LẬP TRÌNH JAVA CĂN BẢN

LẬP TRÌNH JAVA CĂN BẢN

30 tiết lý thuyết

Mục tiêu:

- Tìm hiểu ngôn ngữ Java.
- Xây dựng các ứng dụng đơn giản và các applet với Java

Nội dung môn học

- Chương 1- Tổng quan về ngôn ngữ Java
- Chương 2- Giới thiệu ngôn ngữ Java
- Chương 3- Đối tượng và lớp
- Chương 4- Quản lý lỗi và gom rác
- Chương 5- Lập trình I/O
- Chương 6- Tạo giao diện người dùng
- Chương 7- Mô hình biến cố với AWT
- Chương 8- Lập trình đồ họa với Java
- Chương 9- Tạo Applet
- Chương 10- Lập trình đa luồng

Tài liệu tham khảo

- Bước đầu làm quen Java, Phương Lan
- Java: Lý thuyết và bài tập, Trần Tiến Dũng
- Beginning Java 2 SDK 1.4 Edition, by Ivor Horton; Wrox Press; ISBN: 1861005695
- Thinking in Java, 3/e by Bruce Eckel; Prentice Hall; ISBN: 0131002872

Chuẩn bị môi trường lập trình

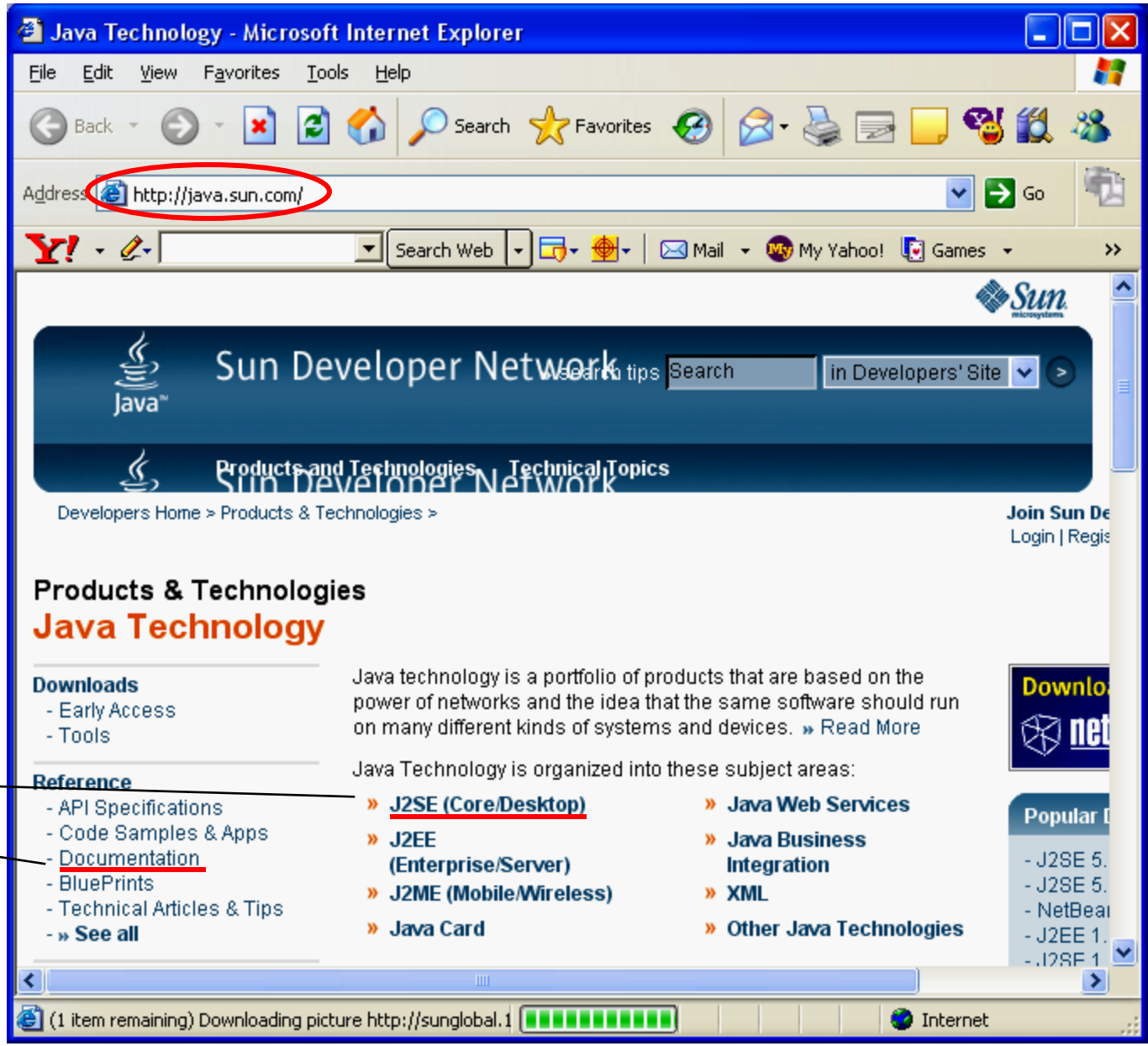
Download Java Software:

Java 2 Standard Edition. Documentation.

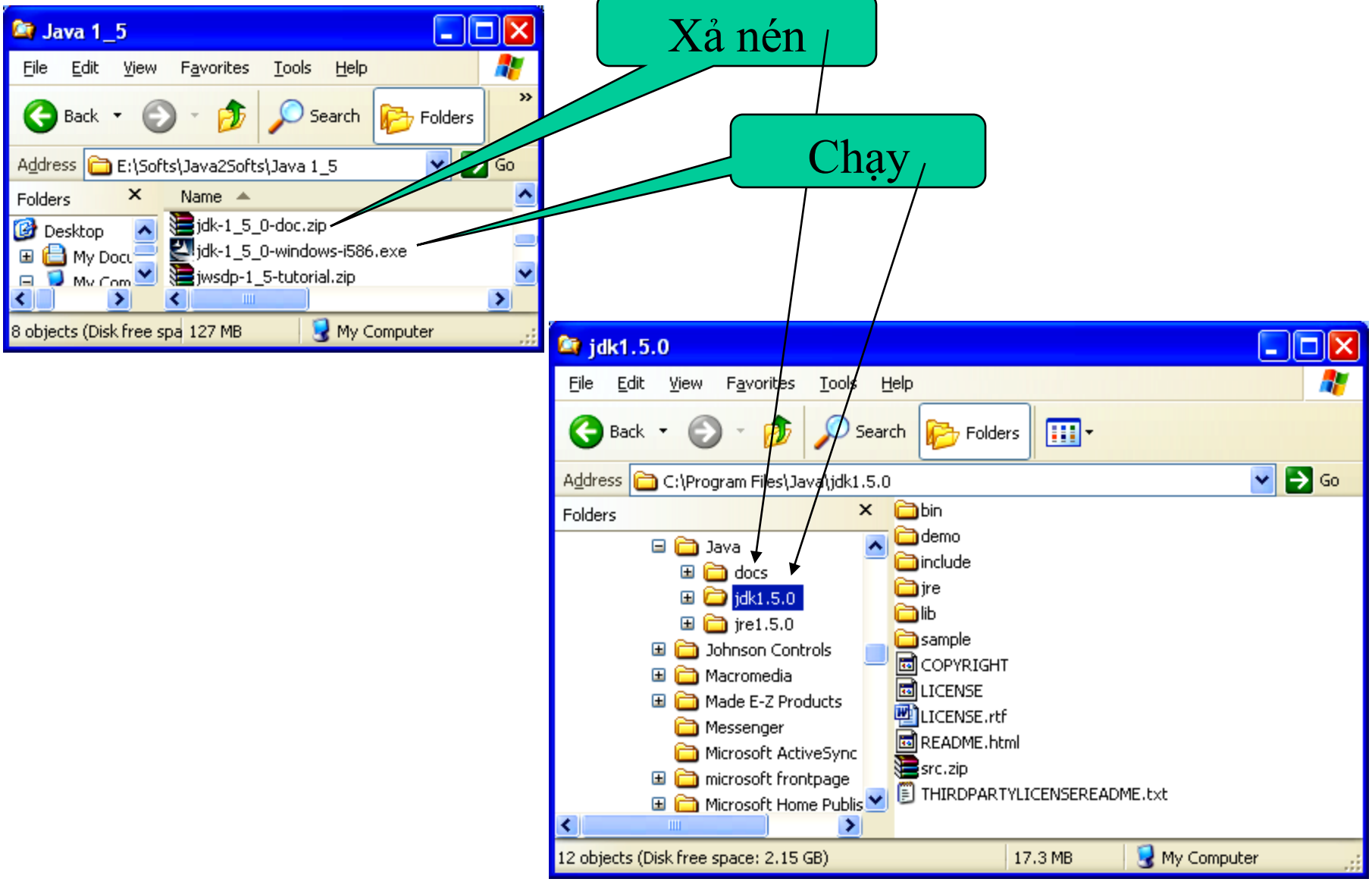
jdk-1_5_0-windows-i586.exe

jdk-1_5_0-doc.zip

Có thể mua đĩa CD



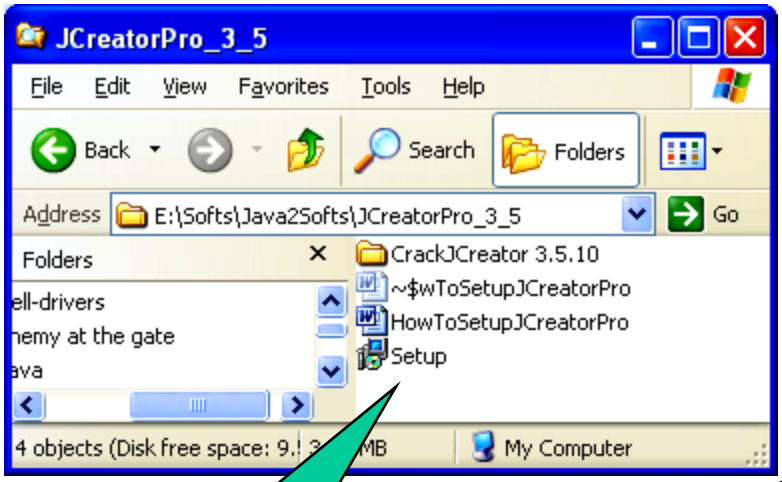
Cài đặt môi trường Java



Download JCreator- Môi trường lập trình

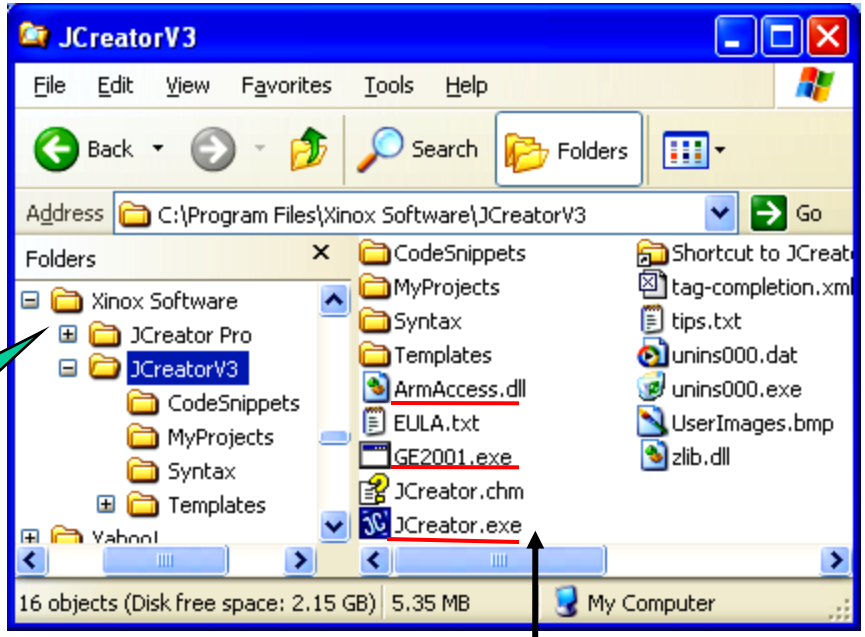
Có thể mua đĩa CD

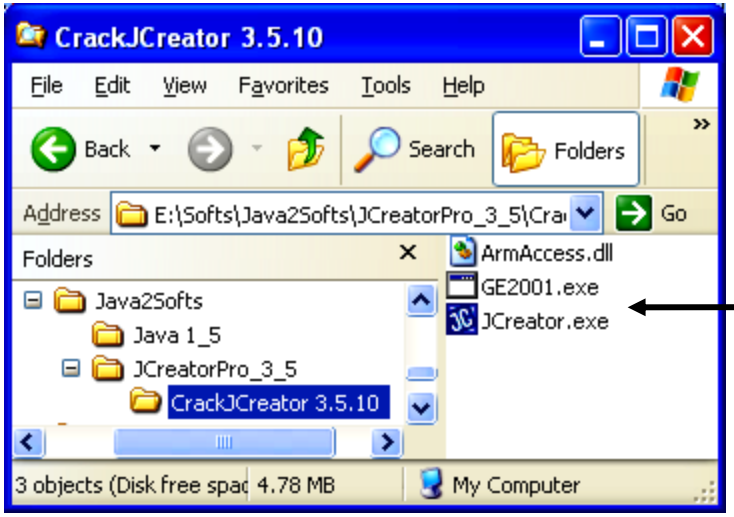
Cài đặt JCreator Pro



Chạy

Các thư mục kết qủa

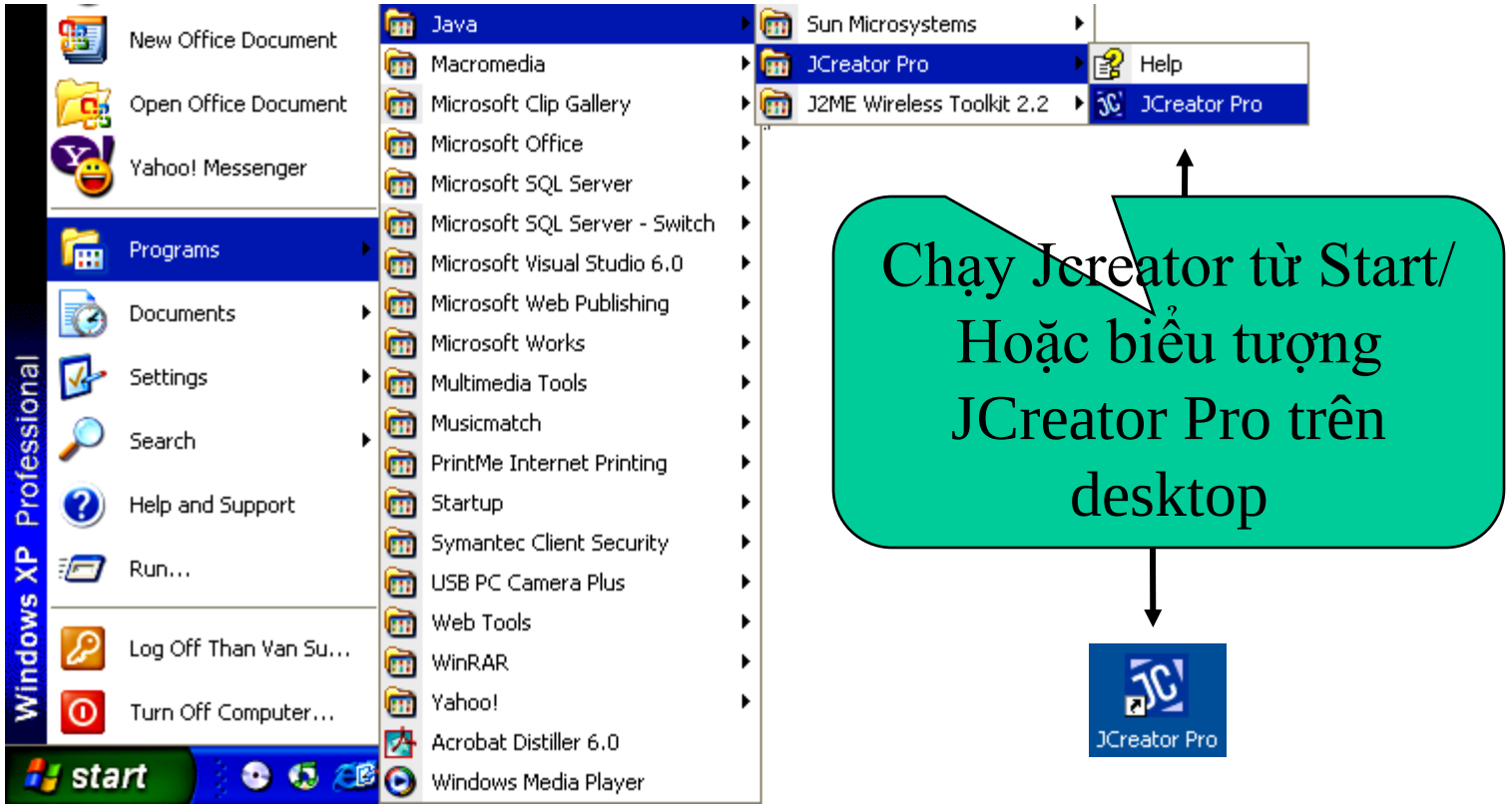




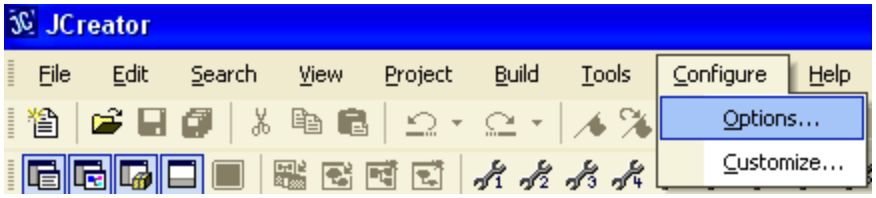
Mở thư mục
Lấy 3 file crack, chép đè vào 3 file đã cài đặt

Cấu hình JCreator-slide 1

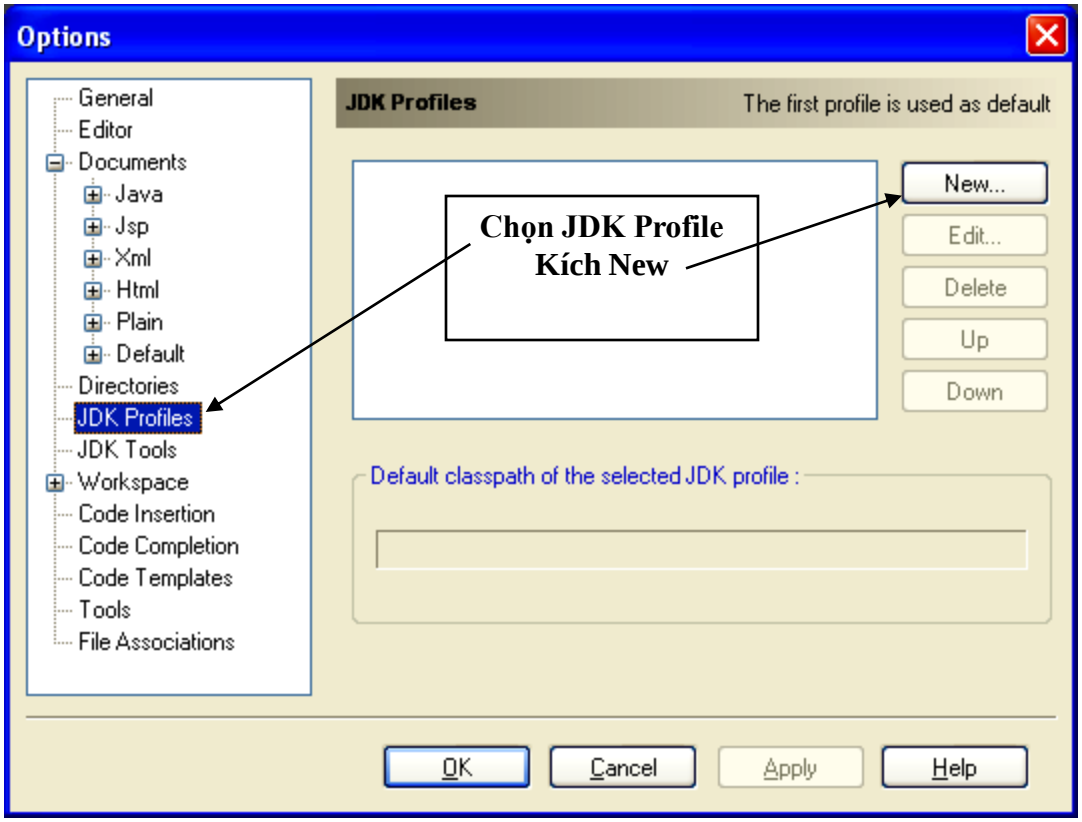
JCreator chỉ là môi trường cho ta xây dựng các ứng dụng Java. Do vậy, ta cần cấu hình cho JCreator bao gồm: Thư mục chứa các lớp của Java, thư mục nguồn, thư mục chứa các file trợ giúp.

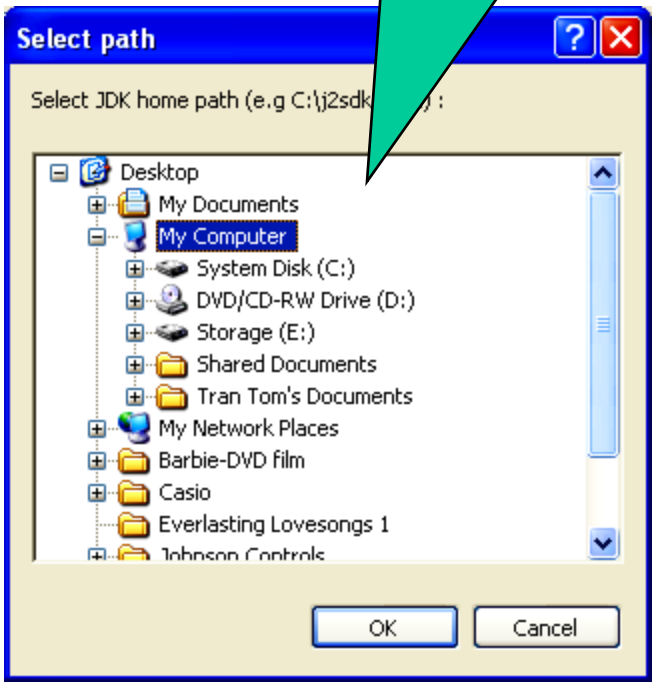


Cấu hình JCreator-slide 2

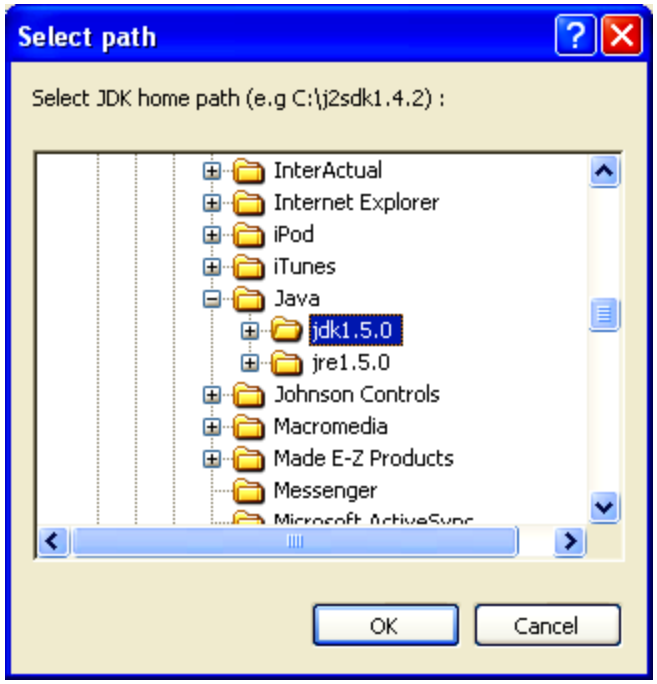


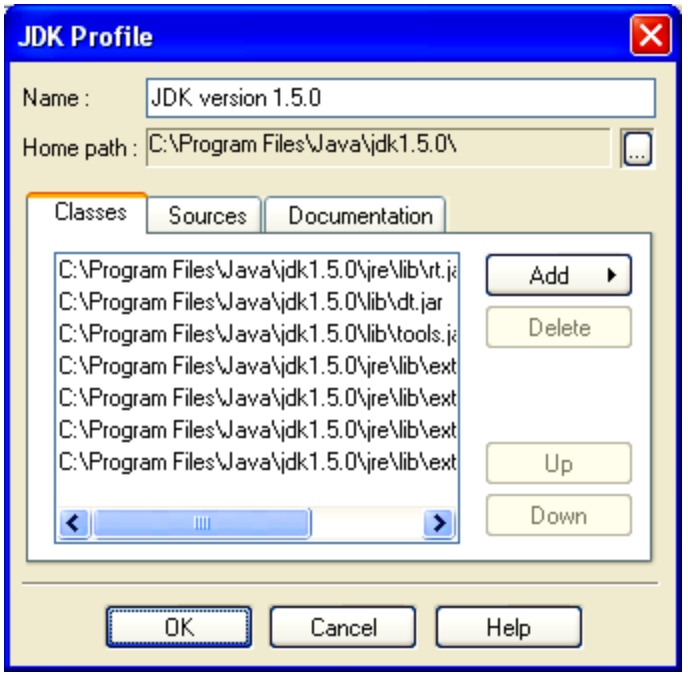
Chọn thư mục đã cài đặt JDK





Cấu hình JCreator-slide 3

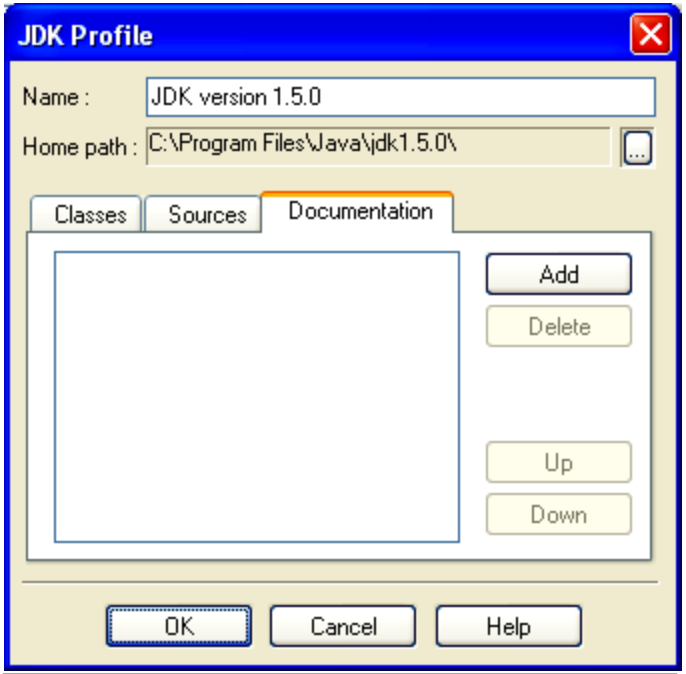




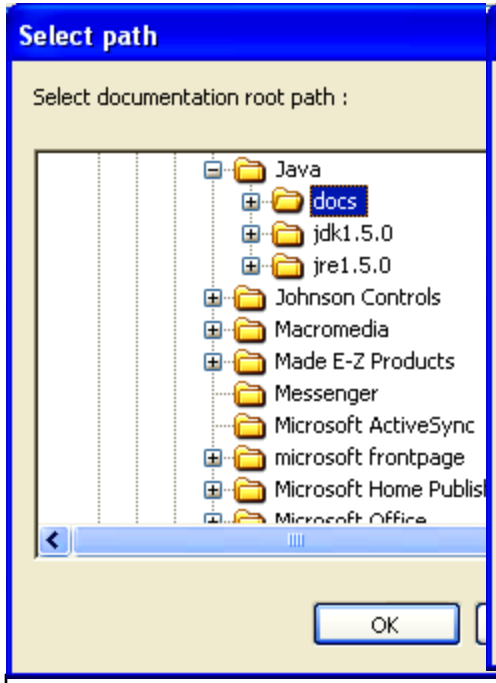
Trong thí dụ này là thư mục C:\Program files\Java\jdk1.5.0

Tất cả các gói của Java jdk 1.5.0 trong thư mục đã được chỉ định sẽ được đưa vào Classes

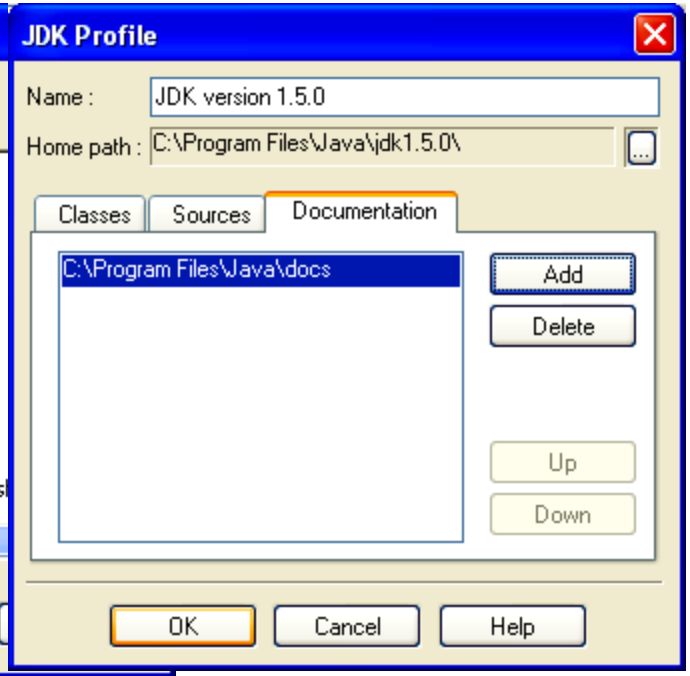
Cấu hình JCreator-slide 4



Chọn thẻ Documentation để chỉ định thư mục chứa tài liệu giúp đỡ của Java. Kích nút Add

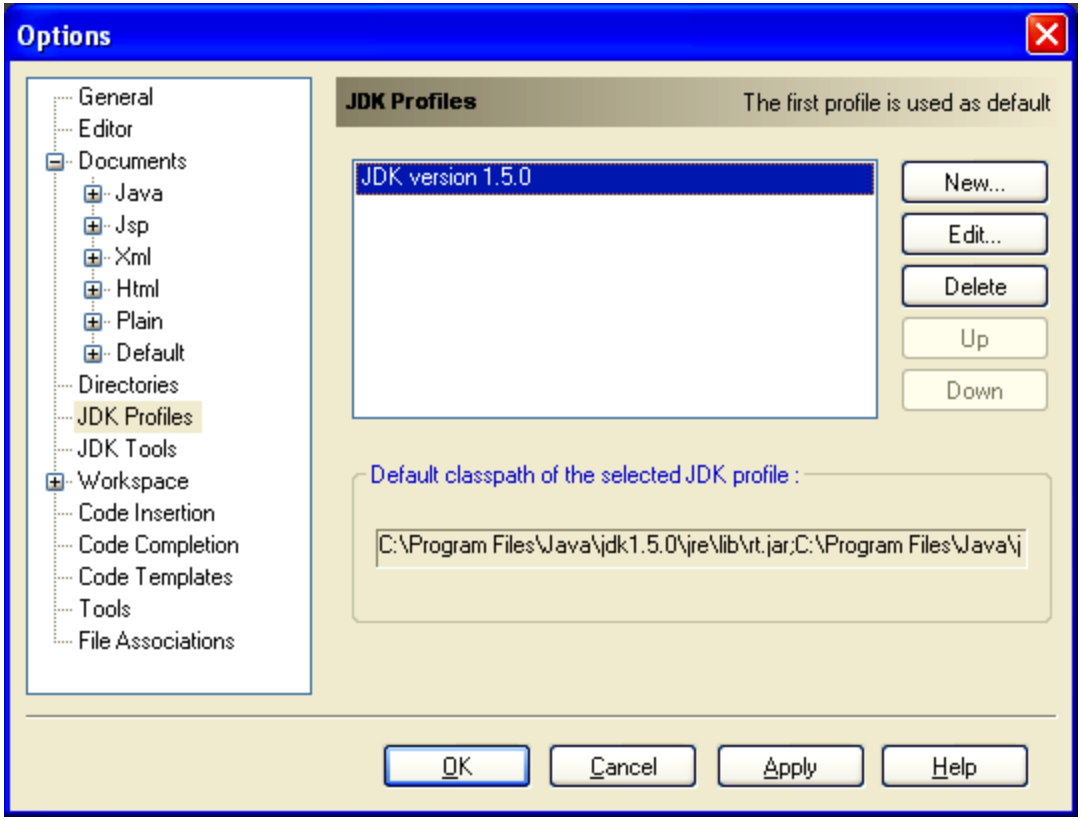


Chọn thư mục Docs là thư mục chứa các file.htm (các file help của Java 2) đã tải về và xả nén ở phần trước)



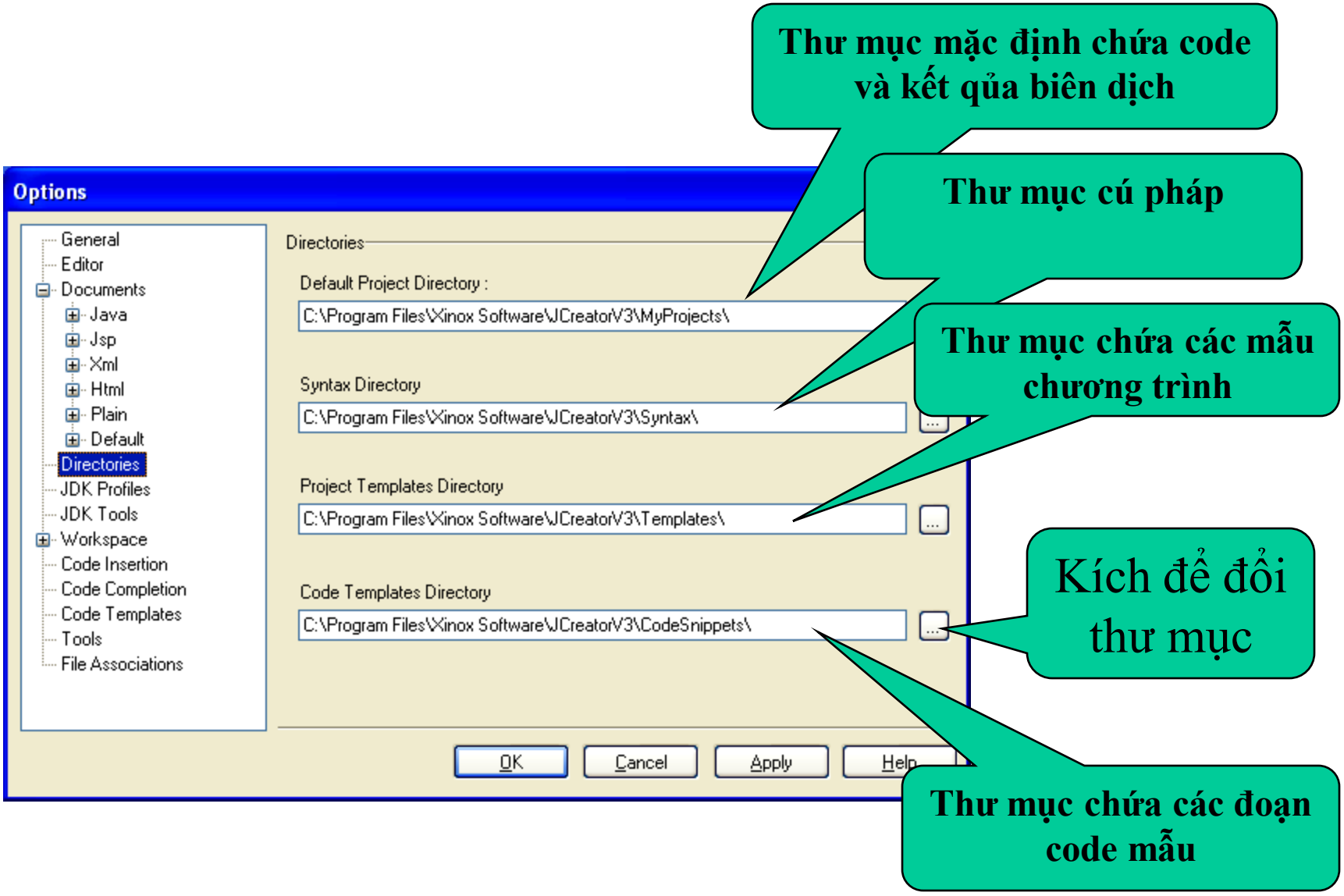
Kết qủa

Cấu hình JCreator-slide 5

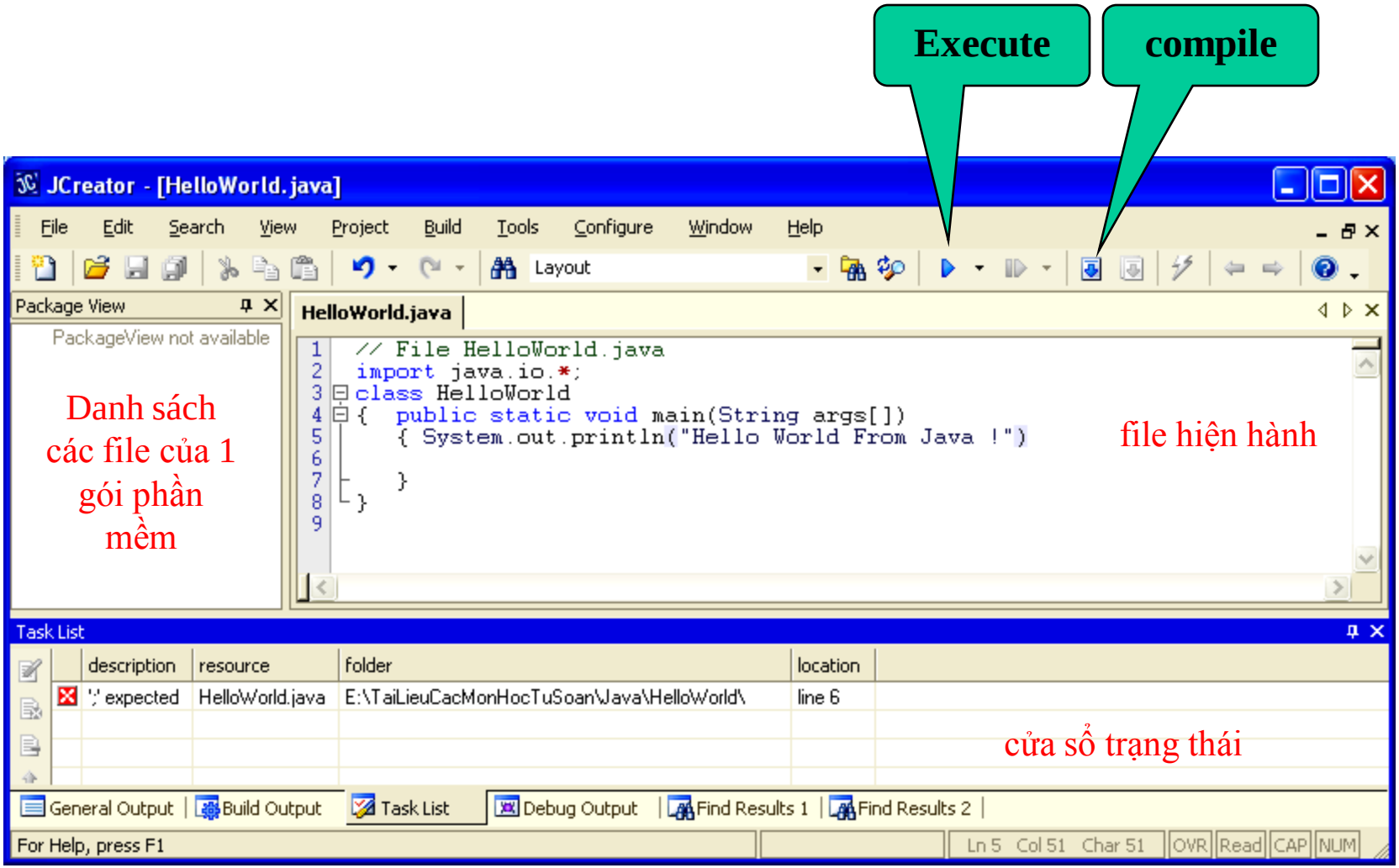


Đến đây, việc cấu hình JDK Profile cho JCreator Pro đã xong,
Nếu muốn cấu hình thư mục thì chọn mục Directories.
Kích Apply rồi kích OK.

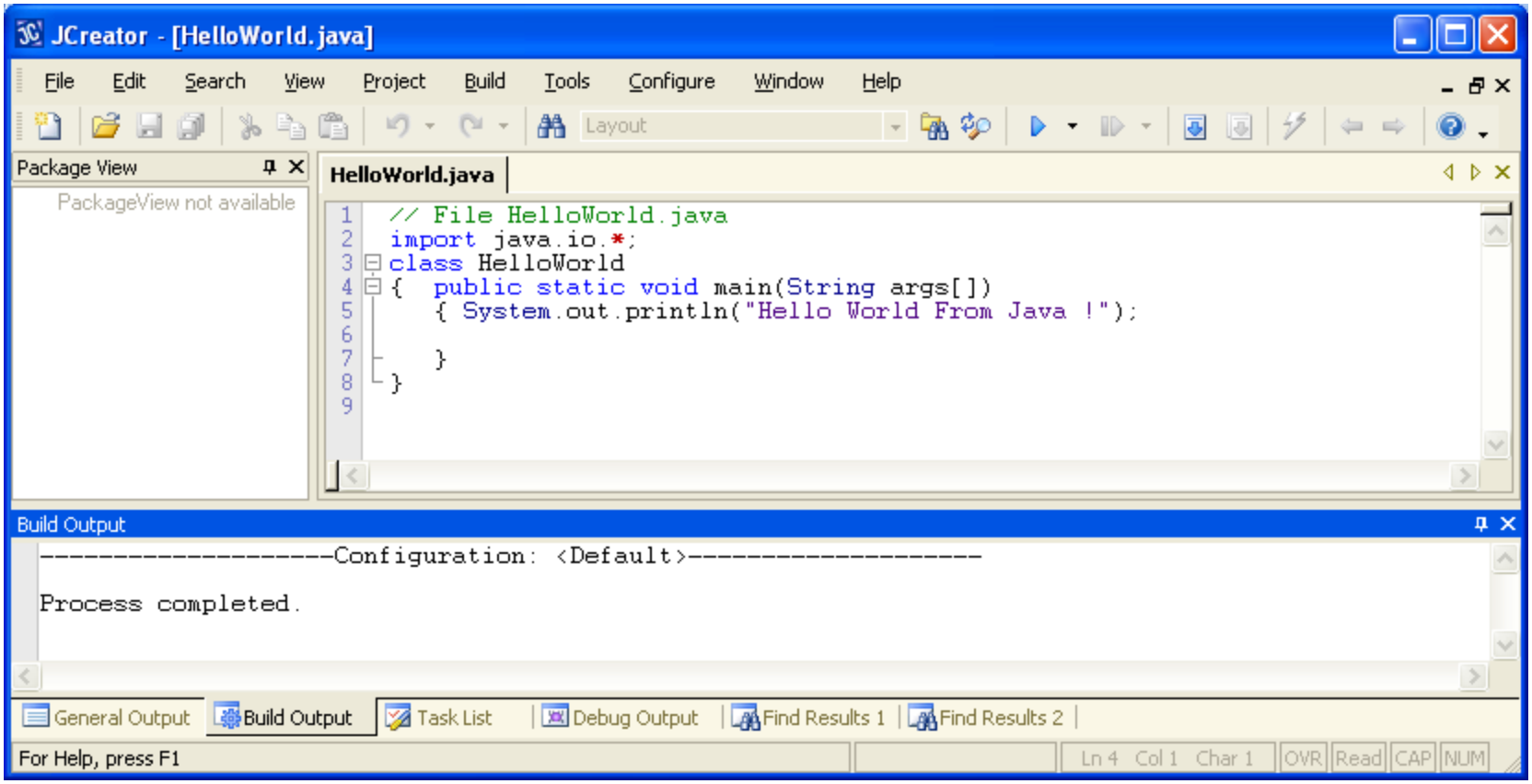
Cấu hình thư mục- slide 6

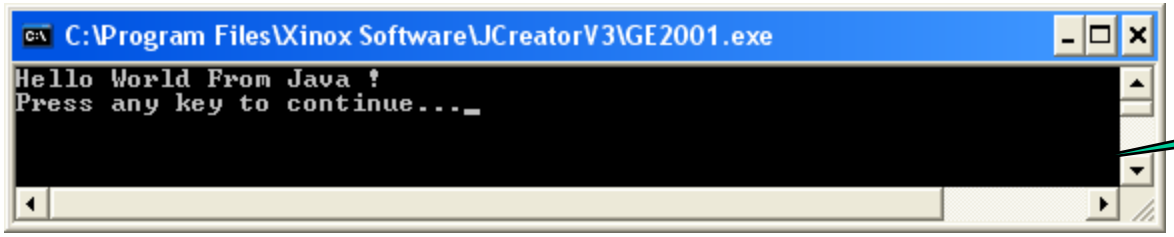


Giới thiệu môi trường JCreatorPro



Biên dịch thành công và chạy chương trình

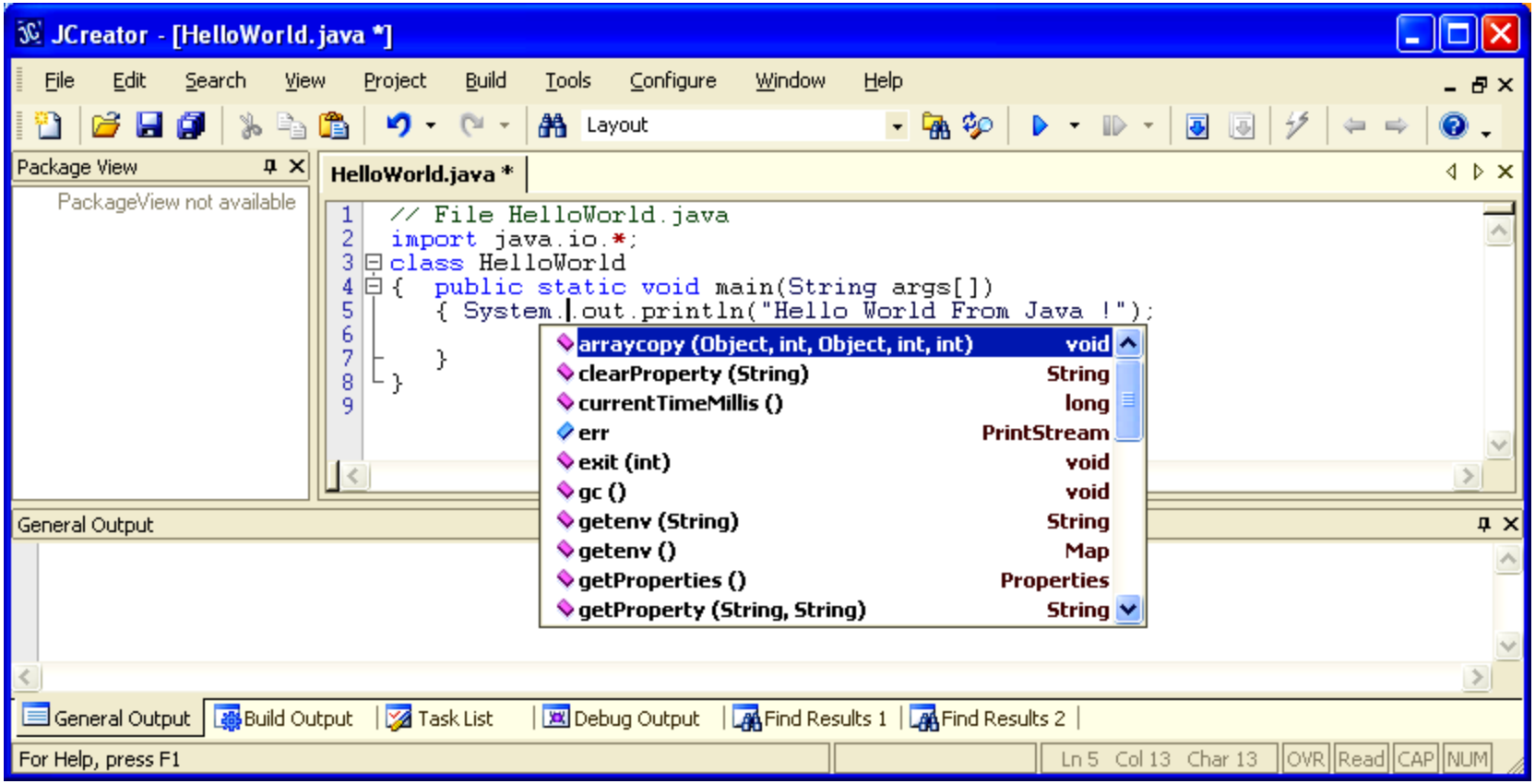




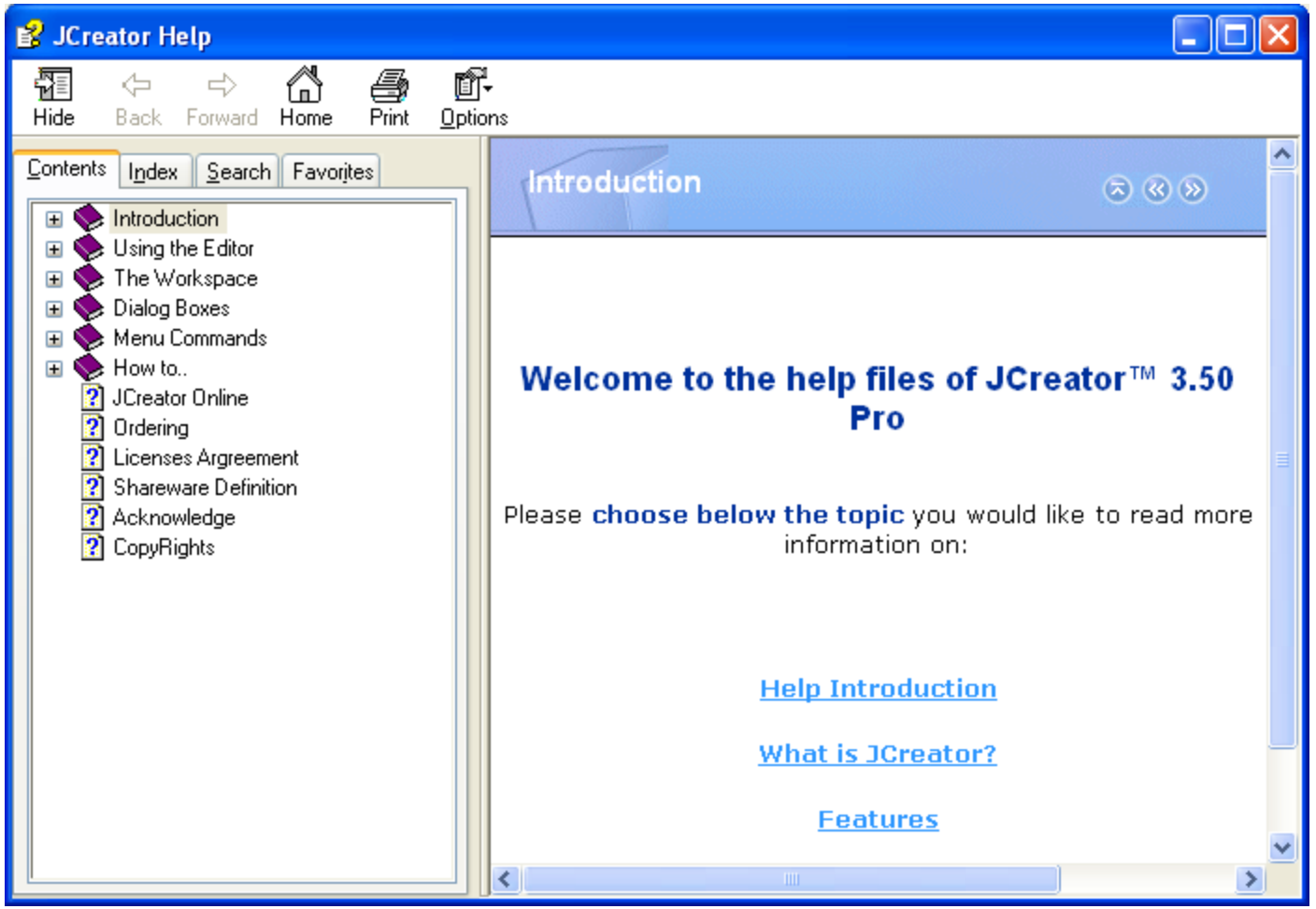
Màn hình kết qủa

Trợ giúp trực tuyến trong JCreator Pro

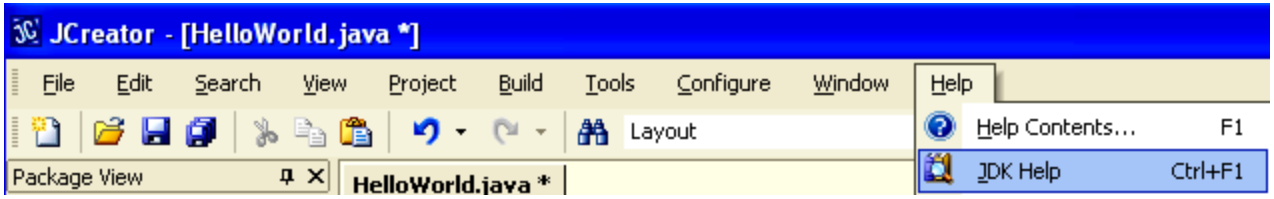
Sau tên đối tượng, ta gõ dấu chấm, các dữ liệu và hành vi public sẽ được JCreator trợ giúp ngay để user có thể chọn

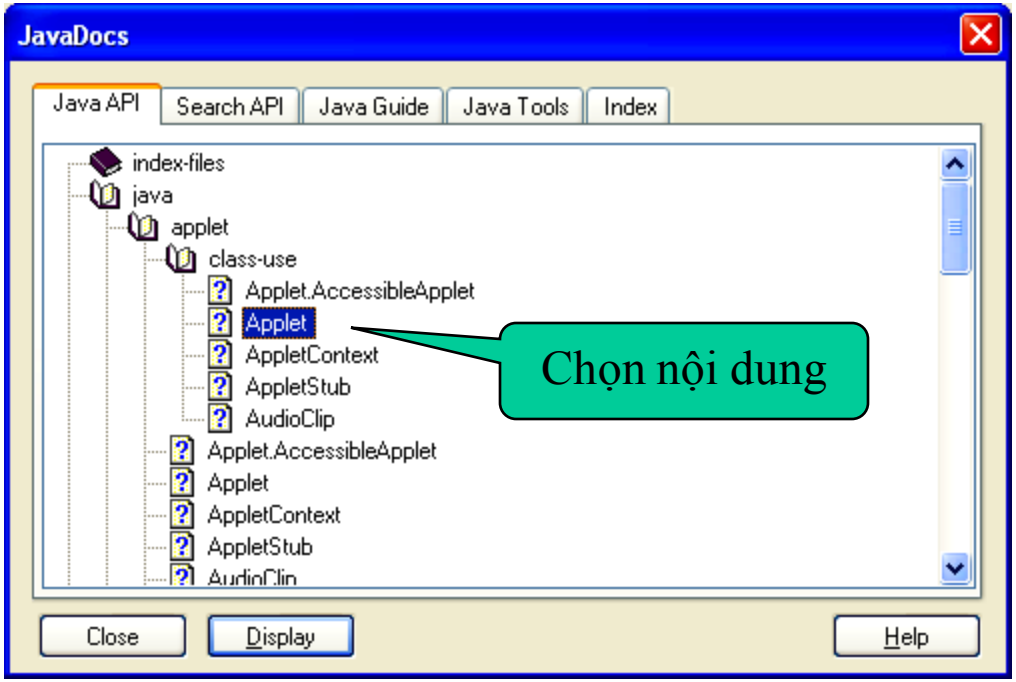


Trợ giúp về sử dụng JCreator

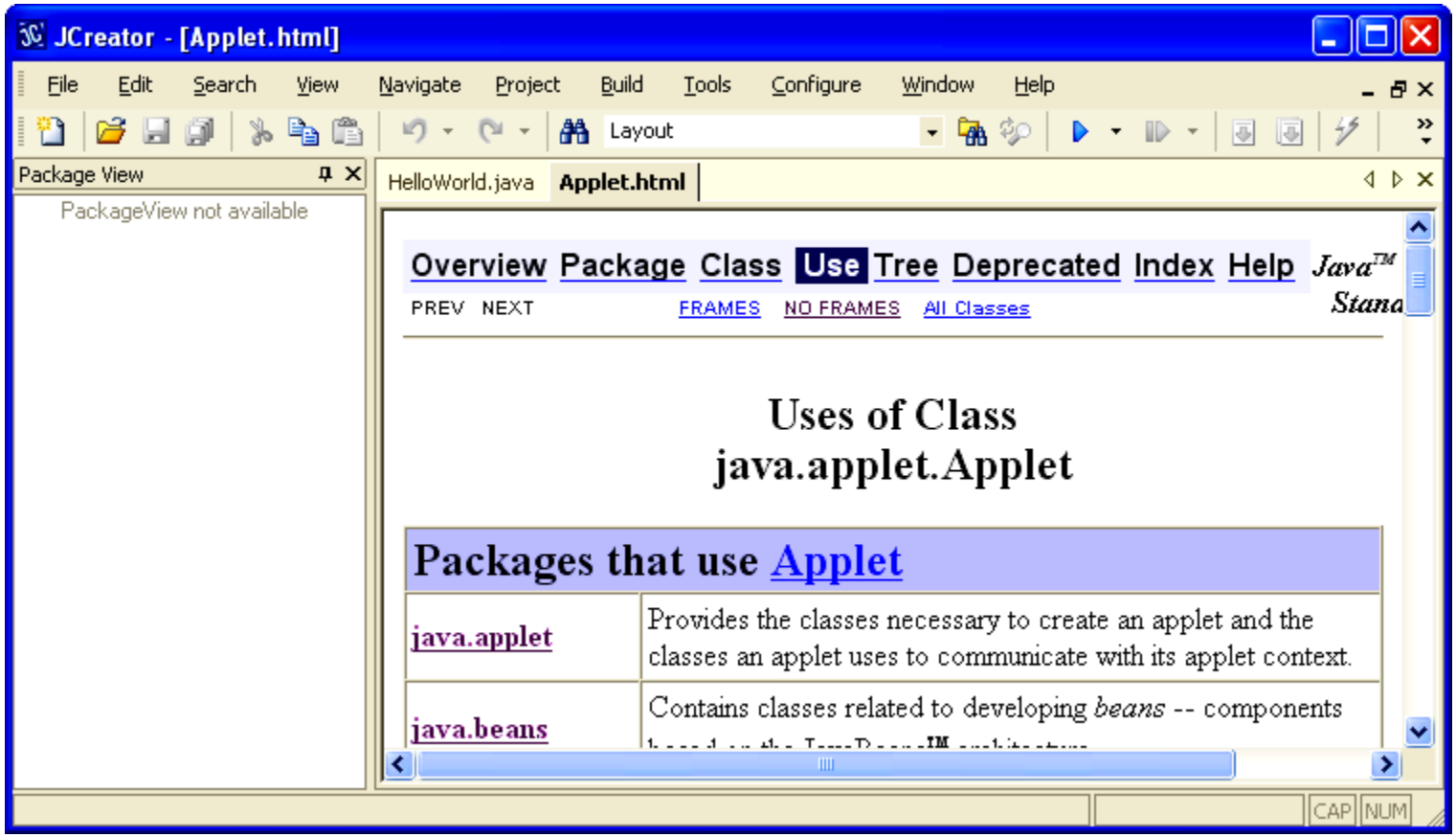


Trợ giúp về sử dụng ngôn ngữ Java



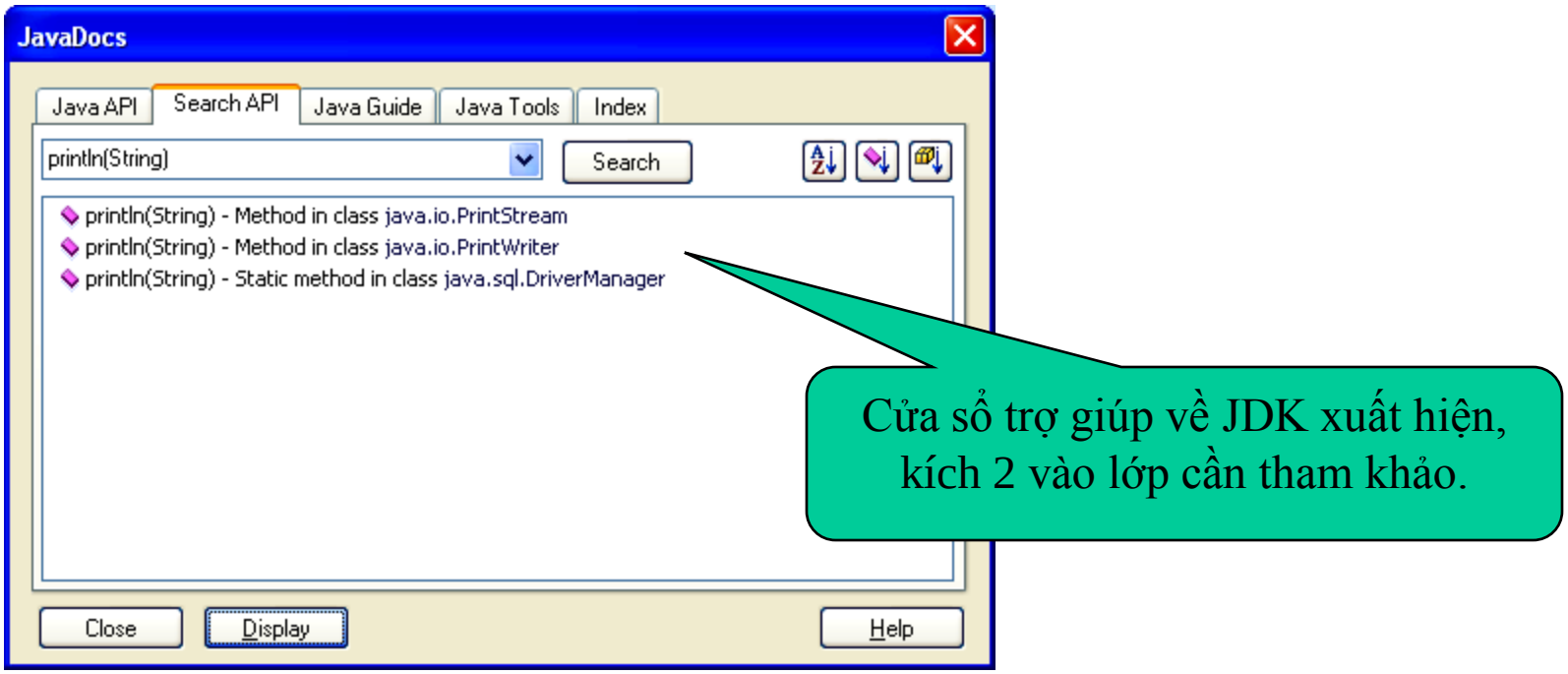


Trợ giúp về sử dụng ngôn ngữ Java



Nôi dung tập tin trợ giúp (file.htm) sẽ xuất hiện trong cửa sổ file

Trợ giúp nóng về 1 method của đối tượng



Chương 1- Tổng quan về JAVA

Mục tiêu: Đến cuối chương bạn có thể

(1) Hiểu những đặc điểm và lợi ích của Java
(2) Hiểu cơ chế thực thi của Java
(3) Hiểu cơ chế biên dịch và chạy 1 chương trình Java
(4) Hiểu cấu trúc cơ bản của một chương trình Java

Nội dung chương 1

1.1- Lịch sử của Java

1.2- Những đặc điểm cơ bản của Java

1.3- Máy ảo Java- JVM

1.4- Môi trường lập trình Java

1.5- Chương trình Java đầu tiên

1.7- Tóm tắt

1.8- Trắc nghiệm và bài tập

1.1- Lịch sử của Java

- Năm 1990, James Gosling, Bill Joy, Patrick Naughton(Sun MicroSystem) phát triển ngôn ngữ Oak nhằm mục đích cài chương trình vào các bộ xử lý của các thiết bị như VCR, lò nướng, PDA (personal data assistant), Oak đòi hỏi:

  - Độc lập cấu trúc nền (phần cứng, OS) do thiết bị có thể do nhiều nhà sản xuất khác nhau (Platform independent)
  - Phải tin cậy tuyệt đối (extremely reliable)
  - Nhỏ gọn, chắc chắn (compact)

Lịch sử Java (tt)

- 1993, TV tương tác và PDA thất bại, Internet và Web bùng nổ, **Sun** chuyển Oak thành một môi trường lập trình Internet với tên dự án là Java.
- 1994, ***HotJava Browser*** của Sun xuất hiện (viết bằng Java chỉ sau vài tháng)→minh họa thế mạnh của các applet cũng như khả năng phát triển nhanh một ứng dụng của Java.

Lịch sử Java (tt)

- Cùng với sự bùng nổ của Internet, Java trở thành phần mềm ưu thế trong việc phát triển ứng dụng chạy trên internet.
- Tuy nhiên, những bản Java đầu chưa đủ mạnh theo yêu cầu của người sử dụng. Thí dụ: Đồ họa trong bản Java 1.0 thô và vụng về hơn so với đồ họa khi được xử lý bằng C hoặc ngôn ngữ khác.
- Tuy lúc đầu Java chưa thành công trong việc xây dựng các ứng dụng mức người dùng, Java vẫn là ngôn ngữ rất thông dụng mức doanh nghiệp, các ứng dụng mức trung gian như: Lưu trữ trực tuyến, xử lý giao tác, giao tiếp với database,… và càng thông dụng trên những cấu trúc nền nhỏ (small platform) như điện thoại di động, PDA.

Java là gì?

- Là một ngôn ngữ OOP đầy đủ, không thể viết 1 ứng dụng hướng thủ tục trong Java.
- Có thể giải các họ bài toán như những ngôn ngữ lập trình khác.
- Cho phép tạo Application hoặc Applet.
- Applet là những chương trình nhỏ chạy trong tài liệu HTML với điều kiện trình duyệt có hỗ trợ Java (như IE, Netscape Navigator, HotJava,…)
- Sử dụng 2 cơ chế: Interpreter | Compiler
- Write code one, run it anywhere, anytime, forever

1.2- Đặc điểm của Java

- Đơn giản(simple). Tương tự như C++ nhưng bỏ bớt các đặc tính phức tạp của C++ như: quản lý bộ nhớ, pointer, overload toán tử, không dùng include, bỏ struct, union
- Hướng đối tượng (OO). Mọi thứ trong Java là đối tượng
- Phân tán (Distributed). Nhắm đến phân bố ứng dụng trên mạng, ứng dụng độc lập platform.
- Mạnh (Robust). Định kiểu mạnh, tường minh, kiểm tra lúc biên dịch và kiểm tra khi thông dịch trước khi thực thi → Giới hạn được lỗi; kiểm tra truy xuất phần tử của mảng, chuỗi lúc thực thi, kiểm tra ép kiểu run-time. Có trình *gom rác – garbage collection*- programmer không cần phải lo toan đến việc hủy đối tượng.

Đặc điểm của Java (tt)

- Bảo mật (Secure): Kiểm tra an toàn code trước khi thực thi, có nhiều mức kiểm tra bảo mật → Môi trường thực thi an toàn

  Mức 1: Mức ngôn ngữ: Nhờ tính bao gói dữ liệu của OOP, không cho phép truy cập trực tiếp bộ nhớ mà phải thông qua method.

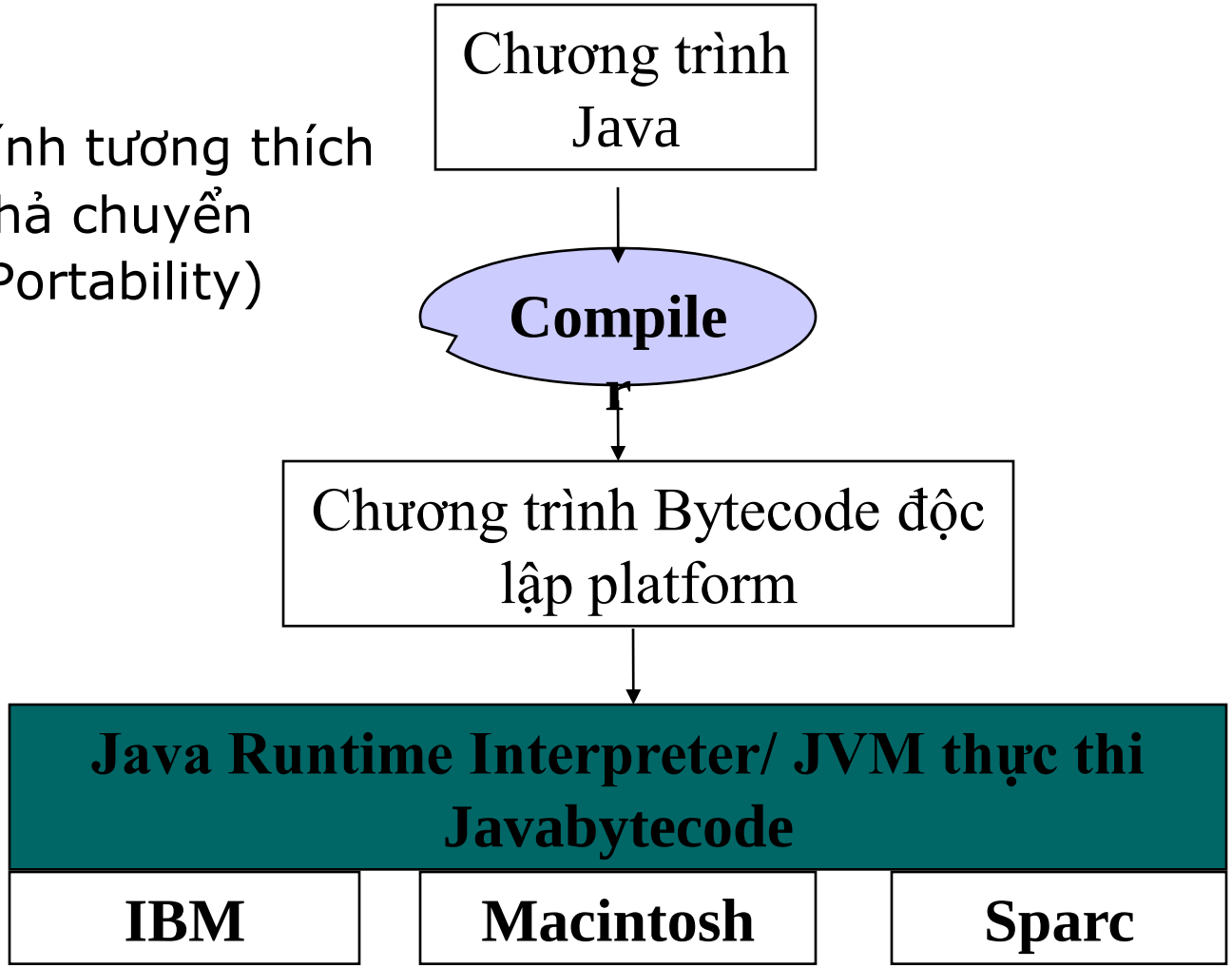
  Mức 2: Mức Compiler, kiểm tra an toàn cho code trước khi biên dịch.

  Mức 3: Mức Interpreter, trước khi bytecode được thực thi, được kiểm tra an toàn.

  Mức 4: Mức Class, các class trước khi nạp được kiểm tra an toàn.

Đặc điểm của Java (tt)

- Tính tương thích khả chuyển (Portability)



Đặc điểm của Java (tt)

Thực thi dạng thông dịch:

(Interpretive execution) Chỉ thị chỉ được dịch sang lệnh máy lúc thực thi

Chương trình độc lập platform → Write Once Run Anywhere (WORA)

- Các file tài nguyên → trình biên dịch ***javac*** → class file độc lập thiết bị
- Class file → trình thông dịch ***java*** → mã máy thực thi, không cần liên kết (link)

→ **Lợi ích**

(1) Java class file có thể được dùng ở bất kỳ flatform nào.

(2) Tính module hóa cao, dùng bộ nhớ tốt hơn với class file hơn là file thực thi vì class file

Đặc điểm của Java (tt)

- Hiệu suất cao (high performance):

  bytecode → native machine code dễ dàng nhờ Just-in-time compiler.

- Đa luồng (multithreading)

  Cho phép lập trình đa luồng (nhiều chương trình đồng hành nhờ lớp Thread : khởi tạo, ngưng 1 luồng, kiểm tra trạng thái của luồng)

  thread: một luồng thực thi của CPU → là 1 chương trình

- Linh động (dynamic): Cho phép tương thích với sự thay đổi của môi trường, Trong CT java có các thông tin run-time → Kiểm tra truy xuất lớp an toán, → an toàn để liên kết caùc lớpvào CT → dynamic

1.3- JVM- Java Virtual Machine

- 5 thành phần của môi trường Java

  (1) Java language

  (2) Bytecode definitions

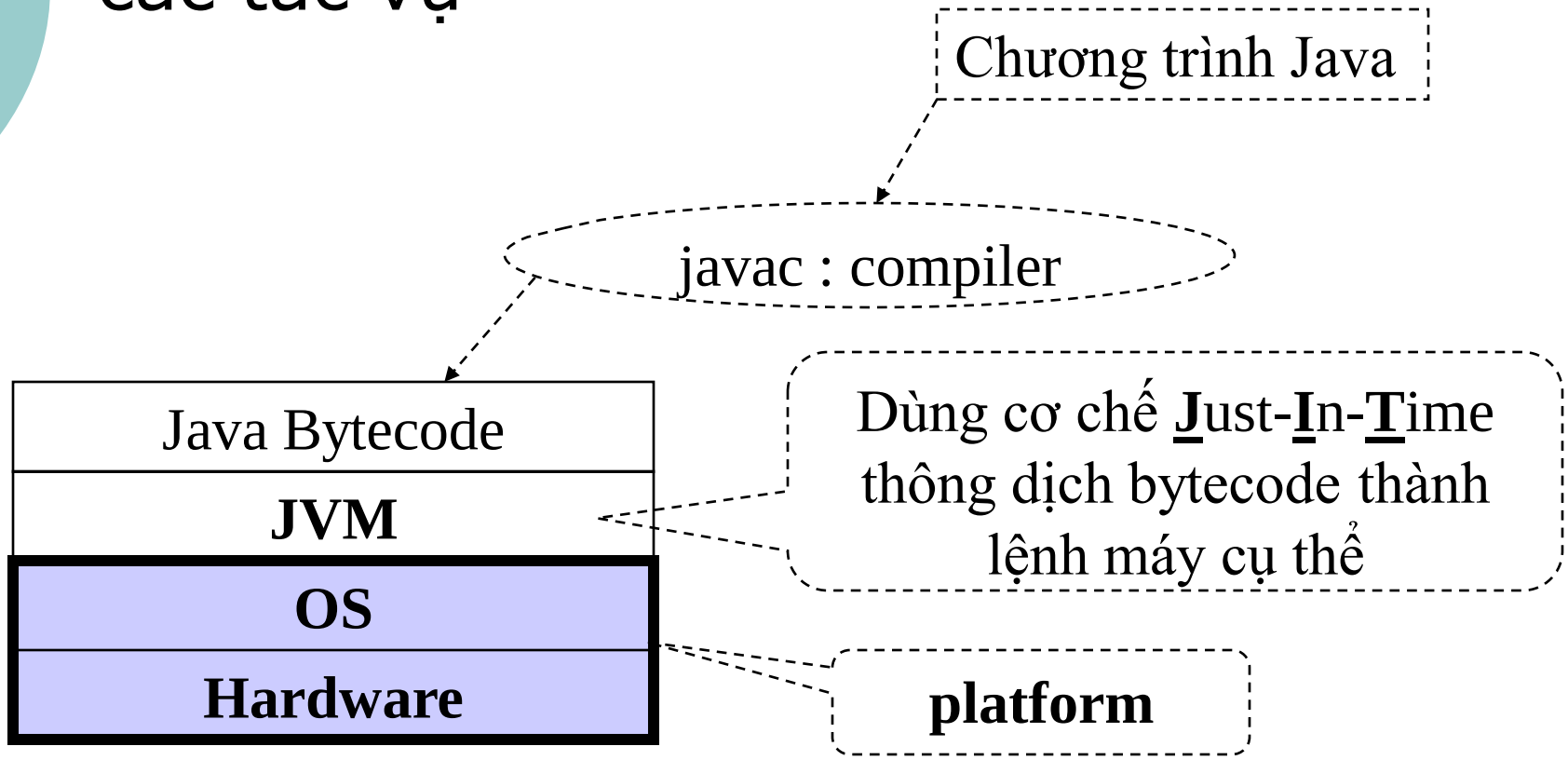
  (3) Java/ Sun Class libraries

  (4) The Java Virtual Machine

  (5) The structure of .*class* file
- JVM là trung tâm của Java
- Các thành phần dẫn đến sự thành công của Java: Bytecode definitions, the structure of .*class* file, JVM.

1.3.1- JVM là gì?

- Là một phần mềm giả lập một máy tính trong đó : có tập lệnh định nghĩa các tác vụ

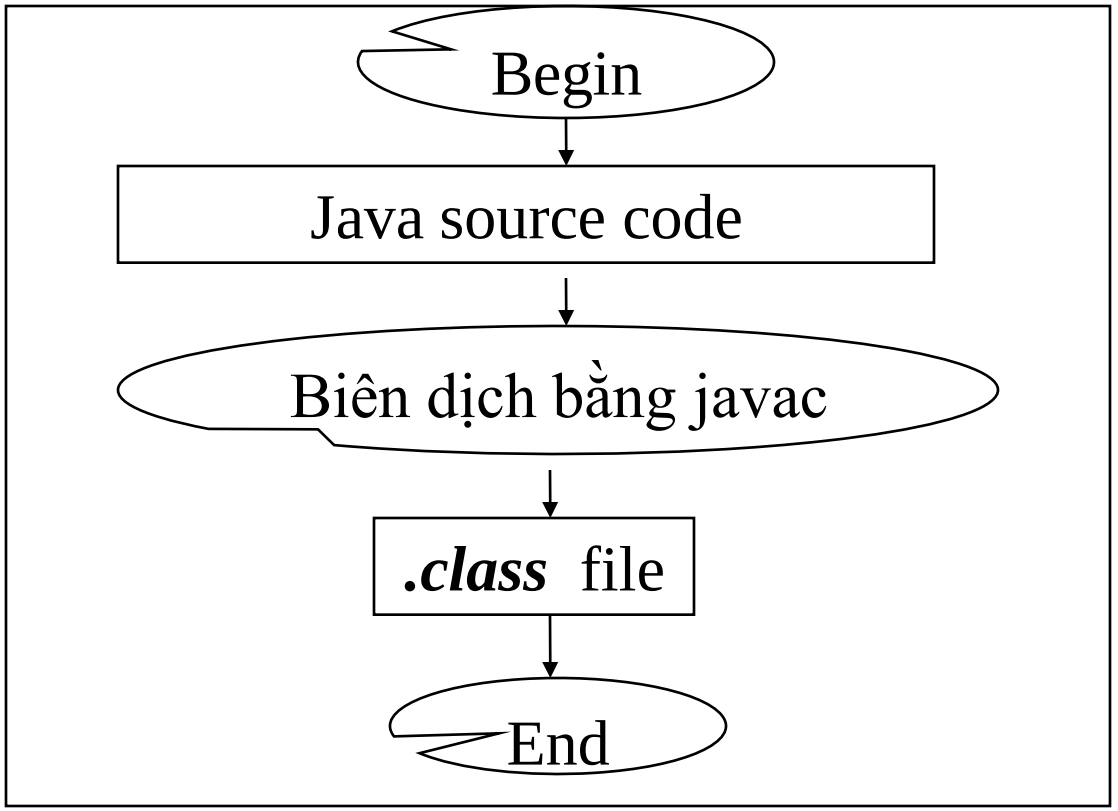


JVM là gì? (tt)

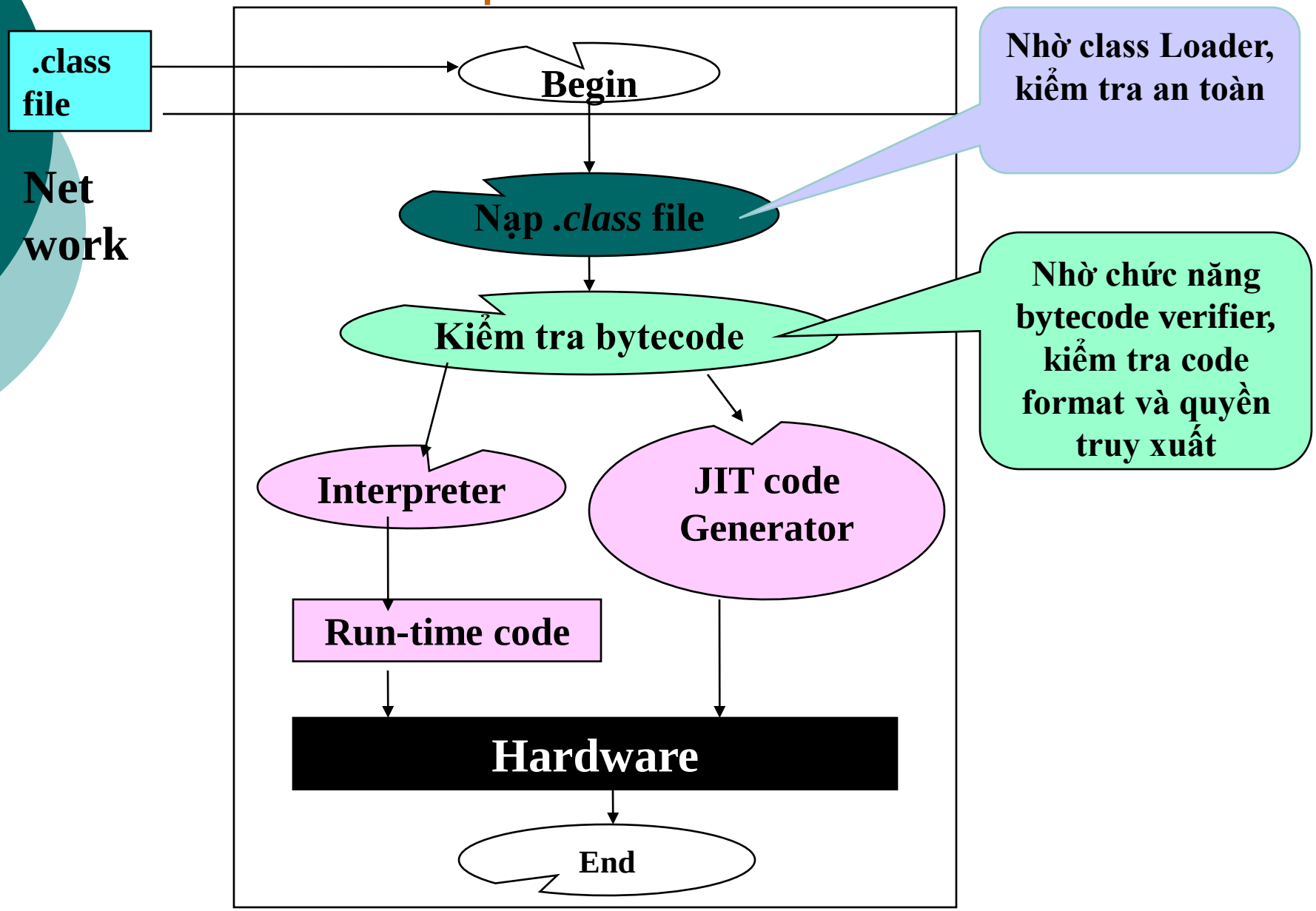
- JVM tạo ra 1 hệ thực thi phụ thuộc platform bao gồm các tác vụ:
  (1) Nạp *.class* file
  (2) Quản lý bộ nhớ
  (3) Thực thi gom rác

1.3.2- JRE-Môi trường run-time của Java

- JRE:
- Hai giai đoạn của 1 Java application: Compile-time, Run-time
- **Compile-time phase**: Viết và biên dịch chương trình



JRE- Run-time phase



1.3.3- Trình gom rác- Garbage Collection

- **Heap**: Vùng nhớ chia sẽ thông tin giữa các qúa trình. Với C, C++, Pascal, programmer phải tự quản lý vùng nhớ cấp phát động này bằng các hàm cơ bản.
- **Cơ chế quản lý heap**

  Heap được quản lý bằng 2 danh sách:

  Free block list và **Allocated Block List**.

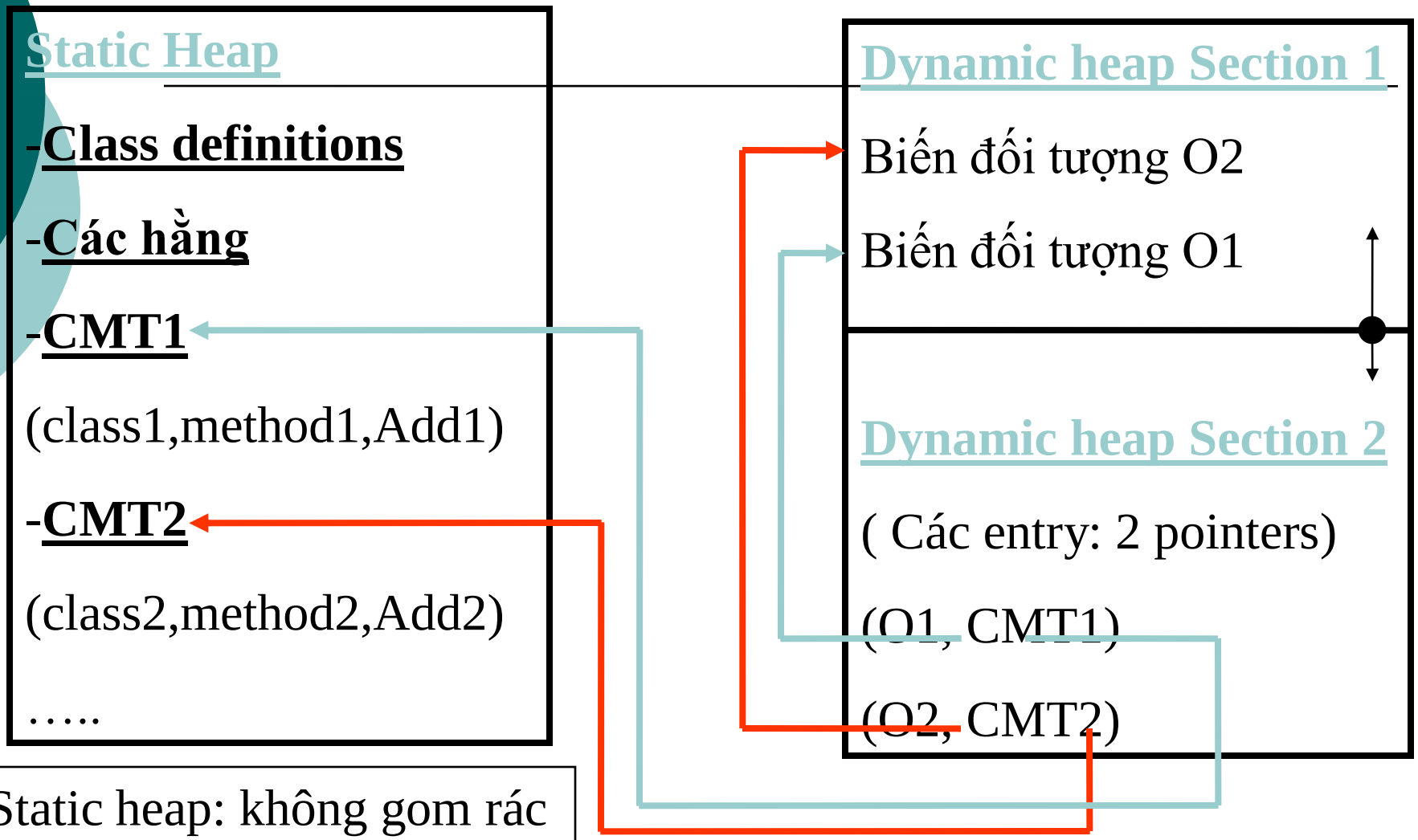
  - Cách cấp phát: **"first-fit block"**

  - Khi khối bộ nhớ được yêu cầu lớn hơn khả năng của các khối tự do: **Compaction** - dồn vùng nhớ để tạo ra vùng lớn hơn.
- **Heap trong Java** : 2 heap

  heap cấp phát tĩnh và heap cấp phát động.

Trình gom rác (tt)

Dynamic heap: có gom rác



Static heap: không gom rác

CMT: class method table

Section 2: Theo dõi hoạt động của các đối tượng

Cơ chế gom rác

Cơ chế cấp bộ nhớ

1/ Nhận yêu cầu cấp bộ nhớ

2/ **if** (*Free-Block list đủ*) cấp bộ nhớ cho yêu cầu (First-fit)

3/ **else if** (*máy rảnh*) thực thi gom rác

4/ **else** ứng dụng phải gọi tường minh tác vụ gom rác: *System.gc();*

Trình gom rác được ấn định độ ưu tiên rất thấp→ Gọi tường minh có ý nghĩa chấp nhận ứng dụng này tạm dừng để chờ gom rác.

Cơ chế gom rác (chỉ gom rác ở Dynamic heap)

1/ Xem đối tượng nào không có entry trong section2→ Không còn dùng đối tượng này nữa.

2/ Garbage Collector sẽ gọi method finalize() để thu tài nguyên của đối tượng (file, stream kết hợp, bộ nhớ)

1.4- Môi trường lập trình Java

- **JDK**- Java Development Kit- Bộ công cụ phát triển ứng dụng Java bao gồm 4 thành phần:

(1) Classes

(2) Compiler

(3) Debugger

(4) Java Runtime Environment

Hiện nay đã có bản Java 1.6(Beta)

Các công cụ chính của môi trường Java

Trong thư mục BIN của JDK (sau khi cài đặt) có:

- **Javac.exe** : Java Compiler:

  Dịch source code → Independent Bytecode
- **Java.exe** : Thực thi class file trong JVM
- **Appletviewer.exe** : cho phép chạy applet mà không cần Browser.

1.5- Chương trình java đầu tiên

Yêu cầu: Viết chương trình xuất chuỗi "Hello world from java!" ra màn hình.

- **Phân tích**:

  - Cần tạo 1 lớp có chức năng xuất chuỗi này (lớp HelloWorld) . Vì chức năng của chương trình đơn giản -> lớp này chỉ có 1 hành vi main(…), nội dung hành vi là xuất chuỗi được yêu cầu.

- **Cách làm 1**: Viết code bằng 1 editor, về dấu nhắc Command Prompt biên dịch, chạy chương trình.
- **Cách làm 2**: Nhờ 1 IDE như Jcreator, JPadPro, Jbuilder, … cho phép vừa viết code vừa thực thi.

HelloWorld - Notepad

File Edit Format Help

Dùng NotePad, biên dịch dòng lệnh

```
// File Helloworld.java
import java.io.*;
class Helloworld
{   public static void main(string args[])
    { System.out.println("Hello World From Java !");
    }
}
```

Lưu trữ với tên HelloWorld.java- tên lớp là tên file

```
C:\PROGRA~1\JavaSoft\JRE\1.2\bin>javac e:\BaiGiang2004\java\HelloWorld.java
e:\BaiGiang2004\java\HelloWorld.java:1: ';' expected.
import java.io.*
            ^
1 error
```

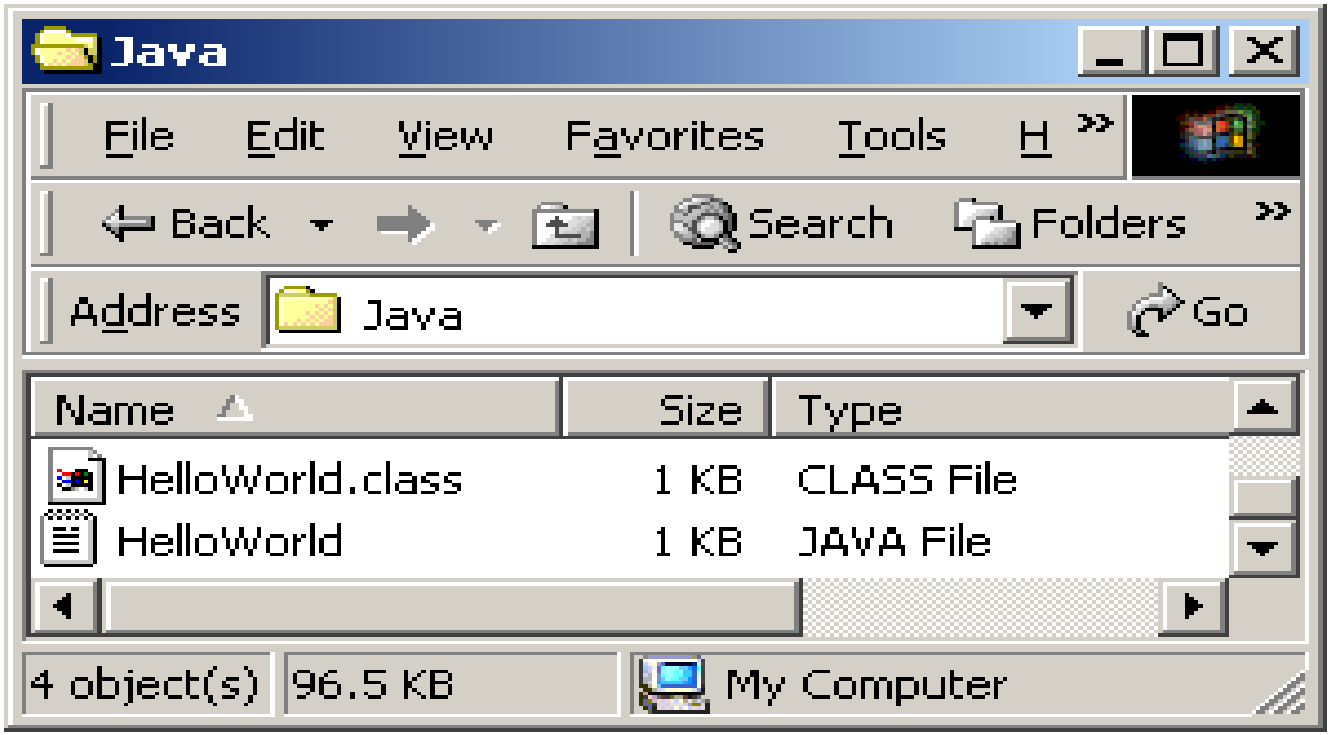
Lỗi thiếu dấu chấm phẩy, code trên đã sửa

```
E:\BaiGiang2004\Java>javac HelloWorld.java
E:\BaiGiang2004\Java>java HelloWorld
Hello World From Java !
E:\BaiGiang2004\Java>
```

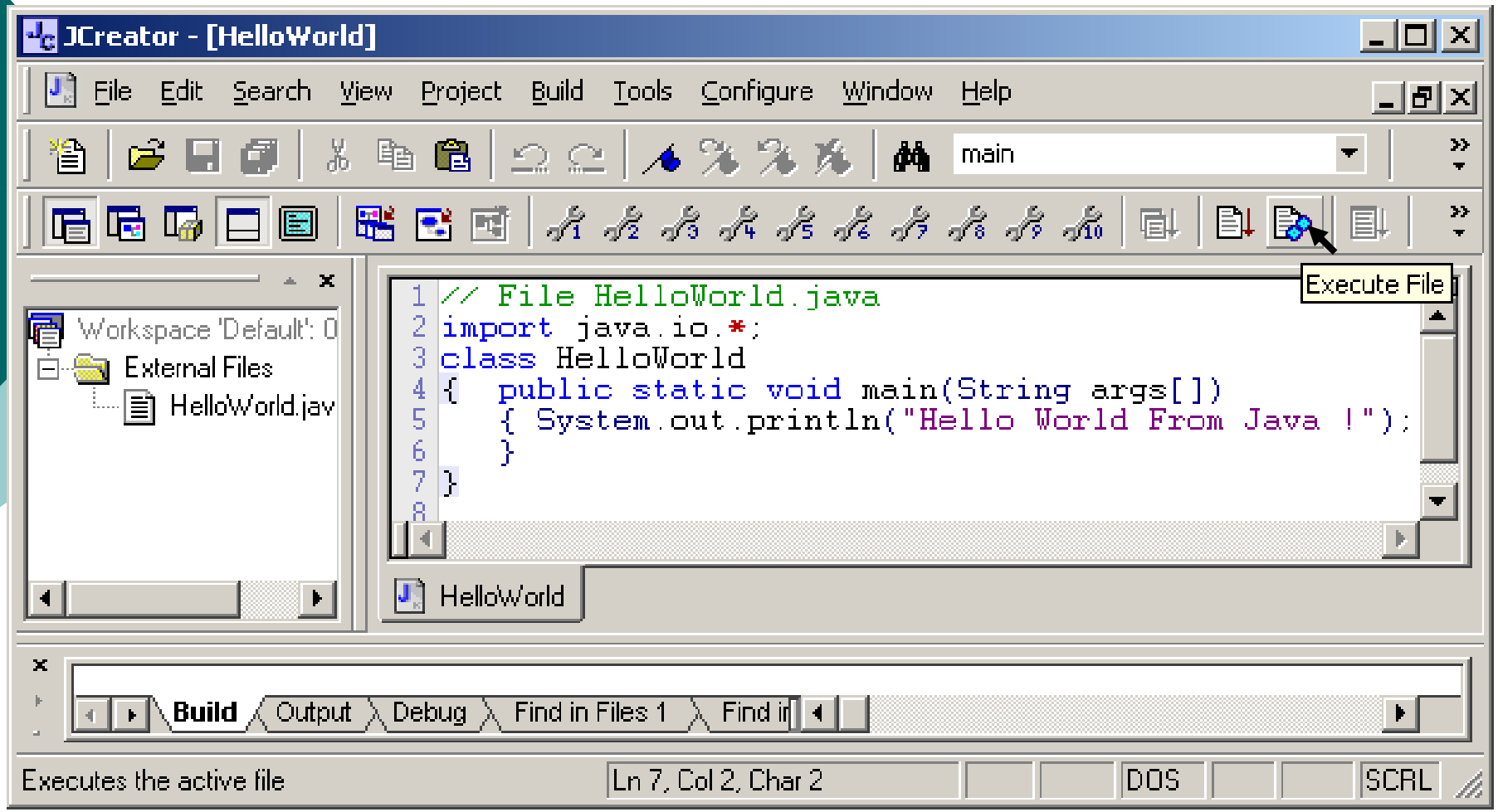
Hoặc biên dịch với thư mục hiện hành là thư mục chứa source code .
Biên dịch thành công và chạy ứng dụng

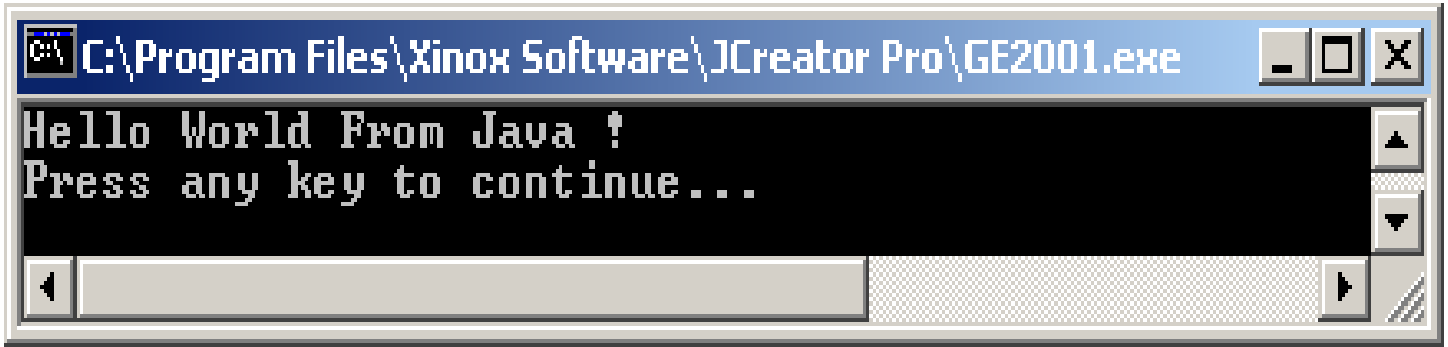
Chú ý: Tên file .java có tính chất case-sensitive

Kết qủa biên dịch



Dùng JCreatorPro





1.6- Tóm tắt

- Java là ngôn ngữ OOP chủ yếu được dùng để phát triển các ứng dụng Internet với các đặc điểm: Simple, OO, Distributed, Robust, Secure, System Structure neutrality, Portability, Interpretive execution, High Performance, Multithreading, Dynamic.
- JVM là trái tim của Java.
- JDK là bộ công cụ hỗ trợ lập trình.
- JDK cung cấp một số công cụ được để trong thư mục BIN khi cài đặt JDK gồm 3 chức năng chính: javac: trình biên dịch, java: JVM, appletviewer.
- Tài liệu API của Java rất cần cho người lập trình java vì chứa các tài liệu hướng dẫn về các lớp (class), các gói phần mềm (package), các giao tiếp (interface)

1.8- Trắc nghiệm

Câu 1

Java hỗ trợ những đặc điểm nào sau đây?

A) OO

B) Độc lập platform

C) Bảo mật

d➔ D) Tất cả các đặc điểm trên

Câu 2

Source code của java có tên mở rộng là :

b➔

A) .class
B) .java
C) .com
D) Tất cả đều sai.

Câu 3

Java source code được trình biên dịch java biên dịch thành:

a➔

A) Bytecode
B) Executable code
C) Machine code
D) Tất cả đều sai

Câu 4

Trình nào sẽ chuyển đổi java source code thành file *.class*

a➔ A) javac

B) java

C) appletviewer

D) Tất cả đều sai

E) Tất cả đều đúng

Câu 5

Chương trình nào thực thi 1 class file trong JVM

A) javac

B) java

b➔

C) appletviewer

D) Tất cả đều sai

E) Tất cả đều đúng

Câu 6

Trình nào cho phép ta chạy applet bên ngoài browser

a➔

A) appletviewer
B) WWW
C) java
D) Tất cả đều sai
E) Tất cả đều đúng

Câu 7

Chọn phát biểu sai.

a➔

a) Có thể xây dựng một ứng dụng hướng thủ tục trong Java.

b) Không thể xây dựng một ứng dụng hướng thủ tục trong Java

Câu 8

Cơ chế quản lý bộ nhớ của Java gồm 2 heap, static heap và dynamic heap. Chọn các phát biểu sai.

a➔ a- Static heap chứa các định nghĩa class + dữ liệu của các đối tượng + code chương trình.

b- Static heap chứa các định nghĩa class + code chương trình.

c➔ c- Dynamic heap chứa các định nghĩa class + dữ liệu của các đối tượng + code chương trình.

d➔ d- Dynamic heap chứa các định nghĩa class + code chương trình.

Câu 9

Chọn các phát biểu đúng.

a➔ a- Dynamic heap trong Java chứa dữ liệu của đối tượng.

b➔ b- Dynamic heap trong Java chứa thông tin về mối quan hệ giữa đối tượng trong dynamic heap và code trong static heap.

c- Dynamic heap trong Java chứa thông tin về mối quan hệ giữa đối tượng và code trong dynamic heap.

d- Dynamic heap trong Java chứa thông tin về mối quan hệ giữa đối tượng trong static heap và code trong static heap.

Câu 10

Chọn các phát biểu sai

Các đối tượng của Java

a➔ a) Được cấp phát động nên ta cần chủ động trả bộ nhớ khi không dùng đến đối tượng nữa.

b) Được cấp phát động, ta không cần chủ động trả bộ nhớ khi không dùng đến đối tượng nữa.

c➔ c) Máy ảo không tự động thu hồi bộ nhớ đối với những đối tượng không dùng đến đối tượng nữa.

Câu 11

Có thể dùng một trình editor chuẩn bất ký để viết code java.

a- Đúng

a➔ b- Sai

Câu 12

Cơ chế nào cho phép 1 ứng dụng Java độc lập với platform (chọn 2)

a➔ a- Mỗi platform có một trình Java.exe riêng để thông dịch file.class.

b➔ b- file.class có cấu trúc độc lập với flatform.

c- Mỗi nhà cung cấp hệ điều hành tạo ra các cách riêng để chạy ứng dụng Java

Bài tập

- Viết chương trình xuất ra màn hình các thông tin sau

"Hello! I'm <your name>."

"This is my first java program."

"This is common technology today."

"I will work hard to enhance my skill in Java"

Chương 2- Ngôn ngữ JAVA

Mục tiêu

- Biết cách định nghĩa 1 tên trong java
- Biết các từ khóa của java.
- Hiểu các kiểu dữ liệu cơ bản của java.
- Nhận biết được cú pháp java gần hoàn toàn giống C.
- Giải thích được cơ chế điều khiển chương trình
- Biết các đặc tính về mảng với java
- Sử dụng được các hàm toán trong gói java.lang
- Sử dụng được các hàm nhập xuất dữ liệu cơ bản.

Nội dung

2.1- Chú thích trong java

2.2- Từ khóa của java- Cách đặt tên

2.3- Kiểu cơ bản trong java

2.4- Biến: Gán trị và khởi tạo.

2.5- Toán tử- Operators

2.6- Gói *java.lang*

2.7- Cấu trúc điều khiển – Phát biểu

2.8- Mảng – Array

2.9- Nhập xuất dữ liệu.

2.10- Tóm tắt dạng trắc nghiệm

2.11- Bài tập

2.1- Chú thích trong java

```
// Chú thích đến cuối dòng
/*  Chú thích nhiều dòng
    .....
*/
```

➔ Cách viết chú thích giống C++

Chú thích là công cụ:

(1) Giải thích chương trình.

(2) Lập tài liệu cho chương trình: Tác gỉa, version, những đặc điểm của chương trình

2.2- Từ khóa- Cách đặt tên

- Từ khóa cho các kiểu dữ liệu cơ bản : byte, short, int, long, float, double, char, boolean
- Từ khóa cho phát biểu lặp: do, while, for, break, continue
- Từ khóa cho phát biểu rẽ nhánh: if, else, switch, case, default, break
- Từ khóa đặc tả đặc tính một method: private, public, protected, final, static, abstract, synchronized, volatile, strictfp
- Hằng (literal): true, false, null
- Từ khóa liên quan đến method: return, void
- Từ khoá liên quan đến package: package, import

2.2- Từ khóa- Cách đặt tên (tt)

- Từ khóa cho việc quản lý lỗi: try, catch. finally, throw, throws
- Từ khóa liên quan đến đối tượng: new, extends, implements, class, instanceof, this, super
- **Cách đặt tên (identifier):**
- Bắt đầu bằng ký tự, ký tự gạch dưới (underscore '_') hay ký tự '$'
- Sau đó là các ký tự ký số hay '_', '$' , không dùng các ký tự khác như: khoảng trống, ký hiệu phép toán
- Từ khóa và tên có tính chất case-sensitive

Nhận xét: Gần như y hệt C++

2.3- Kiểu dữ liệu cơ bản trong java

Type	Default	Size (bytes)	Range	Description
byte	0	1	-128..127	số nguyên
short	0	2	-32768..32767	số nguyên
int	0	4	-2 tỉ mốt.. 2 tỉ mốt	số nguyên
long	0	8	- 9 tỉ tỉ .. 9 tỉ tỉ	số nguyên
float	0.0	4	+/- 1.45 E-45 .. +/-3.4 E+38, +/- infinity, +/-0, NAN	số thực (Not A Number
double	0.0	8	+/- 1.79E-324 .. +/-3.4 E+308, +/- infinity, +/-0, NAN	số thực
char	\u0000	2	\u0000 .. \uFFFF	ký tự Unicode

Thí dụ

Dùng các ký tự đặc tả việc buộc phải xem xét trị thuộc 1 kiểu nào đó: i, I, *l*, L, f, F, d, D

nhưng L thường dùng thay cho *l* vì sợ nhầm với 1.

178 → int (default) **45.62 → double (default)**

178L → long **44.21f → float**

11.19e8 → double (default)

'z' → char , hằng klý tự để trong cặp nhát đơn (single quote character)

Nhận xét: Gần như C++

2.4- Biến- Định nghĩa, khởi tạo

- Biến = Trị có thay đổi theo thời gian
- 3 đặc điểm của biến:

  Tên biến, Trị khởi tạo, tầm vực (scope)
- Scope của biến: khối chương trình mà biến có ý nghĩa (tham khảo được)
- Cú pháp định nghĩa biến:

  DataType [[identifier [= InitValue]],…] ;

int count , age1= 21, age2= 2*age1;

char ch1='z', ch2;

➔ Giống C

2.5- Toán tử- Operators

- Ký hiệu mô tả phép toán
- Arithmetic ops: +, -, *, /, %, ++, --
- Relational ops : <, <=, ==. >=, >, !=
- Logical ops: && ||
- Bitwise ops: ~, &, |, ^ (xor), >>, <<, &=, |=, ^=, >>=, <<=
- Assignment ops : = , +=, -=, *=, /=, %=
- Ternary op:
- Condition ? TrueExp : FalseExp

➔ Giống C

2.5- Toán tử- Operators (tt)

- **instanceof : toán tử kiểm tra 1 đối tượng có thuộc 1 lớp ? → true | false**

```
class InstanceOfDemo
{ public static void main (String args[])
  {  InstanceOfDemo t = new InstanceofDemo();
     if  ( t instanceof InstanceOfDemo)
       System.out.println(“ t la 1 doi tuong thuoc lop nay”);
     else
       System.out.println(“ t KHONG la 1 doi tuong thuoc lop nay”);
  }
}
```

2.6- java.lange package

- Gói cơ bản của ngôn ngữ java (language)
- Chứa các lớp cơ bản đóng vai trò trung tâm đối với các tác vụ của java.
- Các lớp cơ bản nhất: class **Object**, class **Class** là các lớp cơ sở của mọi lớp khác.
- Nếu muốn xem các dữ liệu thuộc kiểu cơ bản là các đối tượng, các lớp *gói* (wrapper) mang các tên: ***Boolean, Character, Integer, Long, Float, Double, Void*** dùng cho mục đích này.
- Lớp **Void** là lớp không thể khởi tạo, lưu trữ 1 tham khảo tới 1 đối tượng thuộc lớp Class biểu diễn cho kiểu void

java.lange package(tt)

- Chứa lớp Math cho các xử lý toán học
- Chứa các lớp Loader, Process, Runtime, SecurityManager, System để cung cấp caùc taùc vụ mức hệ thống như: quản lý nạp các đối tượng, tạo qúa trình, quản lý an toàn, nhập xuất dữ liệu, tham khảo thời gian của hệ thống.
- Một số hàm toán học

  abs(TrịSố) : lấy trị tuyệt đối

  Nếu trị số kiểu byte, short thì kết qủa là kiểu int

  ```
  int n= -5, m ;  m = Math.abs(n); // m=5
  ```

➔ Để ý cách dùng hàm toán:
Math.TênHàm(thamSố)

java.lange package(tt)

- Hàm ceil(x) → Số tròn sát trên <=x (trần)

  Math.ceil(8.02) → 9.0 Math.ceil(-1.3) → -1.0

  Math.ceil(100) → 100.0

- Hàm floor(x) → Số tròn chận dưới (sàn)

  Math.ceil(-5.63) → -6.0 Math.ceil(100) → 100.0

- Hàm max(x,y) → Trị lớn trong 2 số
- Hàm min(x,y)
- Hàm random() → **trả về 1 số ngẫu nhiên từ 0.0 đến 1.0**
- Hàm round (sốthực) Math.round(34.5) → 35

2.7- Cấu trúc điều khiển – Phát biểu

- Cấu trúc rẽ nhánh

```
if (Condition)
    { Statements;
    }
 else
    { Statement;
    }
```

→ Giống C

- Cấu trúc rẽ nhánh

```
switch (Expression)
{ case Cons1: Statements; break;
   case Cons2: Statements; break;
   . . .
  default : Statements;
 }
```

→ Giống C

Cấu trúc điều khiển – Loops

```
while (condition)
  { Statements;
  }
```

```
do
  { Statements;
   }
while (condition);
```

```
for ( varInit ; Condition ; GroupStatements2)
  { Statements1;
  }
```

Bỏ qua 1 lần lặp : **continue;**

➔ **Giống C**

Cắt vòng lặp: **break** [label_name];

Minh họa phát biểu break

```
import java.io.*;
class BreakDemo
 { public static void main(String ags[])
   {  boolean t = true;
      FIRST:
       { SECOND:
          { THIRD:
             {  System.out.println("Executed before break");
                if (t) break SECOND;
                System.out.println("Out of break, Not executed");
             }
             System.out.println("Out of break, Not executed");
          }
          System.out.println("Executed after the second break");
       }
   }
 }
```

Kết qủa

```
Executed before break
Executed after the second break
Press any key to continue...
```

2.8- Mảng – Array

- Mảng= Nhóm trị cùng kiểu, kề nhau, cùng tên gọi.
- Định nghĩa có chỉ định số phần tử (size)-> cấp bộ nhớ
  char ch[] = new char [5];
- Định nghĩa mảng tức thời (in-line initialization):
  int a[] = { 1,4,2,7,8}; // hoặc
  int [] a = { 1,4,2,7,8};

- Phần tử được tham khảo qua chỉ số bắt đầu từ 0

ch[0]	ch[1]	ch[2]	ch[3]	ch[4]

- Chỉ khai báo: không chỉ định size : **long a[];**
- Khi sử dụng phải cấp bộ nhớ: **a= new long [20];**
- Trị mặc định : Toán tử new sẽ xóa bộ nhớ, các bit = 0)

Mảng - minh họa

```
// file  ArrayDemo.java
import java.io.*;
class ArrayDemo
{
   public static void main(String args[])
   { int a1[] = { 1, 2,3,4,5 };//In-line initialization
     int a2[];  // just declaration
     a2 = new int [5]; // mem. allocation
     int i;
     for (i=0;i<5;++i) a2[i] = 2*a1[i];
     long S= 0;
     for (i=0;i<5;++i) S+= a1[i] +a2[i];
     System.out.print("Sum of 2 arrays:");
     System.out.println(S);
      }
}
```

Kết qủa:

Sum of 2 arrays:45

Press any key to continue...

2.9- Nhập xuất dữ liệu.

- Nhập xuất dữ liệu là tác vụ mức hệ thống
- Gói ***java.io*** chứa các lớp cho việc xuất nhập.
- Cần tham khảo gói này.
- Java cung cấp class System mô tả hệ thống
- System.out là đối tượng xuất mặc định (màn hình)
- System.in là đối tượng nhập mặc định (bàn phím)

Methods xuất dữ liệu ra màn hình:

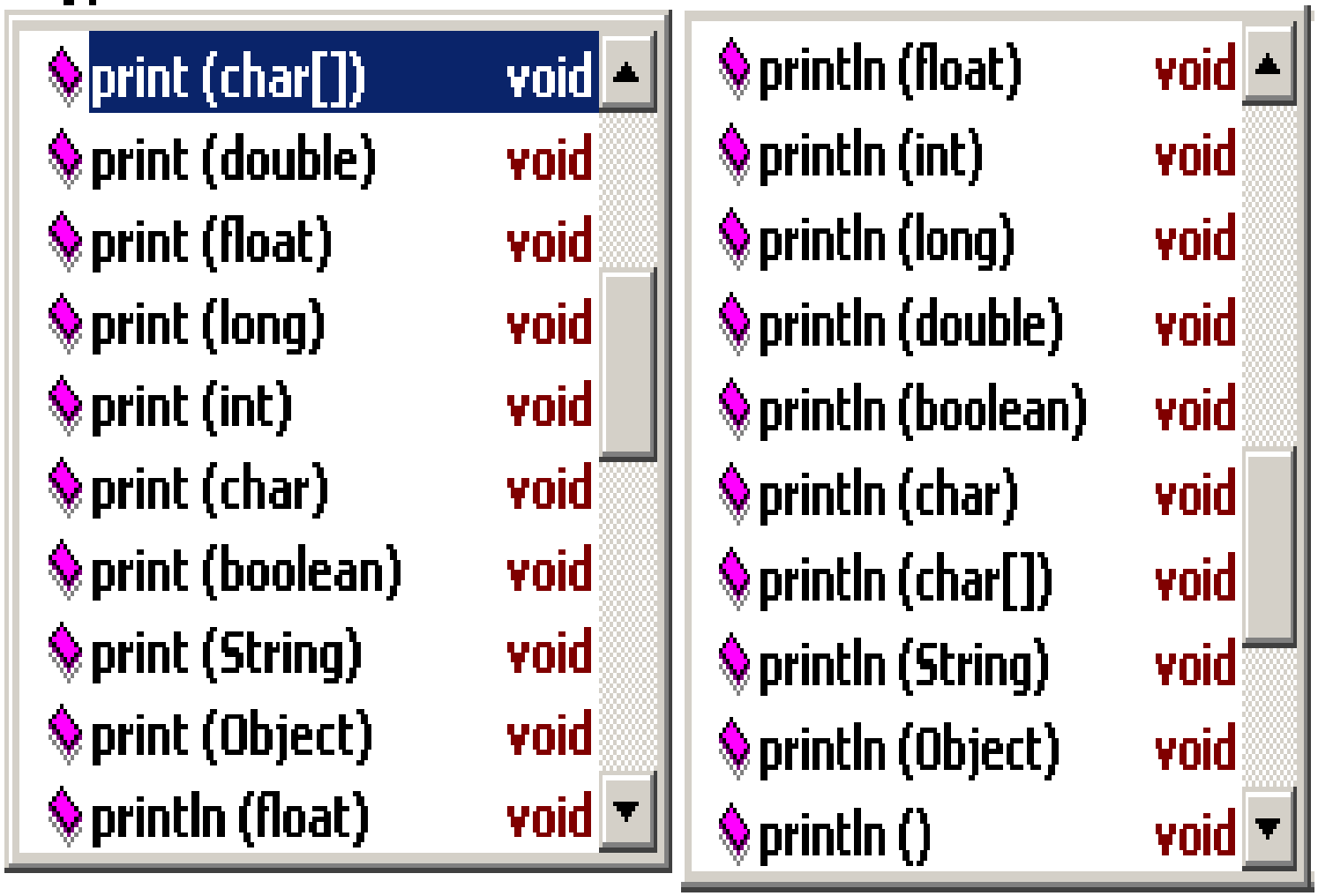
System.out.print(Dữ liệu xuất);

System.out.println(Dữ liệu xuất);

Dữ liệu xuất có thể là : ký tự, số, chuỗi,…

Xuất dữ liệu (tt)

```
System.out.p
```



Nhập dữ liệu với bàn phím

- Nhập dữ liệu từ bàn phím khá phức tạp vì với mỗi dữ liệu có cách nhập khác nhau: Ký tự thì chỉ cần 1 phím, số nguyên, số thực có thể nhập với nhiều phím nên các phím gõ cần giữ lại (đệm, buffer), có thể cần kiểm tra phím gõ (nhập số mà gõ phím chữ → sai).
- Tham khảo tài liệu về gói *java.io*
- *Nếu viết ứng dụng hướng giao diện của sổ, không xuất nhập trực tiếp mà thông qua các đối tượng trong giao diện người sử dụng.*

Minh họa xuất nhập

```
import java.io.*; // file InOutDemo.java
class InOutDemo
{ public static void main(String args [])   throws java.io.IOException
  {
     Reader inputChar_Obj = new InputStreamReader(System.in);
     System.out.print("Input a character:");
     char c = (char)inputChar_Obj.read() ;
     System.out.println("     character read :" + c);
     BufferedReader input_Obj= new BufferedReader(new InputStreamReader( System.in));
     System.out.print("Input an Integer:");
     int n= Integer.valueOf( input_Obj.readLine()).intValue() ;
     System.out.println("     integer read :" + n);
     System.out.print("Input a Double:");
     double x= Double.valueOf( input_Obj.readLine()).doubleValue() ;
     System.out.println("     Double read :" + x);
     System.out.print("Input a string:");
     String s = input_Obj.readLine();
     System.out.println("     String read :" + s);
     System.out.print("Input a character:");
     int m = System.in.read() ;
     System.out.println("     Code of this character :" + m);
  }
}
```

```
Input a character:r
    character read :r
Input an Integer:123
    integer read :123
Input a Double:12.908
    Double read :12.908
Input a string:Hello
    String read :Hello
Input a character:A
 Code of this character:65
```

2.10-Tóm tắt dạng câu hỏi

- Liệt kê 6 kiểu số cơ bản của Java và số bit bộ nhớ sẽ chiếm dụng của chúng khi lưu trữ trị.
- Kiểu luận lý (logic) trong java tên là gì?
- Kiểu ký tự tên là gì? chiếm mấy byte?
- Biến là gì? Hãy cho biết 3 tính chất của biến.
- Hãy liệt kê 4 loại toán tử trong java.
- Hãy cho biết kết qủa của biểu thức sau :

  a) 4 && 7 b) 4&7 c) 0 || 8 d) 0 | 8 e) 8 ^ 3

  f) 4 >> 3 g) 9 << 2 h) (9 >>3) ? 5 : -3
- Trong java có phát biểu **goto** hay không ?
- Mảng là gì ? Mảng in-line là gì ?
- **int a[] = new int { 1,2,3,9,0};** đúng hay sai?
- System.out.println(12 & 9); sẽ xuất trị bao nhiêu?
- Phân tích dòng code: **if** (5&7>0 && 5|3) System.out(“Hello”) ;
- int m= System.in.read(); nếu gõ vào phím ‘C’, trị biến m là bao nhiêu?

2.11- Bài tập

Viết các chương trình sau:

(1) Xuất bảng cửu chương từ 2 đến 9.

(2) Xuất trị bình phương , lập phương từ 1 đến 10.

(3) Tạo 1 mảng số int dạng in-line 10 phần tử, xuất mảng này tăng dần.

(4) Nhập 1 mảng int các số mang trị là mã của các ký tự nhập từ bàn phím. Xuất mảng này dạng chữ rồi xuất mã của chúng.

(5) Xuất 100 số Fibonacci đầu tiên. Dãy Fibonacci : 1,1,2,3,5,8,… 2 số đầu là 1, các số sau bằng tổng 2 số trước nó.

CHƯƠNG 3
LỚP VÀ ĐỐI TƯỢNG

Nội dung chương 3

3.1- Khaí niệm về lớp và đối tượng
3.2- Cú pháp tạo lớp
3.3- Xây dựng và khởi tạo đối tượng.
3.4- Tính thừa kế (Inheritance)
3.5- Tính đa hình (Polymorphism)
3.6- Lập trình với dữ liệu nhập
3.7- Một số lớp có sẵn của Java.
3.8- Giao diện (Interface)
3.9- Lớp trừu tượng (Abstract class)
3.10- Lớp nội (Inner class)
3.11- Gói phần mềm (Package)
3.12- Tóm tắt và câu hỏi.
3.13- Bài tập

3.1- Khái niệm về lớp và đối tượng

- Đối tượng (Object): vật,người,… cụ thể
- Đối tượng = Properties + Methods
- Lớp (class): Mô hình (template) mô tả cho 1 nhóm đối tượng → Đối tượng là 1 hiện hữu, thực thể (instance) của class.
- Một lớp có thể là lớp con (derived class- lớp dẫn xuất, lớp thừa kế, lớp mở rộng-extend) của 1 lớp khác → Quan hệ cha-con
- Class Hierarchy- Phân cấp các class:Cấu trúc 1 lớp cùng các lớp con của nó (tree)

3.2- Cú pháp khai báo class

- Khai báo 1 class là khai báo một mẫu (template) chung mô tả cho 1 nhóm đối tượng cùng đặc tính.
- Thực thể (entity): Một biểu diễn cho một đối tượng bao gồm properties và behaviors → Là một biểu diễn cho một đối tượng vật lý hoặc quan niệm của tự nhiên.
- Mỗi ngôn ngữ OOP hỗ trợ khác nhau về cách khai báo class cũng như các hỗ trợ các kỹ thuật OOP khác nhau.

Cú pháp khai báo class trong Java

```
class  CLASSNAME  extends  FATHERCLASSNAME
{   DataType1   Property1 [=Value];
    DataType2   Property1 [=Value];
   CLASSNAME (DataType  Arg,…)        // constructor
   {… }
    [Modifier] DataType MethodName( DataType  Arg,…)
    { … }
}
```

- public - private- protected : giống C++
- final : Không cho phép con mở rộng(override)
- Không có modifier : Mặc định là friend, cho phép các đối tượng thuộc các class cùng package truy cập

Đặc tính truy xuất

Modifier	private	friendly	protected	public
Cùng class	YES	YES	YES	YES
Cùng gói, khác class	NO	YES	YES	YES
lớp con trong cùng gói với lớp cha	NO	YES	YES	YES
Khác gói, khác lớp	NO	NO	NO	YES
Lớp con khác gói với lớp cha	NO	NO	YES	YES

Ôn lại về chỉ thị static

static property: Dữ liệu chung cho mọi đối tượng cùng lớp→ Nằm ngoài vùng nhớ của đối tượng (mang ý nghĩa của 1 biến toàn cục)

```
class STATIC_DEMO
{ static int Count =0 ;
  STATIC_DEMO() { Count++;}
}
```

Tham khảo static property của 1 lớp:

(1) Tham khảo qua 1 đối tượng của lớp này.

```
STATIC_DEMO D1= new STATIC_DEMO();
D1.Count=100;   TestVar1 = D1.Count ;
```

(2) Tham khảo qua tên lớp.

```
TestVar2 = STATIC_DEMO.Count ;
```

Ôn lại về chỉ thị static

static method: Phương thức cho phép sử dụng mà không cần khai báo đối tượng thuộc lớp.

```
import java.io.*;
class STATIC_CLASS
{ static void Test() { System.out.println("Hello1!");}
}
class STATIC_CLASS2 extends STATIC_CLASS
{  void Test(){ System.out.println("Hello2!");}
}
class STATIC_TST
{  public static void main (String args[])
   { STATIC_CLASS.Test();      }
}
```

Lỗi:
Static method can't overridden

Sửa lại

```
class STATIC_CLASS
{ static void Test() { System.out.println("Hello1!");}
}
class STATIC_CLASS2 extends STATIC_CLASS
{  static void Test(){ System.out.println("Hello2!");}
}
class STATIC_TST
{  public static void main (String args[])
    {  STATIC_CLASS.Test();
       STATIC_CLASS2.Test();
     }
}
```

3.3- Xây dựng và khởi tạo đối tượng.

Chú ý về constructor:

- **Default Constructor: Nếu 1 lớp không hiện thực constructor, constructor mặc định của Java sẽ thực thi khi định nghĩa đối tượng (xóa trống bộ nhớ, các bit đều là 0 cho mọi properties).**

  ClassName ObjName = new ClassName () ;

- **User-defined Constructor: Nếu 1 lớp có hiện thực constructor, Java sẽ thực thi constructor tự tạo này mà không dùng constructor mặc định nữa→ Phải định nghĩa đối tượng theo cú pháp của constructor tự tạo.**

  ClassName ObjName = new ClassName (Args) ;

Thí dụ:

```
import java.io.*;  // file FruitDemo.java
class FRUIT
{    boolean Seedness; // có hột hay không
     boolean Seasonal; // có theo mùa hay không
     int Price ;                    // Giá
     public FRUIT ()  // constructor 1- override default constructor
     { Seedness= false; Seasonal= false; Price= 0;      }
     public FRUIT (boolean aSeedness, boolean aSeasonal,
                                    int aPrice)
     { Seedness= aSeedness; Seasonal= aSeasonal; Price= aPrice;   }
     public void SetProperties(boolean aSeedness,
                                    boolean  aSeasonal, int aPrice)
     {  Seedness= aSeedness; Seasonal= aSeasonal;Price= aPrice;}
```

```
 void PrintPropertes()  // friend method
   { if (Seedness) System.out.println("Fruit is seedness.");
     else System.out.println("Fruit is seeded.");
     if (Seasonal) System.out.println("Fruit is seasonal.");
     else System.out.println("Fruit is not seasonal.");
     System.out.println("Price is :" + Price);
   }
} // end of  FRUIT class
class FruitDemo
{ public static void main (String args[])
  { System.out.println("First fruit:");
    FRUIT f1 = new FRUIT();    f1.PrintPropertes();
    System.out.println("Second fruit:");
    FRUIT f2 = new FRUIT(true, false,10000);
     f2.PrintPropertes();
  }
}
```

Thí dụ (tt)- Kết qủa chạy chương trình

First fruit:

Fruit is seeded.

Fruit is not seasonal.

Price is :0

Second fruit:

Fruit is seedness.

Fruit is not seasonal.

Price is :10000

Press any key to continue...

Chú ý:

Method PrintProperties() có tính chất Friend nên class FruitDemo nằm cùng file với class FRUIT (cùng gói) nên được phép truy xuất method này.

Chỉ thị final

- Từ khóa final có thể đứng trước 1 khai báo class, 1 khai báo method, 1 khai báo property mang ý nghĩa "Đây là cái cuối cùng" → Không cho lớp con mở rộng.
- 1 final class là 1 class không thể có lớp con.

```
import java.io.*;
final class FINAL_CLASS1
{ int t=6;
  void Show() { System.out.println(t);}
}
class FINAL_CLASS extends FINAL_CLASS1
{ public static void main(String args[])
  { }
}
```

Sửa lại bằng cách bỏ final trong khai báo FINAL_CLASS1 hoặc xây dựng mới class FINAL_CLASS

LỖI: 6-Cannot inherit from final class FINAL_CLASS1

Chỉ thị final (tt)

Một final method là 1 method không thể override ở lớp con

```
import java.io.*;
class FINAL_CLASS1
{ int t=6;
  void Show() { System.out.println(t);}
}
class FINAL_CLASS extends FINAL_CLASS1
{ int t2=8;
   public static void main(String args[])
   { FINAL_CLASS obj= new FINAL_CLASS(); obj.Show(); }
}
```

chương trình này OK

Chỉ thị final (tt)

```
import java.io.*;
class FINAL_CLASS1
{ int t=6;
  final void Show() { System.out.println(t);}
}
class FINAL_CLASS extends FINAL_CLASS1
{ int t2=8;
  void Show() { System.out.println(t2);}
  public static void main(String args[])
   { FINAL_CLASS obj= new FINAL_CLASS();
     obj.Show();}
}
```

Lỗi: Cannot override Show() vì Show() là final method đã khai báo trong lớp cha

Chỉ thị final (tt)

final property là 1 hằngcục bộ, KHÔNG thể gán lại trị.

```
import java.io.*;
class FinalVar
{   public static void main (String args[])
    { final int t=1;
      t=2;
      System.out.println(t);
    }
}
```

LỖI: 5- Cannot assign a value to final variable t

3.4- Tính thừa kế (Inheritance)

- Thừa kế: Kỹ thuật cho phép tái sử dụng thông tin (properties+methods).
- Lớp con = Lớp cha + một tí
- Lớp con **không** thể truy xuất thành phần private của lớp cha.
- Cú pháp:

class SON extends FATHER

{ …

}

Chú ý: khi hiện thực code của class

- Tham số của các method: chỉ có dạng **THAM TRỊ** (pass by value) vì Java với định hướng lập trình mạng, hướng OOP, bao gói triệt để → Không thể truy cập trực tiếp properties của 1 đối tượng.
- Từ khóa ***this*** : Đối tượng hiện hành. khi truy xuất *member chính là dạng viết tắt của this.member.*
- Từ khóa ***super*** tham khảo đến lớp cha
- Cho phép ***overload method*** –các method cùng tên nhưng khác tham số.

Thí dụ về this và super

```
import java.io.*;
class T_This1
{ int x1, y1;
  T_This1(int xx,int yy) { x1=xx; this.y1=yy;}
  void OutData()
       {System.out.println("x1="+x1+","+"y1="+y1);}
}
class T_This2 extends T_This1
{ double x2, y2;
  T_This2(int xx1,int yy1,double xx2, double yy2)
          { super(xx1,yy1); x2=xx2; this.y2=yy2;}
  void OutData()
    { super.OutData();
      System.out.println("x2="+x2+","+"y2="+y2);}
}
```

Gọi constructor của lớp cha

Gọi method của lớp cha

this- super (tt)

```
class TestThis
{
  public static void main (String args[])
  {  T_This2 t= new T_This2(4,5,6,7);
     t.OutData();
  }
}
```

Kết qủa

x1=4,y1=5

x2=6.0,y2=7.0

Press any key to continue...

Thí dụ về overloading methods

```
import java.io.*;                                   Obj1     0 do default
class C1                        constructor
{ int x,y,z;
  void SetData(int t1, int t2) { x=t1; y=t2;};
  void SetData(int t1, int t2,int t3) { x=t1; y=t2; z=t3;}
  void OutData() { System.out.println(x+","+y+","+z); };
}

class OverLoad1
{ public static void main(String args[])
  { C1 Obj1= new C1();  Obj1.SetData(3,4); Obj1.OutData();
    C1 Obj2 = new C1();   Obj2.SetData(7,8,9); Obj2.OutData();
  }
}
```

3.4- Tính Đa Hình (Polymorphism)

- Đa hình: Kỹ thuật tạo những sắc thái khác nhau trên cùng 1 methods của các lớp trong phân cấp thừa kế, bảo đảm thực thi đúng code của 1 hành vi của 1 đối tượng trong 1 phân cấp.

➔Đa hình chỉ có trong 1 phân cấp thừa kế và các class của phân cấp có cùng method.Kỹ thuật đa hình cho phép 1 lớp con ***override*** 1 method ở lớp cha (***cùng 1 method nhưng code trong lớp cha và code trong lớp con khác nhau***)

- ***overload methods***: methods cùng tên nhưng khác tham số trong cùng 1 class.

Thí dụ về toán tử instanceof- Kiểm tra lớp của đối tượng

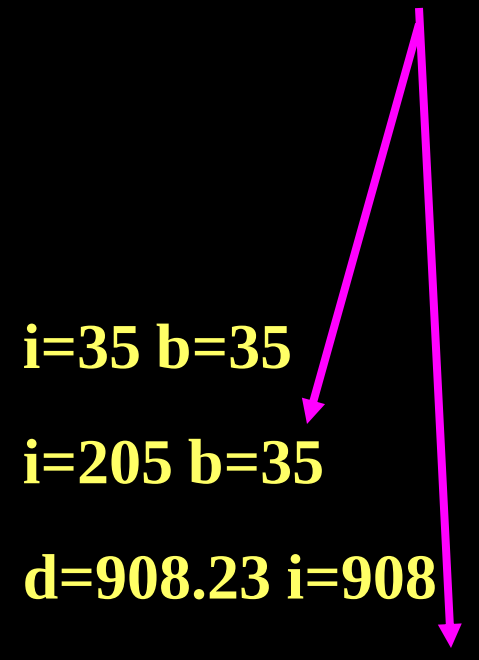
```
import java.io.*; // InstanceOfDemo.java
class Student
{  String Name;    int Score1, Score2, Score3;
    public Student(String aName, int S1, int S2,int S3)
        { Name= aName; Score1=S1; Score2=S2; Score3=S3;}
    String GetName() { return Name;}
}
public class InstanceOfDemo
{ public static void main(String args[])
  { Student st= new Student("Hoa", 5,6,7) ;
    if (st instanceof Student)
            System.out.println(st.GetName()+" is a student.");
    else System.out.println("This isn't a student.");
  }
}
```

Thí dụ về ép kiểu (type casting)

- Nhiều khi cần phải ép kiểu khi viết code

```
import java.io.*; // TypeCaseDemo.java
class TypeCastDemo
{ public static void main(String args[])
  { byte b ; int i= 35; double d= 908.23;
    b= (byte)i ;
    System.out.println("i=" + i +" b=" + b);
    i=205;
    System.out.println("i=" + i +" b=" + b);
    i= (int)d;
    System.out.println("d=" + d +" i=" + i);
    b= (byte)d;
    System.out.println("d=" + d +" b=" + b);
  }
}
```

Để ý tình huống tràn số (overflow)



i=35 b=35

i=205 b=35

d=908.23 i=908

d=908.23 b=-116

Thí dụ về tính đa hình

```
import java.io.*;
class SHAPE
{ double Area() { return 0; }
}
class CIRCLE extends SHAPE
{ double x,y,r;
  CIRCLE(double rr) { r=rr>0?rr:0;}
  double Area() { return Math.PI* r*r; }
}
class RECTANGLE extends SHAPE
{ double a,b;
  RECTANGLE (double aa, double bb) { a=aa>0?aa:0;
     b=bb>0?bb:0;}
  double Area() { return a*b; }
}
```

Thí dụ về tính đa hình (tt)

```
class PolyTest1
{  public static void main (String args[])
   { SHAPE S[]= { new SHAPE(), new CIRCLE(5),
                          new RECTANGLE(2,3)};
     for (int i=0;i<3;++i) System.out.println(S[i].Area());
   }
}
```

Kết qủa

```
0.0
78.53981633974483
6.0
Press any key to continue...
```

3.6- Chạy ứng dụng với tham số

- Chương trình Java có thể đưa vào dữ liệu nhập khi chạy chương trình→ 1 công cụ nhập dữ liệu.
- Cú pháp: java File.class arg0 arg1 …

```
import java.io.*;
class InputCommandLine
{ public static void main (String args[])
    { for (int i=0;i< args.length; ++i)
        System.out.print(args[i]+ (i< args.length?",":""));
    }
}
```

```
D:\Su\BaiGiang2004\Java\BtCh3>java InputCommandLine Mat Uot Mi
Mat,Uot,Mi
D:\Su\BaiGiang2004\Java\BtCh3>
```

3.7- Một số lớp có sẵn của Java.

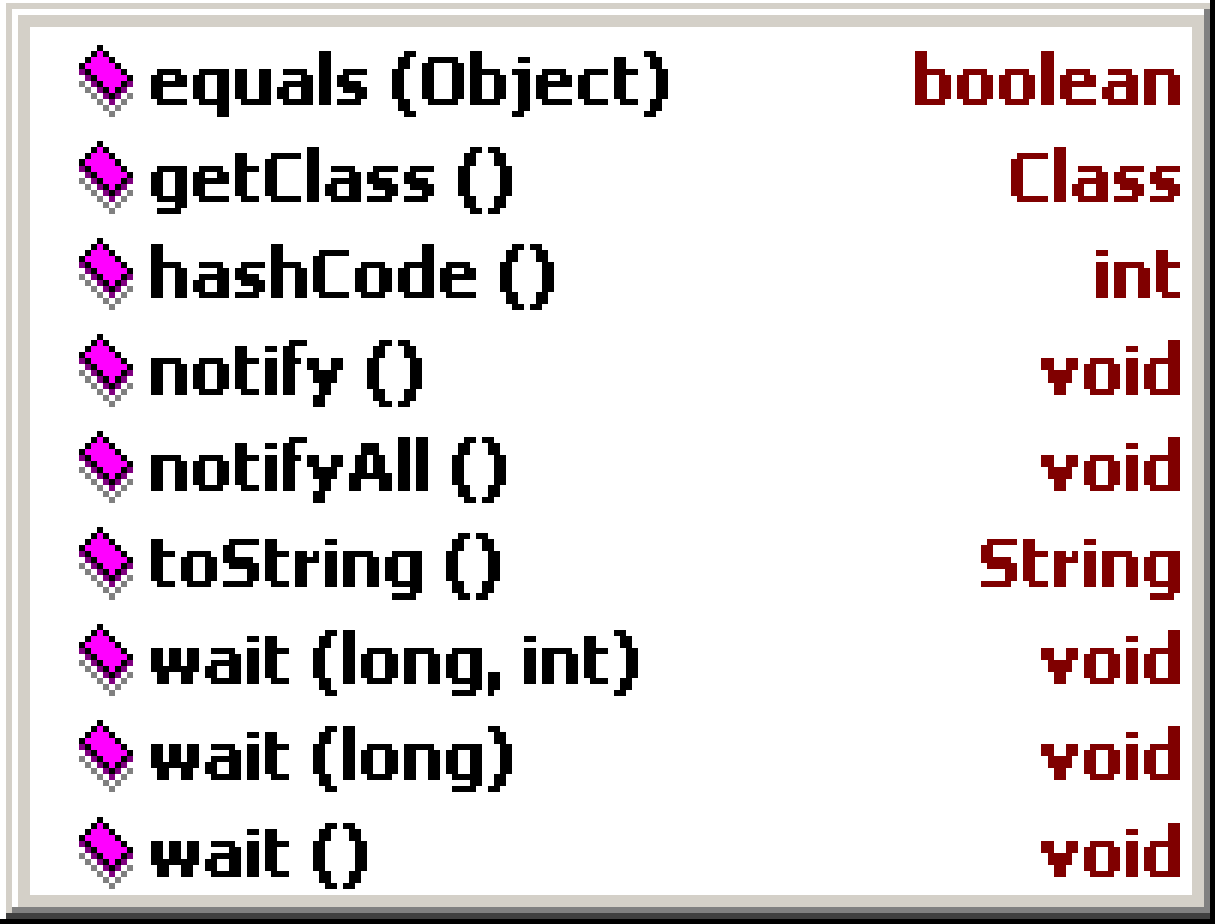
- Lớp Object
- Lớp String
- Các lớp gói (wrapper)

3.7.1- Lớp Object

- Là lớp cha của mọi lớp trong java (trực tiếp/gián tiếp)
- **Được để trong gói *java.lang* *(java.lang.Object)***
- Định nghĩa các trạng thái cơ bản và các phương thức cơ bản của mọi lớp phải có như: So sánh nó với 1 đối tượng khác (equals), chuyển đổi mô tả thành chuỗi (toString), đợi (wait) 1 biến điều kiện, nhận biết (notify) các đối tượng khác khi biến điều kiện có thay đổi, lấy Class (getClass)

Lớp Object (tt)

```
Object Obj= new Object();
Obj.
```



Lớp Object (tt)

```
import java.io.*; // ObjectDemo.java
class Student2
{ String Name; int t1,t2;
  Student2(String aName, int tt1, int tt2)
        { Name=aName; t1=tt1; t2=tt2;}
}
```

```
Student2@111f71
```

Kết qủa của method toString() :

Tên lớp + @ + Địa chỉ hệ 16 của thực thể

```
class ObjectDemo
{ public static void main(String args[])
  { Integer InObj1= new Integer (1);
    Integer InObj2= new Integer (1);
    Integer InObj3= new Integer (3);
    if (InObj1.equals(InObj2))
        System.out.println("Obj1 and Obj2 are the same");
    else System.out.println("Obj1 and Obj2 are Separately");
    if (InObj1.equals(InObj3))
        System.out.println("Obj1 and Obj3 are the same");
    else System.out.println("Obj1 and Obj3 are separately");
    Student2 St= new Student2("Hoa", 5,6);
    System.out.println(St.toString());
    System.out.println(St.getClass().getName()+ "@" +
                       Integer.toHexString(St.hashCode()));
  }
}
```

```
Obj1 and Obj2 are the same
Obj1 and Obj3 are separately
Student2@111f71
Student2@111f71
```

3.7.2- Lớp String - chuỗi ký tự

- **Định nghĩa 1 String:**

  String Str1="Hello";

  String Str2= new String("Hi");

- **Nối String**

  String Str3= Str1 + Str2; // Str3=*"HelloHi"*

  String Str4 = Str3 + 1; // Str4= *"HelloHi1"*

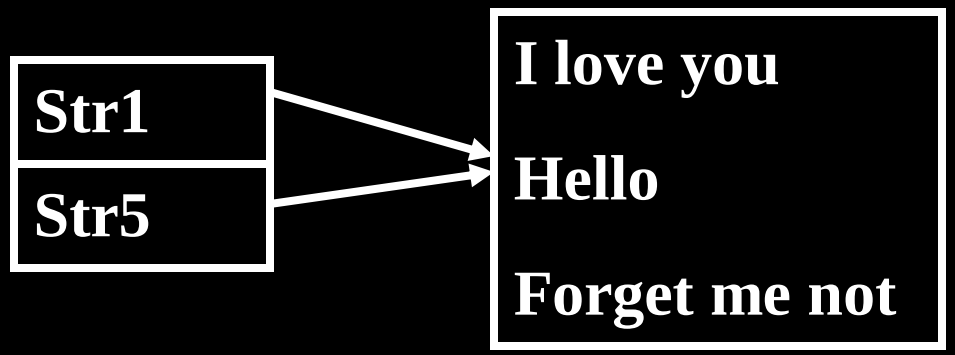
- **String pool (hồ/ bảng chứa chuỗi)**

  Khi nhiều biến String cùng mang 1 nội dung, chúng cùng chỉ đến 1 phần tử trong String pool

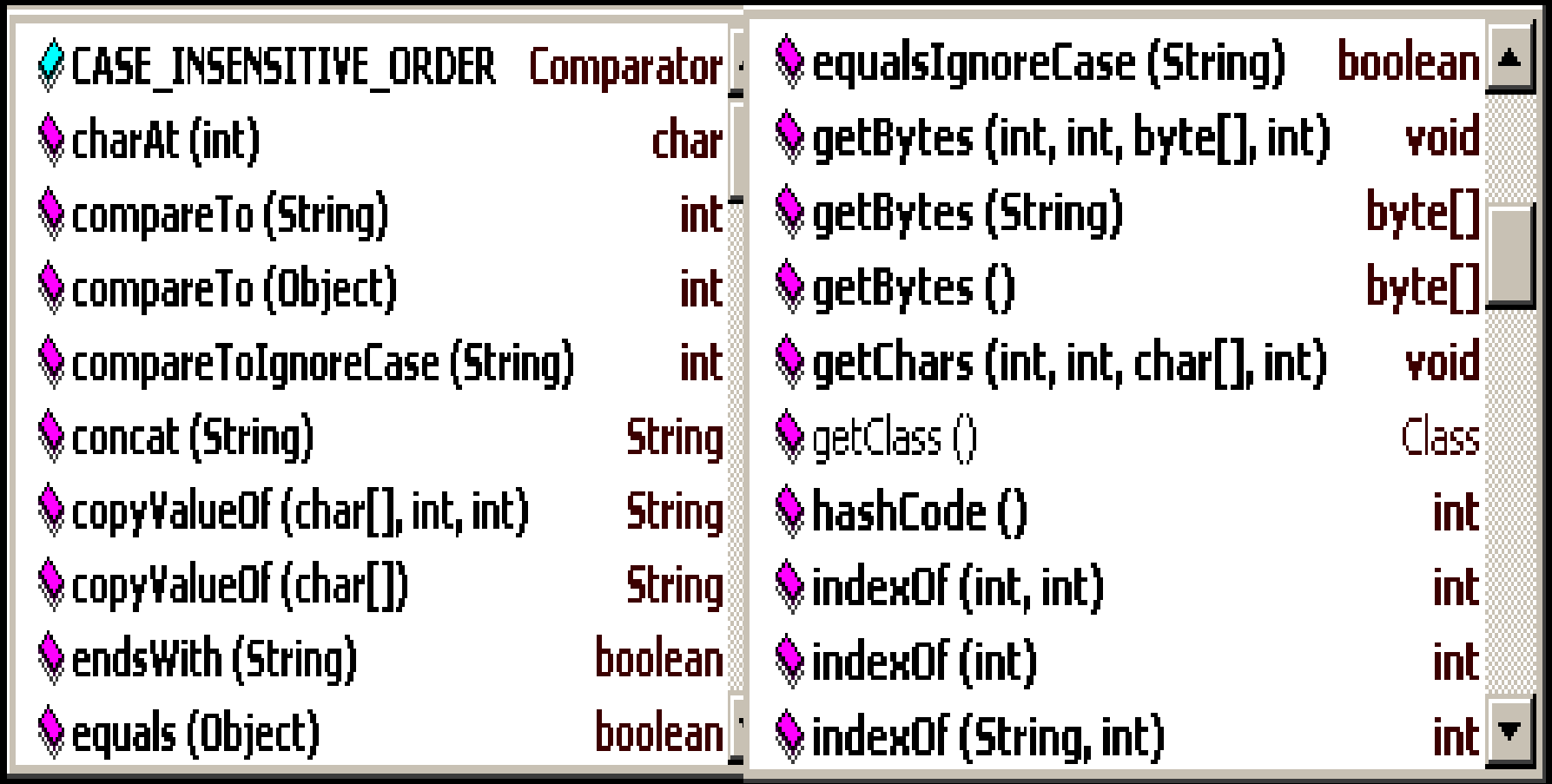
  Thí dụ:

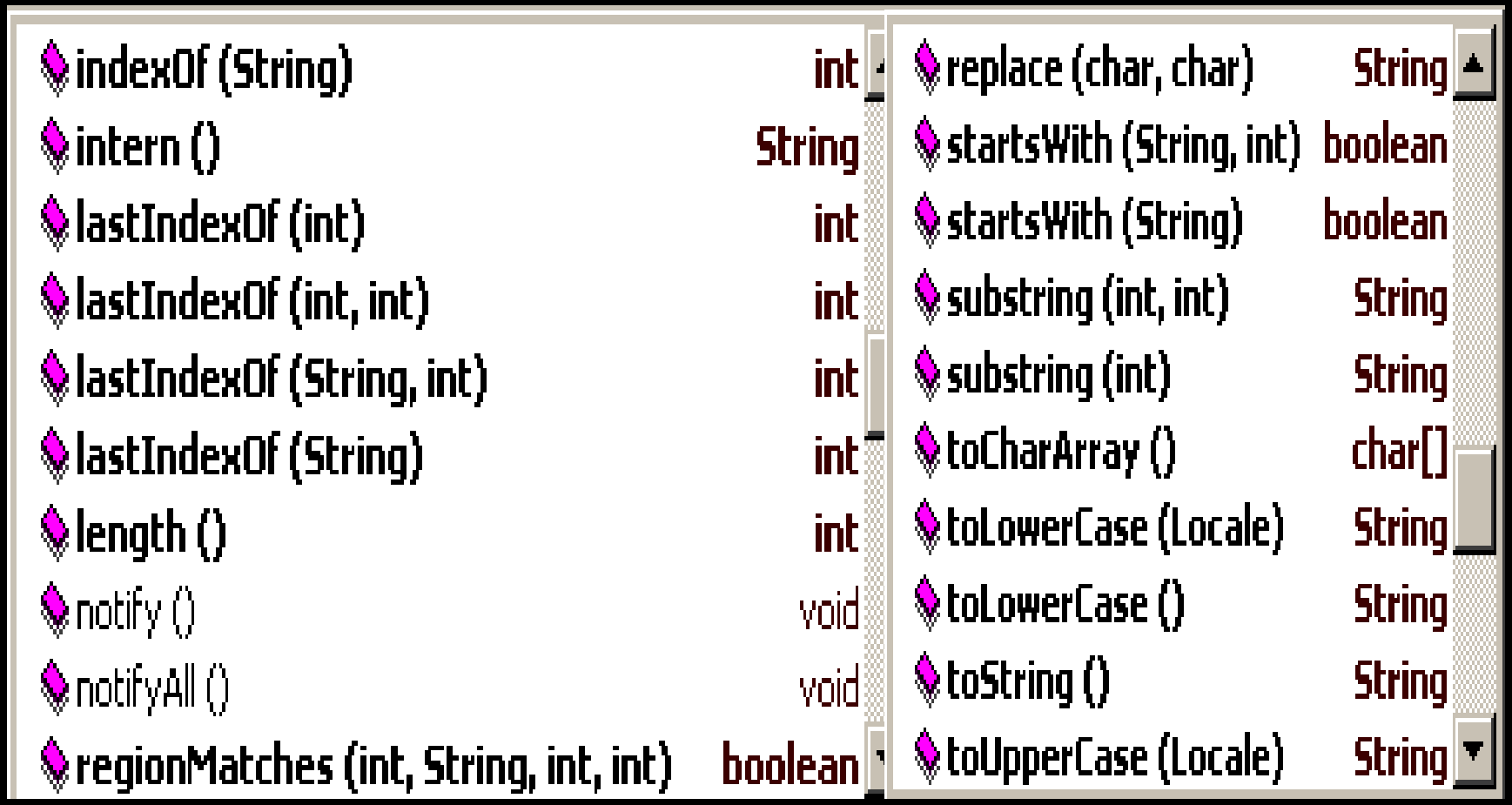
  String Str1 = "Hello";

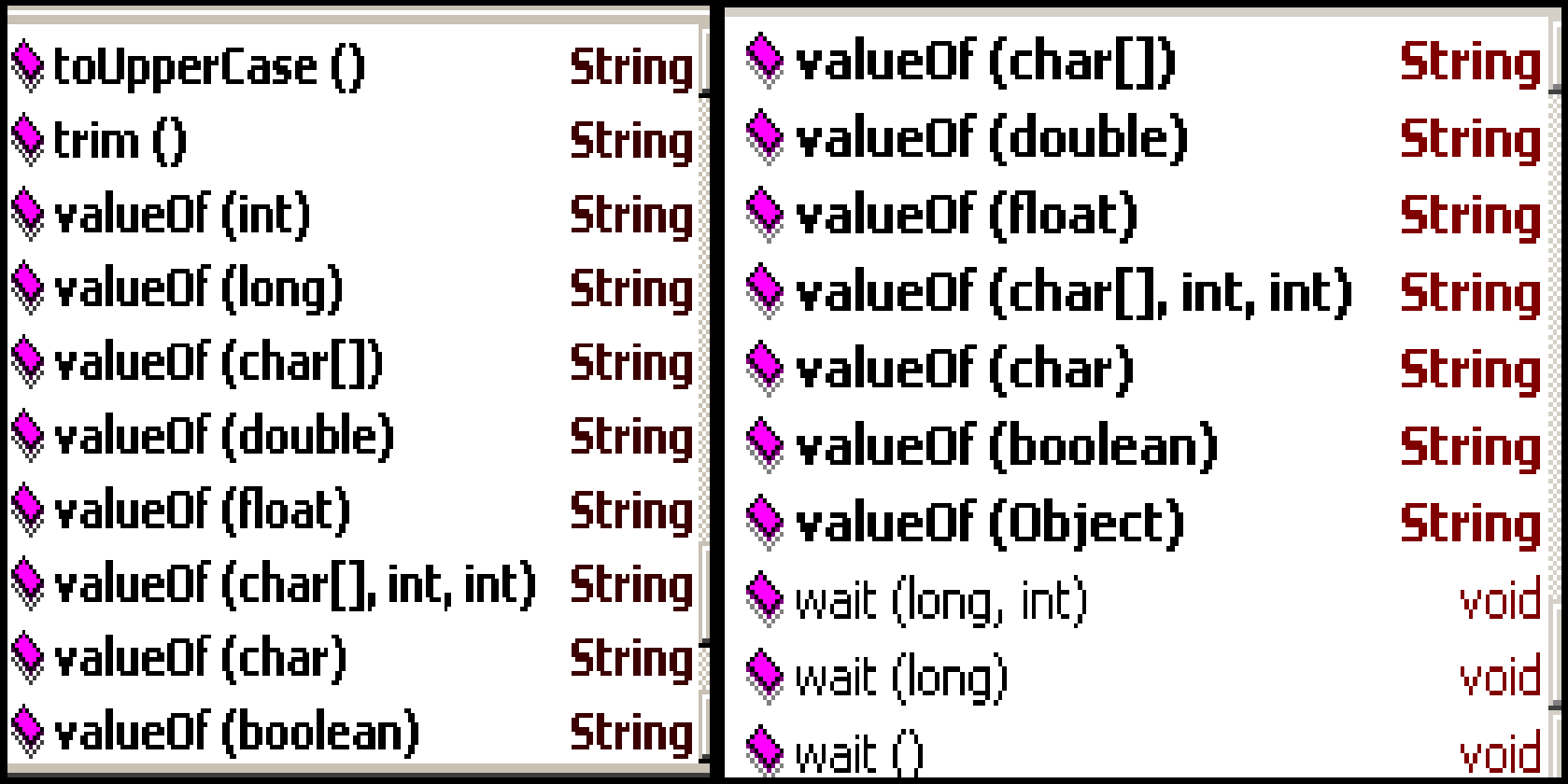
  String Str5= "Hello";



Lớp String (tt)- Methods




indexOf (String) int
intern () String
lastIndexOf (int) int
lastIndexOf (int, int) int
lastIndexOf (String, int) int
lastIndexOf (String) int
length () int
notify () void
notifyAll () void
regionMatches (int, String, int, int) boolean
replace (char, char) String
startsWith (String, int) boolean
startsWith (String) boolean
substring (int, int) String
substring (int) String
toCharArray () char[]
toLowerCase (Locale) String
toLowerCase () String
toString () String
toUpperCase (Locale) String


toUpperCase () String
trim () String
valueOf (int) String
valueOf (long) String
valueOf (char[]) String
valueOf (double) String
valueOf (float) String
valueOf (char[], int, int) String
valueOf (char) String
valueOf (boolean) String
valueOf (char[]) String
valueOf (double) String
valueOf (float) String
valueOf (char[], int, int) String
valueOf (char) String
valueOf (boolean) String
valueOf (Object) String
wait (long, int) void
wait (long) void
wait () void

Thí dụ về lớp String

```
import java.io.*; // StringDemo.java
class StringDemo
{  public static void main (String args[])
   { String Str="Halogen";
     System.out.println(Str.charAt(4));
     System.out.println(Str.toUpperCase());
     System.out.println(Str.indexOf("oge",2));
     System.out.println(Str.toLowerCase());
    }
}
```

```
g
HALOGEN
3
halogen
```

indexOf: tìm vị trí xuất hiện đầu của 1chuỗi con

3.7.3-Các lớp gói (wrappers)

Data type	Wrapper class
boolean	Boolean
byte	Byte
char	Character
double	Double
float	Float
int	Integer
long	Long
short	Short

Đọc Documentation để biết về các hành vi của các wrapper class

```
import java.io.*; // WrapperDemo.java
class WrapperDemo
{ public static void main(String args[])
  { Boolean varBool = new Boolean ("true");
    System.out.println(varBool);
    Integer varInt1= new Integer(125);
    System.out.println(varInt1);
    Integer varInt2= new Integer ("809");
    System.out.println(varInt2);
    int Sum= varInt1.intValue() + varInt2.intValue();
    System.out.println("Sum=" + Sum);
    String S1= "1024";
    int an_int= Integer.parseInt(S1);
    System.out.println( an_int);
  }
}
```

3.8- Interface- Giao tiếp

- Là một **khai báo trước** cho sự khái quát hóa của một nhóm xử lý.

  Thí dụ: Xử Lý Hồ Sơ bao gồm:

  (1) Nhận hồ sơ. (2) Kiểm tra tính hợp lệ của hồ sơ (3) Lưu trữ hồ sơ.

  Thí dụ: Xử lý biến cố gồm:

  (1) Chờ tác vụ (ActionListener)

  (2) Xử lý khi có cập nhật dữ liệu

  (3) Xử lý khi bàn phím bị gõ

  (4) Xử lý biến cố chuột…
- Là một khai báo cho một khái niệm (một tập đặc điểm gồm constants và methods) mà không muốn xây dựng lớp.

Interface- cont.

- Là một đặc điểm chính của java.
- Interface chỉ mới khai báo các hành vi.
- Hiện thực (implement) 1 interface là xây dựng 1 lớp có đặc điểm này trong đó các methods đã khai báo trong interface được hiện thực cụ thể trong lớp này. Ở mỗi tổ chức có cách xử lý hồ sơ riêng → Lúc xây dựng lớp cụ thể hóa các hành vi .
- Một lớp có thể có nhiều đặc điểm khác nhau→ Một lớp có thể là implement của nhiều interface.
- Java không cho phép đa thừa kế nhưng cho phép một lớp có thể là một cụ thể hóa của nhiều interface → Interface là công cụ để hiện thực tính ĐA THỪA KẾ của java.

Cú pháp khai báo 1 interface

public hoặc bỏ qua

public: Mọi nới đều có thể truy xuất

Không có modifier: (default) Chỉ trong cùng gói mới có thể truy xuất

```
Modifier  interface InterfaceName
{  return_type Method1( param_list);
   Datatype  final_var1 = Value1; // khai báo hằng
   return_type Method2( param_list); // prototype
   Datatype  final_var2 = Value2;
   ....
}
```

Một public interface đòi hỏi: Tên file chứa khai báo interface phải cùng tên với tên interface

Một thí dụ về interface

```
// Khai báo các interfaces
import java.io.*; // InterfaceDemo.java
interface CheckRecord // Kiem tra ho so
{ boolean Valid();
  public void OutRecord();
}
interface ScoreTable    // Bang diem
{ public double Avg(); // calculate avg of scores
}
```

```
// Sử dụng interfaces  vào 1 lớp
class Student3 implements CheckRecord, ScoreTable
{   String Name;  int s1,s2;
    public Student3(String name, int t1, int t2)
        { Name= name; s1=t1; s2=t2; }
    public boolean Valid() // implementing CheckRecord interface
     { return (s1>=0 && s1<=10 && s2>=0 && s2<=10);  }
    public void OutRecord()
      { System.out.println(Name + " " + s1 + "," + s2 + "
                                    avg=" + Avg());  }
    public double Avg() // implementing ScoreTable interface
      { return (s1+s2)/2.0;  }
}
```

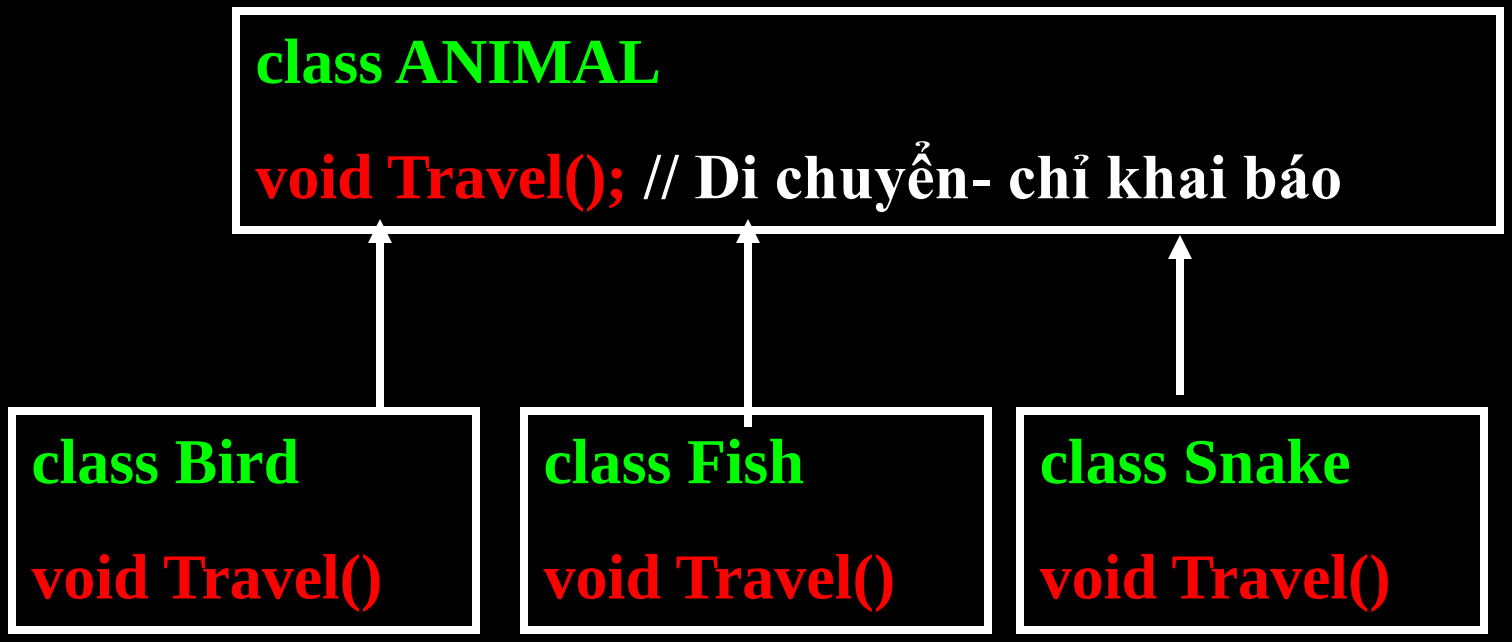
```
// chương trình minh họa
class InterfaceDemo
{ public static void main(String args[])
    { Student3 St= new Student3( "Hoa", 5,6);
      if (St.Valid()) St.OutRecord();
      Student3 St2 = new Student3( "Tuan", -1,6);
      if (St2.Valid()) St2.OutRecord();
    }
}
```

KẾT QỦA

```
Hoa 5,6 avg=5.5

Press any key to continue...
```

3.9- Lớp trừu tượng (Abstract class)

- Là lớp có những methods chưa biết code thế nào. Nên chỉ khai báo methods, để dành việc hiện thực các methods ở lớp dẫn xuất (override).
- Là kết qủa của qúa trình khái quát hóa qúa cao.



Abstract class (cont.)

Một class là class trừu tượng nếu:

- Có phương thức trừu tượng ➔Lớp con của 1 lớp trừu tượng lại có phương thức trừu tượng (có thể là phương thức trừu tượng kế thừa từ lớp cha nhưng chưa hiện thực) thì lớp con này cũng là lớp trừu tượng.
- Lớp này có khai báo implement cho 1 interface mà lại chưa viết code cụ thể cho 1 method.

Cú pháp tạo và sử dụng lớp trừu tượng

```
import java.io.*; // AbstractClassDemo.java
abstract class Employee
{  int BasicSalary = 100;
   String Name;
   abstract void OutSalary();
}
class Manager extends Employee
{   Manager(String aName, int salary)
            { Name=aName; BasicSalary=salary;}
     void OutSalary()
      { System.out.println(Name +" :" + BasicSalary*5); }
}
```

Bỏ abstract → Lỗi: Method void OutSalary() requires a method body. Otherwise declare it as abstract.

Cú pháp tạo và sử dụng lớp trừu tượng(tt)

```
class Worker extends Employee
{   Worker(String aName) { Name=aName;}
    Worker(String aName, int Salary)
              { Name=aName; BasicSalary=Salary;}
    void OutSalary()
     { System.out.println(Name +" :" + BasicSalary*3);}
}
class AbstractClassDemo
{ public static void main(String args[])
  { Manager m = new Manager ("Trung", 200);
    m.OutSalary();
    Worker w= new Worker("Hoang");
    w.OutSalary();
  }
}
```

3.10-Lớp con(trong/inner)

- Là 1 lớp khai báo bên trong 1 lớp khác.
- Quan hệ : lớp ngoài (enclosing, outter class) , lớp trong (nested, inner class).
- Lớp trong có quyền truy xuất lớp ngoài.
- Lớp ngoài chỉ truy xuất được lớp trong khi có một instance *của lớp trong*

Inner class(cont.)

- Cú pháp:

```
class Outter
{   ....
    class Inner
    { …
    }
}
```

Lớp ngoài muốn truy cập lớp trong thì phải định nghĩa 1 đối tượng lớp trong (bằng toán tử new)

Lợi ích:

Có thể viết code truy xuất lớp ngoài từ lớp trong mà không cần định nghĩa đối tượng lớp ngoài

Inner class (cont.)

```
import java.io.*; // InnerClassDemo.java
class Outter
{ String Str="This is outter" ;
  boolean OutterAccess=true;
  Outter()
     {
        System.out.println(Str+", Outter Access=" +
                                   OutterAccess);
        System.out.println("InnerAccessible=" +
                                          InnerAccess);
  }
```

Thêm code:
Inner I=new Inner();

Sửa thành
I.InnerAccess

Có lỗi:Undefined variable: InnerAccess

Inner class (cont.)

```
class Inner
  {  String str="I'm inner" ;
     boolean InnerAccess=true;
     Inner()
      { System.out.println("Inner access outer:");
        System.out.println(Str + ", Outter Access=" +
                                          OutterAccess);
        System.out.println("Inner Accessible=" +
                                           InnerAccess);
      }
  }
} // hết Outter class
class InnerClassDemo
{ public static void main (String args[])
  { Outter Obj = new Outter();  }
}
```

Do constructor lớp trong thực hiện

Kết qủa

Inner access outer:

This is outter, Outter Access=true

Inner Accessible=true

This is outter, Outter Access=true

InnerAccessible=true

Do constructor lớp ngoài thực hiện

3.11- Gói phần mềm (Package)

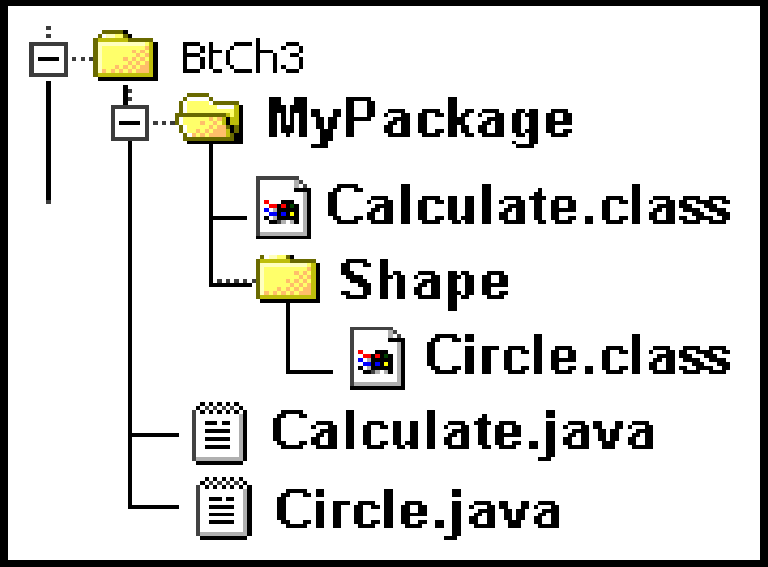
- Là một nhóm các class, interface, các gói khác đã được biên dịch thành Java bytecode.
- Tổ chức của 1 package là 1 thư mục có tên là tên của package → Sub-package là 1 gói con (thư mục con) của 1 package mức cao hơn (giống cấu trúc thư mục).
- Gói là công cụ tạo khả năng tái sử dụng mã (reusable code).

Cú pháp tạo package

```
package  PackageName;
import  OtherPackage.*;
public class Class1
{ …
}
```

Phải là dòng code đầu tiên của tập tin .java (không kể chú thích)

Thí dụ: Tạo gói MyPackage có cấu trúc:



```
package MyPackage;  // file BtCh3/Calculate.java
  public class Calculate
 {    public static double Volume(double l,
                                  double w, double h)
        {return l*w*h;}
      public static double Add(double n1,
                                       double n2)
              { return n1+n2;}
 }
```

Sau khi biên dịch, cấu trúc gói theo yêu cầu sẽ được tự động sinh ra.

```
package MyPackage.Shape;  // BtCh3/Circle.java
  public class Circle
  { double r;
    public Circle(double rr) { r=rr;}
    public double Circumference()
         { return 2*Math.PI*r;}// chu vi
    public double Area()
          { return Math.PI* r*r; }
  }
```

Sử dụng gói với chỉ thị import

- Dùng chỉ thị import

```
import Package.*;
import Package.Class;
import Package.Sub_Package.*;
import Package.Sub_Package.Class;
```

Thí dụ:

```
import java.io.*;
import MyPackage.
import MyPackage.Shape.Circle;
class TestMyPackage
{  public static void main(String args[])
   { System.out.println(Calculate.Volume(3,4,5));
     Circle C= new Circle(5);
     System.out.println(C.Area());
   }
}
```

60.0

78.53981633974483

Chương 4
Quản lý lỗi và gom rác

Mục tiêu

- Định nghĩa được exception là gì.
- Phân loại được các exception
- Sử dụng được cú pháp try..catch..finally
- Biết cách tự quản lý exception
- Giải thích được cơ chế gom rác của Java

Nội dung

4.1- Ôn tập.

4.2- Exception là gì?

4.3- Cấu trúc quản lý lỗi của Java

4.4- Mô hình *try catch finally*

4.5- Sử dụng throws

4.6- Tự định nghĩa exceptions

4.7- Cơ chế gom rác

4.8- Tóm tắt-trắc nghiệm-bài tập

4.1- Ôn tập

- Lớp là mô hình biểu diễn cho 1 tập các đối tượng có cấu trúc giống nhau.
- interface là 1 tên gọi cho một tập các **KHAI BÁO** dữ liệu hằng và hành vi hình thành nên một mô hình xử lý, các hành vi chưa được hiện thực **cần hiện thực** ở các lớp.
- Lớp trừu tượng là lớp khai báo với từ khóa abstrat và có ít nhất 1 hành vi abstract
- Hành vi abstract là hành vi chỉ mới được khai báo mà chưa hiện thực.

Ôn tập

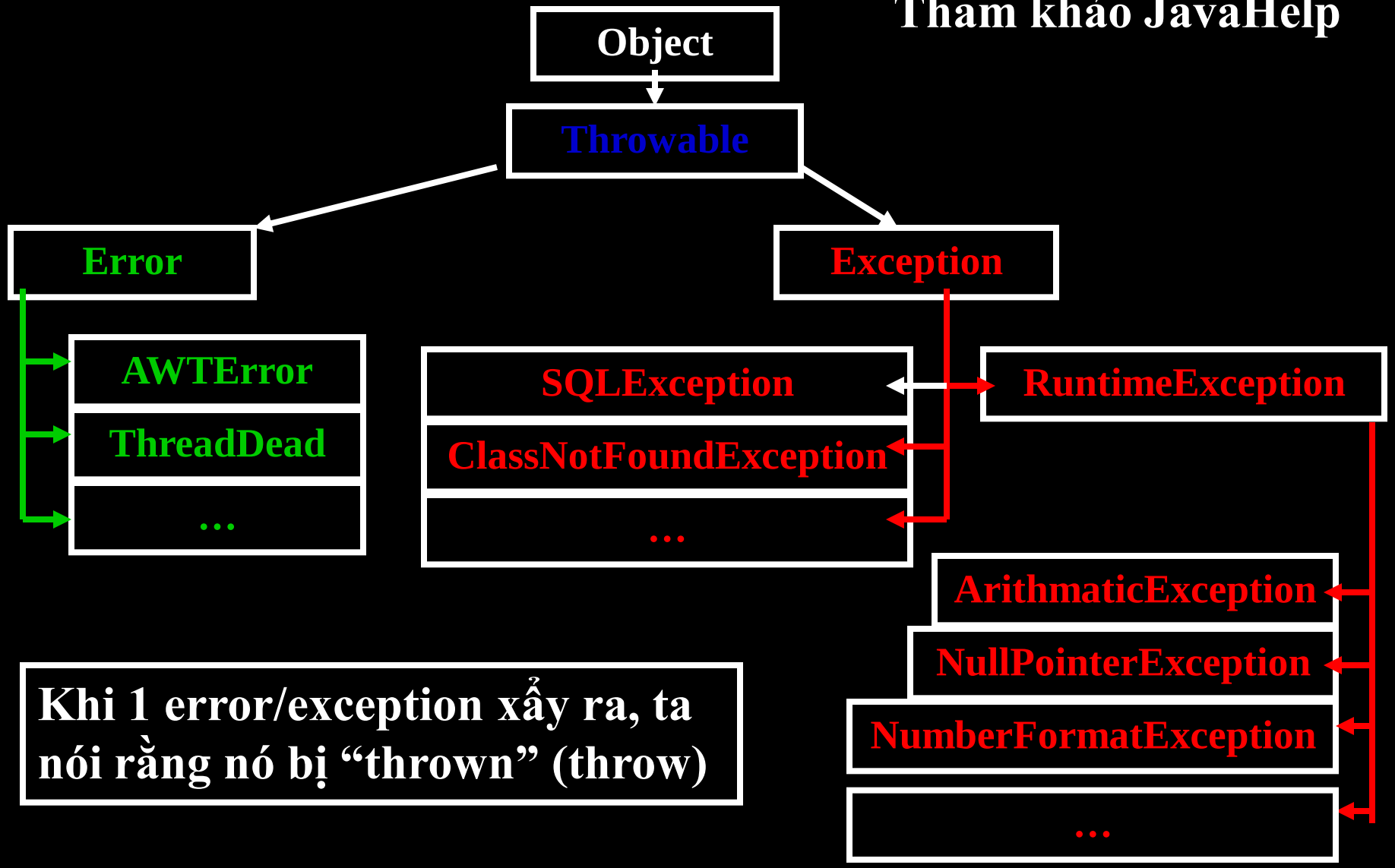
- Lớp abstract và interface khác nhau ở chỗ: 1 lớp chỉ có thể thừa kế từ 1 lớp kah1c nhưng lại có thể là 1 hiện thực của nhiều interface.
- Gói là 1 khai báo cho 1 tập các lớp, các interface và các gói cấp thấp hơn.
- Gói là 1 thư mục có tên trùng với tên gói.

4.2- Exception là gì?

- Java là ngôn ngữ mạnh, có nghĩa là tối thiểu hóa được lỗi và khi có lỗi thì chúng có thể được quản lý.
- Lỗi có 2 loại: Lỗi lúc biên dịch (compile-time error-lỗi cú pháp), lỗi lúc thực thi (run-time error- giải thuật sai, không dự đoán được tình huống).
- Exception= runtime-error
- Thí dụ: thực hiện phép chia mà mẫu số là 0
- Khi 1 exception xẩy ra, chương trình kết thúc đột ngột và điều khiển được trả lại cho OS ➔ Cần phải quản lý được các tình huống này.

4.3- Cấu trúc các class quản lý lỗi của Java

Tham khảo JavaHelp



Khi 1 error/exception xẩy ra, ta nói rằng nó bị "thrown" (throw)

Cấu trúc các class quản lý lỗi của Java(tt)

- **Exception: lớp nền của phân cấp exception.**
- **RuntimeException: Lớp nền của nhiều lớp trong java.lang.exceptions.**
- **ArithmaticException: Quản lý lỗi tóan học như chia cho 0.**
- **IllegalArgumentException : Lỗi sai đối số.**
- **ArrayIndexOutOfBoundsException: Lỗi sai chỉ số của mảng**
- **NullPointerException: Lỗi đã truy xuất 1 đối tượng chưa khởi tạo**
- **SecurityException : Lỗi không được quyền truy cập.**
- **ClassNotFoundException: Lỗi không thể nạp 1 lớp vì không có lớp này**

Cấu trúc các class quản lý lỗi của Java(tt)

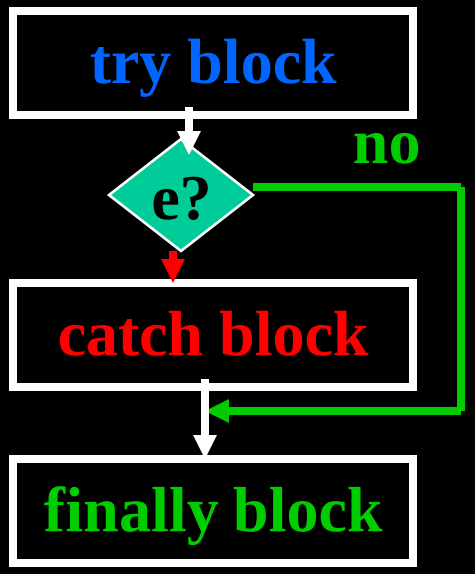
- **NumberFormatException: Lỗi chuyển String ->float.**
- **AWTException: Lỗi Abstract Windowing Toolkit**
- **ArithmaticException: Quản lý lỗi tóan học như chia cho 0.**
- **IOException : Lớp nền của IO exception.**
- **FileNotFoundException: Lỗi không có file đã đặc tả**
- **EOFException: Lỗi cố truy xuất dữ liệu mà file đã hết**
- **IllegalAccessException : Lỗi truy xuất đến 1class bị cấm.**
- **NoSuchMethodException: Lỗi không có method đã đặc tả**
- **InterruptedException: luồng bị ngắt**

4.4- Mô hình *try catch finally*

- Có thể thử thực thi 1 tác vụ (try), nếu xuất 1 lỗi thì bẫy lỗi (catch) để xử lý tình huống lỗi và cuối cùng thực thi tiếp (finally).
- Cú pháp

(Không có các cặp {} → Lỗi: '{' expected)

```
try {….}
catch (Exception e) { …}
finally { …}
…
```



Mô hình *try catch finally*

```
import java.io.*; // TryCatchDemo.java
class TryCatchDemo
{ static double Divide ( double a, double b) { return a/b;}
  public static void main(String args[])
  {  try { System.out.println(Divide(5,0));}
     catch( Exception e)
      { System.out.println("System exception:"+ e.toString());}
     finally
      { System.out.println("I tried to divide 5 by 0");}
    System.out.println("End!");
  }
}
```

```
Infinity
I tried to divide 5 by 0
End!
```

```
import java.io.*; // ArrayCatch.java
class ArrayCatch
{ static String S;
  static void Out() { System.out.println(S);}
  public static void main(String args[])
  {  try  { Out();      }
     catch (NullPointerException e)
           { System.out.println("Exception occured:");}
      finally  { int a[]= { 1,2,3,4,5};
                 try  { System.out.println(a[7]);}
                 catch(ArrayIndexOutOfBoundsException e2)
                      { System.err.println("Out of bounds");}
                }
  }
}
```

```
null
Out of bounds
```

4.5- Sử dụng throw

- Tóan tử *throw* cho phép chỉ thị 1 exception đã xẩy ra. Kết qủa là 1 đối tượng của 1 lớp dẫn xuất của lớp *Throwable.*
- *Tình huống:* method X() gọi method Y(), Y() thực thi gây ra 1 exception mà không được quản lý, exception này lan về cho phương thức gọi là X() . Nếu trong X() cũng không quản lý lỗi ➔ Lỗi truyền về cho nơi đã gọi X()
- Cách giải quyết:

  a) Trong Y() có quản lý lỗi;

  b) Trong X() có có cấu trúc try…catch để quản lý lỗi.

Sử dụng throw/throws ...

- **Cú pháp sinh 1 Exception trong hàm**

```
ReturnType  Method (…) throws  ExceptionClass
{    if (…) throw ExceptionClass(“Message”);
     else { …… }
}
```

Thí dụ về lan truyền exception

```
import java.io.*; // ArrayCatch.java
class ArrayCatch
{ int a[]= { 1,2,3,4,5};  int n=5;
  public static void main(String args[])
  { ArrayCatch Obj= new ArrayCatch();   Obj.OutElement(7); }
  void OutElement(int i) { System.out.println (a[i]);   }
}
```

Kích thước của mảng: 5

Chỉ số truy xuất: 7

```
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException
        at ArrayCatch.OutElement(ArrayCatch.java:10)
        at ArrayCatch.main(ArrayCatch.java:7)
```

Cách sửa 1- Try catch bẫy lỗi trong main(…)

```
import java.io.*; // ArrayCatch.java
class ArrayCatch
{ int a[]= { 1,2,3,4,5};  int n=5;
  public static void main(String args[])
  {   ArrayCatch Obj= new ArrayCatch();
      try { Obj.OutElement(7);}
      catch (ArrayIndexOutOfBoundsException e)
            { System.out.println("Illegal index");}
  }
  void OutElement(int i) { System.out.println (a[i]);   }
}
```

```
Illegal index
Press any key…
```

Bẫy lỗi

Cách sửa 2- Try catch bẫy lỗi trong OutElement(…)

```
import java.io.*; // ArrayCatch.java
class ArrayCatch
{ int a[]= { 1,2,3,4,5};   int n=5;
  public static void main(String args[])
  {  ArrayCatch Obj= new ArrayCatch();Obj.OutElement(7);}
  void OutElement(int i) throws
                              ArrayIndexOutOfBoundsException
   { if (i<0 || i>=n) throw new
        ArrayIndexOutOfBoundsException
        ("\nIndex is out of array");
     else System.out.println (a[i]);
   }
}
```

Try catch bẫy lỗi trong OutElement(…) - tt

Kết qủa:

```
Exception in thread "main"
    java.lang.ArrayIndexOutOfBoundsException:
Index is out of array
        at ArrayCatch.OutElement(ArrayCatch.java:9)
        at ArrayCatch.main(ArrayCatch.java:6)
```

4.6- Tự định nghĩa exceptions

- **User-defined Exception: Tạo ra 1 lớp con của các lớp Error hoặc Exception**

```
import java.io.*; // ArrayCatch.java
class MyException extends ArrayIndexOutOfBoundsException
{ MyException() { super("\nIndex is out of array");}}
class ArrayCatch
{ int a[]= { 1,2,3,4,5};  int n=5;
  public static void main(String args[])
  {  ArrayCatch Obj= newArrayCatch();Obj.OutElement(7);}
  void OutElement(int i) throws MyException
   { if (i<0 || i>=n) throw new MyException();
     else System.out.println (a[i]);
   }
}
```

4.7- Cơ chế gom rác

- Là cơ chế tự động của Java để hủy bỏ các đối tượng không còn dùng nữa.
- Hiện thực bằng heap động (xem lại chương 1).
- Dù có thủ công gọi trình gom rác bằng System.gc() cũng không bảo đảm việc gom rác được thực thi ngay lập tức.
- Có thể thủ công tắt trình gom rác bằng chỉ thị

  java –noasyncgc File.class

  nhưng có thể phải trả giá là thiếu bộ nhớ do số đối tượng sinh ra trong chương trình khó tiên liệu → Hiệu suất chương trình kém.

Cơ chế gom rác- finalize() method

- Java cung cấp phương thức *finalize()* hoạt động như 1 Destructor của C++ để hủy các quá trình trước khi qúa trình trả điều khiển về cho hệ điều hành.
- Chú ý: Chỉ có đối tượng mới bị gom rác chứ tham khảo đến đối tượng không bị gom.
- Cú pháp:

protected void finalize() throws Throwable

Thí dụ:

Object a= new Object(); Object b=a; a=null;

→a, b là các references của 1 đối tượng.

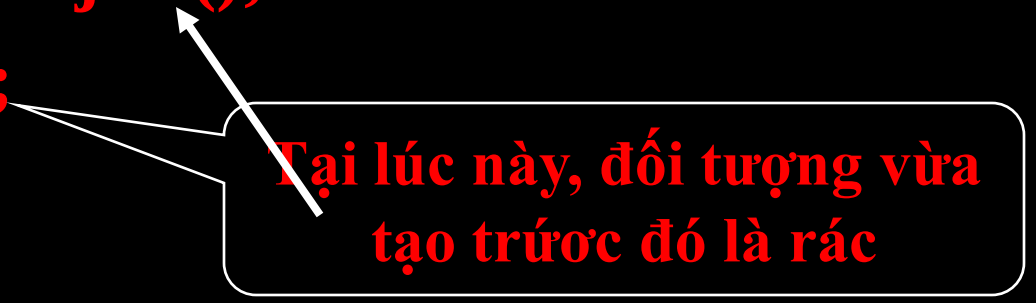
→a=null, nghĩ rằng đối tượng không còn dùng nữa → Sai vì b vẫn tham khảo đến nó

Object c= new Object();

c=null ; // đối tượng này bây giờ là rác

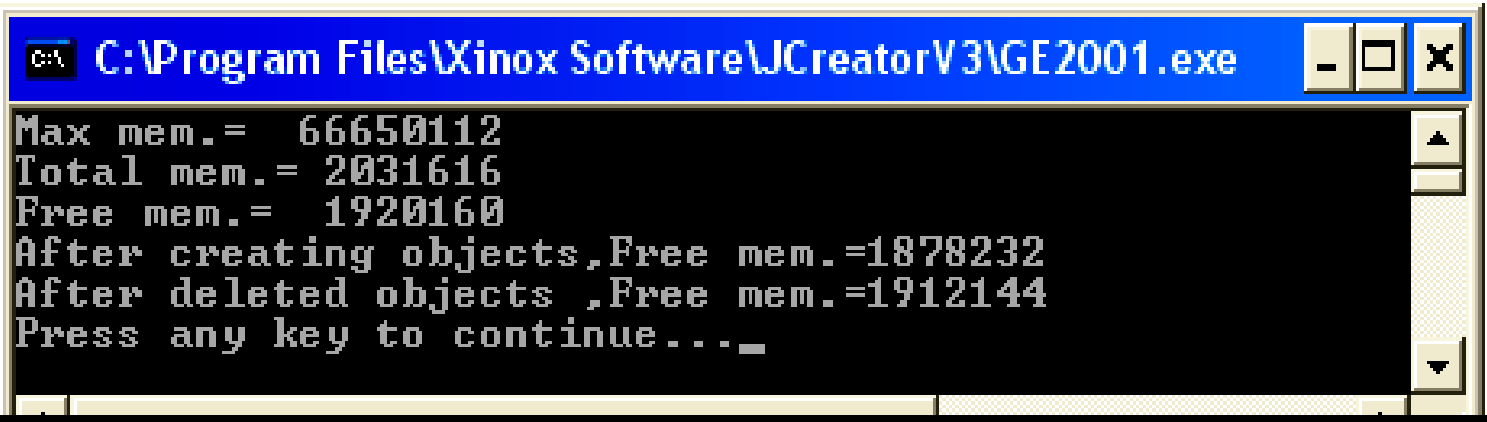
Object d= new Object();

d=new Object();



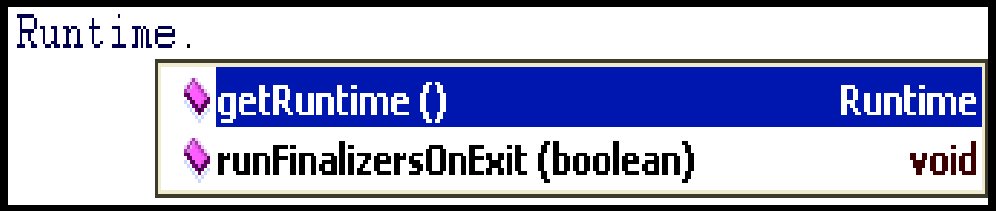
Thí dụ:

```
1  import java.io.*; // GarbageDemo.java
2  class GarbageDemo
3  { public static void main (String args[])
4    {int i; long a; Runtime r= Runtime.getRuntime();
5      r.gc();// call garbage collector before initiating objects
6      System.out.println("Max mem.=   " + r.maxMemory());
7      System.out.println("Total mem.= " + r.totalMemory());
8      System.out.println("Free mem.=  " + r.freeMemory());
9      Long Values[]= new Long [2000]; // array of Long objects
10
11     for (a=10000, i=0; i<2000; a++,i++) Values[i]=new Long(a);
12     System.out.println("After creating objects,Free mem.="+
13                                                  r.freeMemory());
14     for (i=0;i<2000;++i) Values[i]=null; // delete objects
15     r.gc();// call garbage collector after deleting objects
16     System.out.println("After deleted objects ,Free mem.="+
17                                                    r.freeMemory());
18   }
19 }
20
```



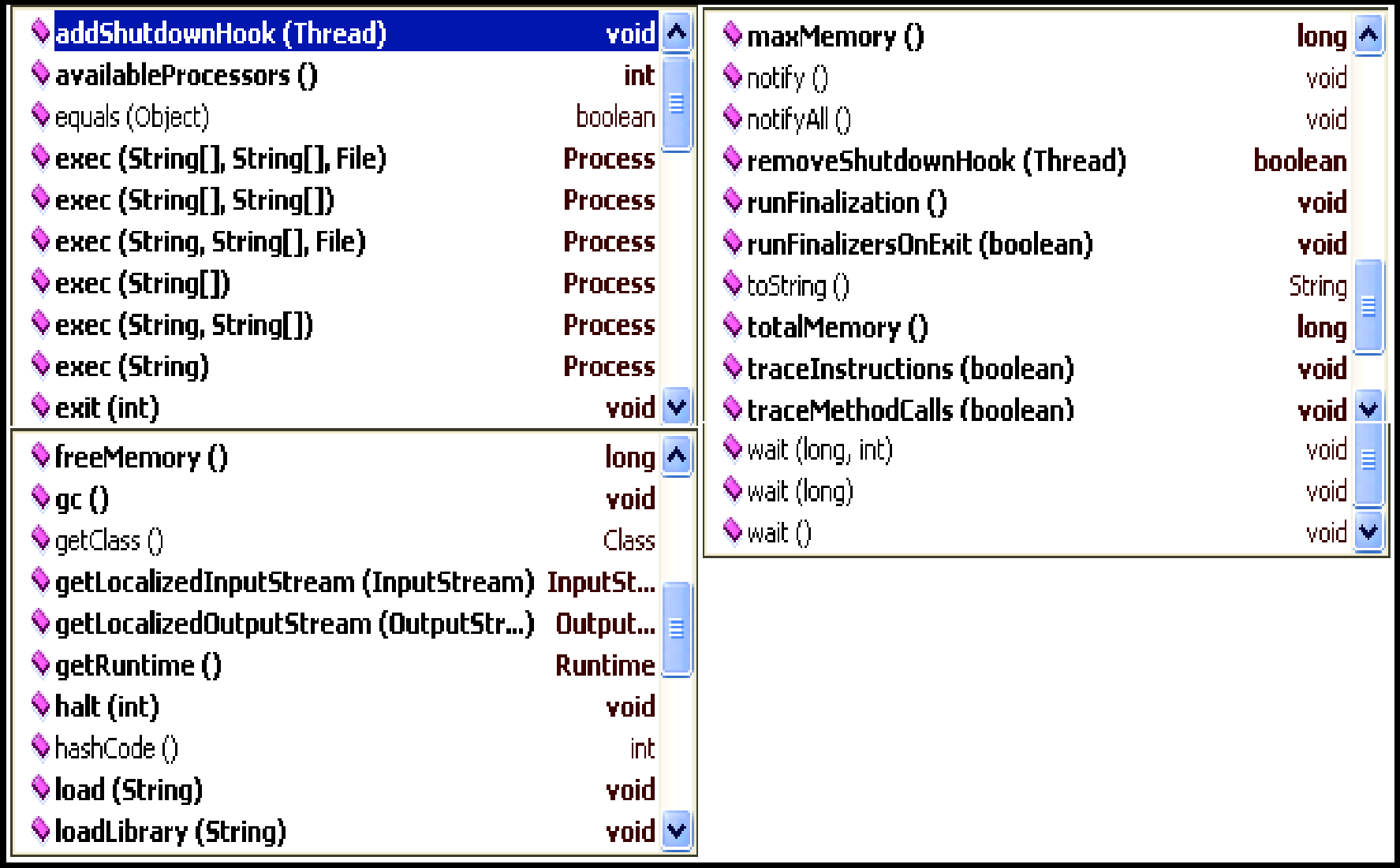
Trong chương trình trên

- Lớp RunTime mô tả hệ thống lúc thực thi
- Dùng hành vi static getRunTime để khởi tạo 1 đối tượng RunTime



- **Một số methods của lớp Runtime**

Methodes của lớp RunTime



4.8- Tóm tắt-trắc nghiệm-bài tập

1. **Có hai lọai lỗi: Lỗi lúc biên dịch và lỗi khi thực thi.**
2. **Lỗi biên dịch là lỗi ……**
3. **Lỗi lúc thực thi còn gọi là ……..**
4. **Nếu không quản lý Exception, chương trình sẽ ngắt đột ngột và điều khiển được trả về cho ……..**
5. **Hòan tòan có thể bẫy được các ……..**
6. **Quản lý exception cho phép xử lý lỗi đúng lúv (true/false)**
7. **5 từ khóa được dùng để bẫy lỗi : ……..**
8. **Từ khóa throws cho phép dùng đối với các exception mà 1 hàm có thể xử lý (true/false)**
9. **Từ khóa throw chỉ thị rằng 1 exception đã xẩy ra (true/false)**
10. **Từ khóa finally chỉ thị nơi bắt đầu 1 khối phát biểu không phụ thuộc vào 1 lỗi có xẩy ra hay không (true./false)**
11. **Ta có thể tự tạo ra 1 Exception class (true/false)**
12. **System.gc() sẽ yêu cầu hệ thống ……..**

Bài tập

1- Viết chương trình chạy bằng đối số dòng lệnh buộc nhập các tham số cho chương trình là các ký số. Nếu nhập ký tự thì báo lỗi "Không nhập ký tự"

Lưu ý: Cú pháp java file.class arg1, arg2, …

2- Viết chương trình nhập vào 1 mảng số int, nhập 1 vị trí i, xuất phần tử thứ i nếu i hợp lệ. Ngược lại xuất thông báo "Ngòai tầm phủ sóng". Gợi ý: xem lại chương 2 về nhập số int, xem trong chương này về Exception ứng với tình huống ngòai tầm phủ sóng:" này.

Chương 5

Tạo giao tiếp người dùng Graphic User Interface- GUI

Mục tiêu

- Hiểu mục đích của gói AWT và cấu trúc của gói này.
- Biết cách sử dụng các đối tượng của gói AWT.
- Biết cách bố cục một GUI.
- Biết cách thiết kế một GUI.

Nội dung

5.1- Ôn tập

- 2 loại lỗi của 1 chương trình: Compile-time error / Run-time error .
- Exception = Run-time error
- Có thể bẫy 1 exception bằng cấu trúc

  try {…}

  catch (ExceptionClass e) {…}

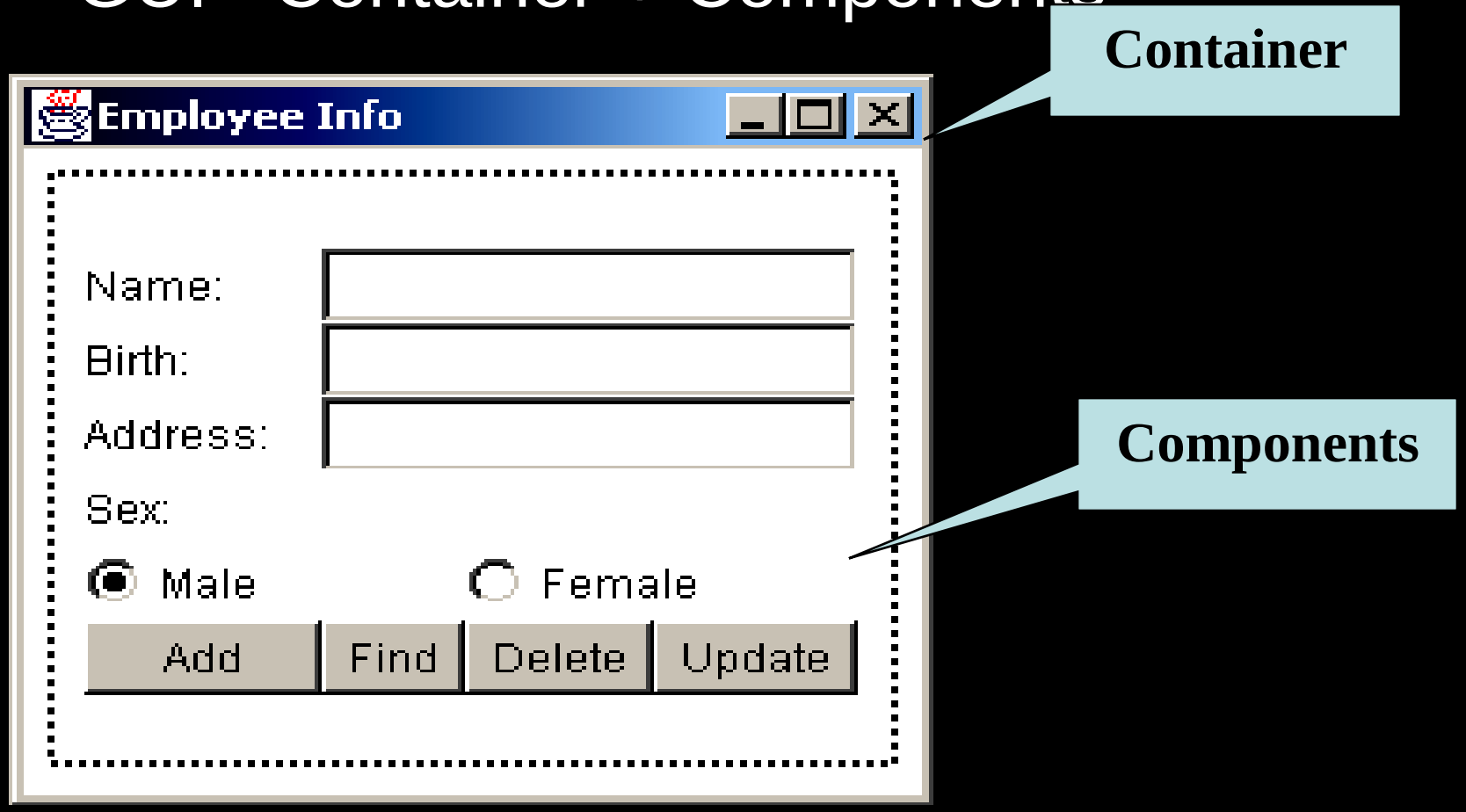
  finally { … }
- Lỗi được truyền từ method gây exception lên các method gọi nó.
- Có thể tự định nghĩa 1 class Exception kế thừa từ các lớp Exception của Java.

5.2- GUI là gì?

- GUI = Graphic User Interface – mô hình giao tiếp kiểu tương tác giữa ứng dụng và user dạng đồ họa.
- Mỗi ngôn ngữ hỗ trợ cách tạo GUI khác nhau: VB, VC++ dùng dạng *drag and drop*, C++ đòi hỏi programmer viết toàn bộ code để tạo GUI, Java hỗ trợ sẵn các lớp tạo GUI cho Programmer sử dụng.

GUI là gì?...

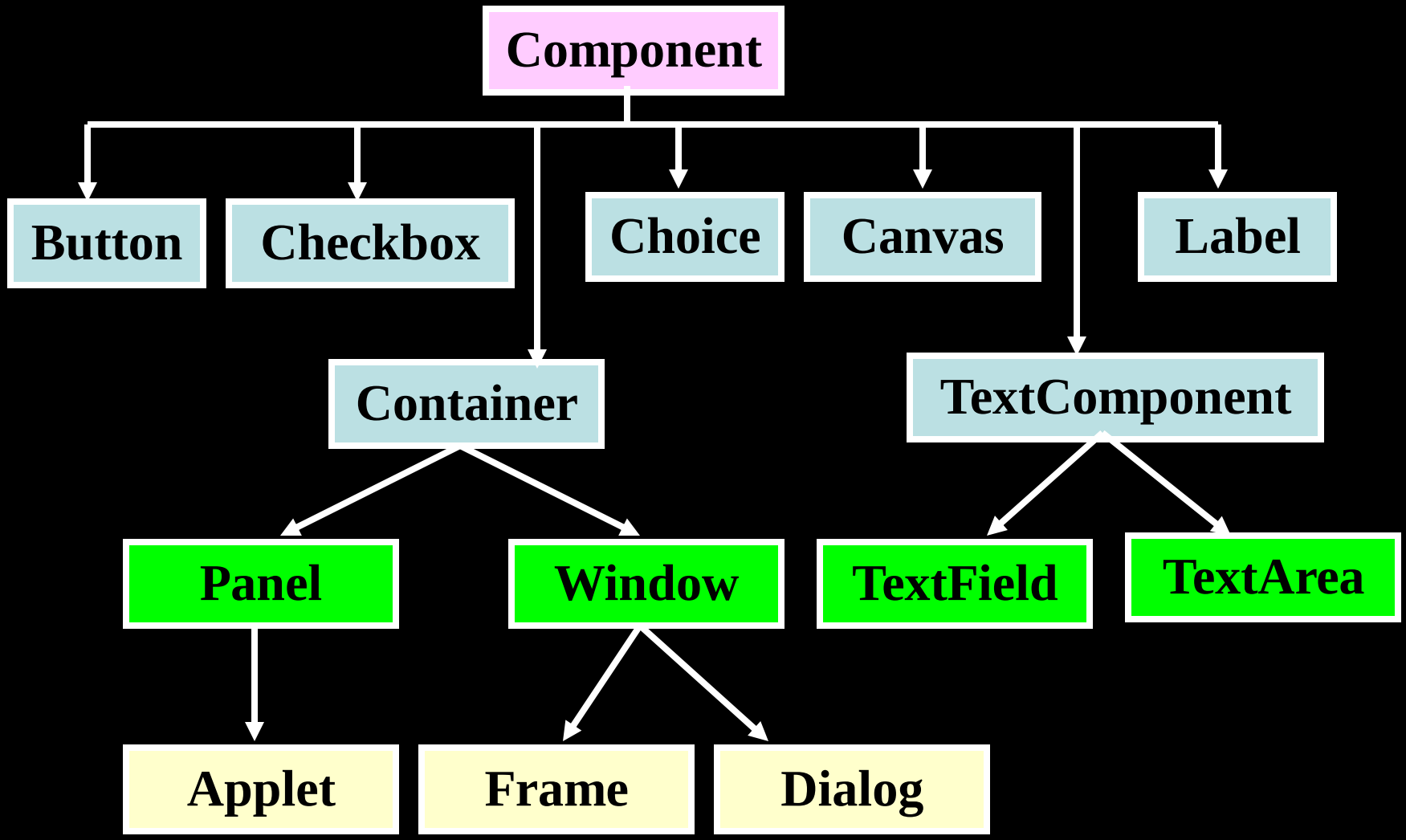
- GUI= Container + Components



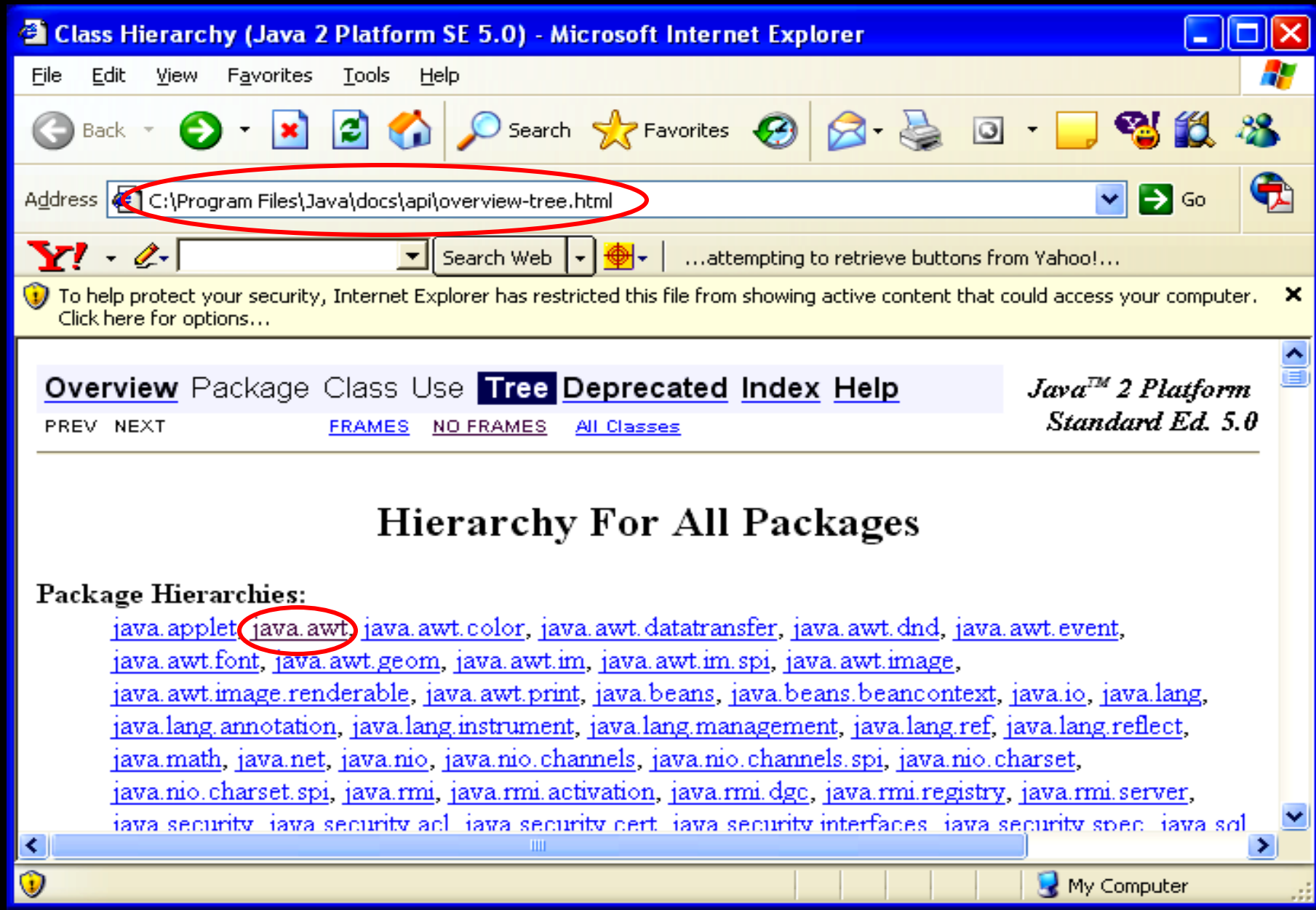
5.3- Gói AWT của Java

- AWT : abstract windowing toolkit - bộ công cụ chứa các lớp để tạo cửa sổ.
- AWT là 1 phần của JFC- Java Foundation Classes.
- Sử dụng: import java.awt.*;
- Gồm nhiều phần tử (class) để tạo GUI.
- Có các lớp quản lý việc bố trí các phần tử.
- Có (event-oriented application) mô hình ứng dụng hướng sự kiện.
- Có các công cụ xử lý đồ họa và hình ảnh.
- Các lớp sử dụng các tác vụ với clipboard (vùng nhớ đệm) như cut, paste.

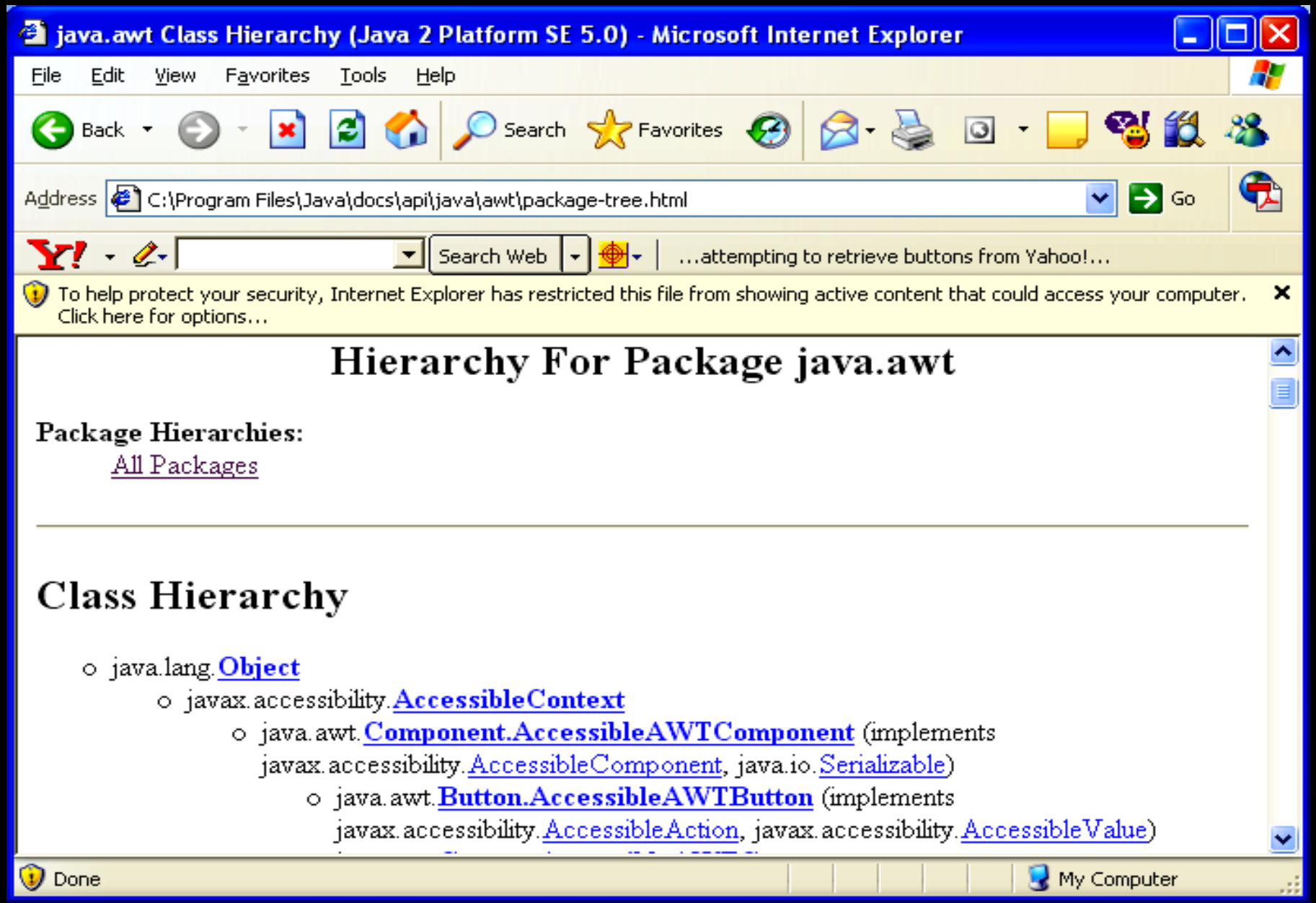
Cấu trúc gói AWT



Tham khảo gói java.awt



Tham khảo gói java.awt



Yêu cầu của GUI

- Thân thiện với user.
- Số phần tử (element, component) trên GUI thay đổi tùy thuộc vào ứng dụng.
- Khi user tương tác với phần tử của GUI, ứng dụng phải có phản ứng.
- Lập trình sự kiện sẽ bàn đến trong chương sau.

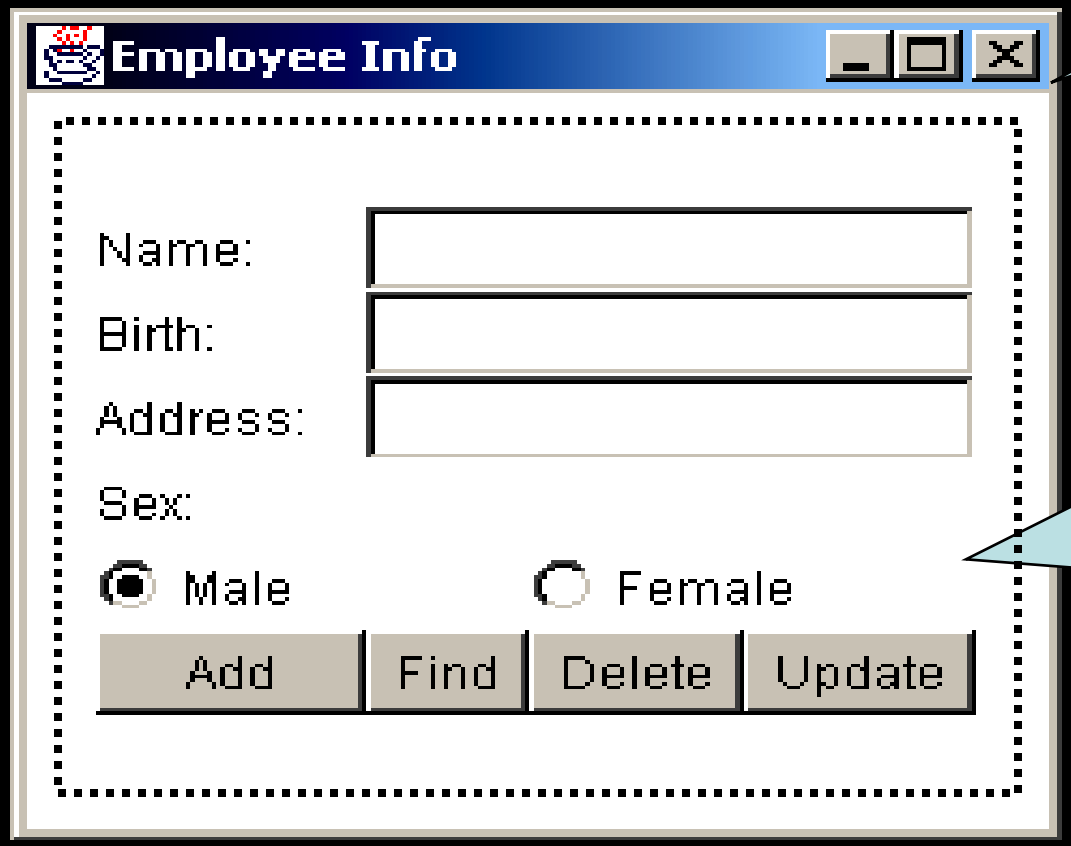
5.4- Đưa 1 component vào GUI

Các bước để đưa 1 component vào GUI

(viết code)

- Tạo 1 đối tượng component phù hợp.
- Xác định hình thức bên ngoài lúc đầu của component.
- Định vị component này trên GUI.
- Thêm component này vào GUI.

5.5- Một thí dụ



Container

Components

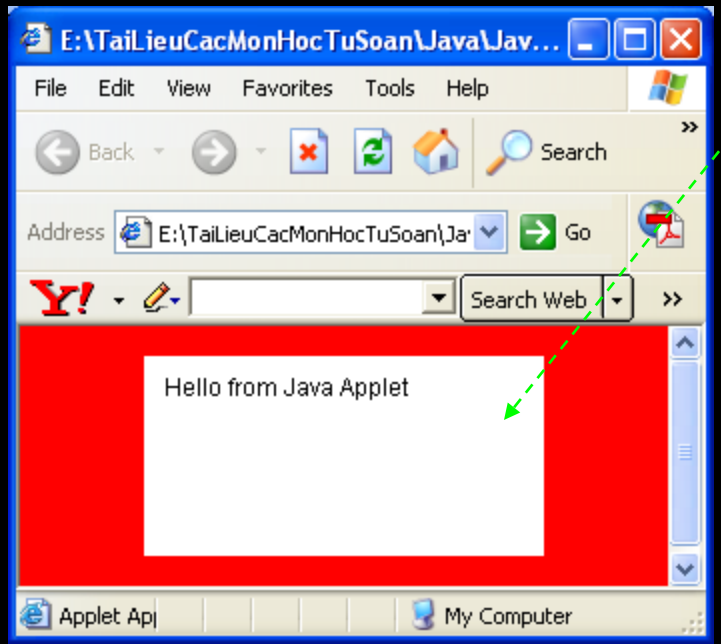
- 3 label,
- 3 text-field
- 1 checkboxgroup chứa 2 check-box
- 4 button

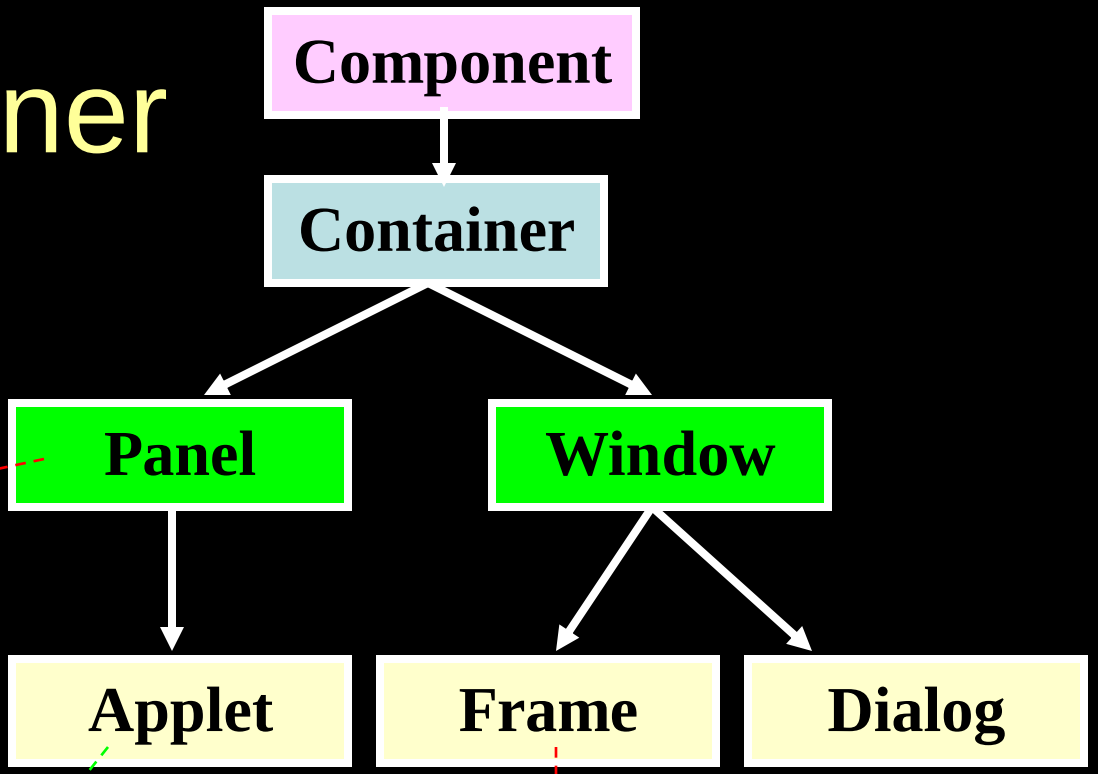
5.6- Sử dụng các lớp của awt

5.6.1- Container

Phân cấp thừa kế

Panel là 1 vùng chữ nhật, không có đường viền





Panel là 1 khung chữ nhật, có đường viền, có các nut1 điều khiển cửa sổ

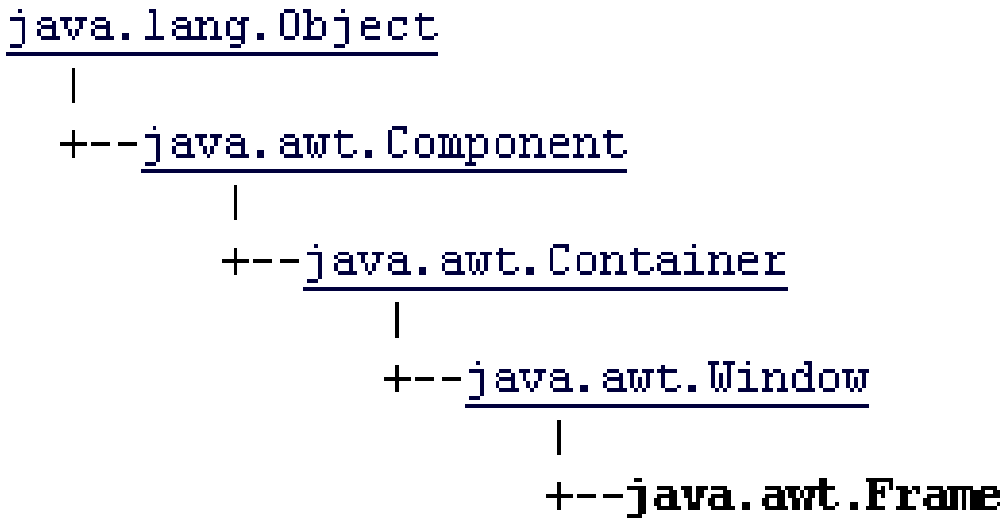
Container...

- Container: Đối tượng chứa các element, cho phép vẽ, tô màu lên container.
- Frame và Panel là các class thường dùng.
- Panel thường dùng để chứa các element trong 1 GUI phức tạp, 1 Frame có thể chứa nhiều Panel.
- Panel, Applet thường dùng để tạo 1 ứng dụng nhúng vào Browser.

5.6.2- Frame

java.awt

Class Frame



Constructors:

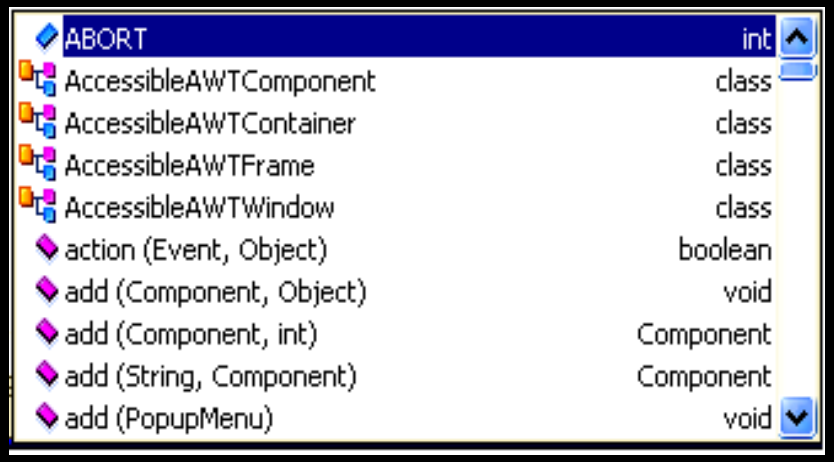
Frame() → **Make invisible frame**

Frame(String) → **Make a visible frame with title**

Frame...

Common methods

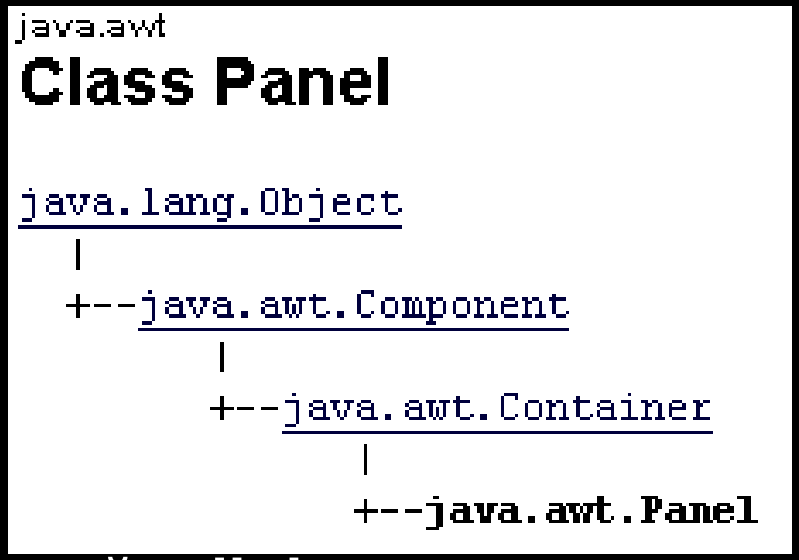
void SetSize(int width, int Height)
public String getTitle();
public void setTitle(String title);
public void setResizable(boolean resizable)
public boolean isResizable()
public void setVisible(boolean)
public boolean isShowing()
void show(boolean)
void add (...) // add component
...



Click for Demo

5.6.3- Panel

Panel phải được đưa vào Frame khi viết application vì Frame mới có border



Constructors

Panel(): tạo 1 panel với bố cục mặc định.

Panel(LayoutManager layout): tạo 1 panel với bố cục đã biết.

Methods:

add (component) // thêm 1 component vào panel

setLayout(LayoutManager layout) //chọn kiểu bố trí components

Click for Demo

5.6.4- Label

- Nhãn nhằm giải thích, chứa dữ liệu chỉ xuất.

Constructor:

Label() : tạo label trống

Label(String) : tạo label có chuỗi

Label (String, int Align) Tạo label có gióng hàng: Align=LEFT,RIGHT,CENTER

Common Methods:

void setFont (Font f)

void setText(String S)

String getText()

5.6.5- TextField

- Chứa dữ liệu nhập 1 chuỗi ký tự.
- User chỉ được nhập 1 dòng.

Constructors

TextField() : tạo text field trống
TextField(int cols) : tạo text field trống có n cột chữ
TextField(String S) : tạo text field có chuỗi S
TextField (String S, int cols) : text có chuỗi S, n cột

Common Methods:

void setEchoChar (char c) - ấn định ký tự thể hiện (password)
void setText(String S)
String getText()
void setEditable(boolean b)
boolean isEditable()

5.6.6- TextArea

- Chứa dữ liệu nhập 1 chuỗi ký tự.
- User chỉ được nhập nhiều dòng.

Constructors

TextArea()
TextArea(int cols, int rows)
TextArea(String S)
TextArea(String S, int rows, int cols)
TextArea(String,int cols, int rows, int Scrollbars)

Common methods

void setText(String)
String getText()
void setEditable(boolean)
boolean **isEditable**()
vois **insert**(String S, int Index)
void **replaceRange**(String S, int begin, int end)

5.6.7- Button

- Công cụ để user chọn 1 tác vụ.

Constructors

Button() - tạo nút không có nhãn

Button(String S) - tạo nút có nhãn

Common Methods

void setLabel(String) - đổi nhãn

String getLabel() - lấy nội dung nhãn

5.6.8- Checkbox/CheckboxGroup

- **Công cụ nhập yes/no**
- **Checkbox: multi-option (cho phép chọn nhiều) thể hiện dạng hộp vuông.**
- **CheckboxGroup chứa nhiều checkbox nhưng chỉ cho phép chọn 1/n. Phần tử trong CheckboxGroup là đối tượng thuộc lớp Checkbox nhưng lại thể hiện dạng nút tròn (radio button)**

Checkbox/CheckboxGroup...

Checkbox constructors

Checkbox() Tạo checkbox không nhãn

Checkbox(String) Tạo text box có nhãn

Tạo 1 checkbox có nhãn, có ấn định trị chọn lựa đưa vào 1 nhóm trong 1 nhóm

Checkbox(String label , boolean state, CheckboxGroup group)

Checkbox(String label, CheckboxGroup group, boolean state)

Common methods

void setLabel(String); String getLabel();

void setState(boolean); boolean getState();

5.6.9- List

- Công cụ nhập bằng cách chọn trong 1 danh sách chuỗi ký tự.
- Có thể chọn 1 hoặc nhiều.

Constructor

Choice() – tạo 1 danh sách trống

Common methods

void add(String) ; void addItem(String);

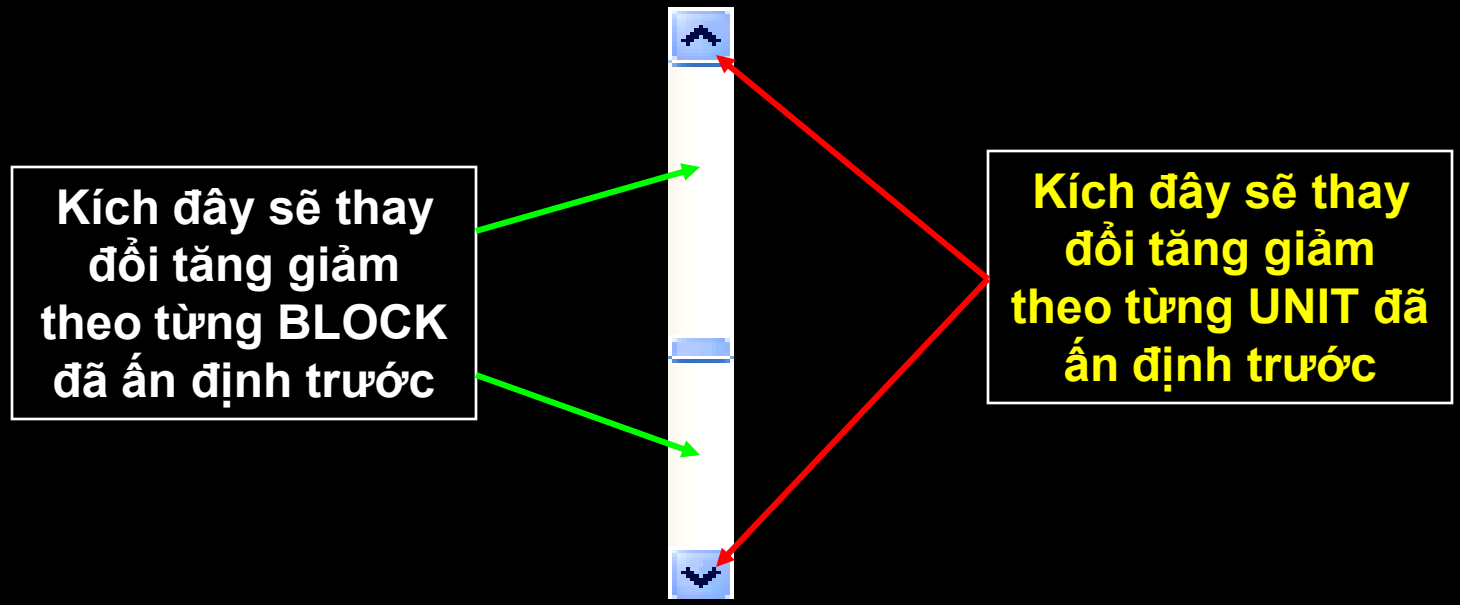
void insert(String item, int index)

int CountItems(); int getItemCount(); - lấy sốphần tử

String getItem(int Index);
int getSelectedInddex();
String getSelectedItem();
void remove(int position)
void removeAll()
void select(int pos) – áp đặt 1 mục chọn theo vị trí
void select(String str) – áp đặt chọn 1 mục chọn theo nội dung

5.6.10- Scrollbar- Thanh cuộn

- Công cụ nhập 1 trị trong 1 khoảng số (biểu diễn bằng Maximum, Minimum) bằng cách kéo con trượt.
- Tại 1 thời điểm, con trượt ở tại vi trí mô tả cho trị hiện hành (Value)
- Có thể có hướng ngang hoặc dọc (Orientation)



Scrollbar...

Constructors

Scrollbar() - tạo thanh cuộn dọc

Scrollbar(int orientation) // VERTICAL|HORIZONTAL

Scrollbar(int orientation, int value, int visible, int minimum, int maximum)

Common methods

void setMaximum(int v)	**int getMaximum() ;**
void setMinimum(int v)	**int getMinimum()**
int getOrientation()	
void setUnitIncrement(int v)	**int getUnitIncrement()**
void setBlockIncrement(int v)	**int getBlockIncrement()**
void setValue(int v)	**int getValue()**
void setVisibleAmount(int newAmount)	
int getVisibleAmount()	

Minh họa

Click for Demonstration

5.7- Bố trí các components lên GUI

- Layout : Cách bố trí các components lên container.
- Không dễ dàng gì để tự quản lý vị trí của các components trên GUI.
- LayoutManager là interface mô tả về các layout.
- Trong gói AWT có hiện thức sẵn một số layout, các lớp layout này đều implement LayoutManager interface.

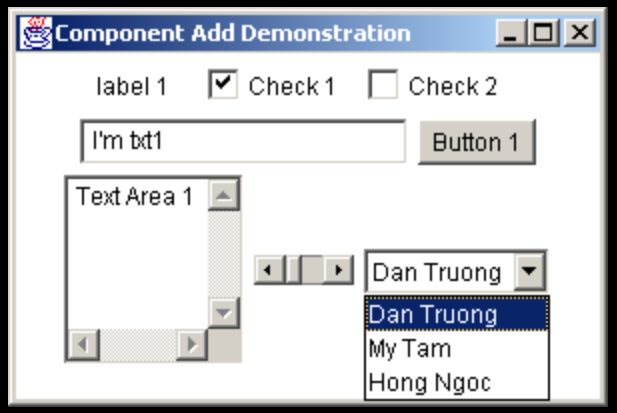
5.7.1-Layouts có sẵn trong AWT

- java.awt.**FlowLayout** (bố trí dạng dòng chảy)
- java.awt.**BorderLayout** (bố trí về biên khung)
- java.awt.**GridLayout** (bố trí dạng lưới đều nhau)
- java.awt.**GridBagLayout** (bố trí dạng lưới không đều)
- java.awt.**CardLayout** (bố trí dạng lưng quân bài)
- **Tham khảo**

docs\api\java\awt\package-tree.html

vớo docs là thư mục Documantation của Java 2

5.7.2- FlowLayout



- Bố trí các component theo dạng chảy xuôi theo thứ tự mà phần tử này được add vào container.
- Đây là layout mặc định của Panel.
- Nếu có nhiều component trên container → Các component có thể được đặt trên nhiều dòng → Vấn đề gióng hàng (Align)
- Giữa các component, chúng hở nhau theo chiều dọc (vgap) bao nhiêu, theo chiều ngang (hgap) bao nhiêu?

FlowLayout...

Constructors

FlowLayout()
Tạo FlowLayout mặc định: align= center, vgap=hgap=5 unit.

FlowLayout(int align)
Tạo FlowLayout với align đã biết, vgap=hgap=5 unit (default).

FlowLayout(int align, int hgap, int vgap)
Tạo FlowLayout với 3 tham số

Trị của align: các dữ liệu static của class FlowLayout

LEFT CENTER RIGHT

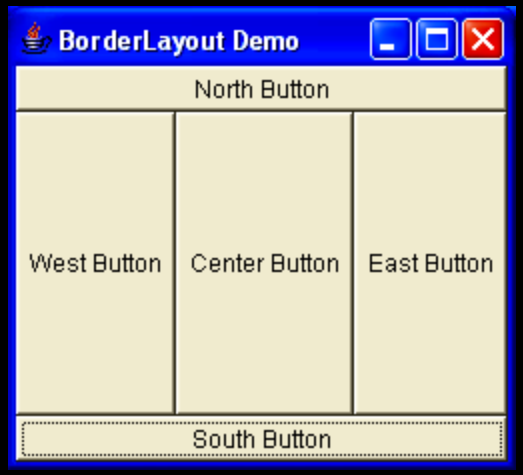
LEADING (phía đầu, tương tự LEFT)

TRAILING (phía đuôi, tương tự RIGHT)

Click for Demo

5.7.3- BorderLayout

- Bố trí các component theo dạng ra biên của khung tạo ra 5 vị trí: EAST, WEST, SOUTH,NORTH, CENTER.
- Đây là layout MẶC ĐỊNH của Frame.
- Nếu container chỉ có 1 component và đặt nó ở vị trí CENTER thì component này phủ kín container.
- Cú pháp thêm 1 component vào container tại 1 vị trí:
  Container.add("East", componentName); // hoặc
  Container.add(BorderLayout.EAST, componentName);
- Tương tự cho “West”, “South”, “North”, “Center”



Click for Demo

5.7.4- GridLayout

- Bố trí các component thành 1 lưới rows dòng, cols cột đều nhau.

GridLayout...

Constructor

GridLayout()

Tạo grid layout mặc định 1x1

GridLayout(int rows, int cols)

Tạo grid layout rows x cols

.**GridLayout**(int rows, int cols, int hgap, int vgap)

Click for Demo

5.7.5- GridBagLayout

- Layout dạng lưới cho phép 1 component chiếm 1 số ô kề nhau theo cả 2 chiều.
- Hình Empoyee Info sau là GridBagLayout 6x4, các label bên trái chiếm 1ô, các textbox chiếm 3 ô ngang. Dòng “Sex” chiếm 4 ô ngang, 2 checkbox chiếm 2 ô ngang.

Các thí dụ khác

Class java.awt.GridBagLayout

- **Constructors**: **GridBagLayout**()
- **Áp đặt GridBagLayout cho 1 container:**

```
GridBagLayout gb= new GridBagLayout();
FrameName.setLayout(gb);
PanelName.setLayout(gb);
```

Viết ngắn gọn:

```
FrameName.setLayout(new GridBagLayout());
PanelName.setLayout(new GridBagLayout());
```

class GridBagConstraints

- Làm sao để có thể đưa 1 component vào 1 vị trí nhưng trải dài trên nhiều ô kề nhau?
- 1 component vào 1 vị trí nhưng trải dài trên nhiều ô kề nhau là 1 sự “ràng buộc” 1 component vào các ô này. Một đối tượng thuộc lớp GridBagConstraints sẽ đảm nhiệm việc này.

class GridBagConstraints...

- **Properties – Đa số là static data**

  int gridx, gridy : ô sẽ đặt component vào

  int gridwidth, gridheight : số ô sẽ phủ theo 2 chiều khi thêm 1 component vào ô <gridx,gridy>

  double weightx, weighty : Khoảng hở của lưới, mặc định là 0.

  int anchor : Vị trí đặt component, mặc đi6nh là CENTER, các static int được khai báo sẵn: GridBagConstraints.NORTH, EAST,WEST, SOUTH, NORTHEAST, SOUTHEAST, NORTHWEST, SOUTHWEST.

  int fill: Xác định kiểu đặt khi component có kích thước lớn hơm ô sẽ được đặt vào. Các hằng được dùng: GridBagConstraints.NONE, HORIZONTAL, VERTICAL, BOTH.

  Insets insets : Đặc tả khoảng hở <top, bottom,left, right> giữa các phần tử được đưa vào, mặc định là 0.

  int ipadx, ipady: Khoảng độn (số pixel trống) bên trong của phần tử theo 2 chiều. Mặc định là 0. Khi vẽ phần tử, sẽ thêm 2*ipadx và 2*ipady vào chiều rộng tối thiểu và chiều cao tối thiểu của phần tử.

class GridBagConstraints...

- **Constructor:**

GridBagConstraint() // tạo object với các dữ liệu mặc định.

GridBagConstraints(int gridx, int gridy, int gridwidth, int gridheight, double weightx, double weighty, int anchor, int fill, Insets insets, int ipadx, int ipady)

- **Thao tác với GridBagConstraint object bằng static data**
- **Làm sao kết hợp GridBagConstraints với GridBagLayout?**

```
GridBagLayout gbLayout = new GridBagLayout();
GridBagConstraints gbc = new GridBagConstraints();
gbLayout.setConstraints(gbc);
```

class GridBagConstraints...

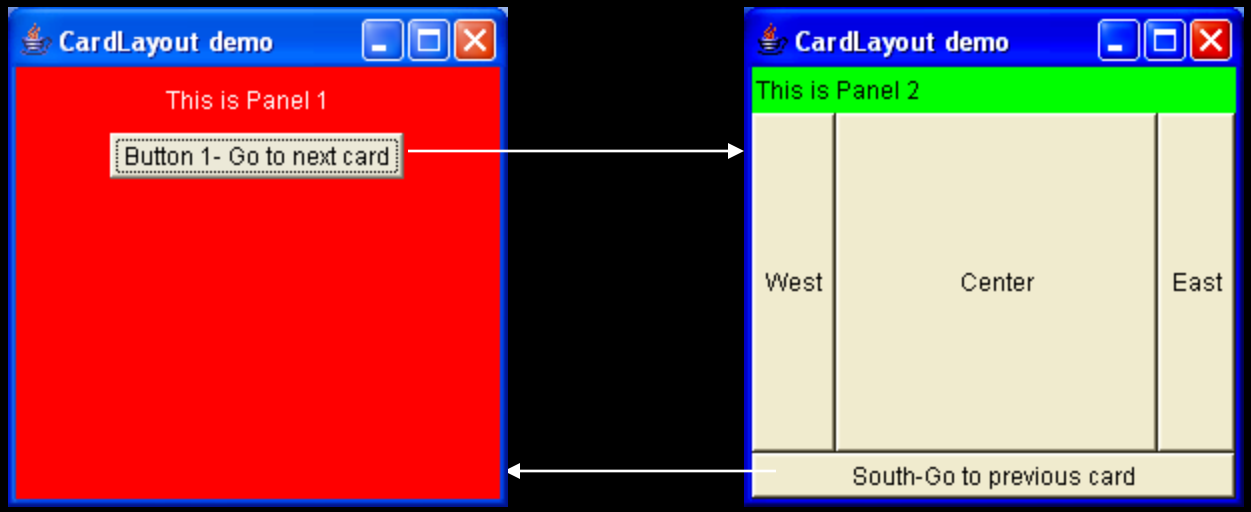
Method sau đây sẽ thêm component c vào vị trí ô (row,col) kéo tràn dọc nrow ô và tràn ngang ncol ô. Trong đó gbc là GridBagConstraints và gbc là GridBagLayout

```
void addComponent(Component c,int row, int col,
                                        int nrow, int ncol)
 { gbc.gridx= col;    gbc.gridy=row;      // định vị ô sẽ thao tác
   gbc.gridwidth=ncol; gbc.gridheight=nrow; // định vùng tràn
  // ràng buộc c vào lưới thông qua gbc
   gb.setConstraints(c,gbc);
   add(c);

  }
```

Click for Demo

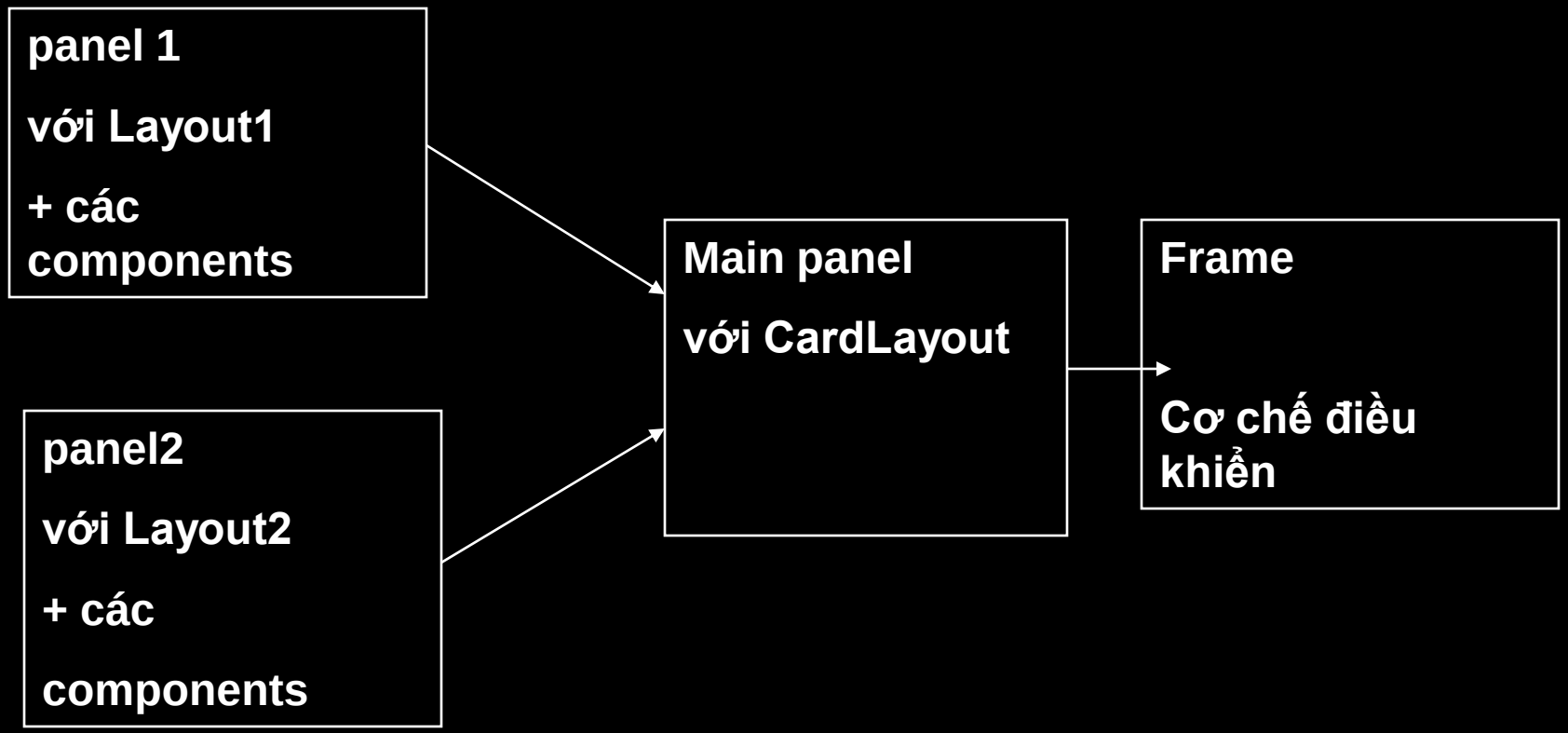
5.7.6- CardLayout

- Bố trí các component thành từng lớp như lưng các quân bài (card).
- Thường dùng Panel để chứa các component.
- Tại 1 thời điểm chỉ có 1 panel hiện hành (quân bài trên cùng).
- Có thể chuyển qua lại giữa các Panel.



CardLayout...

- Cách tạo GUI với card layout



CardLayout...

- CardLayout Constructors:

CardLayout()

CardLayout(int hgap, int vgap)

- Đưa 1 panel con vào panel cha

FatherPanel.add (sonPanel);

FatherPanel.add (“Alias”,sonPanel);

- Chọn 1 panel sẽ hiển thị

Card.first(FatherPanel);

Card.last(FatherPanel);

Card.next(FatherPanel);

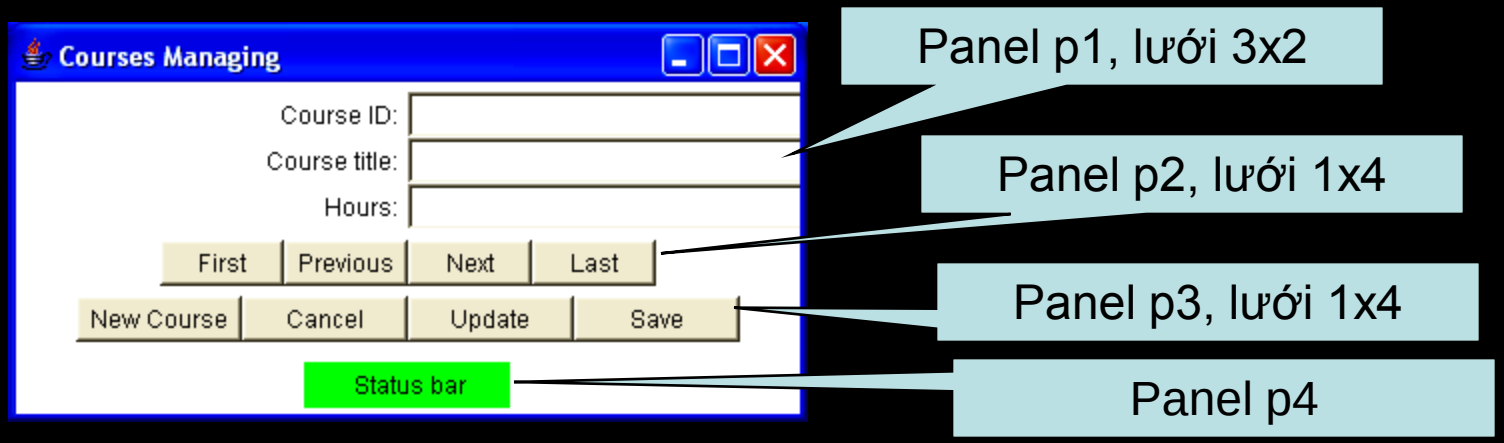
Card.previous(FatherPanel);

Card.show(FatherPanel, “Alias_of_sonPanel”);

Click for Demo

5.7.7- Layout phức tạp

- Có thể phải kết hợp nhiều Layout trên 1 GUI.
- Chia GUI thành nhiều Panel, mỗi panel 1 Layout riêng.



Click for Demo

5.8- Hướng dẫn tạo GUI cho ứng dụng

- GUI là khuynh hướng của các ứng dụng hiện nay. Nhờ GUI, giao diện với người sử dụng đẹp hơn và có được cơ hội kiểm tra dữ liệu nhập trước khi chuyển giao vào biến. Ta nói rằng "tách biệt cơ chế điều khiển của chương trình và dữ liệu).
- User ←→ GUI ←→ Biến

Hướng dẫn tạo GUI: Chọn components

- **Bài toán**
  ➔ Dữ liệu nhập/xuất ➔ Chọn component cho việc nhập/xuất
  ➔ Các tác vụ ➔ Mỗi tác vụ là 1 nút lệnh

Dữ liệu	Đối tượng
Chuỗi nhập 1 dòng	TextField
Chuỗi nhiều dòng	TextArea
Chọn trong nhiều chuỗi	Choice
Chọn Yes/No (nhiều)	Checkbox
Chọn Yes/No (1/n)	CheckboxGroup + Checkbox
Dữ liệu chỉ xuất 1 dòng	Label, TextField-cấm nhập
Dữ liệu chỉ xuất nhiều dòng	TextArea – cấm nhập
Chuỗi nhập + xuất	TextField/TextArea

Hướng dẫn tạo GUI: Chọn Layout

- **Các cơ sở chọn Layout**

  ➔ Thân thiện :Ưu tiên tạo Layout giống mẫu hồ sơ mà user thường dùng.

  ➔ Trật tự nhập liệu tự nhiên của bài toán.

  ➔ Nếu GUI phức tạp thì phân bổ các component vào nhiều panel, mỗi panel có một layout khác nhau.

- **Thói quen tốt khi đặt tên đối tượng**

  ➔ Dùng tiếp đầu ngữ

  txt cho TextField,
  lbl cho Label,
  chk cho Checkbox,
  btn cho Button, ...

Một thí dụ: -Phân tích bài toán

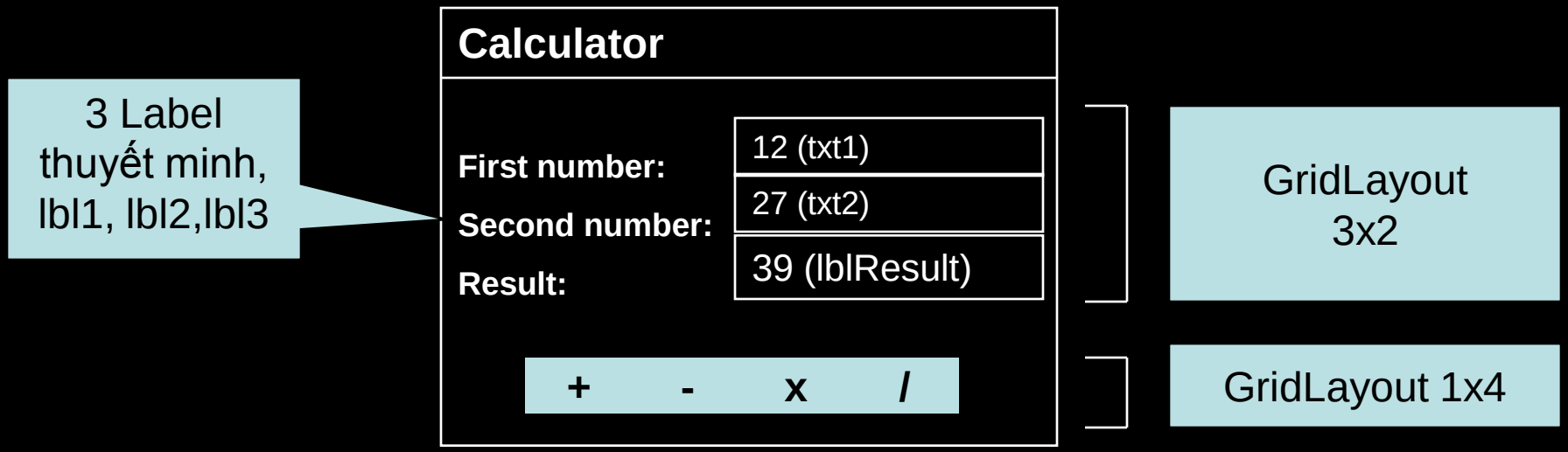
Bài toán:

Xây dựng ứng dụng cho phép làm các phép tính +, -, *, /

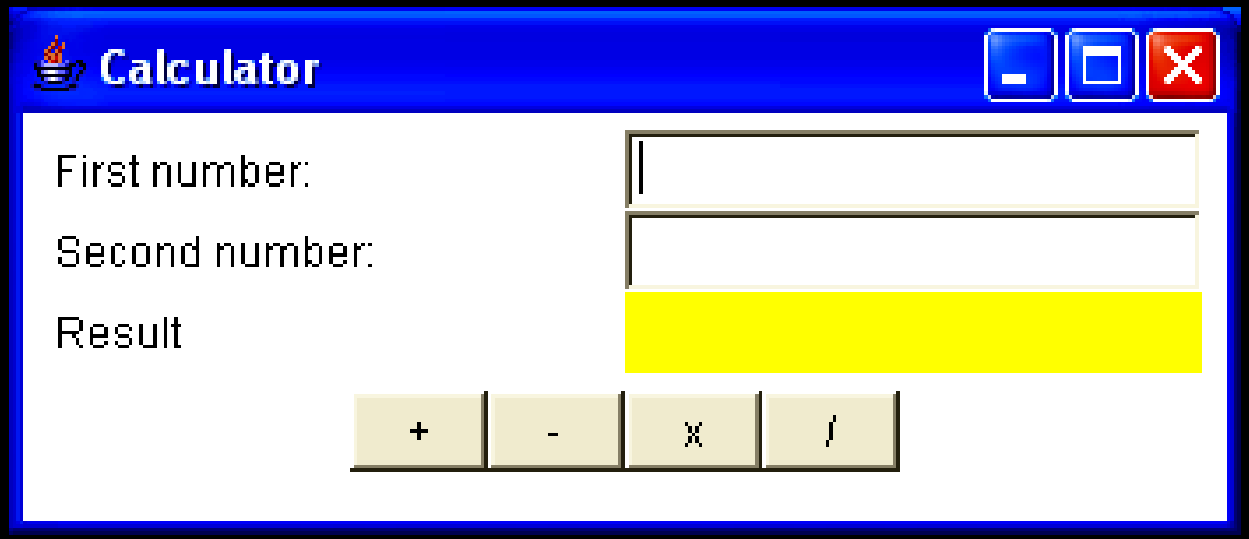
- Dữ liệu nhập: 2 số ➔ 2 TextFiled, tên txt1, txt2.
- Dữ liệu xuất: Kết qủa của phép tính: Label, tên lblResult
- 4 tác vụ: Cộng, trừ, nhân, chia ➔ 4 Button, tên btnAdd, btnSub, btnMul, btnDiv
- 3 lời thuyết minh: Label, tên lbl1, lbl2, lbl3

Một thí dụ: THiết kế GUI

- Hình thức GUI



Kết qủa



Click for Demo

5.9- Tóm tắt

- GUI- Graphic User Interface.
- GUI là khuynh hướng của các ứng dụng hiện nay.
- Nhờ GUI, người lập trình tách được điều khiển của chương trình và dữ liệu của chương trình.
- Gói AWT của Java bao gồm một tập các lớp cho phép người lập trình tạo ra GUI của ứng dụng.
- Một component đặt trên GUI có thể được user nhìn thấy (visible) hoâc không nhìn thấy (invisible) và có thể được thay đổi kích thước (resize)
- Frame và Panel là các container thường được dùng để tạo ra ứng dụng chạy độc lập (stand-alone application).
- Một panel thường được dùng để nhóm một số components lại với nhau.

Tóm tắt...

- Các lớp Layout manager giúp bố trí các component lên GUI.
- Trong gói awt, có 5 loại layout khác nhau: FlowLayout, BorderLayout, GridLayout, GridBagLayout,CardLayout.
- Thiết lập layout cho 1 container bằng hành vi setLayout(aLayout) của lớp container này.
- FlowLayout là bố cục mặc định của Panel, Applet. Các component được thêm vào tuần tự từ trên xuống dưới, từ trái sang phải.
- BorderLayout là bố cục mặc định của Frame. Cửa sổ sẽ được chia thành các phần: “East”, “West”, “South”, “North”, “Center”.
- GridLayout là cách bố trí các component vào container dạng 1 lưới rows hàng cols cột. Các component cũng được đưa vào các ô theo thứ tự: trên → dưới, trái → phải. Các component sẽ có cùng kích thước.

Tóm tắt...

- GridBagLayout là cách bố trí cho phép 1 component trải rộng trên nhiều ô kề nhau.Các component có thể có kích thước khác nhau.
- Để dùng GridBagLayout, cần có sự kết hợp của GridBagConstraint để "ràng buộc" 1 component vào lưới.
- CardLayout là cách bố trí các component trên 1 số Panel. Các panel hình thành 1 chồng (stack). Tại 1 thời điểm chỉ có 1 panel hiện hành.
- Một GUI có bố cục phức tạp có thể được phân tích thành nhiều thành phần, mỗi thành phần là 1 panel có layout riêng.
- BorderLayout là bố cục mặc định của Frame. Cửa sổ sẽ được chia thành các phần: "East", "West", "South", "North", "Center".
- GridLayout là cách bố trí các component vào container dạng 1 lưới rows hàng cols cột. Các component cũng được đưa vào các ô theo thứ tự: trên → dưới, trái → phải.

Tóm tắt...

- Trình tự thiết kế GUI:

(1) Đọc kỹ yêu cầu.
(2) Xác định dữ liệu nhập/xuất, chọn đối tượng phù hợp.
(3) Xác định các tác vụ, mỗi tác vụ là 1 nút lệnh.
(4) Xác định các nội dung thuyết minh, mỗi lời thuyết minh là 1 label.
(5) Vẽ các đối tượng lên giấy → Hình dáng GUI.
(6) Phân tích hình dáng GUI để xác định số nhóm panel.

Chương 6
Lập trình Menu với AWT

Mục tiêu

- Định nghĩa được Menu là gì?
- Biết cách tổ chức hệ thống menu của ứng dụng.
- Biết cấu trúc các lớp liên quan đến menu trong gói java.awt
- Biết viết event handler cho hệ thống menu.

Nội dung

6.1- Ôn tập.

6.2- Hệ thống Menu

6.3- Cấu trúc một hệ menu

6.4- Các tính chất của một mục chọn.

6.5- Gợi ý về thiết kế hệ thống menu cho ứng dụng.

6.6- Các lớp liên quan đến menu trong gói awt.

6.7- Phím nóng của MenuItem

6.8- Chuỗi lệnh kết hợp

6.9- Minh họa.

6.1- Ôn tập

- Gói java.awt chứa các lớp giúp tạo ra GUI.
- Gói java.awt.event chứa các lớp về Event Object, chứa các EventListener interfaces.
- Object Status- Trạng thái : Tập trị thuộc tính của đối tượng.
- Event: Tình huống có 1 đối tượng bị đổi trạng thái.
- Event object: Đối tượng được phát sinh động khi 1 object bị thay đổi trạng thái.
- Event source: Đối tượng tạo ra event object.
- Listener : Đối yượng chờ và xử lý sự kiện cho event source.

Ôn tập ...

- Cấu trúc quản lý event của một component:

  (1) Tạo Listener.

  (2) Viết code cho event handler.

  (3) Component.addXXXListeneer(Listener);
- Listener có thể là:

  (1) Chính Container chứa component.

  (2) Một lớp inner của lớp Container.

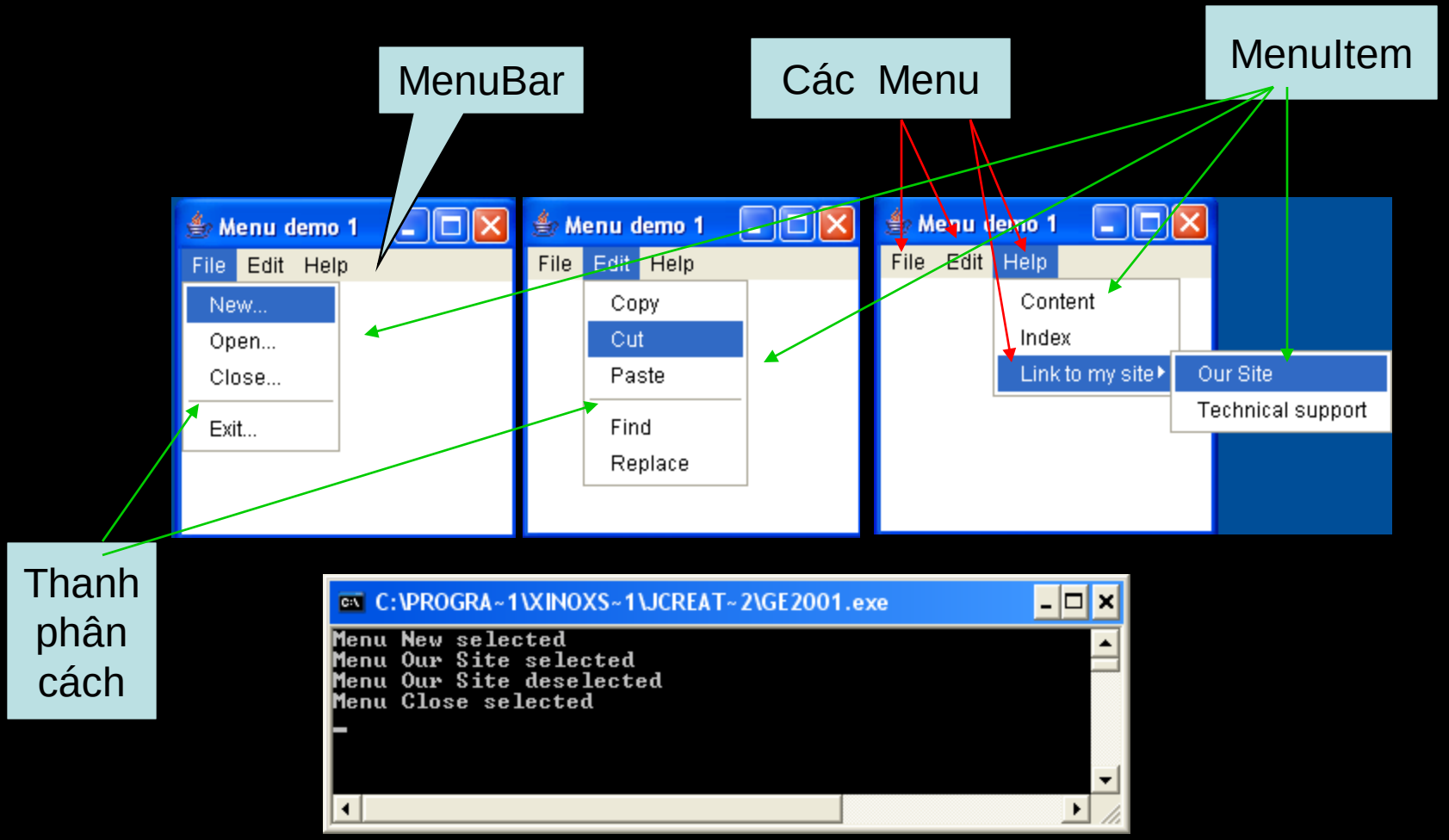
  (3) Một đối tượng xxxListener + Code event handler là thuộc tính của Container.

  (4) Một đối tượng thuộc lớp xxxAdapter + Code Event handler.

6.2- Hệ thống Menu là gì?

- Menu system- Hệ thống menu: Tập các mục chọn chức năng của ứng dụng được tổ chức phù hợp.
- Menu Item : Một mục chọn dạng chuỗi ký tự trong tập mục chọn.
- Hệ menu đơn giản: một Choice, một danh sách nút lệnh.
- Hệ menu phức tạp: Menu phân cấp.

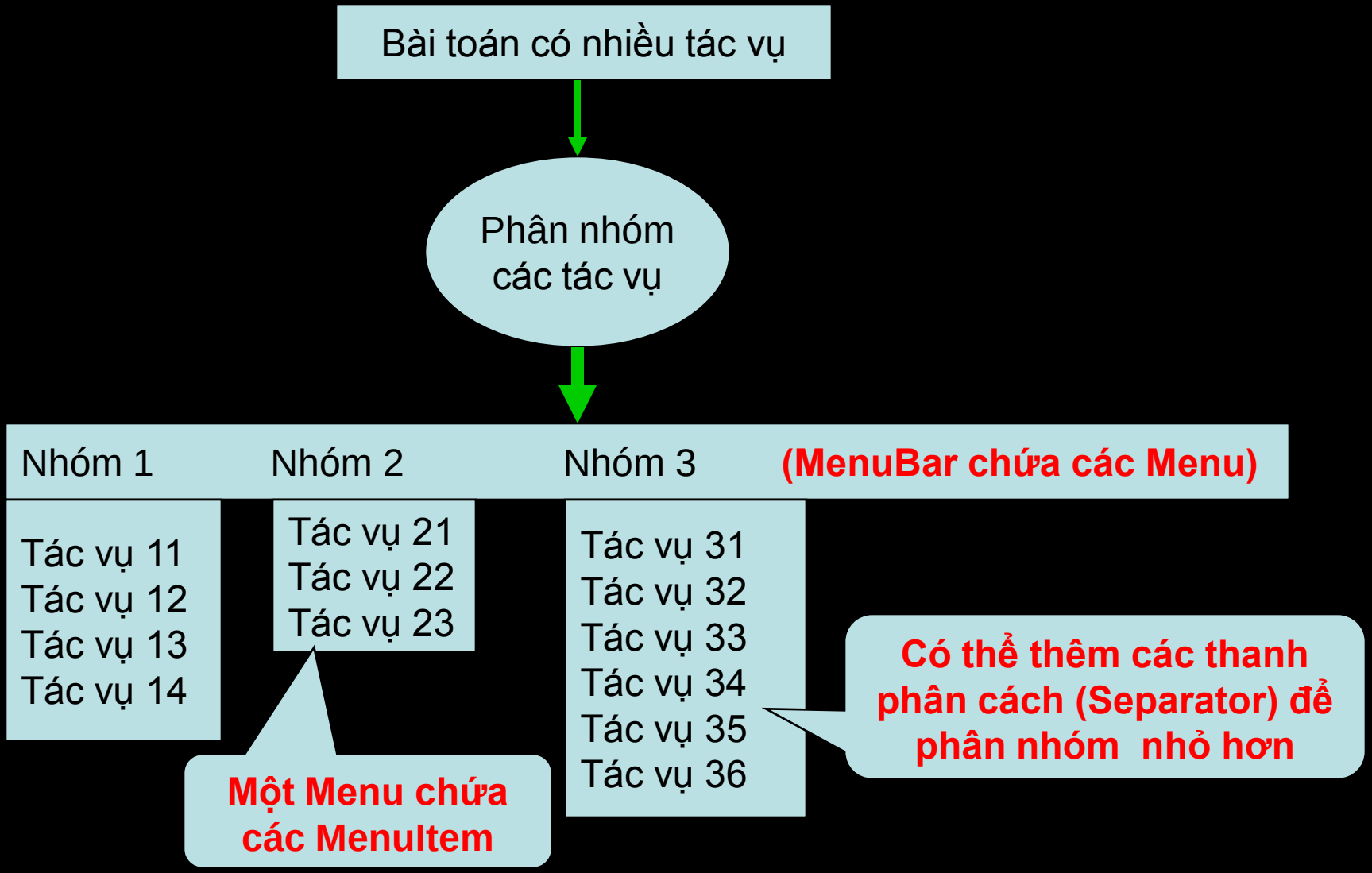
6.3-Cấu trúc một hệ menu



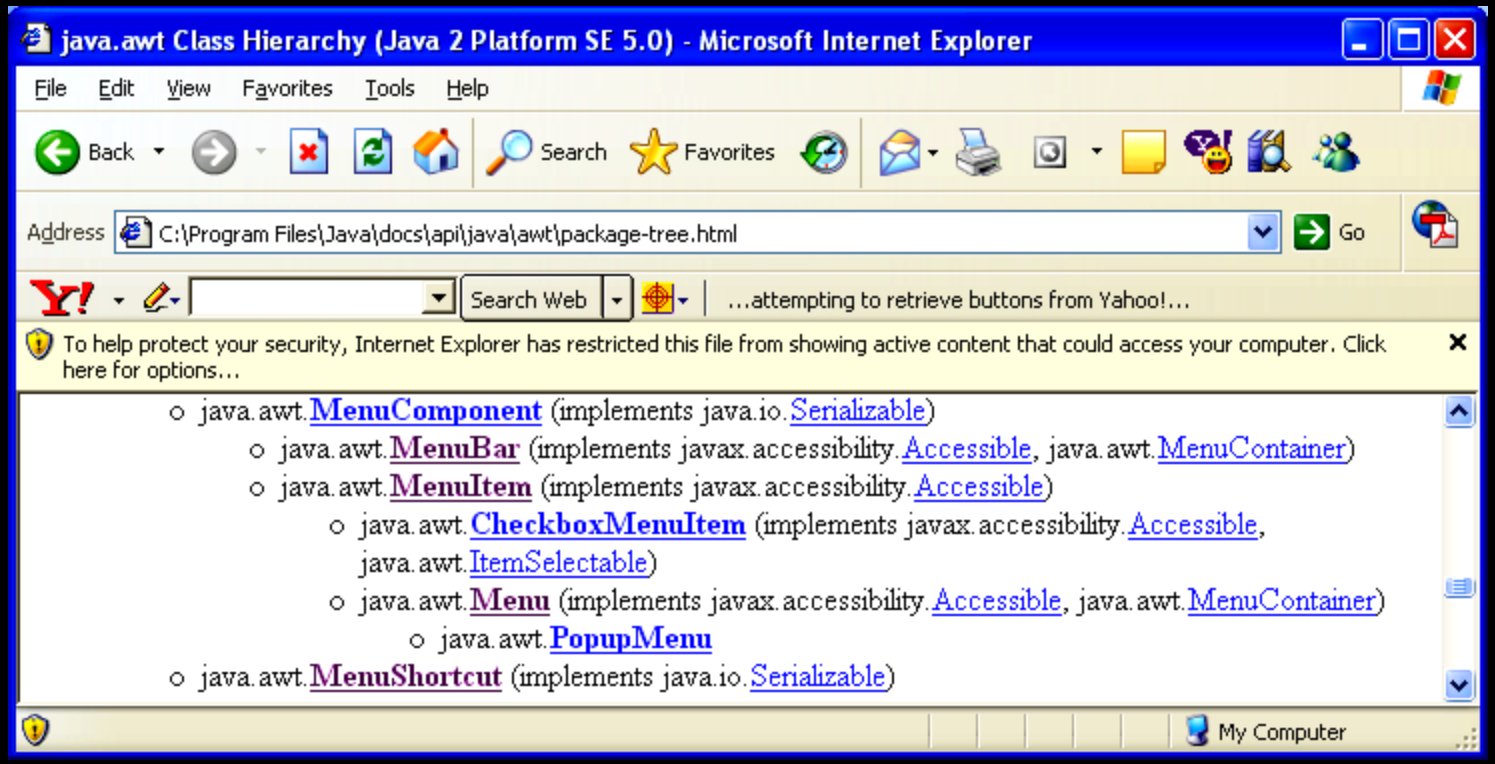
6.4- Tính chất của một menu Item

- Label-Chuỗi mô tả.
- Shortcut key- Phím nóng được kết hợp.
- Enable/ Disable- Cho user tác động?
- Action Command- Chuỗi tên lệnh được kết hợp.
- Ủy thác xử lý sự kiện : ActionListener

6.5- Gợi ý về thiết kế hệ thống menu

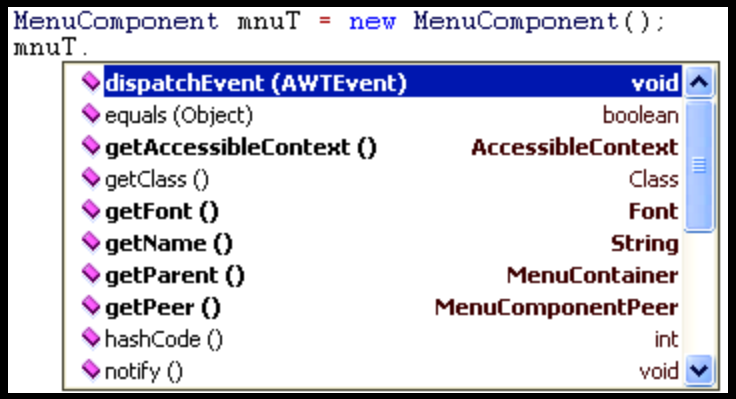


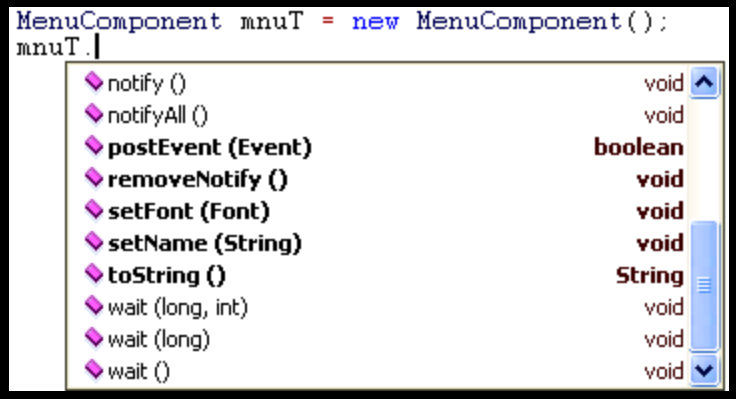
6.6- Các lớp liên quan đến menu trong gói AWT



6.6.1-Lớp MenuComponent

- Là lớp cha của các đối tượng menu
- Constructor: MenuComponent(void)

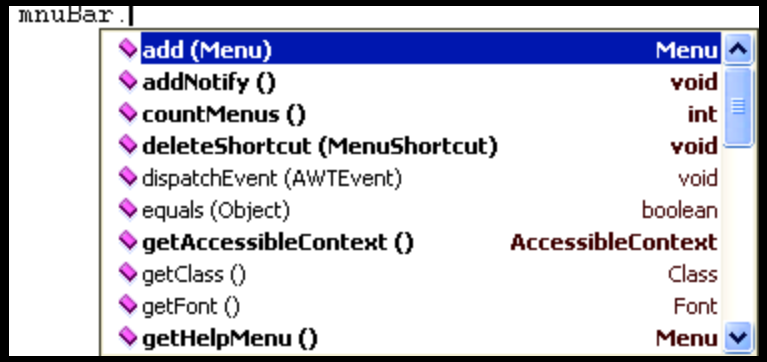


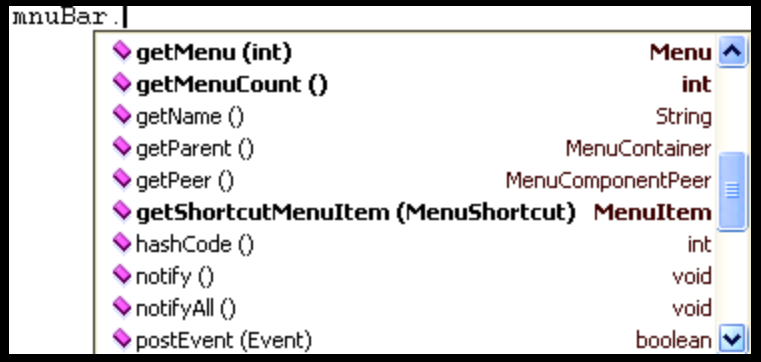


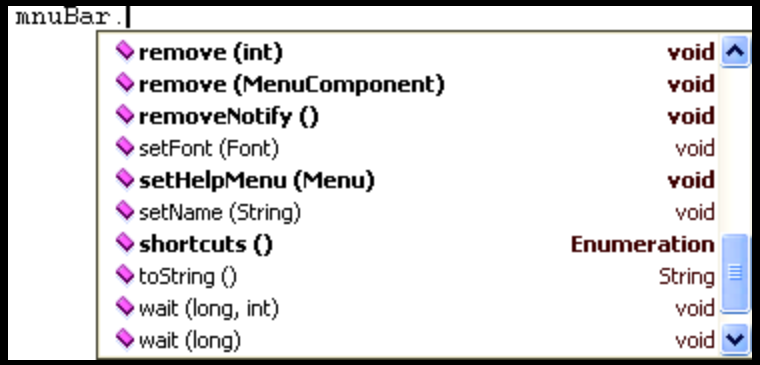
6.6.2- Lớp MenuBar- Thanh ngang

- Contructor:

MenuBar() – Tạo menu bar trống

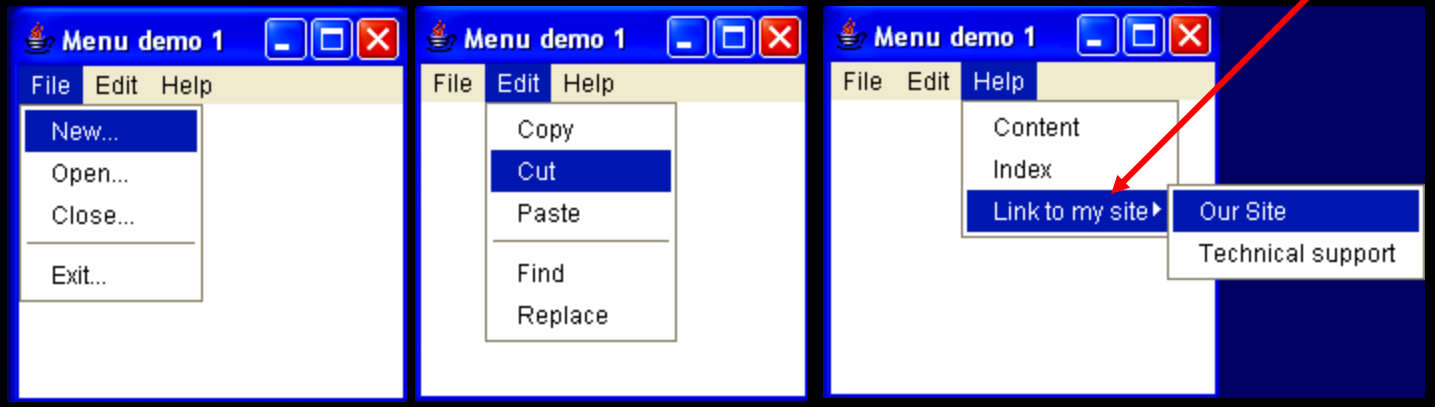






6.6.3- MenuItem – một mục chọn

- Một mục chọn có thể lại là một nhóm. Thí dụ:



MenuItem constructors

MenuItem()

Constructs a new MenuItem with an empty label and no keyboard shortcut.

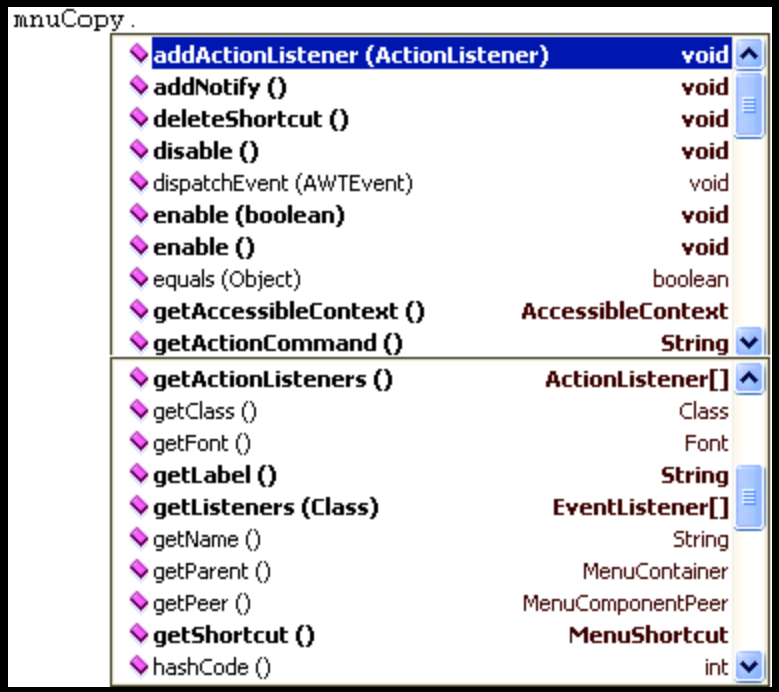
MenuItem(String label)

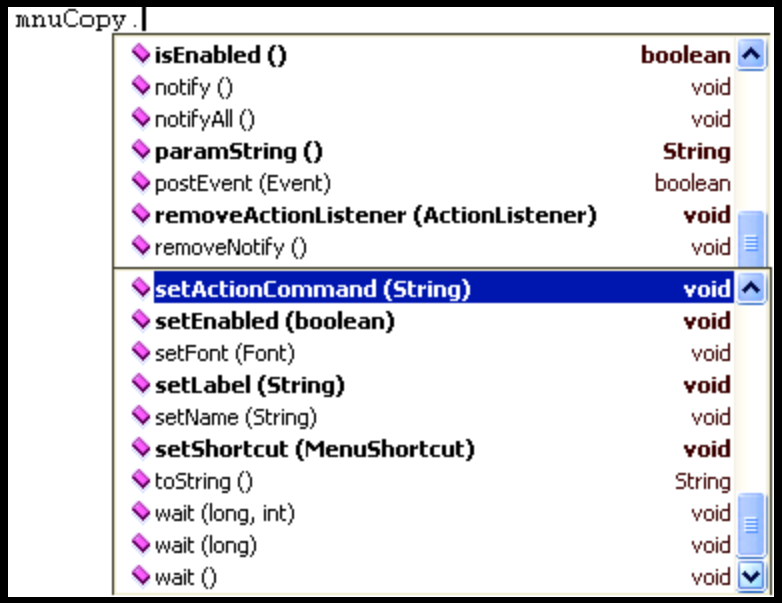
Constructs a new MenuItem with the specified label and no keyboard shortcut.

MenuItem(String label, MenuShortcut s)

Create a menu item with an associated keyboard shortcut

MenuItem methods





6.6.4- Lớp Menu – Nhóm MenuItem

- Lớp con của lớp MenuItem
- Constructors:

Menu()

Constructs a new menu with an empty label.

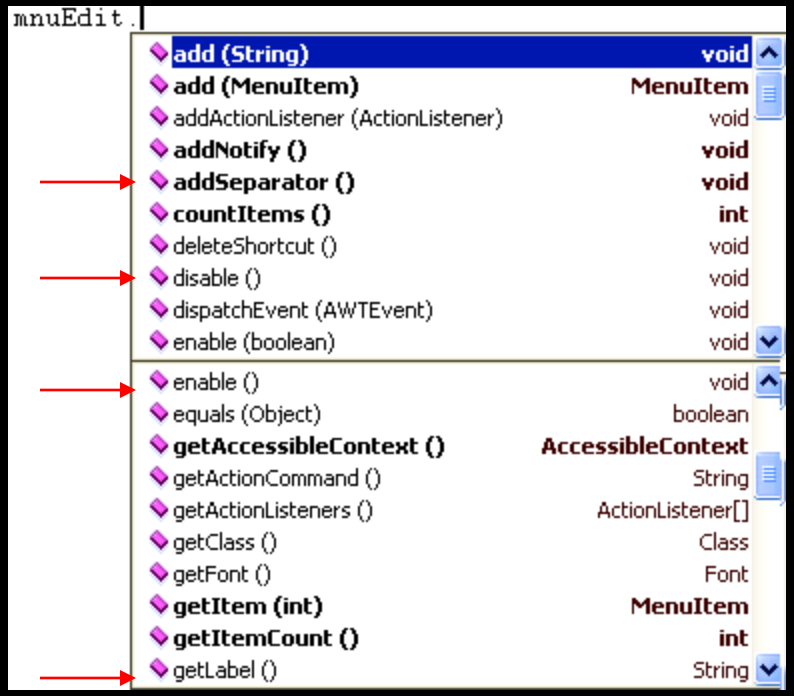
Menu(String label)

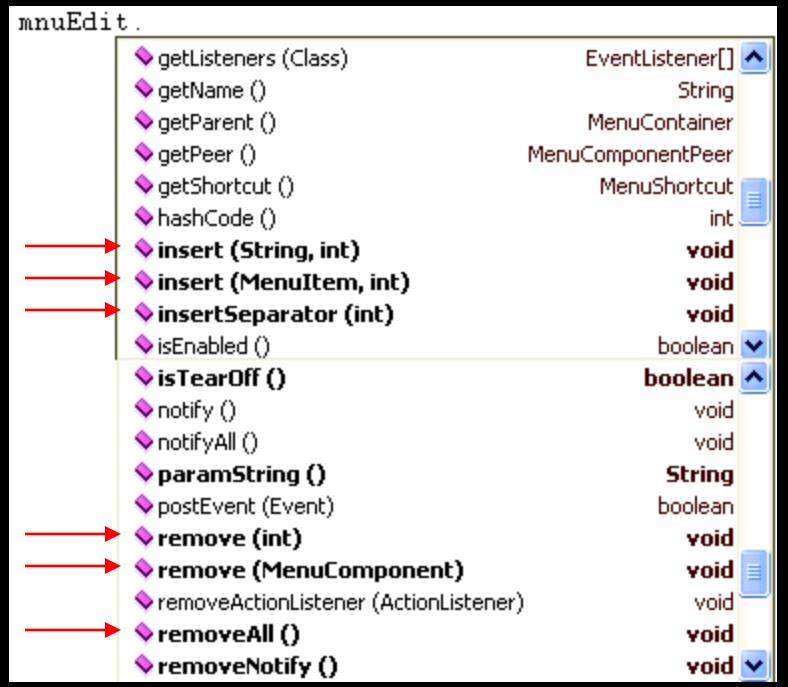
Constructs a new menu with the specified label.

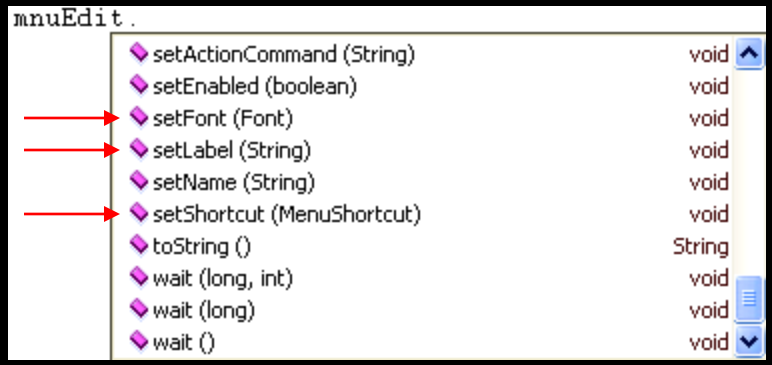
Menu(String label, boolean tearOff)

Constructs a new menu with the specified label, indicating whether the menu can be torn off.

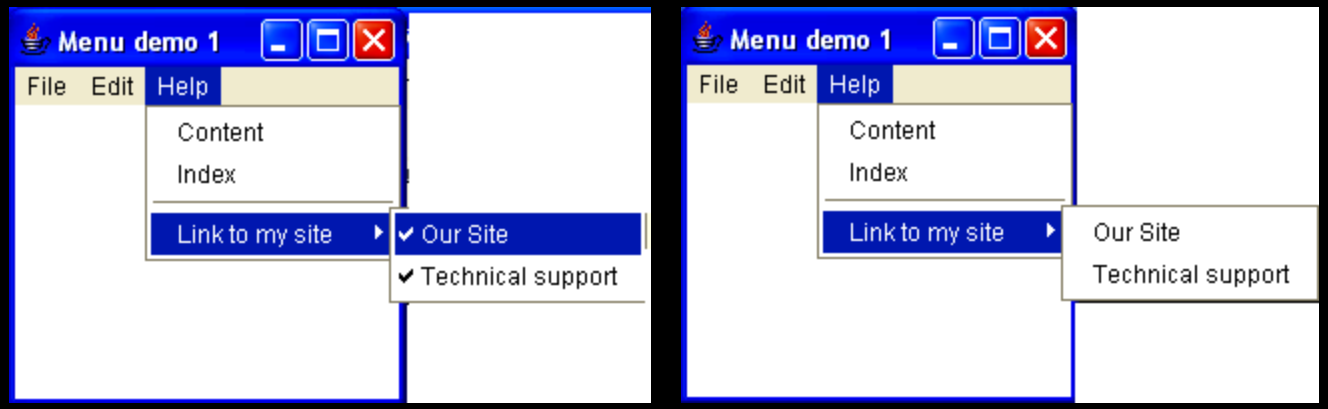
Menu class methods

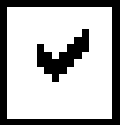






6.6.5- Lớp CheckboxMenuItem



- CheckboxMenuItem chỉ hiển thị  khi mục này được chọn.
- Constructors:

  CheckboxMenuItem()
  CheckboxMenuItem(String label)
  CheckboxMenuItem(String label, boolean state)

CheckboxMenuItem methods

- Ngoài các methods kế thừ từ lớp MenuItem, có thêm các methods:

void **addItemListener**(ItemListener l)

void **addNotify**()

<T extends EventListener> **getListeners**
(Class<T> listenerType)

Object **getSelectedObjects**()

boolean **getState**()

String **paramString**()

void **removeItemListener**(ItemListener)

void **setState**(boolean b)

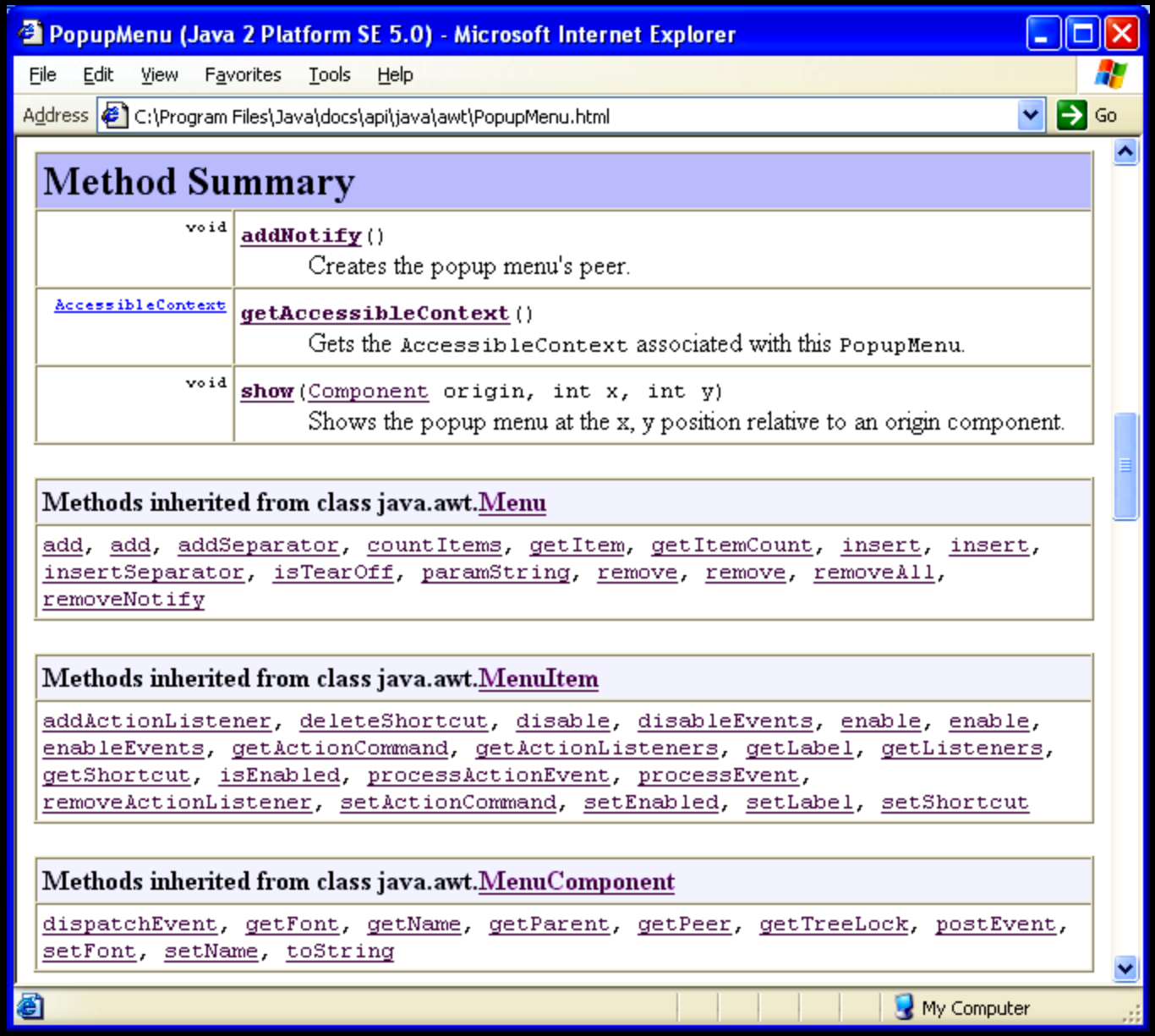
6.6.6- Lớp PopupMenu

- Là menu sẽ xuất khi ta kích chuột phải.
- Là lớp con của lớp java.awt.Menu
- Constructors:

PopupMenu() - Tạo đối tượng popup menu trống.

PopupMenu (String label) - Tạo đối tượng popup menu có nội dung

PopupMenu methods



Minh họa tạo PopupMenu...

```
class PopupMenuDemo extends Frame
{
   PopupMenu pMenu = new PopupMenu();
   MenuItem mnuCopy = new MenuItem("Copy");
   MenuItem mnuCut = new MenuItem("Cut");
   MenuItem mnuPaste = new MenuItem("Paste");
   PopupMenuDemo() // Constructor of a frame
   {  ...
      pMenu.add(mnuCopy); // setup popup menu
      pMenu.addSeparator();
      pMenu.add(mnuCut);
      pMenu.addSeparator();
      pMenu.add(mnuPaste);
      // Add popup menu to the frame
      this.add(pMenu);
```

Minh họa code hiển thị PopupMenu

```
//   In constructor of a frame
//   Add mouse Listener for showing popup menu
addMouseListener ( new MouseAdapter()
   { public void mouseReleased(MouseEvent e)
     { if (e.isPopupTrigger())  // check right clicked
         pMenu.show(e.getComponent(),
                    e.getX(),e.getY());
     }
   } );
```

The right-clicked position

Minh họa code quản lý biến cố cho các mục chọn trong popupmenu

```
ActionListener actListener = new ActionListener()
      { public void actionPerformed(ActionEvent e)
        { Object src=e.getSource();
          if (src==mnuCopy)
                LblStatus.setText("menu Copy is selected");
          if (src==mnuCut)
                LblStatus.setText("menu Cut is selected");
          if (src==mnuPaste)
                LblStatus.setText("menu Paste is selected");
        }
      };
mnuCopy.addActionListener(actListener);
mnuCut.addActionListener(actListener);
mnuPaste.addActionListener(actListener);
```

6.7 Phím nóng của MenuItem

- Shortcut Key: Tổ hợp Ctrl+ Phím sẽ tác động vào 1 mục chọn tương tự như kích chuột vào 1 mục menu.
- Lớp java.awt.MenuShortcut giúp mô tả các phím nóng.
- Lớp java.awt.event.KeyEvent định nghĩa sẵn các phím
- Ấn định phím nóng cho MenuItem:

```
MenuShortcut  CtrlN = new MenuShortcut(KeyEvent.VK_N));
mnuNew.setShortcut(CtrlN); // Ctrl + N
mnuOpen.setShortcut(new MenuShortcut(KeyEvent.VK_O));
```

6.8- Chuỗi lệnh kết hợp

- Action Command string: Một chuỗi được gán cho 1 nút lệnh hay 1 mục menu.

```
mnuNew.setActionCommand("New Command");
```

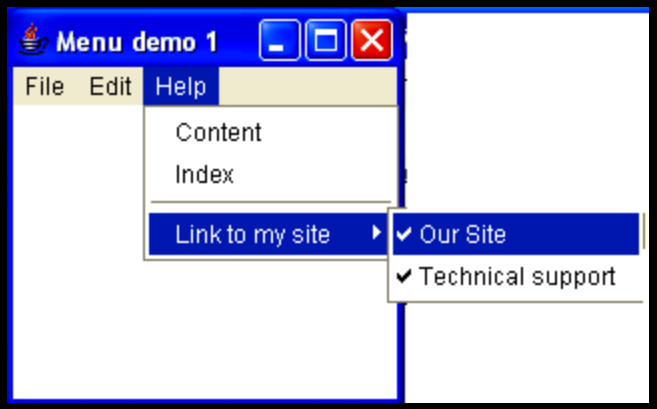
- Các command string của các đối tượng phải khác nhau
- Có thể quản lý sự kiện bằng command string.

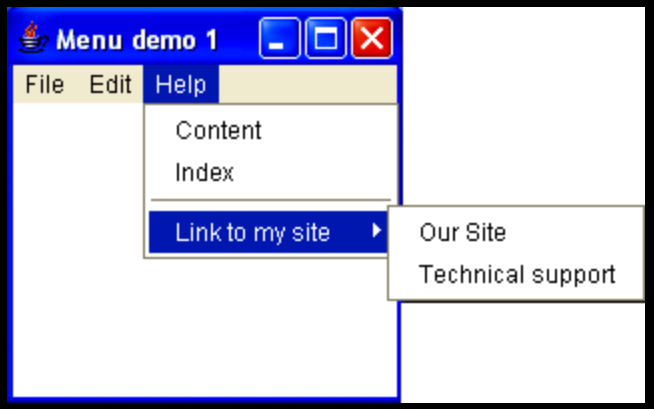
```
public void actionPerformed(ActionEvent e)
  {  String CommandStr= e.getActionCommand();
     if (CommandStr.equals("New Command"))
       { <code>
       }
    .....
  }
```

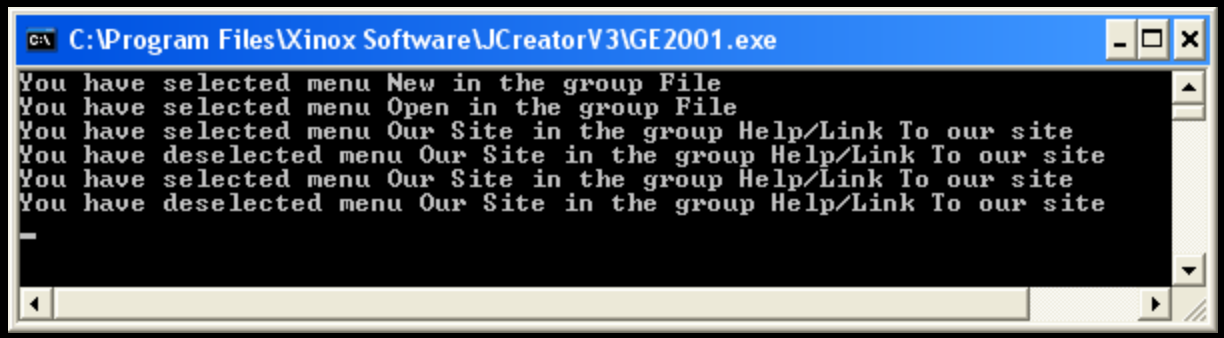
6.9-Minh họa

- Minh họa 1- Tạo menu bar, thiết lập Shortcut key, quản lý các mục chọn bằng e.getSource()
- Minh họa 2- Tạo menu bar, quản lý cac1 mục chọn bằng e.getActionCommand()
- Minh họa 3- Tạo và quản lý PopupMenu

Minh họa 1





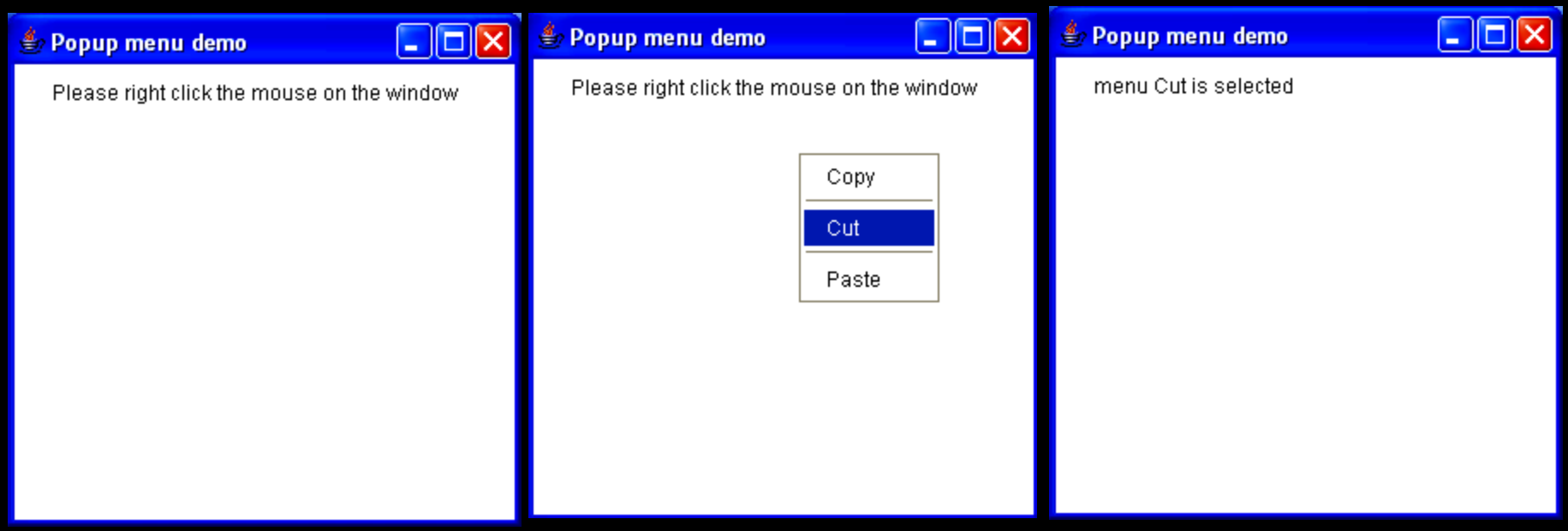


Demo

Minh họa 2-
Vẫn thí dụ trước nhưng
Quản lý sự kiện với Action Command

Demo

Minh họa 3- PopupMenu



Demo

Chương 6
Mô hình sự kiện với AWT

Mục tiêu

- Hiểu sự cần thiết phải kiểm soát các biến cố.
- Biết cấu trúc các biến cố trong gói AWT.
- Nắm bắt cách cơ bản để kiểm soát biến cố.
- Hiểu về lớp vô danh (anonymous class)

Nội dung

6.1- Ôn tập.
6.2- Mô hình ứng dụng hướng sự kiện.
6.3- Cấu trúc các sự kiện trong AWT.
6.4- Các Event Adapter.
6.5- Tóm tắt về cách quản lý sự kiện
6.6- Trò chơi Puzzle.
6.7- Code quản lý biến cố cơ bản.
6.8- Lớp vô danh (Anonymous class)
6.9- Tóm tắt.
6.10- Câu hỏi.
6.11- Bài tập.

6.1- Ôn tập

- AWT cung cấp một tập các lớp để người lập trình tạo GUI cho ứng dụng.
- AWT cung cấp 5 mô hình bố trí các phần tử lên GUI gồm:

FlowLayout :	Bố trí dạng tuần tự,
BorderLayout:	Bố trí ra biên,
GridLayout:	Bố trí dạng lưới 1 phần tử chiếm 1 ô
GridBagLayout:	Bố trí dạng lưới , có thể 1 phần tử chiếm nhiều ô.
CardLayout:	Bố trí dạng phân lớp, tại 1 thời điểm có 1 lớp tích cực.
Bố trí phức tạp:	Kết hợp nhiều panel

6.2- Mô hình ứng dụng hướng sự kiện

- Event-Oriented Application Model: Chương trình có GUI, user tương tác với GUI qua chuột, bàn phím,…, chương trình xử lý, trạng thái mới lại xuất ra cho user xem → thân thiện.
- Event : một tín hiệu mà ứng dụng nhận biết có sự thay đổi trạng thái của 1 đối tượng.
- 3 nguồn phát xuất event:

(1) User(gõ phím, kích chuột vào 1 phần tử,…),

(2) Do hệ thống (do định thời 1 tác vụ)

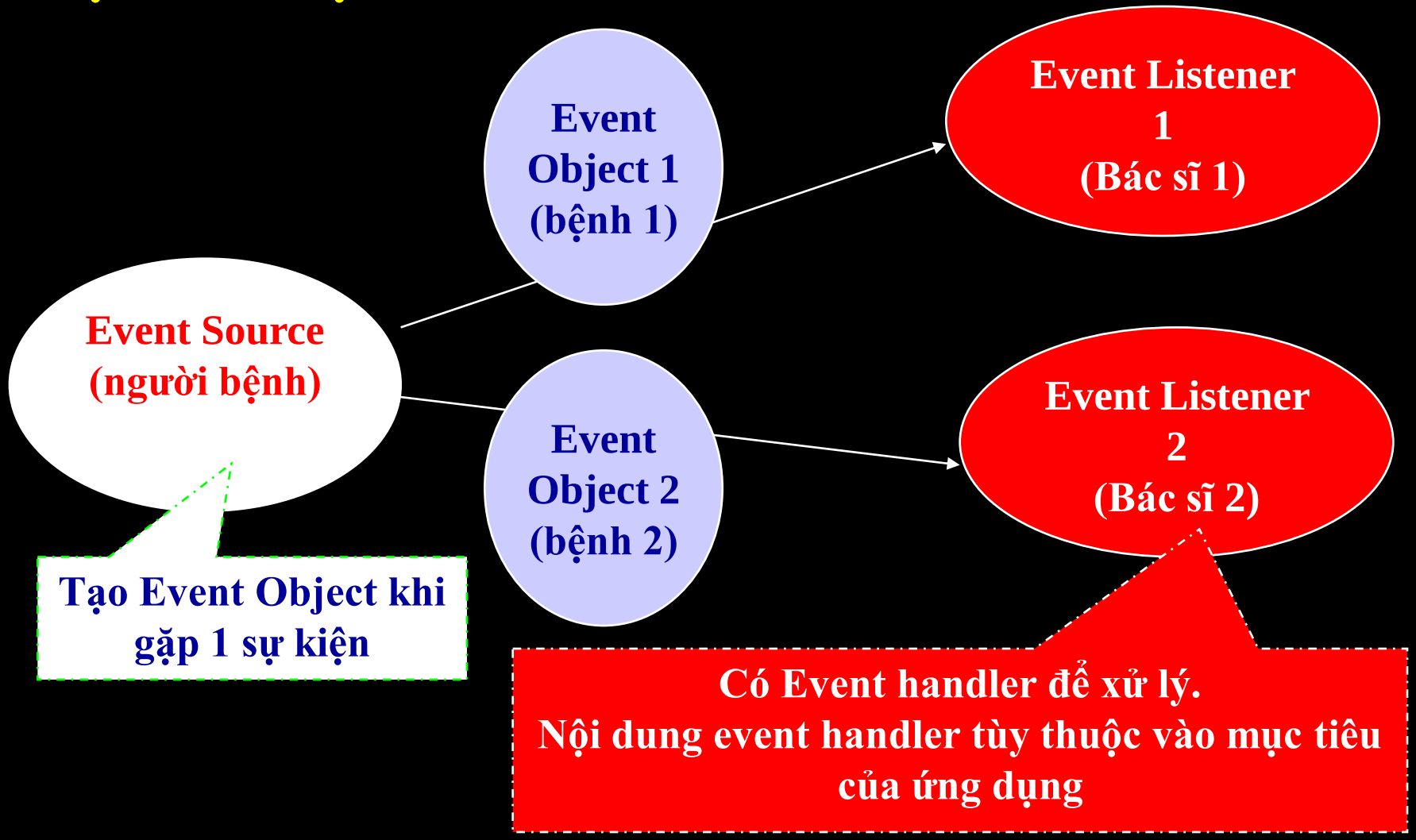
(3) Do 1 event khác (các event kích hoạt nhau)

- Hiện nay, đa số các ngôn ngữ đều cung cấp mô hình này, VC++ cung cấp MFC (Microsoft Foundation Classes), Java cung cấp JFC (Java Foundation Classes).

6.2.1- Một minh họa về ủy thác xử lý sự kiện

- Ta là một **đối tượng**.
- Ta bị bệnh (**sự kiện**)
- Bệnh có trạng thái (**đối tượng sự kiện**).
- Một bác sĩ là một **đối tượng khác**.
- Ta nhờ bác sĩ chữa bệnh (**ủy thác** xử lý sự kiện).
- Bác sĩ chờ (**listen**) ta đưa ra triệu chứng bệnh (**đối tượng event**) rồi dựa vào trạng thái của bệnh (đối tượng event) để **xử lý phù hợp**.
- Có thể ta mắc nhiều bệnh ➔ Có thể phải ủy thác chữa bệnh cho nhiều bác sĩ, mỗi bác sĩ một loại bệnh.
- Một bác sĩ chữa 1 bệnh như thế nào tùy thuộc vào quyết định của bác sĩ đó dựa trên tình hình thực tế của dược phẩm.

Một minh họa...


Event Listener 1 (Bác sĩ 1)
Event Object 1 (bệnh 1)
Event Source (người bệnh)
Event Object 2 (bệnh 2)
Event Listener 2 (Bác sĩ 2)
Tạo Event Object khi gặp 1 sự kiện
Có Event handler để xử lý.
Nội dung event handler tùy thuộc vào mục tiêu của ứng dụng

6.2.2- Một số định nghĩa

- **Event** : Là tình huống ứng dụng nhận biết có 1 đối tượng đã thay đổi trạng thái.
- **Event handler**: Là đoạn code đễn đạt phản ứng của ứng dụng khi gặp 1 event.
- **Event source**: Đối tượng kích hoạt (trigger, fire) 1 event (thí dụ: nút lệnh bị user kích chuột).
- **Listener** : Đối tượng nhận sự ủy nhiệm xử lý sự kiện cho đối tượng khác.
- **Focus**: Trạng thái 1 đối tượng đang bị user nhắm đến để tương tác.

6.2.3- Đối tượng không thể tự quản lý sự kiện ?

- **Mỗi nút lệnh (lớp Button) trong ứng dụng cụ thể sẽ phản ánh trạng thái của ứng dụng khác nhau.**
- **Khi thiết kế lớp Button, người thiết kế không thể biết trước khi user kích vào nút này thì chương trình sẽ phản ứng thế nào.**
- **Event handler phụ thuộc vào ứng dụng cụ thể và tại mỗi ứng dụng cũng có thể có nhiều event handler cho mỗi sự kiện trên 1 đối tượng.**
- **Java định nghĩa sẵn các Listener Interface cho các tình huống khác nhau (*mỗi Event object có listener interface xử lý tương ứng*).**
- **Một lớp có khả năng listener sẽ phải cụ thể hóa – viết code- một số hành vi xử lý một event phù hợp (nhận 1 event làm tham số).**

6.2.4-Java Event Delegation Model

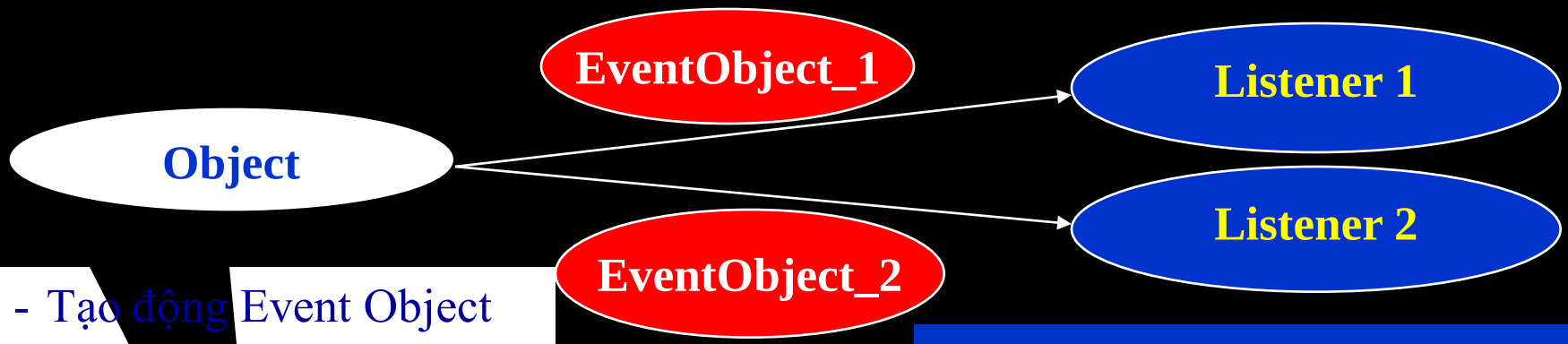
- Java cung cấp các công cụ để quản lý sự kiện:

(1)Tập các lớp mô tả các đối tượng Event

(2)Tập các Interface tương ứng cho 1 lớp event,

(3)Tập các lớp Adapter tương ứng- Các lớp đã khai báo sẵn các hành vi trong Interface tương ứng để tiết kiệm công sức cho người lập trình vì: Có những lúc chỉ cần 1 event handler mà người lập trính phải viết code (dù là code trống) cho các event handler khác đã được khai báo trong interface .

Java Delegation Model



- Tạo động Event Object khi gặp 1 sự kiện.
- Có method addXXXListener (ListenerObject)
- Có method removeXXXListener (ListenerObject).
- Một object có thể ủy thác xử lý event cho nhiều Listener.
- Chỉ có những object có đăng ký Listener mới có thể là event source.

- Implements EventListener phù hợp.
- Mỗi method là một event handler phụ thuộc ứng dụng.
- Một Listener có thể được ủy thác xử lý event cho nhiều object.

1 object sẽ nhận biết Listener của mình khi event xẩy ra. Chính điều này làm cho code event handler được gọi đúng.

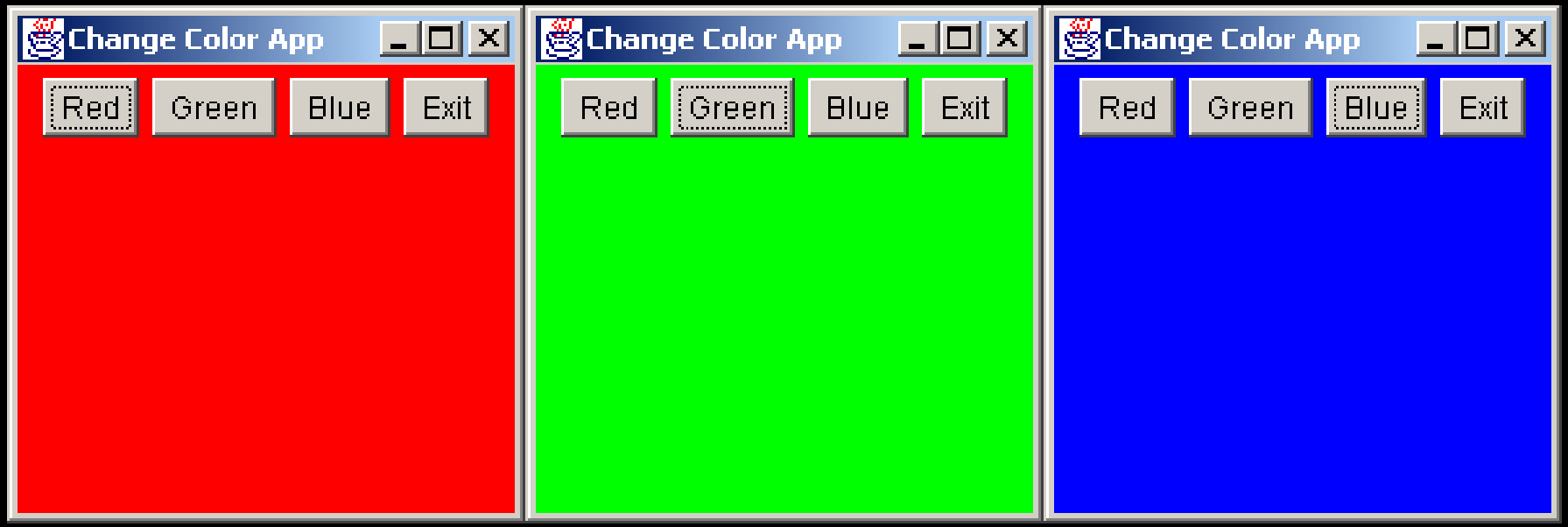
Cơ chế xử lý sự kiện

(1) Event source phát sinh EventObject khi gặp biến cố.

(2) Event source truyền EventObject tới tất cả các Listener của event source.

(3) Các Listener dựa trên thông tin trong EventObject để xác định đoạn code phù hợp và phản ứng của ứng dụng đối với sự kiện được tiến hành.

Một thí dụ:



ChangBKColor.java

6.3-Cấu trúc các đối tượng Event-Dạng phân cấp

- Object
  - EventObject
    - AWTEvent

	Interface xử lý
ActionEvent	(ActionListener interface)
AdjustmentEvent	(AdjustmentListener interface)
ComponentEvent	(ComponentListener interface)
ContainerEvent	(ContainerListener interface)
FocusEvent	(FocusListener interface)
InputEvent	
KeyEvent	(KeyListener interface)
MouseEvent	(MouseListener interface)
PaintEvent	
WindowEvent	(WindowListener interface)
ItemEvent	(ItemListener interface)
TextEvent	(TextListener interface)

6.3.1- ActionEvent class

- **Một ActionEvent object được sinh ra khi: 1 nút lệnh bị kích, một mục chọn trong danh sách bị kích đôi, 1 mục menu bị kích.**

- **Các hằng kiểm tra có 1 phím bị nhấn khi kích chuột hay không: ALT_MASK (phím Alt), CTRL_MASK (phím Ctrl), META_MASK (phím meta, ký tự mô tả về 1 ký tự khác -ký tự escape), SHIFT_MASK (phím Shift).**

ActionEvent class...

Demo- Thí dụ 2
ActionEventDemo.java

- Event source: Button, List Item, Menu
- Constructor:

ActionEvent(Object source, int id, String command)

ActionEvent(Object source, int id, String command, int modifiers)

- Common Methods

public String getActionCommand() - Lấy tên tác vụ kết hợp với Source (xem thí dụ 1 về getActionCommand)

public int getModifiers() - lấy bit mask của phím điều khiển đi kèm (Shift,Alt,Ctrl)

public Object getSource() - Lấy nguồn gây event

ActionListener interface Có 1 event handler

void actionPerformed (ActionEvent e)

6.3.2- AdjustmentEvent class

- Được sinh ra khi 1 thanh cuộn bị thao tác.
- Các hằng int:BLOCK_DECREMENT, BLOCK_INCREMENT: Độ giảm/tăng theo khối khi user kích chuột vào vùng giữa con trượt và 1 biên của thanh cuộn, UNIT_DECREMENT, UNIT_INCREMENT: Đơn vị giảm/tăng khi user kích chuột vào mũi tên ở 2 đầu thanh cuộn. TRACK: Giá trị mô tả thanh cuộn khi bị user kéo.
- Các phương thức thường dùng:

Adjustable getAdjustable()	Lấy đối tượng Source.
int getAdjustableType()	Lấy trị hằng mô tả ở trên.
int getValue()	Lấy trị hiện hành của thanh cuộn

AdjustmentListener interface Có **1** method là event handler của sự kiện: **void adjustmentValueChange(AdjustmentEvent e)**

Demo- Thí dụ 3
AdjustmentEventDemo.java

6.3.3- ComponentEvent class

- Được sinh ra khi 1 componet bị ẩn đi, được hiển thị, bị di chuyển, bị thay đổi kích thước.
- Các hằng mô tả trạng thái gồm: COMPONENT_HIDDEN, COMPONENT_MOVED, COMPONENT_RESIZED, COMPONENT_SHOWN

Hành vi: **Component getComponent()** : Lấy đối tượng phát sinh sự kiện.

ComponentListener interface :Các method được gọi khi Source gặp biến cố tương ứng:

void componentHidden(ComponentEvent e)

void componentMoved(ComponentEvent e)

void componentResized(ComponentEvent e)

void componentShown(ComponentEvent e)

6.3.4- ContainerEvent class

- Được sinh ra khi 1 component được thêm/xóa khỏi 1 container.
- Các hằng mô tả sự kiện: COMPONENT_ADDED, COMPONENT_REMOVED.
- Các hành vi hay dùng

Component getChild()

Lấy component được added/removed

Container getContainer() : lấy source container của sự kiện

ContainerListener interface 2 event handler:

void componentAdded(ContainerEvent e)

void componentRemoved(ContainerEvent e)

6.3.5- FocusEvent class

- Được sinh ra khi 1component có/mất focus.
- Các hằng: FOCUS_GAINED, FOCUS_LOST.
- Hành vi hay dùng:**boolean isTemporary()**: Trả về true nếu việc mất focus là tạm thời. Việc mất focus là tạm thời khi focus ở tại 1 phần tử trên GUI như thanh cuộn, pop-up menu.

FocusListener: interface, 2 event handler:

void focusGained (FocusEvent e)
void focusLost (FocusEvent e)

6.3.6- ItemEvent class

- **Được sinh ra khi 1 mục được chọn/bỏ chọn trong 1 danh sách Listbox, Combobox, checkbox menuitem.**
- **Các hằng:SELECTED, DESELECTED, ITEM_STATE_CHANGED**
- **Các methods hay dùng:**

Object getItem() : Lấy đối tượng bị thao tác

ItemSelectable getItemSelectable() : Lấy source của sự kiện

int getStateChange() : Lấy loại sự kiện (SELECTED/DESELECTED)

ItemListener interface :1 event handler:

void itemStateChanged (ItemEvent e)

6.3.7-InputEvent

- Là lớp cha của 2 lớp con: KeyEvent và MouseEvent.
- Các hằng khai áo trong lớp này mô tả các bit mặt nạ truy xuất phím đi kèm sự kiện hoặc nút chuột nào bị nhấn: ALT_MASK, CTRL__MASK, META_MASK, SHIFT_MASK, BUTTON1_MASK, BUTTON2_MASK, BUTTON3_MASK.
- Meta character : Ký tự mô tả về 1 ký tự khác – Thí dụ: Ký tự backslash (\) chỉ thị rằng ký tự sau nó là thành phần của chuỗi escape trong C, Java

Các methods hay dùng:

int getModifier() : Lấy bit mặt nạ.

boolean isAltDown() : kiểm tra có phím bấm đi kèm

boolean isMetaDown()

boolean isShiftDown()

boolean isControlDown()

KeyEvent class

- Được sinh ra khi user thao tác với bàn phím .
- Các hằng kiển intKEY_PRESSED, KEY_RELEASED, KEY_TYPED. Nếu phím chữ, phím số được gõ, cả 3 loại sự kiện được sinh ra (pressed, released, typed). Nếu phím đặc biệt được thao tác (phím Home, End, PageUp, PageDown- modifier key), chỉ có 2 sự kiện được sinh ra: pressed, released.
- Hai methods thường dùng để truy cập phím bị thao tác:

char getKeyChar() **int getKeyCode()**

KeyListener interface : 3 event handler:

void keyPressed(KeyEvent e)
void keyReleased(KeyEvent e)
void keyTyped(KeyEvent e)

MouseEvent class

- Được sinh ra khi user thao tác chuột với 1 component.
- Các hằng int:MOUSE_CLICKED, MOUSE_DRAGGED, MOUSE_ENTERED, MOUSE_EXITED, MOUSE_MOVED, MOUSE_PRESSED, MOUSE_RELEASED.
- Các methods hay dùng:

Point getPoint() : Lấy vị trí của mouse lúc sự kiện xẩy ra.

int getX() ,int getY() -Lấy tọa độ x,y của vị trí chuột

MouseListener interface :5 event handler:

void mouseClicked(MouseEvent e)

void mouseEntered(MouseEvent e)

void mouseExited(MouseEvent e)

void mousePressed(MouseEvent e)

void mouseReleased(MouseEvent e)

6.3.8- TextEvent class

- Được sinh ra khi các ký tự trong 1 TextField hay 1 textArea bị đổi.
- Hằng int: TEXT_VALUE_CHANGED

TextListener interface 1 event handler

void textValueChanged(TextEvent e)

6.3.9- WindowEvent class

- **Được sinh ra khi 1 cửa sổ: activated, deactivated, iconified,deiconified, opened, closed, closing**
- **Các hằng int: WINDOW_ACTIVATED, WINDOW_DEACTIVATED, WINDOW_OPENED, WINDOW_CLOSED, WINDOW_CLOSING, WINDOW_ICONIFIED, WINDOW_DEICONIFIED.**
- **Method thông dụng:**

Window getWindow() : lấy source window

WindowListener interface 7 event handler cho 7 sự kiện

void windowActivated(WindowEvent e)
void windowDeactivated(WindowEvent e)
void windowOpened(WindowEvent e)
void windowClosed(WindowEvent e)
void windowClosing(WindowEvent e)
void windowIconified(WindowEvent e)
void windowDeiconified(WindowEvent e)

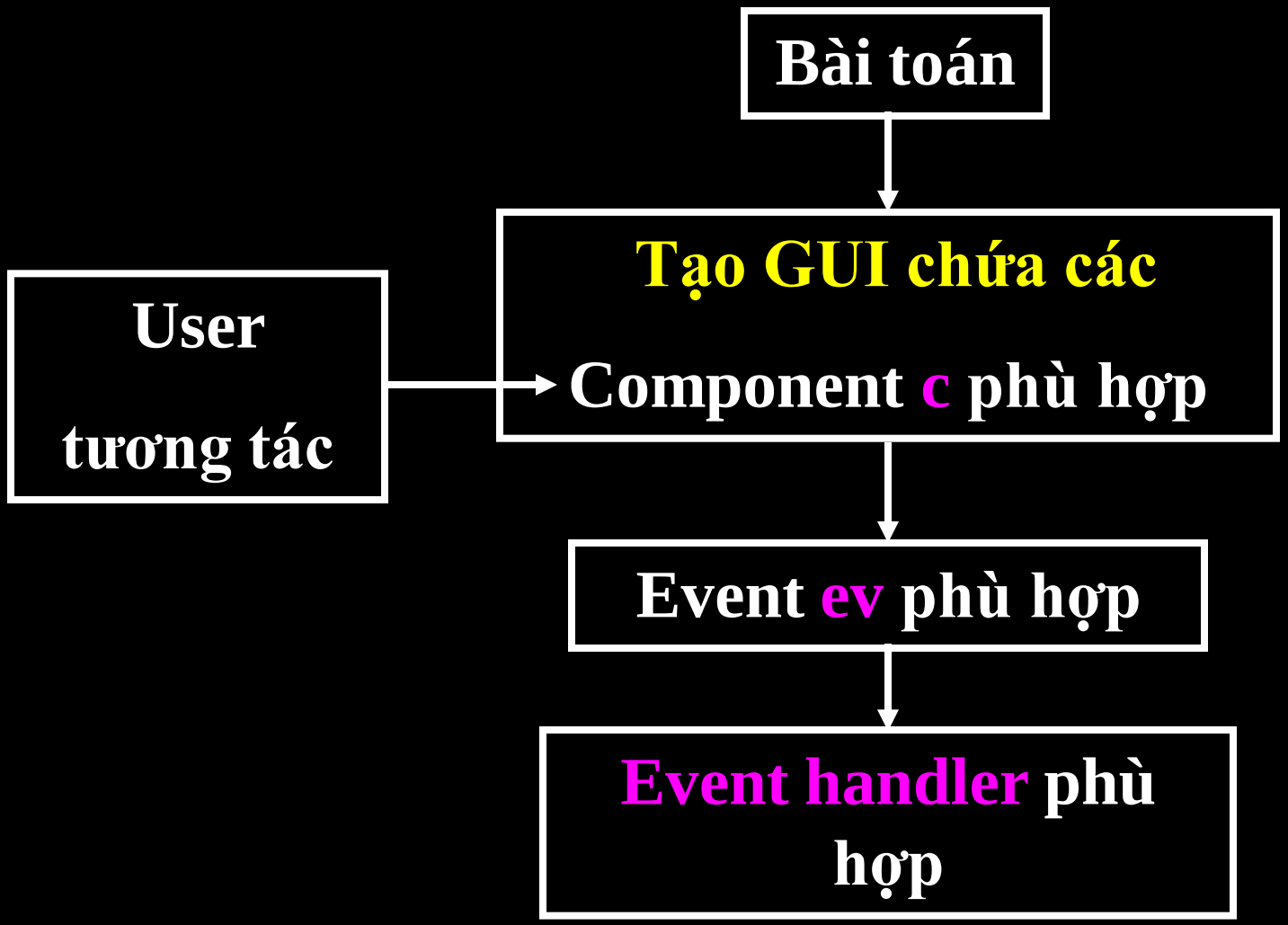
6.4- Các Adapter class quản lý sự kiện.

- Có tình huống chúng ta chỉ cần 1 vài event handler trong khi nếu 1 lớp implements 1 interface thì phải hiện thực toàn bộ các methods đã khai báo trong interface đó.
- Java cung cấp sẵn 1 số lớp có tên <type>Adapter cho tình huống này. Các lớp này đã implement (nội dung trống)các methods của các Listener

➔ User có thể chỉ cần khai báo một lớp con của lớp Adapter này và hiện thực một vài hành vi cần cho ứng dụng mà không cần phải hiện thực toàn bộ các methods của interface tương ứng.

interface	Adapter class
ComponentListener	→ ComponentAdapter
ContainerListener	→ ContainerAdapter
FocusListener	→ FocusAdapter
KeyListener	→ KeyAdapter
MouseListener	→ MouseAdapter
MouseMotionListener	→ MouseMotionAdapter
WindowListener	→ WindowAdapter

6.5- Tóm tắt về cách quản lý sự kiện



Cách 1: Add Listener trực tiếp vào GUI

-Xem thí dụ 2, thí dụ 3

(1) Add 1 đối tượng Listener/Adapter phù hợp, hiện thực event handler;

(2) Liên kết đối tượng này với componrent

Cách 2: Xem thí dụ 1

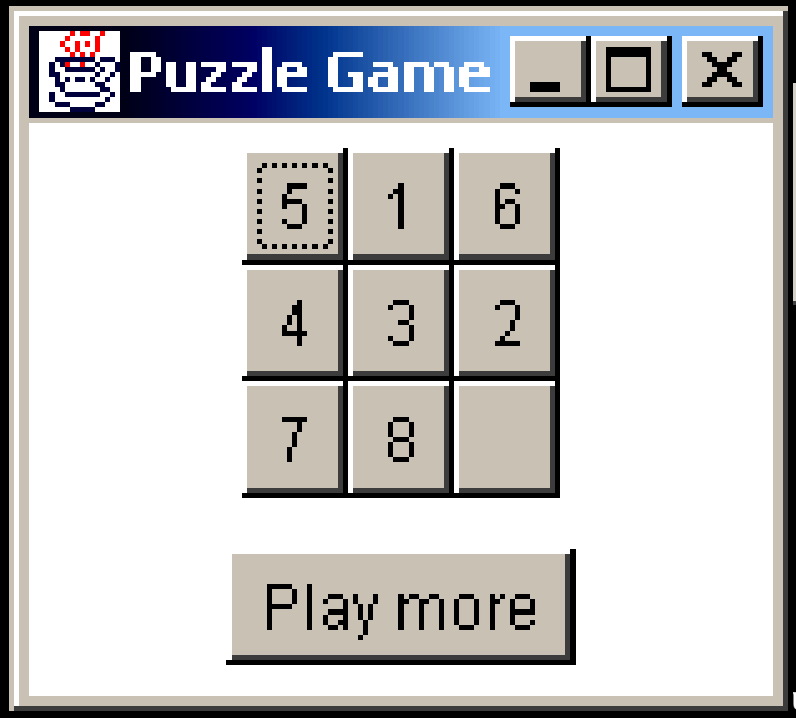
(1) Khai báo 1 class Listener implements 1 Listener interface (hoặc khai báo 1 lớp con của lớp Adapter phù hợp), hiện thức các hành vi phù hợp.

(2) Khai báo đối tượng myListener, add vào GUI

(3) Liên kết Component với đối tượng Listener.

6.6- Trò chơi Puzzle

Trò chơi Puzzle gồm 9 nút. Khi thắng xuất thông báo “You Win”. Khi thắng: trật tự các nút là “123456780”



Demo-PuzzleGame.java

6.7- Vài code quản lý biến cố cơ bản

- Biến cố focus
- Biến cố bàn phím.
- Biến cố chuột

6.7.1- Biến cố focus

Tình huống:

Có hộp textbox txtCode (mã nhân viên) , buộc nhập đòi hỏi phải nhập 1 từ và tối thiểu 4 ký tự.

→Khi txtCode mất focus mà dữ liệu không hợp lệ, buộc con trỏ quay về hộp TextField này.

```
// Thêm vào constructor của frame
FocusAdapter focusListener = new FocusAdapter()
   { public void focusLost(FocusEvent event)
     { txtCodeLostFocusHandler (event); // event handler
     }
   };
txtCode.addFocusListener(focusListener);
```

```
boolean CodeValid()
  {  // cắt khoảng trống đầu đuôi
     txtCode.setText(txtCode.getText().trim());
    int len= txtCode.getText().length();        // lấy độ dài chuỗi
    int Pos= txtCode.getText().indexOf(" ");// tìm vị trí khoảng trống
    if (len<4 || Pos>=0 ) return false;
    return true;
  }
 void txtCodeLostFocusHandler (FocusEvent event)
  { Component Src= event.getComponent();
    if (!CodeValid() && !event.isTemporary())
      { txtCode.setText("");
        txtCode.requestFocus();
       Message("Code is a word with at least 4 characters!");
      }
  }
```

6.7.2- Biến cố bàn phím

Tình huống:

Có hộp txtBirthYear, nhập năm sinh của nhân viên: Đòi hỏi phải nhập số.

```
KeyAdapter NumkeyListener= new KeyAdapter()
  { public void  keyPressed(KeyEvent k_ev)
    { int c = k_ev.getKeyCode();
      if (c < KeyEvent.VK_0 || c > KeyEvent.VK_9)   // <'0' || >'9'
      // Back space to delete the input character
        k_ev.setKeyCode(KeyEvent.VK_BACK_SPACE);
    }
  };
  txtBirthYear.addKeyListener(NumkeyListener);
```

6.7.3- Biến cố chuột

Một trích đoạn về biến cố chuột:

```
public void mouseClicked( MouseEvent e)
{ int x = e.getX();  int y= e.getY();
  int ClickCount= e.getClickCount();
  if ( e.isShiftDown() &&
        e.isControlDown() && ClickCount>3)
    txt1.setText("Shift down, Ctrl down, >3");
}
```

6.8- Đối tượng vô danh (Anonymous object)

- Là đối tượng không gán tên gọi, được new trực tiếp.
- Rất thường dùng trong việc quản lý biến cố. Tạo động 1 Adapter cùng với event handler bên trong constructor của bài toán.

```
class MyPanel extends Panel
{ MyPanel()  // constructor
  { Button  btn= new Button ("Yellow");
    add(btn);
    btn.addMouseListener ( new MouseAdpter()
    {  public void mouseClicked(MouseEvent e)
       {   setBackground(Color.yellow); repaint();
       }
    } );
    ..... // các phát biểu kế tiếp của constructor
```

6.9- Tóm tắt.

- **Event : một tín hiệu mà ứng dụng nhận biết có sự thay đi trạng thái.**
- **Event object : Đối tượng Java mô tả cho một sự kiện.**
- **3 nguồn phát xuất event:**
  (1) User(gõ phím, kích chuột vào 1 phần tử,…),
  (2) Hệ thống (do định thời 1 tác vụ)
  (3) Do 1 event khác (các event kích hoạt nhau)
- **Event handler: Là đoạn code biểu diễn phản ứng của chương trình khi gặp 1 event.**
- **Event source: Đối tượng kích hoạt (trigger, fire) 1 event (thí dụ: nút lệnh bị user kích chuột).**
- Khi trạng thái nội của event source bị thay đổi, event source tạo ra 1 event.
- Event Listener - Đối tượng chờ sự kiện : Là 1 object được Event source nhận biết khi event xẩy ra.

Tóm tắt

- **Focus: Trạng thái 1 đối tượng đang bị user nhắm đến để tương tác.**
- Mô hình ủy thác sự kiện là mô hình trong đó 1 đối tượng khi gặp 1 tình huống phải phát sinh 1 sự kiện, sự kiện này được truyền cho một đối tượng khác xử lý hộ.
- Java xây dựng các class riêng có tên <Type>Event, các lớp <type>Adapter và các interface có tên <type>Listener giúp quản lý các event.
- Quản lý 1 event bằng cách liên kết 1 phần tử trên GUI với 1 đối tượng thuộc lớp <type>Event hoặc lớp <type>Adapter hoặc lớp hiện thực 1 interface tương ứng. Ta nói rằng phần tử này có đăng ký chờ (listener) sự kiện.
- Có hai dạng hiện thực event handler:

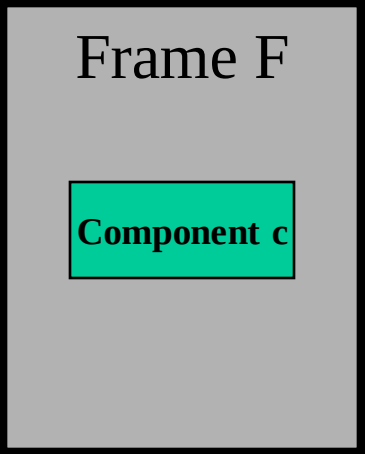
(a) Phân tán: Mỗi event listener là riêng cho mỗi component,

(b) Tập trung: một event listener là chung của một số component.

Tóm tắt về cách quản lý biến cố cho 1 component- Cách 1

- Container chứa component làm luôn vai trò Event Listener cho component mà nó chứa.
- Nếu có nhiều component cùng có chung 1 loại event, event handler của container sẽ xử lý tập trung.
- Tham khảo các bài tập có hướng dẫn.

Bài toán



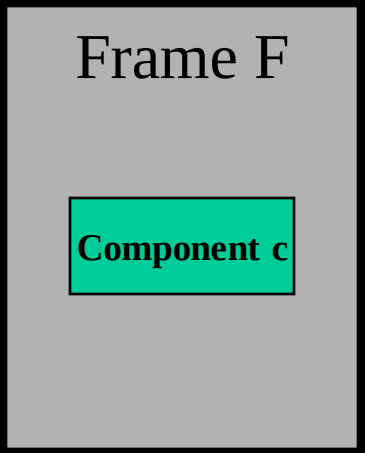
Dự kiến các biến cố cho c

```
class GUI extends Frame implements EventListener
{  Component c = new Componenet(...);
   GUI()
   {   ...
       c.addxxxListener (this);
   }
   public void EventHandler( xxEvent e)
   {  <Code> }
}
```

Tóm tắt về cách quản lý biến cố cho 1 component- Cách 2

- Tạo 1 lớp riêng đóng vai trò Event Listener (hiện thực event handler) cho component mà container đang chứa. Vì event handler phải truy xuất đến dữ liệu của container nên được khai báo là **inner** class.
- Nếu có nhiều component cùng có chung 1 loại event, event handler của container sẽ xử lý tập trung. Tham khảo các thí dụ trong bài.

Bài toán



Dự kiến các biến cố cho c

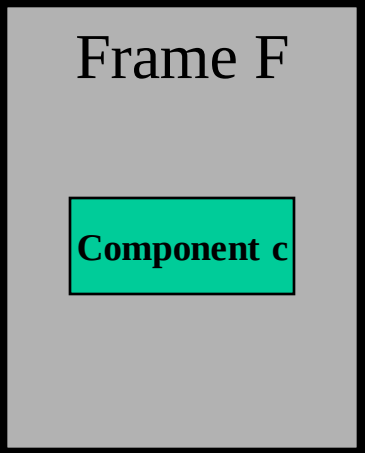
```
class GUI extends Frame
{  Component c = new Component (...)
   MyListener  Lst = new MyListener();
   GUI()
   { ...
     c. addXXXListener (Lst) ;
   }
  class MyListener extends XXXAdapter
  {    public void EventHandler( xxxEvent e)
       {   <code>   }
  }// 
}
```

Tóm tắt về cách quản lý biến cố cho 1 component- Cách 3

- Tạo động các Adapter/Event Listener, hiện thực event handler cho các listener động này – xem lại thí dụ 2 trong bài.

```
class GUI extends Frame
{  Component c = new Component (...)
   MyListener  Lst = new MyListener();
   GUI() // constructor
   { ...
     xxxListener Lst = new xxxAdapter() // xxxListener()
     { public void Handler ( xxxEvent e)
        {  <code event handler> } ;
      }
     c. addXXXListener (Lst);
   }
}
```

Bài toán



Dự kiến các biến cố cho c

6.10-Câu hỏi

1- Biến cố (sự kiện) là gì?

2- Thế nào là mô hình Event Delegation Model.

3- Biến cố nào được tạo ra khi phím Shift và phím Control được nhấn.

4- Mô tả mối quan hệ giữa Event source, Event Listener và Listener Interface.

6.11- Bài tập

Quay lại bài toán quản lý nhân viên của chương trước (thí dụ đầu tiên). Viết các biến cố cần thiết đáp ứng:

Tên: Không cho phép chuỗi trống và ít nhất có 2 từ.

Năm sinh: Chỉ cho phép nhập các ký số từ 0 đến 9 và trị phải thuộc khoảng :

1960 <= năm sinh <= 1986

Địa chỉ: Không cho phép trống

Xin cảm ơn.

Chương 7- Lập trình đồ họa

Mục tiêu

Đến cuối chương bạn có thể

- Hiểu biết về lớp Font, lớp FontMetrics
- Hiểu biết về lớp Color.
- Biết cách vẽ hình ảnh trên GUI. với đối tượng thuộc lớp Graphics, Graphics2D

Nội dung

7.1- Ôn tập
7.2- Các vấn đề cơ bản về vẽ.
7.3- Điều khiển mầu sắc.
7.4- Điều khiển Font.
7.5- Đồ họa với lớp Graphics.
7.6- Paint mode.
7.7- Đồ họa với lớp Graphics2D.
7.8- File ảnh.
7.9- Tóm tắt.
7.10- Chương trình vẽ bằng chuột.

7.1- Ôn tập

- Event : Tín hiệu nhận biết có sự thay đổi trạng thái.
- Nguồn phát sinh event: Hệ thống, user, event khác.
- Có 2 mức sự kiện : low-level events (không do user kích hoạt), semantic-level events (event do user kích hoạt)
- Có 4 loại semantic-level events: trong gói java.awt.event: ActionEvent, AdjustmentEvent, ItemEvent, TextEvent.

Ôn tập...

- **Event Source: Đối tượng kích hoạt 1 sự kiện.**
- **Event là đối tượng mô tả có sự thay đổi trạng thái của đối tượng nguồn.**
- **Event handler: Các method xử lý tình huống dựa trên loại Event object.**
- **Khi 1 event xẩy ra, event source sẽ gọi các method tương ứng được định nghĩa trong đối tượng listener kết hợp với event source.**
- **Quản lý 1 event trong ứng dụng là tạo 1 đối tượng listener phù hợp với event source, viết code event handler, kết hợp event source với listener.**

7.2- Các vấn đề cơ bản về vẽ

- Điều khiển mầu sắc.
- Chọn Font
- Thao tác vẽ : Vẽ chuỗi, vẽ hình, tô mầu.
- Chế độ đồ họa Graphic mode.
- Xuất file ảnh.

7.3- Điều khiển mầu sắc

- Tại 1 thời điểm. có 1 mầu nền (background color, mặc định là white) hiện hành và 1 mầu vẽ hiện hành mặc định là black.
- Thao tác với mầu nền và mầu vẽ của Frame:

  setBackground(aColor); getBackground();

  setForeground(aColor); getForeground();
- Ấn định mầu vẽ g.setColor (aColor); // g:graphic object
- Chỉ định mầu bằng các hằng mầu sắc: Khai báo sẵn trong lớp Color → Color.black , … Các hằng khác : white, gray , lightGray, darkGray, red, pink, orange, yellow , green, magenta, cyan , blue

Điều khiển mầu sắc (tt)

- 1 mầu tự chọn được ấn định bằng bộ 3 (Red,Green,Blue) thông qua constructor của lớp Color :

Color class

public Color(float RedVal, float GreenVal, float BlueVal)

public Color(int RedVal, int GreenVal, int BlueVal) // 0..255

Truy xuất trị 1 mầu hoặc thành phần của 1 mầu

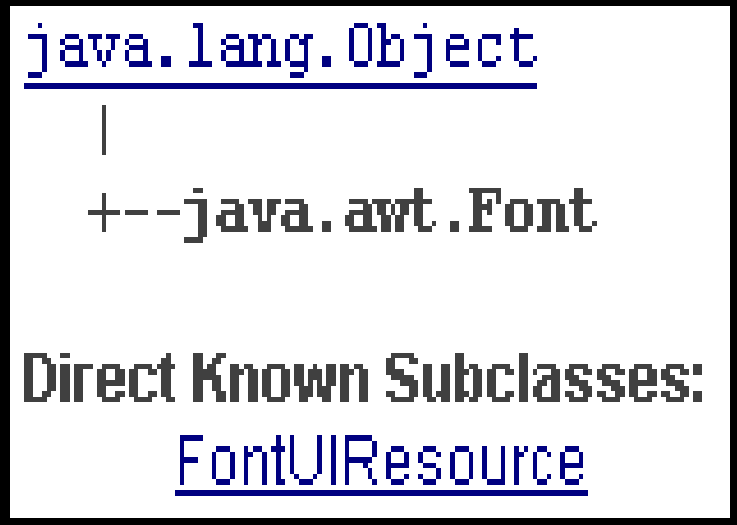
getRed(), getGreen(), getBlue(), getRGB()

- **Thí dụ về ấn định mầu hiện hành:**

```
Color  c = new Color( 255,130,60);
g.setColor(c) ;    // g: graphic object
…
g.setColor(new Color(100,0,200));
```

7.4- Điều khiển Font

- **Font = Kiểu chữ , mô tả nét vẽ (glyphs) của ký tự.**
- **Có ký tự 1 nét (a), 2 nét (á)**
- **3 thuộc tính của font: Font name, font style, font size.**
- **Lớp Font mô tả cho 1 font.**
- **Physical Fonts: Font thực, là các font TrueType hay PostScript Type 1.**
- **Logical Font: chia làm 5 nhóm: Serif, SansSerif, Monospaced, Dialog, và DialogInput**
- Label, TextField, ... chỉ sử dụng Logical Font



Font....

- **Có thể lấy tập font trong máy bằng 1 đối tượng thuộc lớp GraphicEnvironment.**
- **Lấy fonts hệ thống thông qua đối tượng GraphicsEnvironment**
- **Thí dụ: Lấy fonts hệ thống đưa vào choice cFonts**

```
GraphicsEnvironment ge;
ge=GraphicsEnvironment.getLocalGraphicsEnvironment();
Font f[]= ge.getAllFonts();
for (int i=0;i<f.length;++i)   cFonts.add(f[i].getFontName());
```

SystemFonts.java

Font (tt)

- **Lớp FontMetric cho ta kích thước font:**

String getName(): tên font

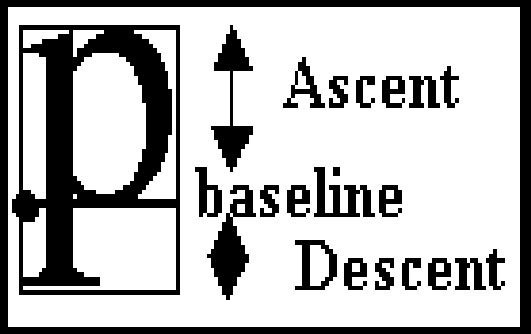
int getHeight() : chiều cao

int getAscend()

int getDescent()

int getHeight()

int getLeading()



FontMetricDemo.java

Khoảng hở đến ký tự kế tiếp

Tham khảo thêm trong Document của lớp này để biết thêm các methods

Thí dụ 2- Truy xuất thuộc tính kích thước font

7.5- Đồ họa với lớp Graphics.

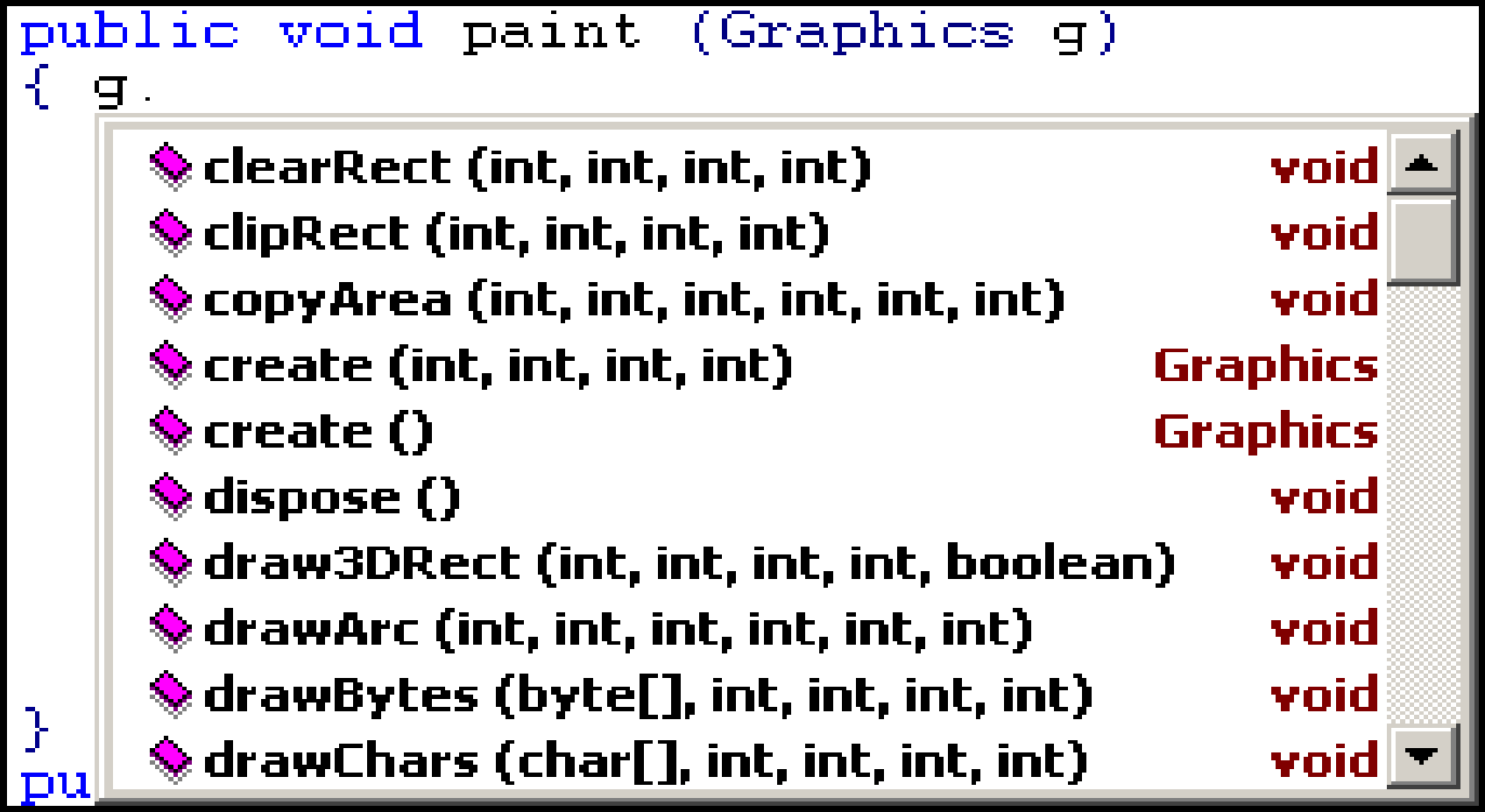
- **Graphic: Hình ảnh do ta vẽ hoặc file ảnh.**
- **Một GUI thiếu hình ảnh là 1 GUI thiếu sinh khí (dull).**
- **Trong gói AWT cung cấp đối tượng Graphics cho ta vẽ và lớp Image cho ta thao tác với file ảnh.**

```
java.lang.Object
   |
   +--java.awt.Graphics

Direct Known Subclasses:
      DebugGraphics, Graphics2D
```

```
java.lang.Object
   |
   +--java.awt.Image

Direct Known Subclasses:
      BufferedImage
```

lớp Graphics (tt)

- Lớp Graphics có các phương thức vẽ hình cơ bản, tô mầu: Hình Oval, Ractangle,Square, Circle, Lines, Text, xuất file ảnh…



lớp Graphics (tt)

- Muốn vẽ : Lấy đối tượng đồ họa kết hợp của Frame (Panel) bằng hành vi getGraphics() hoặc hiện thực 1 trong các phương thức sau:

(1) Phương thức **paint(Graphics g)** được gọi ngay khi nạp class và được gọi bởi phương thức update(..)

(2) Phương thức **repaint()** được gọi khi cần vẽ lại.

(3) Phương thức **update(Graphics g)** được gọi tự động bởi phương thức repaint(). Sẽ xó các đối tượng đồ họa cũ rồi gọi lại paint(g)

→ **Muốn vẽ thêm mà không xóa các hình ảnh cũ, cần override phương thức update như sau:**

```
public void update (Graphics g) { paint (g); }
```

Slide sau cho thấy thí dụ về cách viết chương trình đồ họa

7.5.1- Vẽ ký tự với font và mầu hiện hành

- **void drawString(String str, int x, int y)**

  vẽ chuỗi bắt đầu tại toạ độ (x,y)

- **void drawChars(char[] data, int offset, int length, int x, int y)**

  vẽ length ký tự từ vị trí offset trong mảng ký tự bắt đầu tại toạ độ (x,y)

- **void drawBytes(byte[] data, int offset, int length, int x, int y)**

  Vẽ ký tự có mã ký tự trong mảng data, từ vị trí offset, length ký tự bắt đầu tại toạ độ (x,y)

(Xem thí dụ 3)

Draw1.java

7.5.2- Vẽ hình ảnh – lớp Graphics

- abstract void drawLine(int x1, int y1, int x2, int y2)
- abstract void drawOval(int x, int y, int width, int height)
- abstract void drawPolygon(int[] xPoints, int[] yPoints, int nPoints)
- void drawPolygon(Polygon p)
- abstract void drawPolyline(int[] xPoints, int[] yPoints, int nPoints)
- void drawRect(int x, int y, int width, int height)
- abstract void drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)

7.5.3-Vẽ + tô mầu hình ảnh

void fill3DRect (int left, int top, int width, int height, boolean raised)

abstract void fillArc (int left, int top, int width, int height, int startAngle, int arcAngle)

abstract void fillOval (int left, int top, int width, int height)

abstract void fillPolygon (int[] xPoints, int[] yPoints, int nPoints)

void fillPolygon (Polygon p)

abstract void fillRect (int left, int top, int width, int height)

abstract void fillRoundRect (int left, int top, int width, int height, int arcWidth, int arcHeight)

Minh họa

Vẽ trên Frame – Draw2.java

Vẽ + Tô mầu- Draw3.java

Vẽ biểu đồ khối, biểu đồ quạt- BieuDo.java

Vẽ, Tô mầu đa giác - PolygonDemo.java

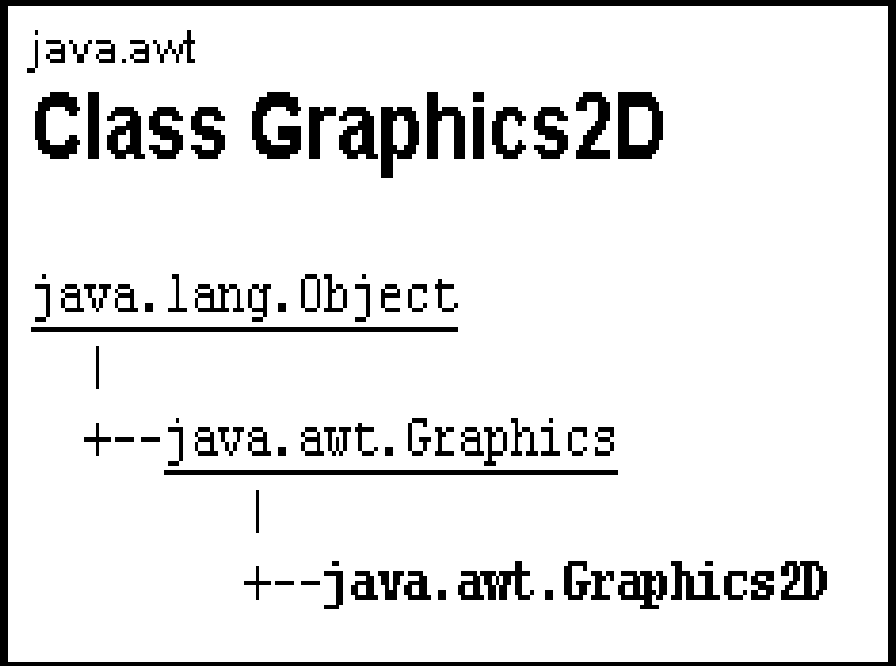
7.6- Paint mode

- 2 chế độ đồ họa:
- Overwrite mode: Nội dung mới xóa nội dung cũ.
- XOR mode : Nội dung mới không xóa nội dung cũ, cả 2 nội dung cùng khả kiến
- g.setXORMode(Color.cyan);
- Thí dụ: Xem Draw4.java trong tài liệu minh họa.

Xor-mode- Draw4.java

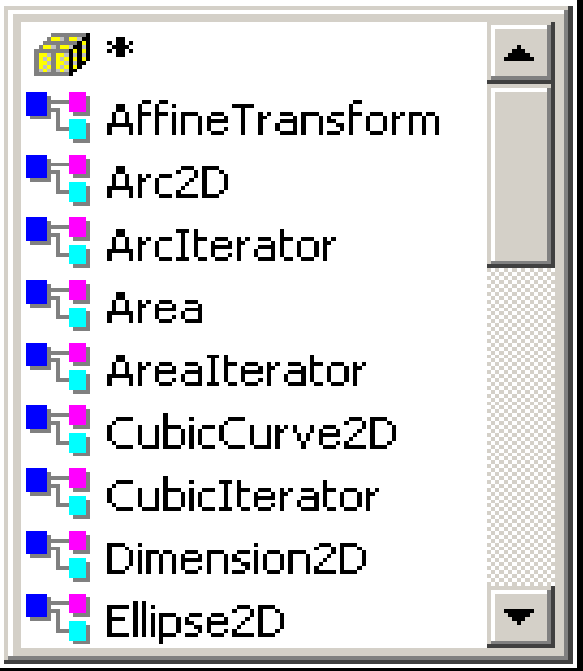
7.7- Đồ họa với Graphics2D

- Lớp Graphics cung cấp các methods đồ họa nhưng không xây dựng các lớp ảnh.
- Lớp Graphics2D kế thừa lớp Graphics nhưng có xây dựng các lớp mô tả ảnh và các phép biến hình …, có sử dụng hệ tọa độ thực



Đồ họa với Graphics2D (tt)

```
import java.awt.geom.
```



- Trong gói geom (geometry- hình học)
- có interface Shape và hiện thực của các lớp Polygon, RectangularShape, Rectangle, Line2D, CubicCurve2D, Area, GeneralPath, QuadCurve2D

Đồ họa với Graphics2D (tt)

java.awt.geom

Class Line2D

java.lang.Object
  |
  +--**java.awt.geom.Line2D**

Direct Known Subclasses:

Line2D.Double, Line2D.Float

java.awt.geom

Class Ellipse2D

java.lang.Object
  |
  +--java.awt.geom.RectangularShape
        |
        +--**java.awt.geom.Ellipse2D**

Direct Known Subclasses:

Ellipse2D.Double, Ellipse2D.Float

java.awt.geom

Class Rectangle2D

java.lang.Object
  |
  +--java.awt.geom.RectangularShape
        |
        +--**java.awt.geom.Rectangle2D**

Direct Known Subclasses:

Rectangle, Rectangle2D.Double, Rectangle2D.Float

- **Có lớp …Double, …Float cho phép mô tả hình trong hệ tọa độ thực.**

Đồ họa với Graphics2D (tt)

java.awt.geom

Class AffineTransform

java.lang.Object

|

+--java.awt.geom.AffineTransform

- Lớp AffineTransform mô tả cho các phép biến hình phẳng

Graph2D1.java

7.8- Thao tác với file ảnh

- Ảnh đơn giản là các hình vẽ.
- Ảnh phức tạp là các file ảnh
- Để xuất ảnh từ file, cần dùng đối tượng **Toolkit** (đối tượng truy xuất một số file ảnh) **và Image**
- **Toolkit có thể truy xuất file *.GIF*, *.JPG*, *.JPEG***
- Chỉ định file ảnh cục bộ bằng tên file.
- Chỉ định file trên mạng bằng địa chỉ URL.

Thao tác với file ảnh (tt)

- Sử dụng Toolkit để truy xuất 1 file ảnh cục bộ:

```
String FileName= “img1.jpg";
Image img;
img= Toolkit.getDefaultToolkit().getImage (FileName);
```

- Sử dụng Toolkit để truy xuất 1 file ảnh từ URL:

```
URL Addr = new URL ( http://www.xyz.com/img1.gif);
Image img ;
img= Toolkit.getDefaultToolkit().getImage (Addr);
```

Xuất file ảnh (tt)

Lớp Graphics, Graphic2D có các methods drawImage , 2 method thông dụng:

public abstract boolean drawImage

(Image img, int x,int y, ImageObserver observer)

abstract boolean drawImage (Image img, int x, int y, int width, int height, ImageObserver observer)

(x,y) vị trí trên trái của vùng xuất ảnh

observer: Đối tượng quan sát qúa trình nạp ảnh. Nếu là null, chỉ thấy ảnh sau khi thay đổi kích thước cửa số. Nên cho là container chư ảnh(this) để thấy ngay ảnh.

(Thí dụ: Xem tài liệu minh họa)

ShowImg1.java

ShowImg2.java

7.9- Tóm tắt

- The Graphics, Graphics2D classes are used to draw objects like text, lines, rectangle, ovals , arcs or show an image on the screen.
- The methods drawXXX of these class will draw graphic on the screen.
- To make an Image object associating to an image file, use java.awt.Toolkit class
- The Font class manages the font of the characters that will be drawn on the screen.
- The FontMetrics class is used to obtain information about a special font.
- The Color class is used to manage color of objects on the screen.

7.10- Chương trình vẽ bằng chuột

DrawWithMouse.java

Xin cám ơn

Chương 8- Applet

Mục tiêu

Sau chương này bạn có thể

- Định nghĩa được applet là gì.
- Phân biệt ứng dụng Java và applet Java.
- Mô tả được chu kỳ sống của 1 applet.
- Giải thích được tham số truyền cho applet.
- Giải thích được làm sao multimedia file có thể được thể hiện trên applet.

Nội dung

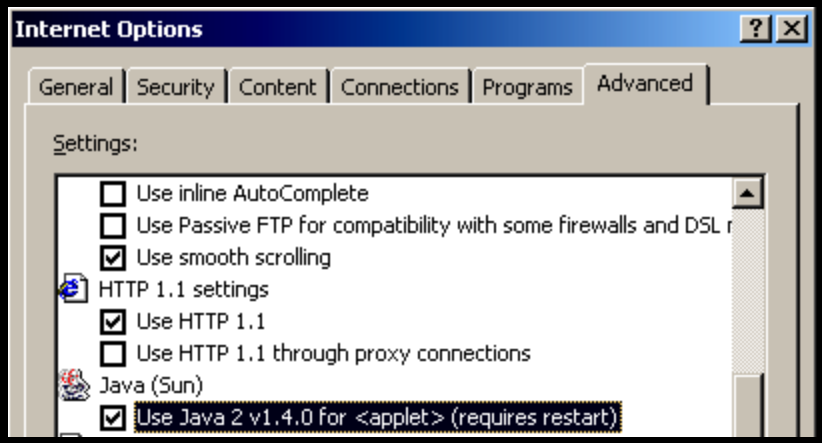
8.1- Cơ bản về applet.
8.2- Chu kỳ sống của một applet.
8.3- Tính bảo mật của applet
8.4- Các bước tạo applet.
8.5- Chạy applet
8.6- Applet trong Frame
8.7- Applet có hàm main
8.8- Truyền tham số cho applet
8.9- Multimedia với applet.
8.10- Cách chuyển 1 ứng dụng thành applet
8.11- Tóm tắt

8.1- Cơ bản về applet.

- Applet là một chương trình Java nhỏ được nhúng vào trang HTML và được thực thi nếu như Browser có hỗ trợ Java Applet (IE từ 3.0 trở lên, Netscape Navigator từ 3.0 trở lên). Applet trở thành 1 thành phần của trang HTML.
- Nhờ các applet, trang Web có thêm được sức sống vì applet có thể thể hiện multimedia, tạo được sự tương tác với user như hỏi-trả lời (còn có thể tạo tương tác trong trang Web với JavaScript, VBScript, những ngôn ngữ khác với Java).

Cơ bản về applet...

- Để có thể chạy applet, trình duyệt của user phải bật chức năng Java Plug-in: (mở Internet Explorer, vào Menu Tools/ Internet Options – chỉ có sau khi bạn cài đặt JRE)



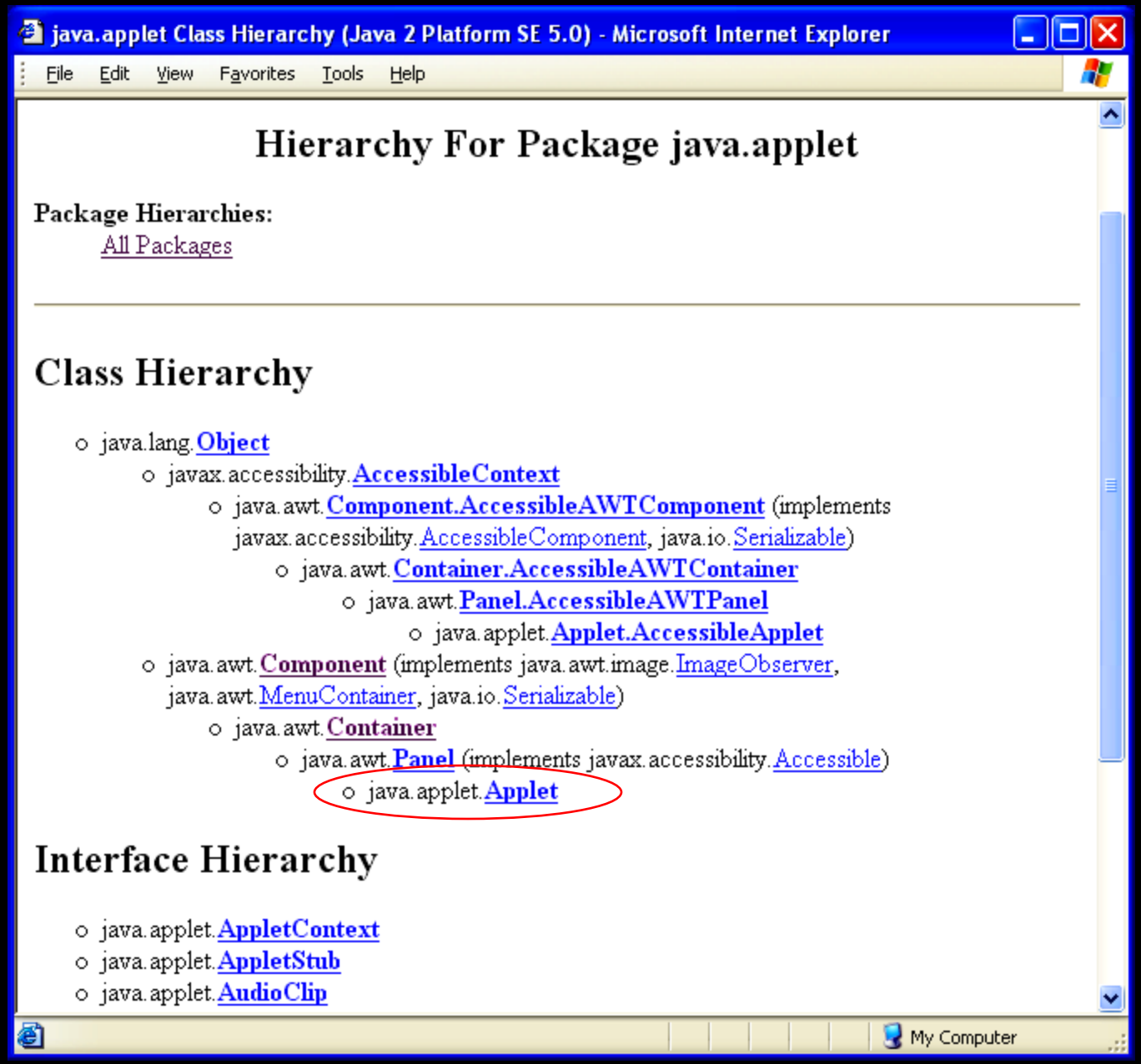
Để sử dụng gói Applet của Java:

import java.applet.*; // để sử dụng lớp Applet

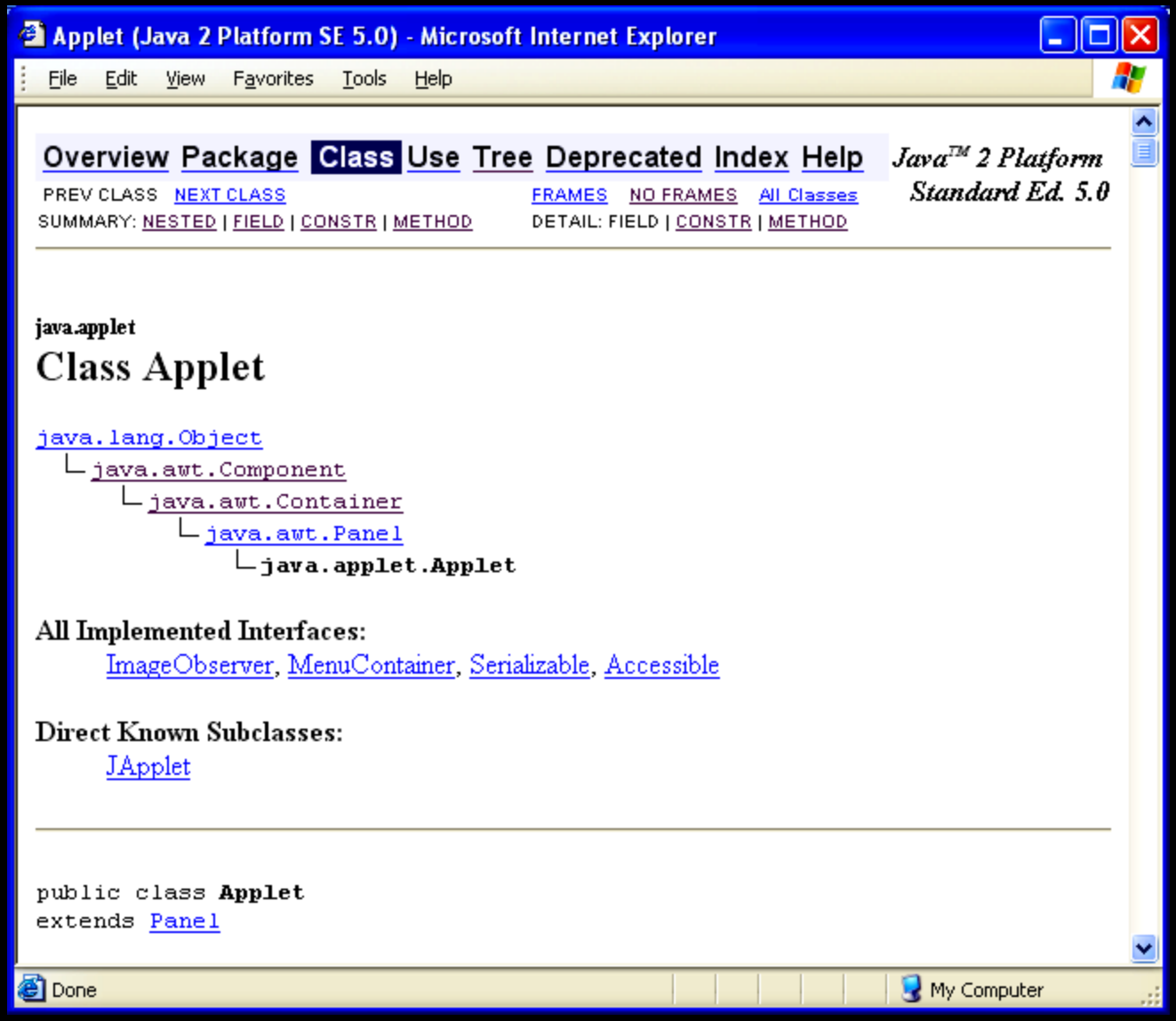
import javax.swing.* ; // để sử dụng lớp JApplet

- Để tạo 1 applet, ta đi xây dựng 1 class con của lớp Applet hoặc lớp JApplet. Người ta thường đặt tên lớp applet này trùng tên với tên file.html. Điều này **không bắt buộc nhưng thường dùng vì để tạo ra khả năng dễ bảo trì trang html cũng như apple**

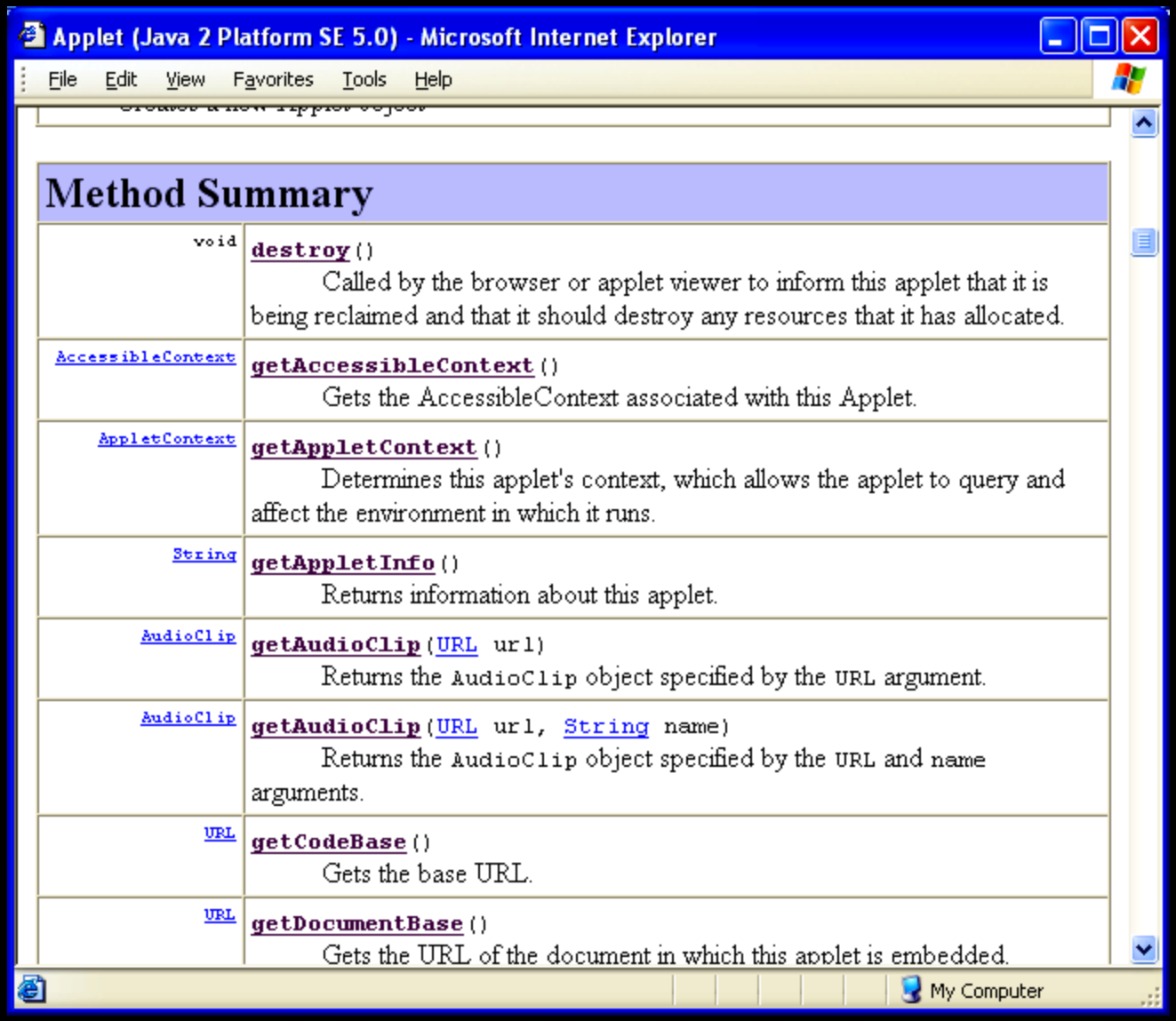
Cơ bản về applet ...



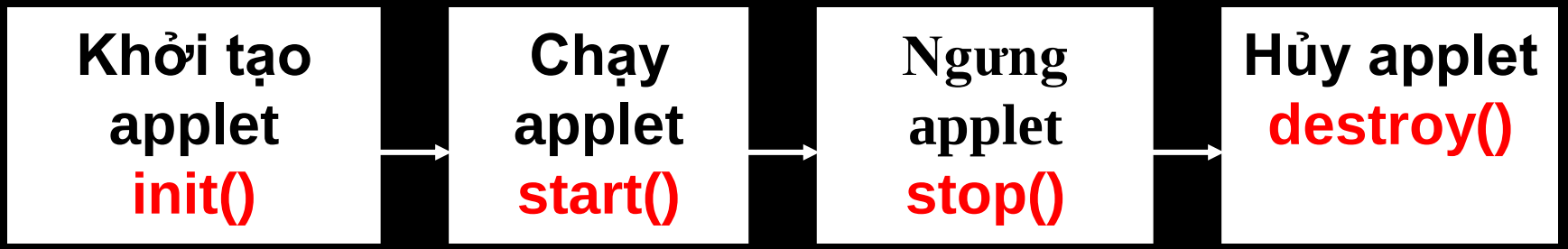
Cơ bản về applet ...



Cơ bản về applet ...



8.2- Chu kỳ sống của một applet



8.3- Tính bảo mật của applet

- Applet được thiết kế để được nạp từ remote site và được thực thi cục bộ trên client site. Do vậy, tính bảo mật là yêu cầu sống còn của applet.
- Nếu **client site browser** có bật chức năng hỗ trợ Java, browser sẽ download mã applet và thực thi ngay mã applet này. User không có cơ hội để xác nhận có muốn chạy applet hay không cũng như ngưng applet khi applet đang thực thi.
- Vì các lý do trên, applet được tải từ Web và thực thi trong Browser của client với 1 môi trường bảo vệ gọi là **sandbox** có những giới hạn để có được tính bảo mật của client system.

Giới hạn của applet

- **Aplet không được phép gọi 1 executable file nào ở hệ thống client.**
- **Applet không được phép giao tiếp với bất kỳ một máy chủ (host) nào khác ngoại trừ server mà applet đã được download.**
- **Applet không được thao tác đọc ghi lên hệ thống file của client.**
- **Applet không được tìm kiếm thông tin ở client system ngoại trừ** : Java version, OS version, các ký tự được dùng để phân cách file, phân cách dòng , phân cách đường dẫn. **Applet không được phép truy tìm thông tin như username, email address, …**
- **Mọi cửa sổ được sinh ra bởi applet** (popped by applet) luôn mang 1 thông điệp cảnh báo.
- **Trình Applet Security Manager sẽ xuất** (throws) **một SecurityException bất cứ khi nào applet v phạm caùc quy tắc về bảo mật.**

Applet và application

- Applet là 1 ứng dụng nhỏ được xây dựng nhằm mục đích phân phối trên Web và được thực thi bên trong (nhúng vào) trình duyệt trong khi application được thực thi độc lập (standalone application).
- Một applet phải được xây dựng 1 lớp con của lớp **java.applet.Applet** còn application thì không bị ràng buộc này.
- Application được thực thi thông qua **Java Interpreter** (máy ảo Java, trình java.exe) trong khi applet phải được thực thi thông qua **Browser có hỗ trợ Java** hay trình **appletviewer của môi trường Java**.

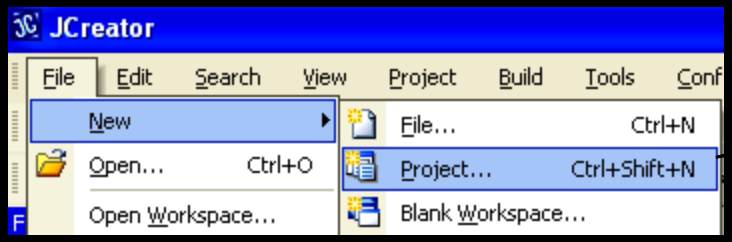
Applet và Application

- Thực thi Application bắt đầu từ **main(…)** trong khi applet bắt đầu thực thi bằng **init()**.
- Phương thức **draw_** dùng để xuất dữ liệu trong applet trong khi Application dùng **System.out.print** để xuất dữ liệu.
- Applet phải bắt đầu bằng ít nhất 1 **public class nếu không có thì sẽ có lỗi lúc biên dịch. Không bắt buộc phải có hàm main(…) trong applet. Nếu applet có hàm main(…) thì có thể chạy với 2 chế độ : (1) Chạy trong Browser, (2) Chạy dạng application**. Còn application luôn phải có hàm main(…).

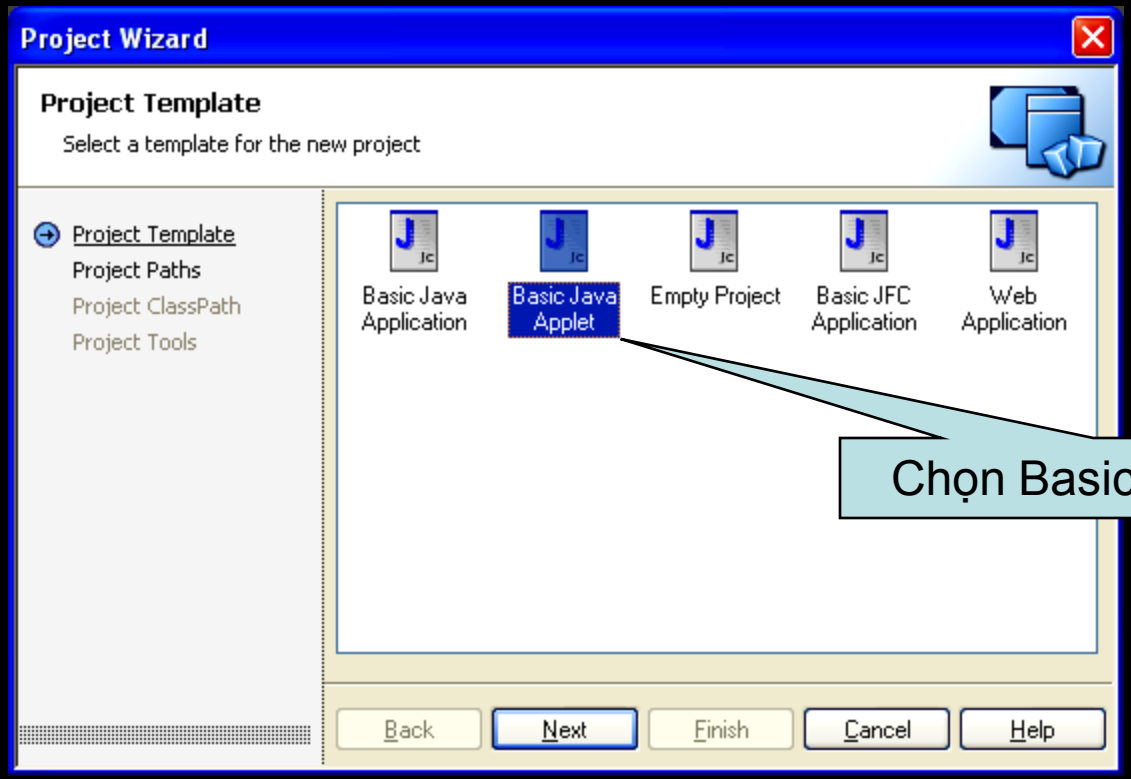
8.4- Các bước tạo applet

- Tạo applet với JCreator
- Tự tạo applet

8.4.1- Tạo applet với JCreator

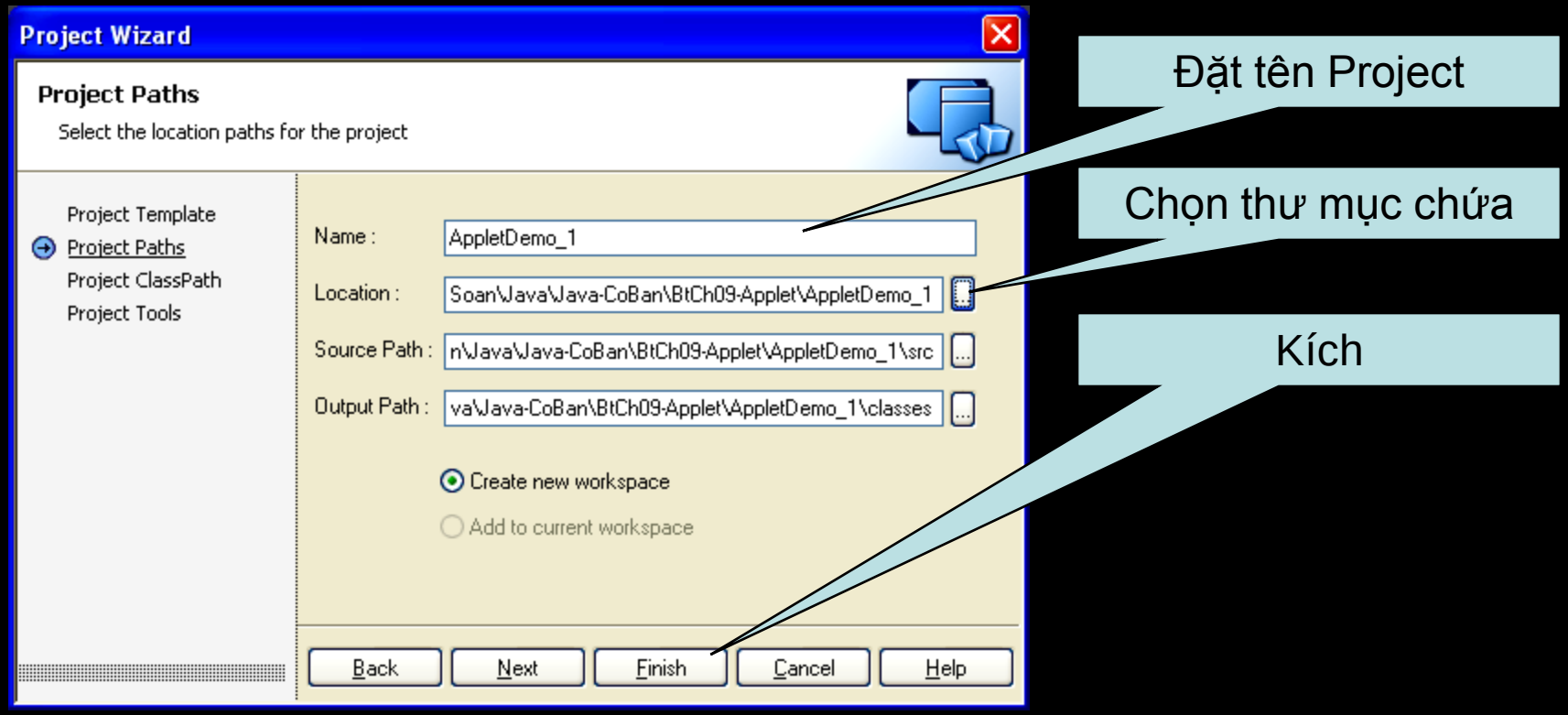


Menu File/New/Project

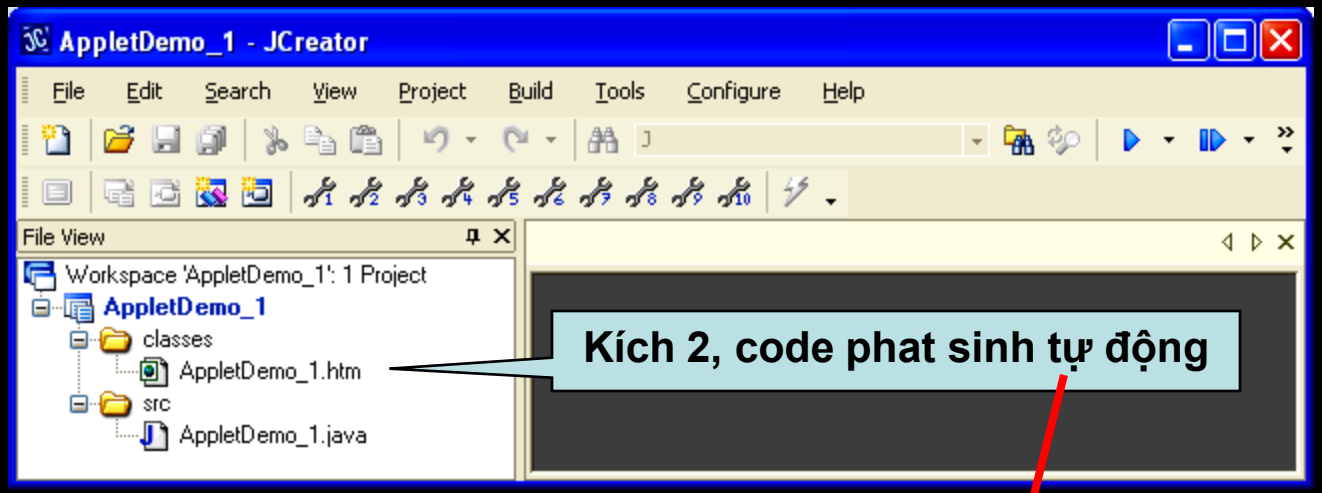


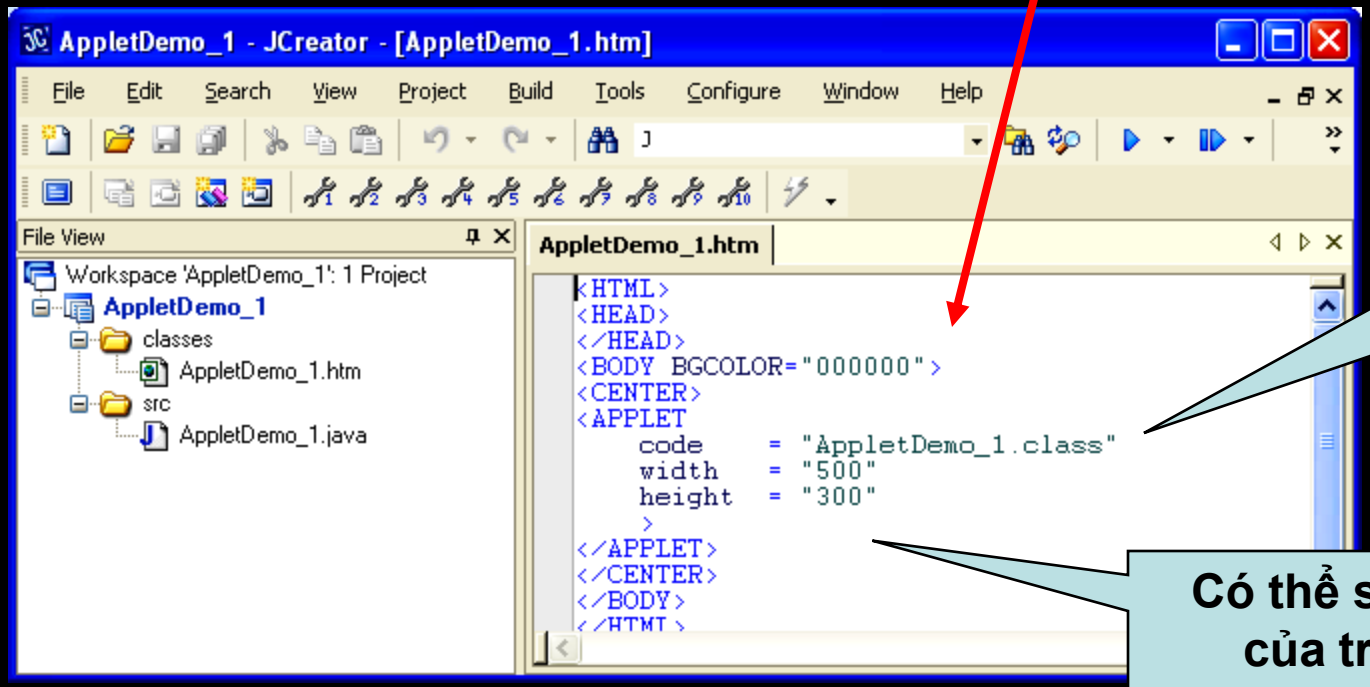
Chọn Basic Java Applet

Tạo applet với JCreator

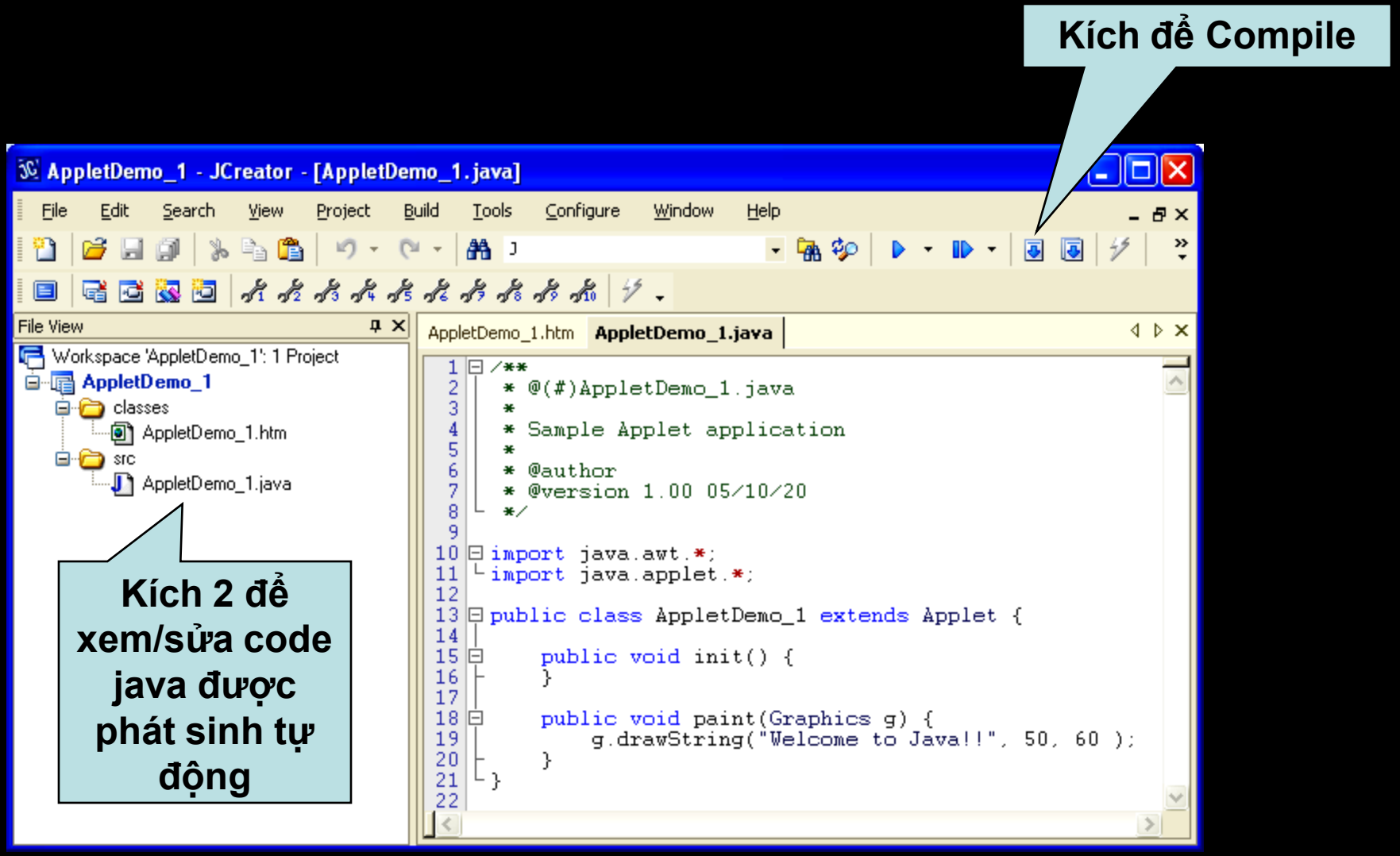


Tạo applet với JCreator

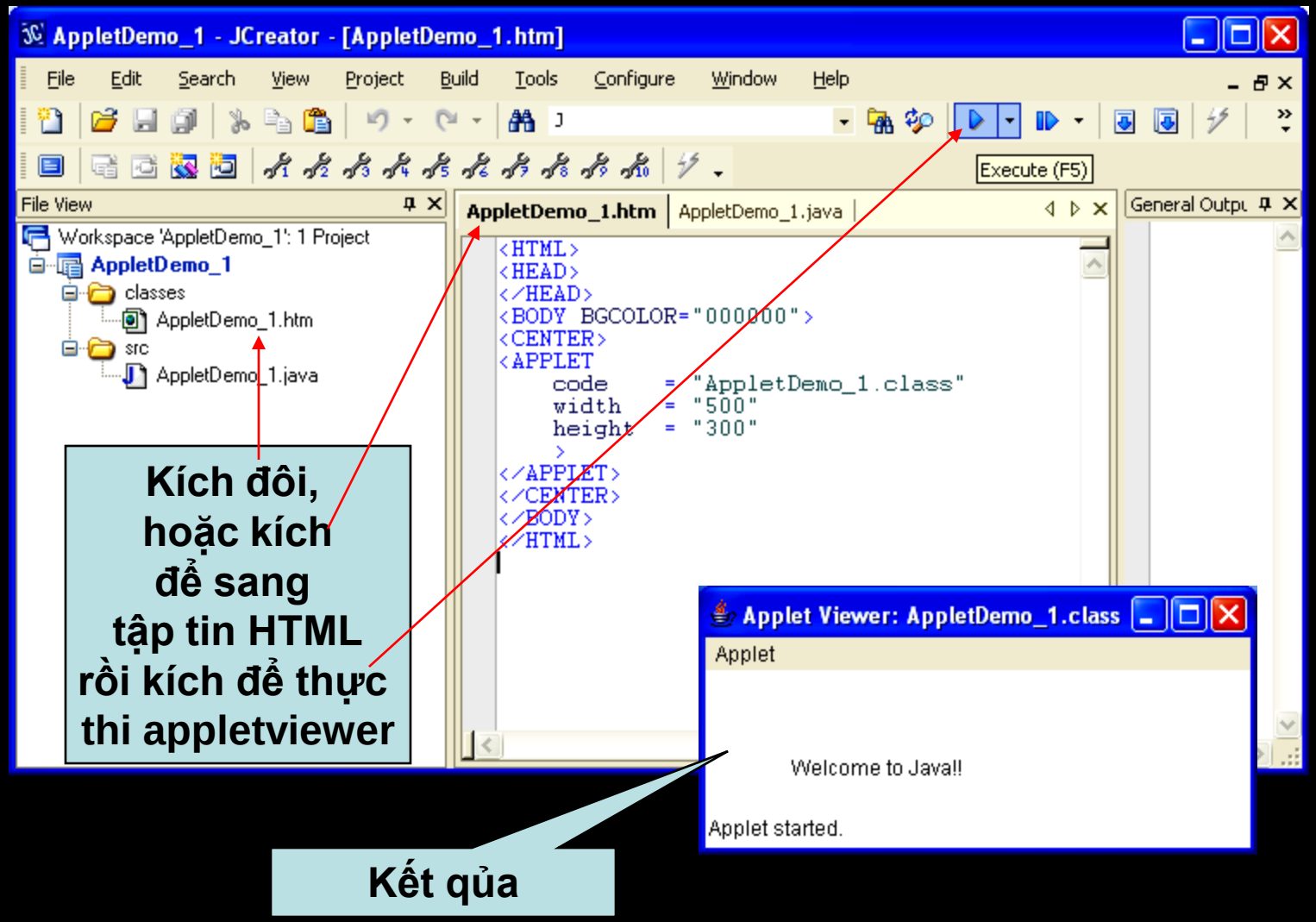




Tạo applet với JCreator

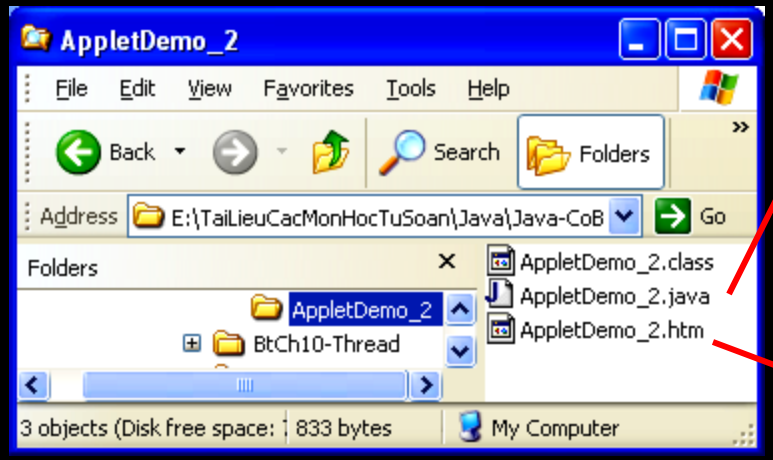


Tạo applet với JCreator



8.4.2- Tạo applet thủ công

Thiết kế : Sẽ tạo cấu trúc thư mục và tập tin

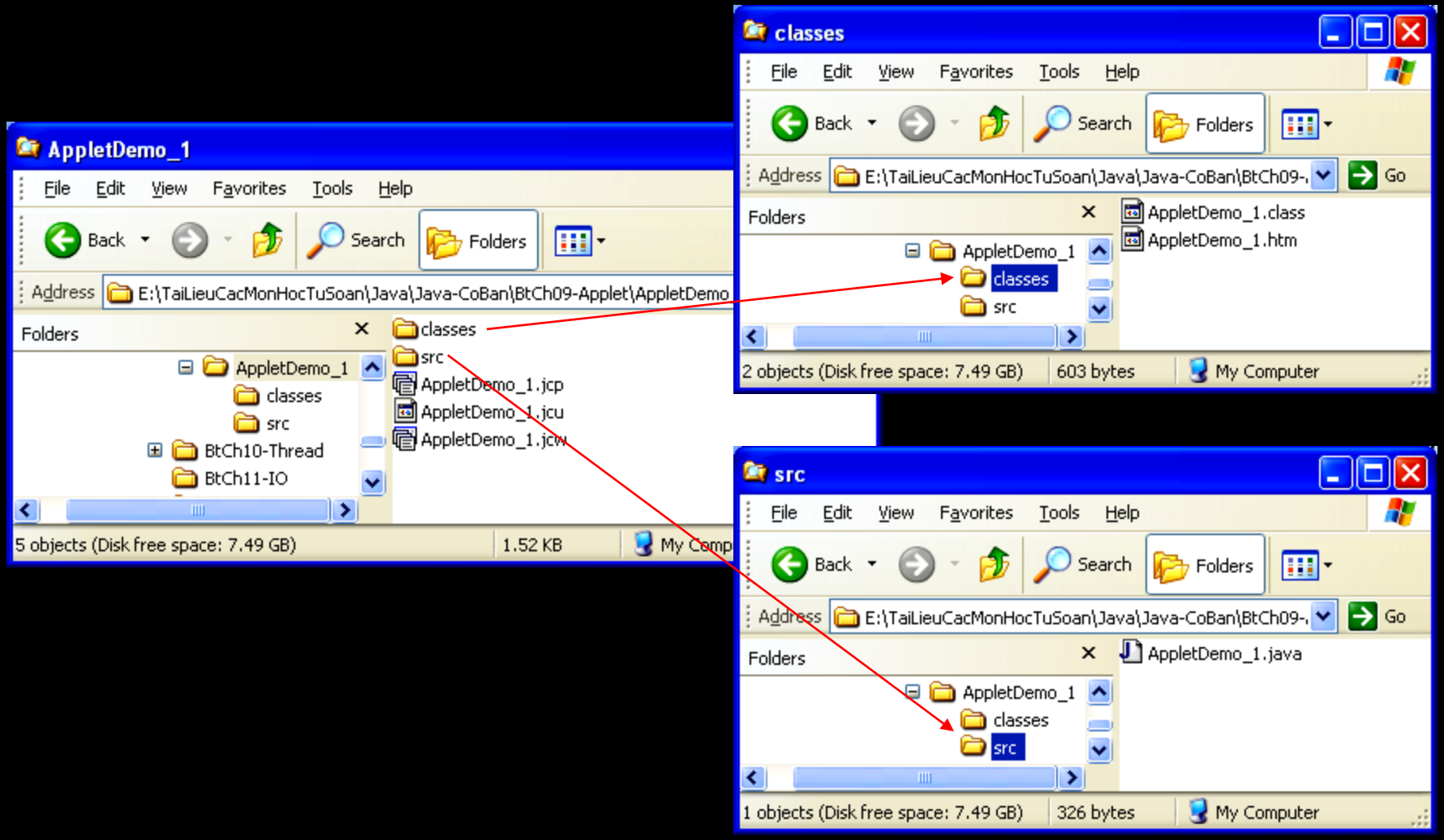


Chú ý:
Tập tin .htm và tập tin .class để trong cùng thư mục

```
import java.applet.*;
import java.awt.*;
public class AppletDemo_2 extends Applet
{   public void init()
    {    }
    public void paint(Graphics g)
    {  g.drawString
       ("Welcome From AppletDemo_2!", 50, 60 );
    }
}
```

```
<HTML>
<HEAD>
</HEAD>
<BODY BGCOLOR="red">
<CENTER>
<APPLET
    code   = "AppletDemo_2.class"
    width  = "200“  height      = "200“ >
</APPLET>
</CENTER>
</BODY>
</HTML>
```

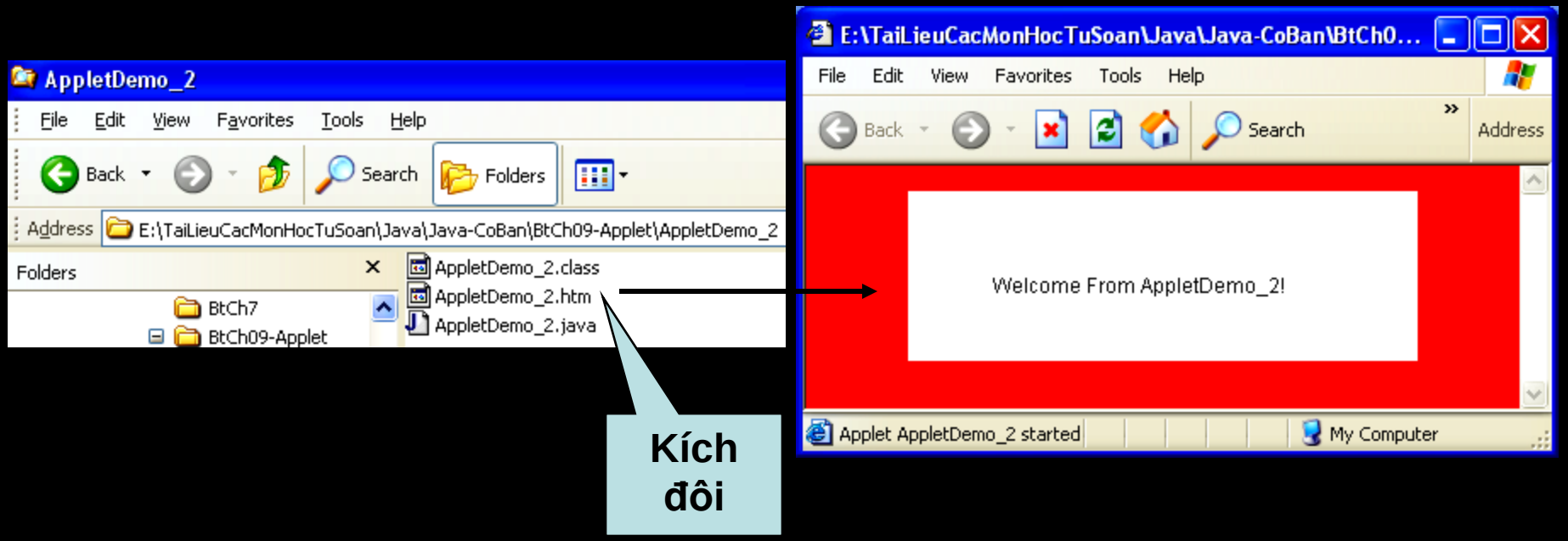
Cấu trúc 1 JCreator project



8.5- Chạy applet

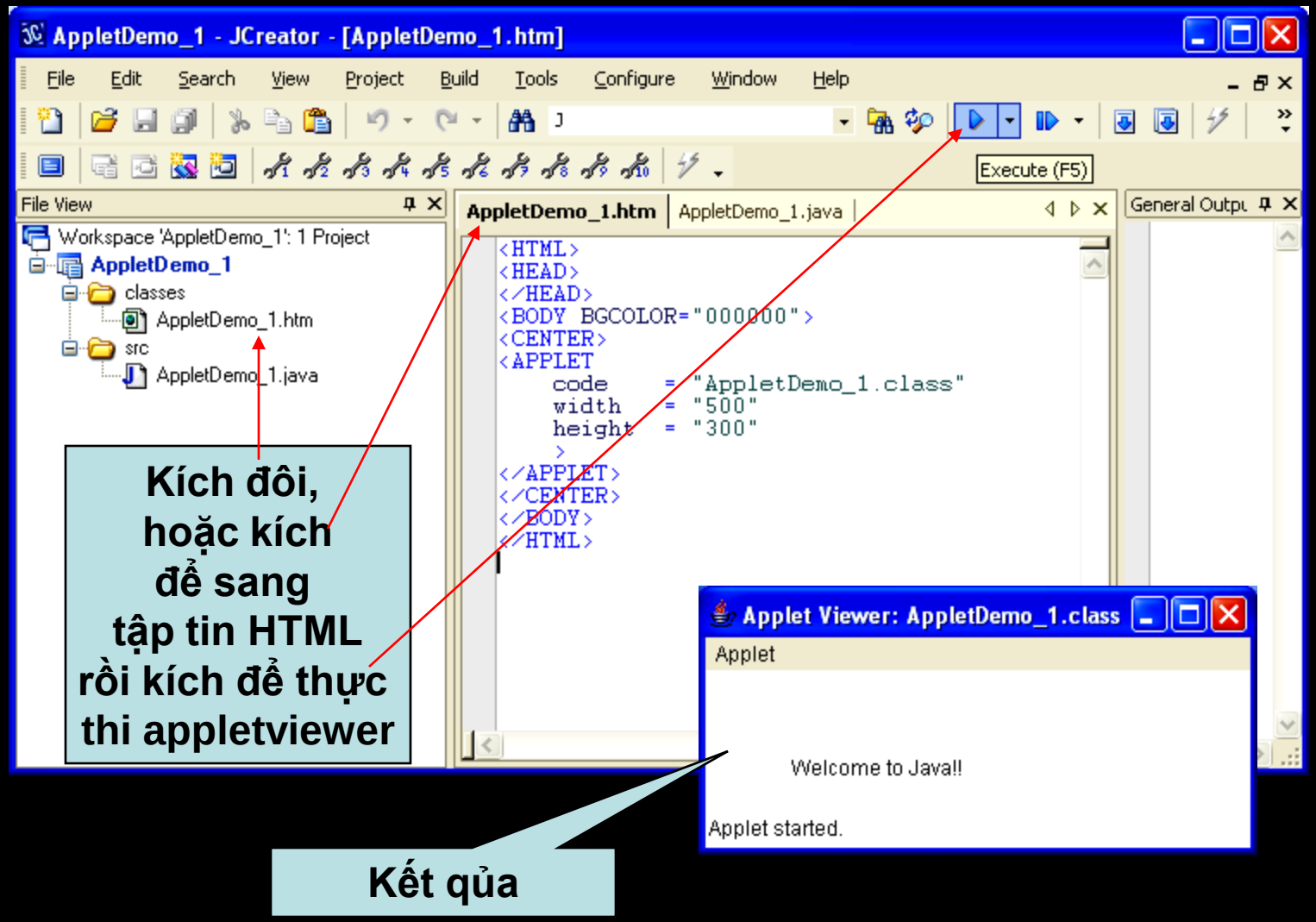
- Chạy applet trong web page
- Chạy applet bằng appletviewer
- Chạy Applet trong môi trường JCreator

Chạy applet trong Browser



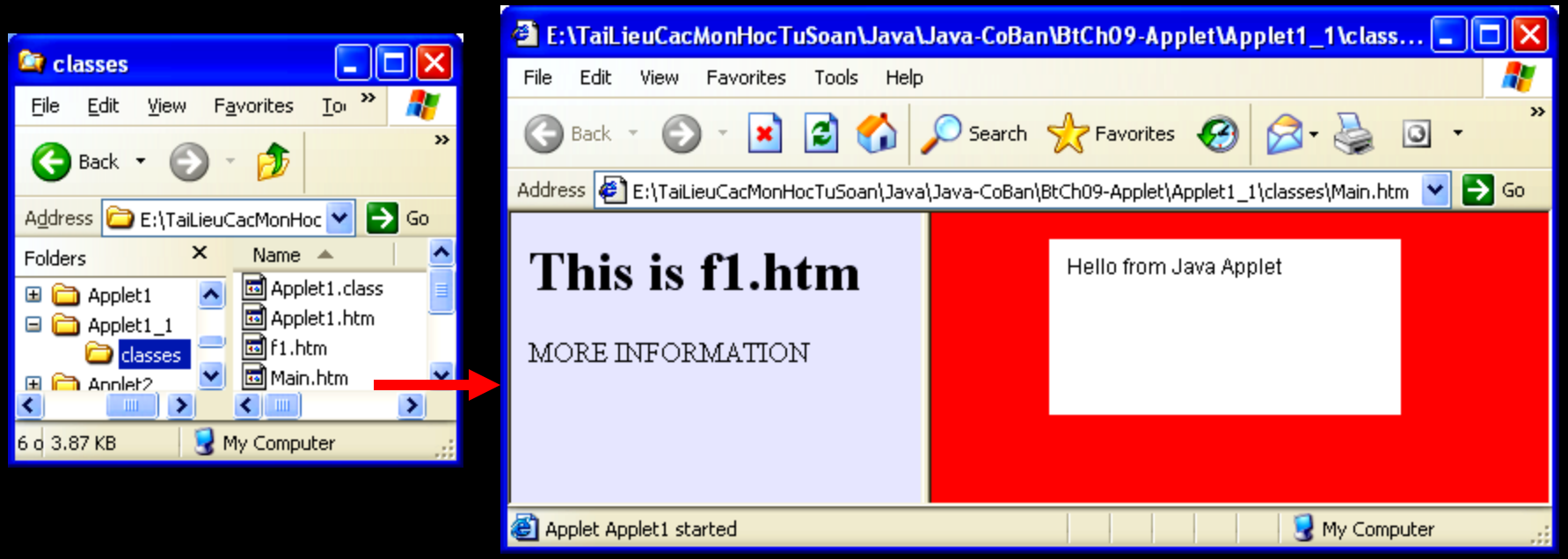
Cú pháp appletview

Chạy applet với JCreator



8.6- Applet trong Frame

- Đặt applet vào 1 frame, các frame khác hiển thị thêm thông tin



2 Frame ➔ 3 file.htm : main.htm, f1.htm, Applet1.htm

Main.htm

```
<HTML>
<HEAD>
</HEAD>
<FRAMESET cols= "40%,*">
  <noframes>
<BODY BGCOLOR="Lavender">
</BODY>
  </noframes>
  <FRAME src = "f1.htm">
  <FRAME src = "applet1.htm">
</frameset>
</HTML>
```

F1.htm

```
<HTML>
<HEAD>
</HEAD>
<BODY
BGCOLOR="Lavender">
<H1> This is f1.htm
</H1>
<P>  MORE
INFORMATION
</BODY>
</HTML>
```

Applet1.htm

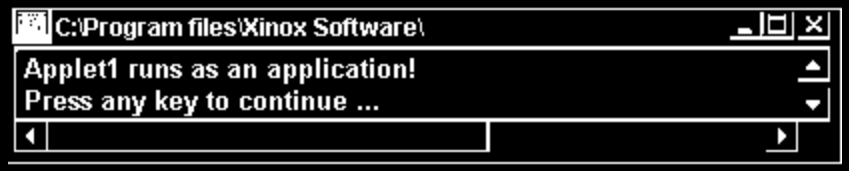
```
<HTML>
<HEAD>
</HEAD>
<BODY BGCOLOR="red">
<CENTER>
   <APPLET
      code          = "Applet1.class"
      width         = "200"
      height        = "100"
    >
   </APPLET>
   </CENTER>
</BODY>
</HTML>
```

Applet1.java

```
import java.awt.*;
import java.applet.*;
public class Applet1 extends Applet
{ public void init()
   { }
  public void paint(Graphics g)
  { g.drawString
    ("Hello from Java Applet", 10, 20 );
  }
}
```

8.7- Applet có phương thức main

- Cho phép chạy applet như một application
- Viết hàm main cho applet




```
import java.awt.*;
import java.applet.*;
public class Applet1 extends Applet
{    public void init()
     {  }
     public void paint(Graphics g)
     { g.drawString("Hello from Java Applet", 10, 20 );
     }
      public static void main (String args[])
     {  Applet1 ap = new Applet1();
        ap.init(); // khởi tạo ap nhưng không gọi appletviewer
        System.out.println("Applet1 runs as an application!");
   }
}
```

8.8- Truyền tham số cho applet

- Truyền tham số cho applet bằng thẻ <param> đặt bên trong thẻ <Applet>
- Một tham số có tên gọi và trị của tham số.
- Thí dụ sau minh họa truyền tham số từ Applet2.htm sang Applet2.class để xuất file ảnh Xuongrong.jpj
- Cú pháp của thẻ applet:

```
<APPLET
  code = "Applet2.class"
  width = "250“ height          = "280">
  <param name = "image"
         value = "xuongrong.jpg">
</APPLET>
```

Truyền tham số cho applet

Applet2 - JCreator - [Applet2.htm *]

File Edit Search View Project Build Tools Configure Window Help

File View

Workspace 'Applet2': 1 Project
Applet2
classes
Applet2.htm
Applet2.java

Applet2.htm * | Applet2.java

```
<HTML>
<HEAD>
</HEAD>
<BODY BGCOLOR="white">
<CENTER>
<APPLET
  code   = "Applet2.class"
  width  = "280" height = "300">
  <param name = "image" value = "xuongrong.jpg">
</APPLET>
</CENTER>
</BODY>
</HTML>
```

Truyền tham số có tên image sang applet với trị là tên file "xuongrong.jpg"

Bên Applet lấy trị của tham số thông qua tên bằng method getParameter ("image")

Kết qủa

Applet Viewer: Appl...

Applet

Applet started.

Applet2 - JCreator - [Applet2.java *]

File Edit Search View Project Build Tools Configure Window Help

File View

Workspace 'Applet2': 1 Project
Applet2
classes
Applet2.htm
Applet2.java

Applet2.htm * | Applet2.java *

```
1  import java.awt.*;
2  import java.applet.*;
3  import java.awt.*;
4  import java.applet.*;
5  public class Applet2 extends Applet
6  {  Image Img;
7     public void init()
8     {  String ImageName = getParameter("image");
9        Img = getImage(getCodeBase(), ImageName);
10    }
11   public void paint(Graphics g)
12    { g.drawImage(Img,0,10,this);
13    }
14 }
```

8.9- Multimedia với applet

- Multimedia – Đa môi trường, thêm hình ảnh, âm thanh vào trang web.
- Multimedia làm tăng tính hấp dẫn của trang Web.
- Các file multimedia có thể lấy ở hệ thống file cục bộ hoặc từ 1 địa chỉ URL.

8.9.1- Đưa ảnh vào Applet

- Các dạng ảnh được chấp nhận: JPG, JPEG, GIF.
- **Các bước hiển thị ảnh trong applet**

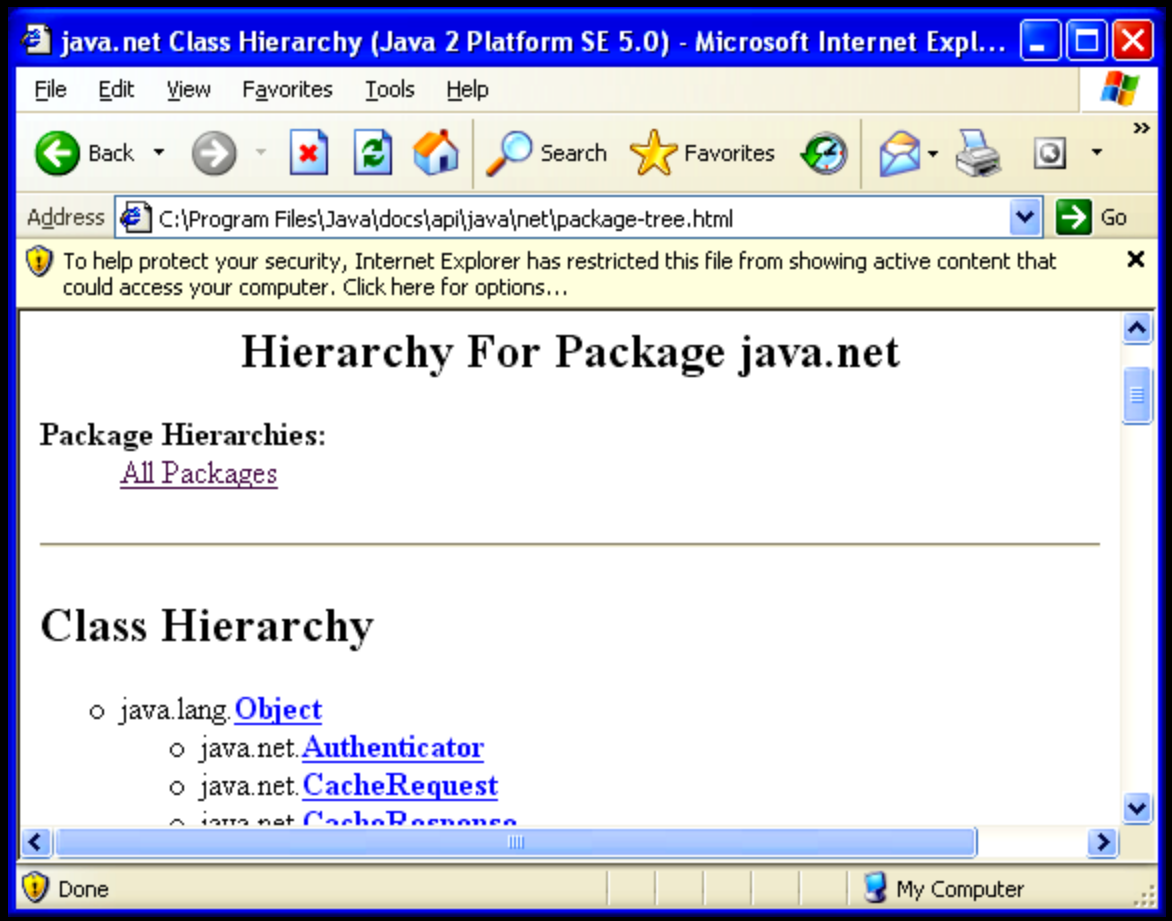
1. Xác định địa chỉ file ảnh (file ảnh trong máy hoặc URL)
2. Quyết định vị trí sẽ hiển thị ảnh.
3. Cung cấp các tham số cần thiết cho phương thức tương ứng. (xem **drawImage(..)**

Xem thí dụ trước về nạp file ảnh cục bộ

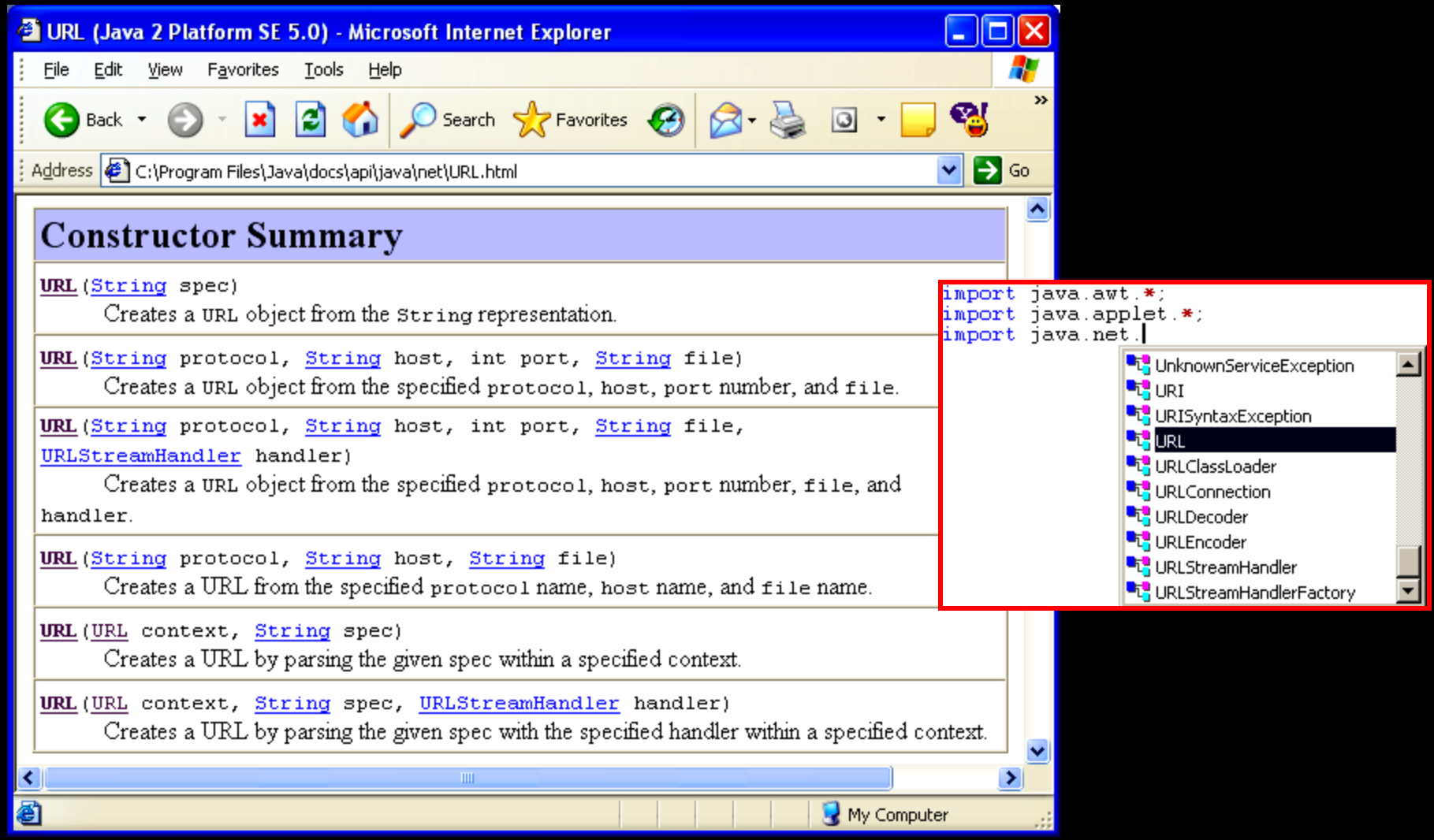
8.9.2- URL với Applet

- URL= Uniform Resource Locator
- “http://www.sun.com/index.html” cho trình duyệt biết dùng hypertext transfer protocol để tải trang index.html từ www.sun.com.
- Java cung cấp lớp URL trong gói java.net cho phép tạo 1 đối tượng quản lý địa chỉ trang Web.

URL với Applet...



Constructors của lớp URL



Thí dụ về mô tả 1 URL

- Mô tả 1 địa chỉ tuyệt đối

  URL url1 = new URL

  (“http://www.sun.com/index.html”);

- Để định vị tương đối 1 trang (1 file)

  URL url2 = new URL (u, “tmc/abc.gif”);

Trong đó mô tả abc.gif nằm trong thư mục tmc và tmc là thư mục con URL u đã có.

8.9.3- Xuất ảnh

- Trích 1 đối tượng ảnh trong applet từ URL

  Image Img1 = getImage

  (“http://www.abc.com/pic1.gif”);
- Trích 1 đối tượng ảnh trong applet từ local file

  Image Img2 = getImage

  (getCodeBase(), “pic2.jpg”);
- **Vẽ ảnh Img1 trong applet tại vị trí (x,y) . Dùng đối tượng Graphics g của Applet**

  g.drawImage (Img1, x,y,this);

Tham khảo lại method drawImage của lớp Graphics

8.9.4- Vẽ trong Applet

Dùng các hành vi của lớp Graphics về vẽ đã được giới thiệu trong chương đồ họa :

- **drawLine (x1,y1, x2, y2);**
- **drawRect (x,y,Width,Height) , fillRect (x,y,Width,Height) ;**
- **drawOval (x,y,width,height) , fillOval (x,y,width,height)**
- **drawArc (x,y,width,Height,startAngle, degree)**
- **fillArc (x,y,width,Height,startAngle, degree)**
- **...**

Applet3.HTM

8.9.5- Âm thanh với Applet

- Interface java.applet.AudioClip cho việc xuất âm thanh.
- Các hành vi:

Âm thanh với Applet

- **Khai báo và nạp AudioClip**

```
AudioClip audioClip;
```

- **Lấy file âm thanh từ local file**

```
AudioClip getAudioClip (getDocumentBase(), String Filename);
```

- **Lấy file âm thanh từ URL**

```
AudioClip  getAudioClip (URL url);
AudioClip  getAudioClip (URL url , String filename );
```

Amthanh1.htm

8.10-Cách chuyển 1 ứng dụng thành Applet

(1) import java.applet.*;

(1) Đổi lớp cha từ Frame sang Applet

(2) Đổi constructor thành public void init(), bỏ phát biểu super(...) nếu có vì đây là hành vi của lớp Frame.

(3) Bỏ Window Listener vì trong Applet không có tác vụ này

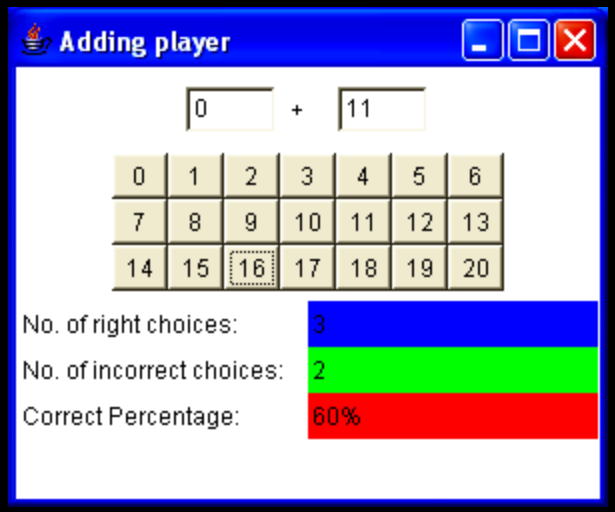
(4) Bỏ hàm main

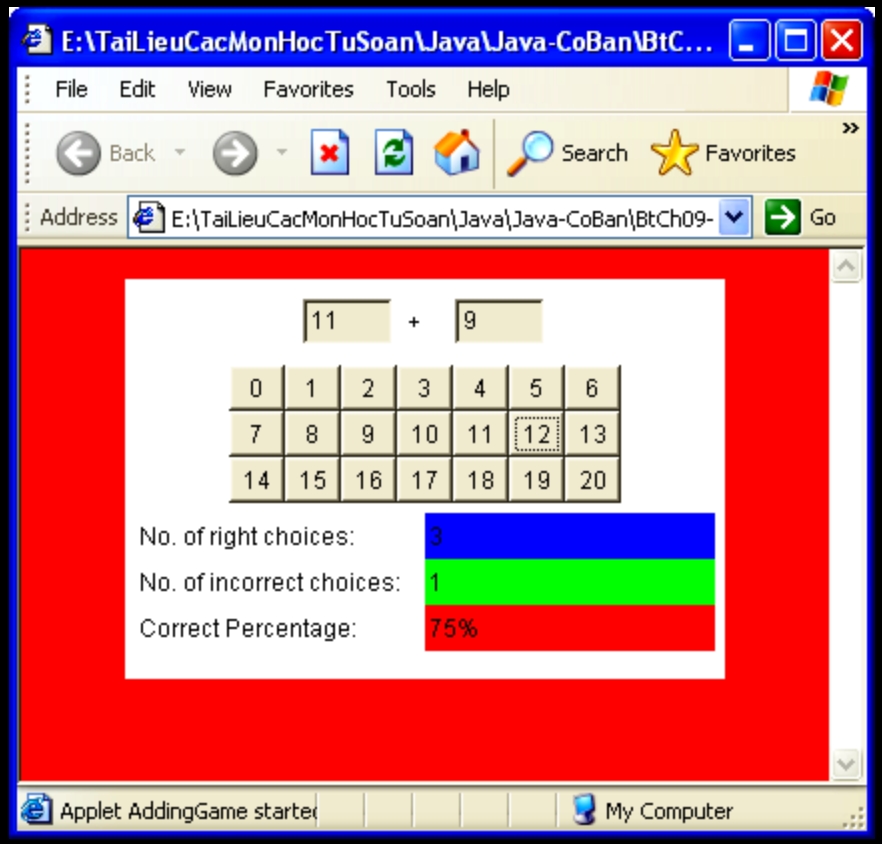
Cách chuyển 1 ứng dụng thành Applet

```
import java.awt.*;
import java.awt.event.*;

class A extends Frame
{ <data>

  A()
 { super(...);
   <code>
   addWindowListener(...)
 }
 < các methods>
 public static void main(...)
 {  ...  }
}
```

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
public class A extends Applet
{ <data>

public void init()
 {  // super(...);
   <code>
  //addWindowListener(...)
 }
 < các methods>
 // Bỏ  main(...)

}
```

Minh họa chuyển application thành applet





Appication
BaiTap3.java
Trò chơi phép cộng
trong chương trước

Xem code

Applet AddingGame
chạy trong Browser

8.11- Tóm tắt

- Applet là 1 ứng dụng Java nhỏ nhúng vào trình duyệt
- Để giữ tính bảo mật, applet không được truy cập hệ thống file của máy khách, không được truy cập server nào khác ngoại trừ server mà từ đó applet này được download.
- Applet có chu kỳ sống: init, start, stop, destroy.
- Gói java.applet chứa lớp Applet giúp ta tạo ra applet.
- Tạo 1 applet tức là hiện thực 1 lớp con của lớp Applet.

Tóm tắt

- Phương thức paint(Graphics g), repaint(Graphics g), update(Graphics g) thường được dùng để vẽ trong panel của applet
- Trong Applet, ta có thể xuất chữ, vẽ các hình, xuất ảnh bằng các hành vi drawXXX(...).
- Muốn applet có thể chạy như 1 application thì viết thêm hàm main cho applet.
- Truyền tham số từ file.htm sang applet bằng thẻ <parameter name="paraName" value ="paraValue">

Tóm tắt

- **Lấy file ảnh bằng**

Img = getImage(getCodeBase(), ImageFileName);

- **Gói java.net chứa lớp URL mô tả về một url**
- **Tạo 1 đối tượng URL**

URL url1 = new URL("http://www.sun.com/index.html");

- **Interface java.applet.AudioClip khai báo các han2h vi xử lý âm thanh gồm start(), loop(), stop()**
- **File âm thanh file.au**
- Nạp file âm thanh

AudioClip audioClip=
getAudioClip(getDocumentBase(), MusicFileName);

Xin cám ơn

Chương 9- THREADS

Mục tiêu

Sau chương này bạn có thể

- Định nghĩa được luồng (thread) là gì.
- Hiểu đa luồng là gì?
- Biết cách tạo luồng trong Java.
- Hiểu về nhu cầu đồng bộ (synchronize) các luồng.
- Biết cách dùng wait() và notify() để giao tiếp giữa các luồng.

Nội dung

9.1- Ôn tập

- Gói AWT cung cấp các lớp cho ta xây dựng GUI nhưng các lớp này sử dụng các hỗ trợ phụ thuộc platform.
- LỚp Graphics và Graphics2D trong gói AWT cho ta các công cụ vẽ hình và xuất file ảnh.
- Lớp Applet và JApplet cung cấp khả năng tạo các ứng dụng nhỏ của Java nhúng vào trang Web và chúng được thực thi trong Browser.
- appletviewer cho phép chạy một Java applet mà không cần đến Browser.

9.2- Luồng và đa luồng

- **Luồng- thread**: **Một dòng các lệnh mà CPU phải thực thi.**
- Các hệ điều hành mới cho phép nhiều luồng được thực thi đồng thời. Chúng ta đã quen với việc mở nhiều ứng dụng trong 1 lần làm việc với máy tính → Nhiều ứng dụng được nạp.
- **Như vậy**
  - Một luồng là một chuỗi các lệnh nằm trong bộ nhớ (chương trình đã được nạp).
  - 1 application thông thường khi thực thi là 1 luồng.
  - Trong 1 application có thể có nhiều luồng. Thí dụ chuyển động của 10 đối tượng hiện hành trong 1 trò chơi là 10 luồng.

Kỹ thuật đa luồng

- Với máy có m CPU chạy m luồng → Mỗi CPU chạy 1 luồng → Hiệu quả.
- Với máy có m CPU chạy n luồng với n>> m→ Mỗi CPU chạy n/m luồng.
- Với 1 CPU chạy đồng thời k luồng với k>1. Các luồng được quản lý bằng 1 hàng đợi, mỗi luồng được cấp phát thời gian mà CPU thực thi là ti (cơ chế time-slicing – phân chia tài nguyên thời gian). Luồng ở đỉnh hàng đợi được lấy ra để thực thi trước, sau ti thời gian của mình, luồng này được đưa vào cuối hàng đợi và CPU lấy ra luồng kế tiếp.
- Với máy chỉ có 1 CPU mà lại chạy k luồng → Hiệu suất mỗi chương trình sẽ kém.

Lợi ích của đa luồng

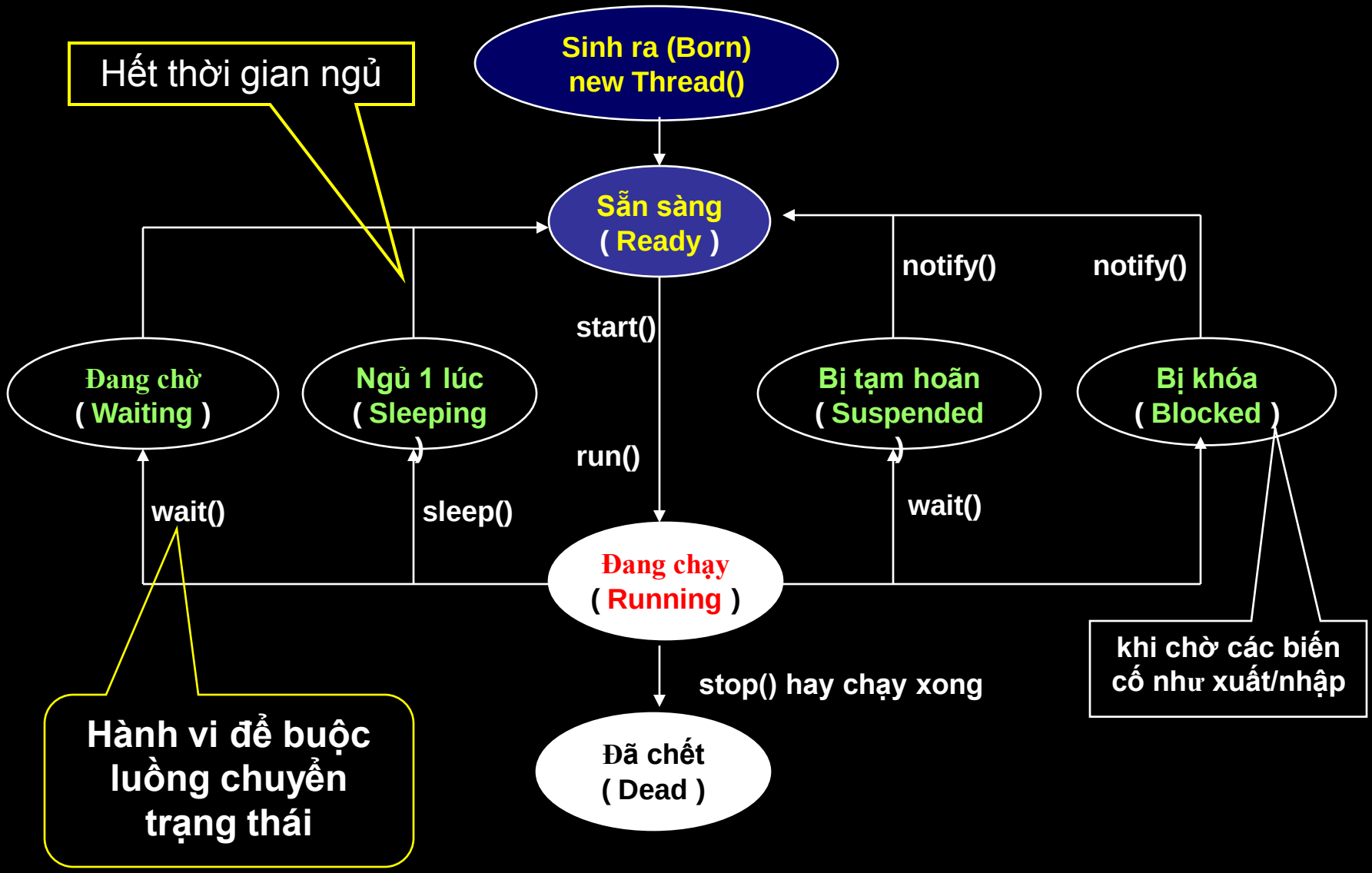
- **Tăng hiệu suất sử dụng CPU:** Phần lớn thời gian thực thi của 1 ứng dụng là **chờ đợi nhập liệu từ user → hiệu suất sử dụng CPU chưa hiệu qủa.**
- **Tạo được sự đồng bộ giữa các đối tượng**: Thí dụ như trong 1 trò chơi, các nhân vật cùng nhau chuyển động. Trong 1 trang Web, tạo được sự đồng thời của các đường diềm (marquee) như thanh tiêu đề động (banner, chữ,ảnh chạy), vừa hiển thị đồng hồ, vừa phát nhạc, vừa chơi game, vừa hoạt ảnh (animated images),… → Trang Web thật bắt mắt (eye-catching) và quyến rũ (captivating).
- Quản lý được thời gian trong các ứng dụng như thi online, thời gian chơi một trò chơi.

9.3- Luồng trong Java

- **Main thread - luồng chính** : là luồng chứa các luồng khác. Đây chính là luồng cho Java. Application hiện hành (mức toàn application).
- **Child thread** - **luồng con** : là luồng được tạo ra từ luồng khác.
- Khi 1 application thực thi, main thread được chạy, khi gặp các phát biểu phát sinh luồng con, các luồng con được khởi tạo. Vào thời điểm luồng chính kết thúc, application kết thúc.
- Java cung cấp lớp **Thread**

  mô tả 1 luồng trong gói **java.lang**

9.4- Trạng thái của luồng



Trạng thái của luồng

- Một luồng sau khi sinh ra (born) không được chạy ngay mà chỉ là sẵn sàng (ready) chạy. Chỉ khi nào phương thức ***start()*** được gọi thì luồng mới thực thi (chạy code phương thức run()).
- Luồng đang thực thi có thể bị tạm ngưng bằng phương thức ***sleep()*** một thời khoảng và sẽ lại ready sau khi đáo hạn thời gian. Luồng đang **ngủ** không sử dụng tài nguyên CPU.
- Khi nhiều luồng cùng được thực thi, nếu có 1 luồng giữ tài nguyên mà không nhả ra sẽ làm cho các luồng khác không dùng được tài nguyên này (đói tài nguyên). Để tránh tình huống này, Java cung cấp cơ chế ***Wait-Notify****(đợi-nhận biết)* và cơ chế này được trình bầy ở mục sau. Phương thức ***wait()*** giúp đưa 1 luồng vào trạng thái chờ.

Trạng thái của luồng

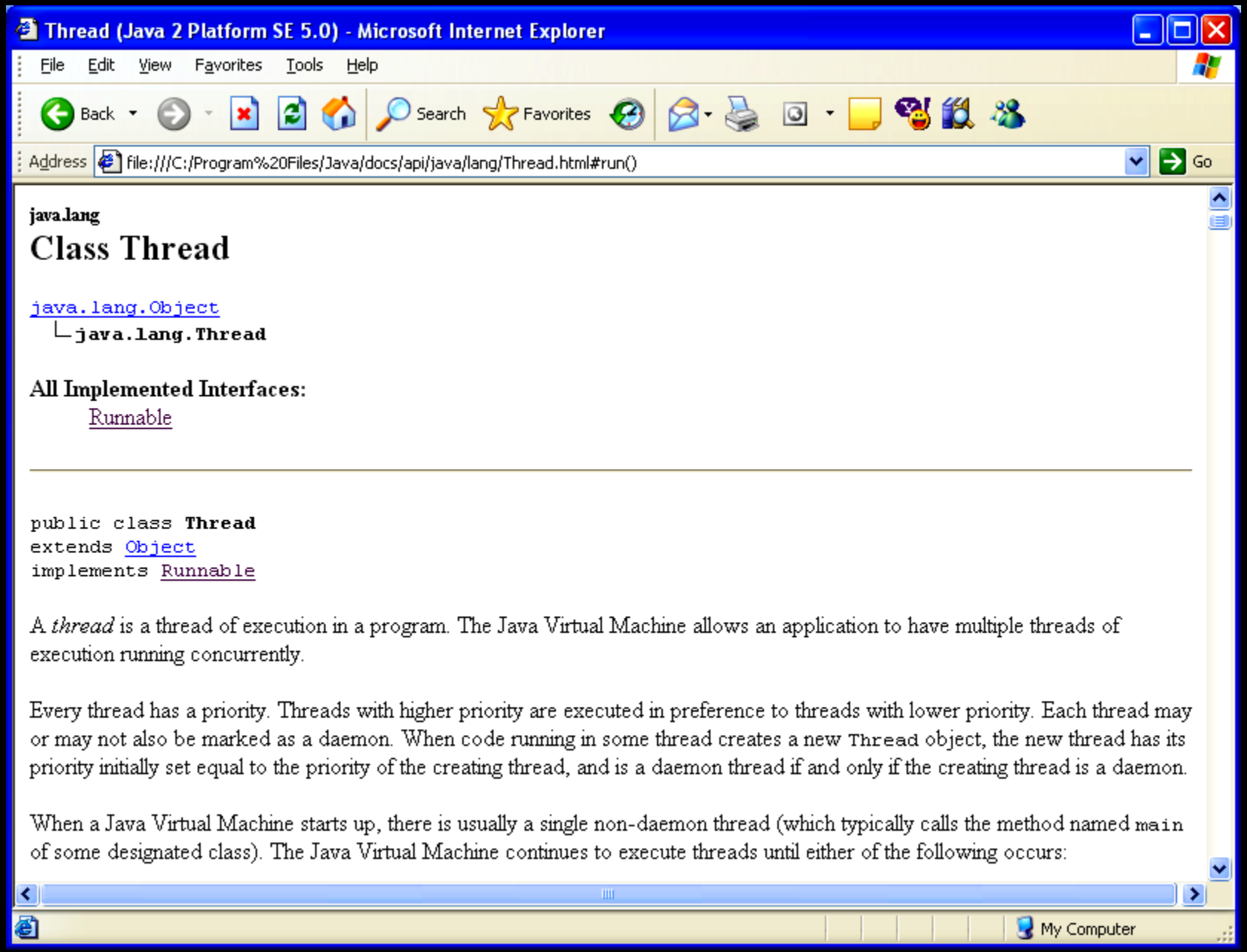
- Khi một luồng bị tạm ngưng hay bị treo, luồng rơi vào trạng thái tạm hoãn (suspended). Phương thức ***suspend()- version cũ/ wait() trong Java 2*** dùng cho mục đích này.
- Khi 1 suspended thread được mang ra thực thi tiếp, trạng thái của luồng là resumed. Phương thức ***resume() – version cũ/ notify() trong Java 2 được dùng cho mục đích này***.
- Khi 1 luồng chờ biến cố như xuất/nhập dữ liệu. Luồng rơi vào trạng thái *blocked*.
- Khi 1 luồng thực thi xong phương thức ***run()*** hay gặp phương thức ***stop()***, ta nói luồng đã chết (dead).

9.5- Lập trình luồng trong Java

- Cách 1: Xây dựng 1 lớp con của lớp java.lang.Thread, override hành vi run() để phù hợp với mục đích bài toán.
- Cách 2: Xây dựng 1 lớp có hiện thực interface Runnable

  - Không cần import java.lang vì là gói cơ bản.
  - java.lang.Thread là lớp Java xây dựng sẵn đã hiện thực interface Runnable.
  - Interface java.lang.Runnable chỉ có 1 method run()
  - Tham khảo thêm trong gói java.lange

Tham khảo lớp Thread



9.5.1- Tạo luồng là lớp con của lớp Thread

```
class MyThread extends Thread
{  // dữ liệu + hành vi của lớp
     public void  run()
    { // hiện thực code phụ thuộc bài toán
     }
}
```

9.5.2- Tạo luồng với interface Runnable

```
class MyThread implements Runnable
{   // dữ liệu + hành vi của lớp
     public void  run()
    { // hiện thực code phụ thuộc bài toán
     }
}
```

9.5.3- Khởi tạo và thực thi 1 luồng

MyThread t = new MyThread(); // tạo 1 luồng
t.start(); // chỉ thị cho luồng thực thi

Hành vi start() sẽ tự động gọi hành vi run()

9.5.4- Thread Constructors

- **Thread ()**
- **Thread (Runnable target)**
- **Thread (Runnable target, String Name)**
- **Thread (String Name)**
- **Thread (ThreadGroup group, Runnable target)**
- **Thread (ThreadGroup group, Runnable target, String Name)**
- **Thread (ThreadGroup group, Runnable target, String Name, long stacksize)**
- **Thread (ThreadGroup group, String Name)**

- **target : luồng cha**
- **name: tên gọi của luồng được tạo ra**

9.5.5- Hai loại luồng

- **Luồng Daemon**: luồng hệ thống, chạy ở mức nền (background- chạy ngầm), là những luồng cung cấp các dịch vụ cho các luồng khác. Các quá trình trong JVM chỉ tồn tại khi các luồng daemon tồn tại. JVM có ít nhất 1 luồng daemon là luồng "garbage collection"
- **Luồng do user tạo ra**.

9.5.6- Methods thông dụng của lớp Thread

Method	Mục đích
static int enumerate (Thread [] t)	Sao chép các luồng đang tích cực vào 1 mảng từ các nhóm luồng và nhóm con của chúng.
final String getName()	Lấy tên của luồng
final boolean isAlive()	Kiểm tra luồng còn sống hay không?
final void setName(String NewName)	Đặt tên mới cho luồng
final void join () throws interruptedException	Chờ luồng này chết
public final boolean isDaemon()	Kiểm tra xem luồng này có phải là luồng daemon
void setDaemon(boolean on)	on=true : luồng là daemon on=false : luồng của user

9.5.6- Methods thông dụng của lớp Thread

Method	Mục đích
static void sleep (long milisec)	Trì hoãn luồng 1 thời gian
void start()	thực thi luồng
static int activeCount()	Đếm số luồng đang tích cực
static void yield()	Tạm dừng luồng hiện hành để các luồng khác tiếp tục thực thi

Minh họa tạo luồng với lớp Thread

```
// Thread1.java – Minh họa tạo luồng với lớp Thread
class Thread1 extends Thread
{ public void Create() // tạo luồng con của luồng cha hiện hành
  { Thread t = new Thread (this);
    t.start();
  }
  public void run() // override hành vi run()
   { System.out.println("This is child thread.");
   }
  public static void main (String args[])
  { System.out.println("This is main thread.");
    Thread1 t= new Thread1();
     t.Create(); // tạo luồng con
  }
}
```

Kết qủa:

```
This is main thread
This is child thread
Pres any key to continue
```

Minh họa tạo luồng với lớp interface Runnable

```
class Thread2 implements Runnable
{ public void Create()
  { Thread t = new Thread(this);
    t.start();
  }
  public void run() // implement the run () method
   { System.out.println("This is child thread."); }
  public static void main (String args[])
  { System.out.println("This is main thread.");
    Thread2 t= new Thread2();   t.Create();
  }
}
```

Khi xây dựng luồng bằng interface Runable, phải khai báo 1 đối tượng Thread và gọi hành vi start() để hành vi này gọi run()

Kết qủa:
This is main thread
This is child thread
Pres any key to continue

MInh họa một số methods của Thread

```
class Thread3 implements Runnable // Thread3.java
{   Thread t1,t2;
    Thread3()
     { t1= new Thread(this);
       t1.start();          // t1 is an user-defined thread
       t2= new Thread(this);
       t2.setDaemon(true);   // t2 is a daemon thread
     }
    public void run()
   { int n= Thread.activeCount(); // Đếm số luồng đang tích cực trong JVM
     System.out.println("Number of active threads:" + n);
     String t1Name = t1.getName(); // lấy tên của 2 luồng
     String t2Name = t2.getName();
     System.out.println("Name of t1 thread:" + t1Name);
     System.out.println("Name of t2 thread:" + t2Name);
     System.out.println("Is t1 thread a daemon? :" + t1.isDaemon());
     System.out.println("Is t2 thread a daemon? :" + t2.isDaemon());
     System.out.println("Is t1 thread alive? :" + t1.isAlive());
     System.out.println("Is t2 thread alive? :" + t2.isAlive());
   }
  public static void main (String args[])
  { System.out.println("This is main thread.");
    Thread3 t= new Thread3();
  }
}
```

Kết qủa là 4 luồng tích cực : luồng gom rác, luồng mẹ và 2 luồng t1,t2.

Kết quả

```
This is main thread.
Number of active threads:4
Name of t1 thread:Thread-1
Name of t2 thread:Thread-2
Is t1 thread a daemon? :false
Is t2 thread a daemon? :true
Is t1 thread alive? :true
Is t2 thread alive? :false
Press any key to continue...
```

Tên mặc định của luồng là Thread-n, với n là số thứ tự khi luồng được tạo ra

Minh họa về trạng thaí của luồng

```
class Thread4 extends Thread// // Thread4.java
{ Thread t;
  Thread4()
  { t= new Thread(this);
    System.out.println("t thread is born and ready.");    t.start();
  }
  public void run()
   { try
      { System.out.println("t thread is running.");
        t.sleep(5000);
        System.out.println("t is awaked and running again after 5 secs.");
      }
    catch( InterruptedException e)
      { System.out.println("thread is interrupted!");
      }
   }
  public static void main (String args[])
  {   new Thread4();
  }
}
```

Dòng này xuất sau 5 giây so với dòng trước

```
t thread is born and ready.
t thread is running.
t is awaked and running again after 5 secs.
Press any key to continue...
```

9.6- Độ ưu tiên của luồng

- Các luồng cùng chia sẻ thời gian của CPU → Luồng ở cuối hàng đợi sẽ lâu được CPU thực thi → Có nhu cầu thay đổi độ ưu tiên của luồng. Java cung cấp 3 hằng mô tả độ ưu tiên của 1 luồng (các độ ưu tiên khác dùng 1 số nguyên từ 1.. 10).
- **NORM_PRIORITY : mang trị 5**
- **MAX_PRIORITY : mang trị 10**
- **MIN_PRIORITY : mang trị 1**
- Độ ưu tiên mặc định của 1 luồng là NORMAL_PRIORITY. Luồng con có cùng độ ưu tiên với luồng cha (do đặc điểm thừa kế).

Thao tác với độ ưu tiên của luồng

- **final void setPriority(int newPriority)**
- **final int   getPriority()**

Như vậy, các điều kiện để 1 luồng không được thực thi:

- Luồng không có được độ ưu tiên cao nhất để dành lấy thời gian của CPU.
- Luồng bị cưỡng bức ngủ bằng hành vi sleep().
- Luồng bị chờ do hành vi wait().
- Luồng bị tường minh nhận hành vi yield().
- Luồng bị khóa vì đang chờ I/O

Minh họa về độ ưu tiên của luồng

```
class Thread5 extends Thread// Thread4.java
{
  public void run()
   { Thread Child = new Thread(this);
     Child.setName("Child thread");
     System.out.println("Name of current thread:" +
                    Thread.currentThread().getName());
     System.out.println("Piority of current thread:"
      + Thread.currentThread().getPriority());
     System.out.println("Name of child:" +
      Child.getName());
     System.out.println("Priority of child:" +
      Child.getPriority());
  }
  public static void main (String args[])
  { Thread5 t = new Thread5();
    t.start();
    t.setName("Parent thread");
  }
}
```

Name of current thread:Parent thread
Piority of current thread:5
Name of child:Child thread
Priority of child:5
Press any key to continue...

Nếu trong main(), thêm dòng t.setPriority (8); trước dòng t.start(); ta có kết qủa là 8 thay vì 5

9.7- Đồng bộ các luồng

- **Tình huống:** Có hai luồng t1, t2 cùng truy xuất 1 đối tượng dữ liệu là biến m. t1 muốn đọc biến m còn t2 muốn ghi biến m. → dữ liệu mà t1 đọc được có thể không nhất quán.

→ Nếu để cho t2 ghi m trước rồi t1 đọc sau thì t1 đọc được dữ liệu nhất quán tại thời điểm đó.

→ Cần có cơ chế để chỉ cho phép 1 luồng được truy xuất dữ liệu chung (shared data) tại 1 thời điểm.

- → Kỹ thuật này gọi là **"ĐỒNG BỘ HÓA – SYNCHRONIZATION"**

Kỹ thuật cơ bản về đồng bộ hóa

- Tạo ra 1 đối tượng quản lý sự đồng bộ của 1 thao tác dữ liệu của các luồng bằng cách thực thi hộ một tác vụ của các luồng mỗi lần chỉ cho 1 luồng bằng từ khóa **synchronized**
- Mọi đối tượng luồng đều được đối tượng quản lý này quan sát (MONITOR) bằng cách cho mọi đối tượng luồng có dữ liệu là đối tượng monitor này và thay vì phải làm 1 tác vụ thì nhờ đối tượng monitor làm hộ hoặc là 1 biến boolean để nhận biết đã có 1 luồng đang thực thi.
- → **Luồng đang được chiếu cố gọi là luồng đang có monitor**

Minh họa về đồng bộ các luồng bằng MONITOR

Chương trình sau sẽ xuất 3 số 10,11, 12 ra màn hình, mỗi số được 1 luồng thực thi.

```
// Monitor1.java – Lớp làm nhiệm vụ xuất hộ 1 số num
class Monitor1
{ synchronized void Display (int num)
  {  System.out.println("Output " + num + " - done.");
     try
     { Thread.sleep(500); // current thread sleep 1/2 sec
     }
     catch (InterruptedException e)
     { System.out.println ("Thread is interrupted!");
     }
  }
}
```

Từ khóa synchronized khai báo có quản lý việc đồng bộ các luồng

Minh họa về đồng bộ các luồng bằng MONITOR

```
class OutNum implements Runnable // luồng
{  Monitor1 monitor; // Luồng có dữ liệu là monitor
    int number;  // dữ liệu cần xuất
    Thread t;
   // hành vi xuất n với Monitor1 có tên moni
    OutNum(Monitor1 moni, int n )
    { monitor= moni;
       number = n;
       t = new Thread(this);
       t.start();
    }
  // khi luồng chạy, số number được xuất bởi monitor
   public void run() { monitor.Display(number);  }
}
```

Minh họa về đồng bộ các luồng bằng MONITOR

```
class Synchro // lớp của chương trình chính
{  public static void main (String args[])
   {  Monitor1 monitor = new Monitor1();
      int num = 10;
      OutNum Obj1 = new OutNum(monitor,num++);
      OutNum Obj2 = new OutNum(monitor,num++);
      OutNum Obj3 = new OutNum(monitor,num++);
      // wait for 3 threads to end
      try
      { Obj1.t.join();
        Obj2.t.join();
        Obj3.t.join();
      }
      catch(InterruptedException e)
      { System.out.println ("Thread was interrupted!");  }
     }
}
```

3 luồng có 3 trị khác nhau là 10,11, 12 nhưng có chung 1 monitor
Ba luồng cùng đơ

```
Output 10 - done.
Output 11 - done.
Output 12 - done.
Press any key to continue....
```

Kỹ thuật đồng bộ luồng theo khối

- Đồng bộ một khối tác vụ.
- Người lập trình có thể không muốn dùng các synchronized method để đồng bộ truy xuất đến đối tượng.
- Các lớp được cung cấp bởi các thư viện hay do “một ai đó” cung cấp – lớp đã xây dựng- nên không thể thêm từ khóa **synchonized** vào được các method này.

Kỹ thuật đồng bộ luồng theo khối

- Cú pháp đồng bộ khối

synchronized (Object)

{ <các lời gọi methods của Object

cần phải được đồng bộ>

}

- **Buộc phải có { } dù chỉ có 1 phát biểu**

Minh họa đồng bộ khối

Chương trình sau viết lại chương trình trước, bỏ qua từ khóa synchronized trong lớp Monitor1 (ở đây gọi là lớp Monitor2)

```
class Monitor2  // Monitor2.java
{ void Display (int num)
  {  System.out.println("Output " + num + " - done.");
     try
     { Thread.sleep(500); // current thread sleap 1/2 sec
     }
     catch (InterruptedException e)
     { System.out.println ("Thread is interrupted!");
     }
  }
}
```

Minh họa đồng bộ khối

```
class Synchro
{  public static void main (String args[])
   {  Monitor2 monitor = new Monitor2();
      int num = 10;
      OutNum Obj1 = new OutNum(monitor,num++);
      OutNum Obj2 = new OutNum(monitor,num++);
      OutNum Obj3 = new OutNum(monitor,num++);
      // wait for 3 threads to end
      try
      { Obj1.t.join();
        Obj2.t.join();
        Obj3.t.join();
      }
      catch(InterruptedException e)
      { System.out.println ("Thread was interrupted!");
      }
   }
}
```

Minh họa đồng bộ khối

```
class OutNum implements Runnable
{ Monitor2 monitor;
  int number;
  Thread t;
  OutNum(Monitor2 moni, int n )
  { monitor= moni;
    number = n;
    t = new Thread(this);
    t.start();
  }
  public void run()
  { synchronized (monitor)
      { monitor.Display(number);
      }
  }
}
```

Minh họa đồng bộ khối

9.8- Deadlock

- Deadlock – tình huống bế tắc, đóng băng- xẩy ra khi các luồng chờ tài nguyên (monitor) của nhau hình thành một chu trình. Deadlock hiếm khi xẩy ra.
- Minh họa: DeadlockDemo.java

Giải thích DeadlockDemo class

- 1 ứng dụng có 2 luồng :Luồng t1 trong đối tượng d1, luồng t2 trong đối tượng d2
- Monitor của t1 lại là d2 và monitor của t2 lại là d1 (tréo nhau).
- Cả 2 luồng cùng gọi hành vi **synchronized run()** và cùng ngủ 300 mili giây.Vì chia sẻ thời gian CPU nên t1 ngủ trước và t2 ngủ sau (xem phương thức run()).
- Khi t1 thức dậy (wake up), phương thức Synchro() của đối tượng monitor của d2 (chứa luồng t2) được gọi nhưng luồng t2 đang ngủ nên phương thức này chưa thể thực thi.
- Khi t2 thức dậy (wake up), phương thức Synchro() của đối tượng monitor của d1 (chứa luồng t1) được gọi nhưng luồng t1 cũng đang ngủ nên phương thức này chưa thể thực thi.
- Như vậy chương trình sẽ đóng băng (blocked) không làm gì được nữa.

9.9- Cơ chế chờ- nhận biết

- Java cung cấp sẵn một cơ chế giao tiếp liên qúa trình (inter-process mechanism) để các luồng có thể gọi nhau (**yêu cầu nhau**) sử dụng các final methods của lớp Object: *wait() , notify() , notifyAll()*. Như vậy mọi lớp đều có thể sử dụng chúng và các phương thức này *chỉ có thể được gọi trong các synchronized methods*.

Cơ chế wait-notify

- **Phương thức wait()** : Luồng nhả monitor để đi vào trạng thái ***sleep*** cho đến khi 1 luồng khác vào cùng monitor và gọi phương thức ***notify***.
- **Phương thức notify()** : Luồng thức dậy (wake up) và nhận biết (notify) rằng luồng thứ nhất đã gọi wait().
- **Phương thức notifyAll()** : Đánh thức tất cả các luồng đang ngủ để chúng biết rằng luồng hiện hành đã gọi phương thức wait(). Khi tất cả các luồng đang ngủ thức dậy, luồng có ưu tiên cao nhất sẽ nắm giữ monitor và thực thi.

Chú ý đối với phương thức wait

- Luồng gọi phương thức wait() sẽ nhả CPU, thôi không dùng CPU nữa.
- Luồng gọi phương thức wait() sẽ nhả monitor, thôi không khóa (lock) monitor nữa.
- Luồng gọi phương thức wait() sẽ được đưa vào danh sách hàng đợi monitor (monitor waiting pool)

Chú ý đối với phương thức notify

- Một luồng đang ngủ được đưa ra khỏi monitor waiting pool và đi vào trạng thái ***ready***.
- Luồng vừa thức giấc (notify) phải giành lại monitor và khóa monitor lại không cho luồng khác chiếm để luồng này được thực thi.

Chú ý đối với phương thức notifyAll

- Luồng đang thực thi cảnh báo cho tất cả các luồng đang ngủ rằng “Tôi đi ngủ đây, các bạn dậy để làm việc”.
- Luồng ở đầu danh sách monitor waiting pool được vào trạng thái ready

Bài toán 5 triết gia ăn tối với 5 chiếc đũa

9.10- Tóm tắt

- Luồng là biện pháp chia công việc thành các đơn vị cụ thể (concrete) nên có thể được dùng để thay thế vòng lặp.
- Lập trình đa luồng làm tăng hiệu suất CPU trên những hệ thống "bận rộn". Tuy nhiên hiệu suất của từng ứng dụng lại bị giảm đang kể (chậm ba bốn lần do các tác vụ đồng bộ hóa), qúa trình biên dịch cũng chậm vì trình biên dịch phải tính toán cơ chế quản lý các luồng. Do vậy trong caùc ứng dụng đòi hỏi yếu tố hiệu suất thời gian là quan trọng, nên tránh sử dụng kỹ thuật đồng bộ hóa. → Nhiều lập trình viên không thích lập trình đa luồng mà chỉ dùng lập trình lập trình đơn luồng để tăng hiệu suất của ứng dụng.
- Java cung cấp kỹ thuật lập trình đa luồng bằng lớp **Thread** và interface **Runnable**.
- Khi 1 ứng dụng Java thực thi, có 1 luồng đang chạy đó là luồng chính (main thread). Luồng chính rất quan trọng vì (1) Đây là luồng có thể sinh ra các luồng con, (2) Quản lý việc kết thúc ứng dụng vì luồng main chỉ kết thúc khi tất cả các luồng con của nó đã kết thúc.

Tóm tắt

- Hiện thực 1 luồng bằng 1 trong 2 cách:
- Hiện thực 1 lớp con của lớp **Thread**, override phương thức **run()** của lớp này.
- Khai báo lớp mà ta xây dựng là implement của interface **Runnable** và định nghĩa phương thức **run()**.
- Mỗi java thread có 1 độ ưu tiên từ 1 (MIN) đến 10 (MAX) với 5 là trị mặc định. JVM không bao giờ thay đổi độ ưu tiên của luồng.
- Có 8 contructor của lớp Thread nhưng 2 constructor thường dùng: **Thread()** và **Thread(String TênLuồng)**, **Thread(ĐốiTượngChứa)**.
- Các phương thức *Thread.suspend(), Thread.resume(), Thread.stop()* không còn được dùng nữa kể từ Java 2.
- Luồng *daemon* là luồng chạy ngầm nhằm cung cấp dịch vụ cho các luồng khác. Nếu muốn 1 luồng là daemon, hãy dùng ***public final void setDeamon (boolean)*** và kiểm tra 1 luồng có là daemon hay không, hãy dùng ***public final boolean isDaemon()***.

Tóm tắt

- Dữ liệu có thể bị mất nhất quán(hư hỏng) khi có 2 luồng cùng truy xuất dữ liệu tại cùng 1 thời điểm.
- Đồng bộ là 1quá trình bảo đảm tài nguyên (dữ liệu, file,…) chỉ được 1 luồng sử dụng tại 1 thời điểm. Tuy nhiên, chi phí cho việc này lại làm giảm hiệu suất thời gian của ứng dụng **xuống 3, 4 lần**.
- Phương thức ***wait()*** sẽ làm 1 luồng đi vào trạng thaí ngủ.
- Phương thức ***notify()*** sẽ đánh thức luồng thứ nhất trong danh sách luồng đang chờ trên cùng 1 đối tượng monitor.
- Phương thức ***notifyAll()*** sẽ đánh thức tất cả các luồng trong danh sách luồng đang chờ trên cùng 1 đối tượng monitor.
- Deadlock xẩy ra khi 2 luồng có sự phụ thuộc vòng trên một cặp đối tượng quản lý việc đồng bộ (synchronized object).

Xin cám ơn