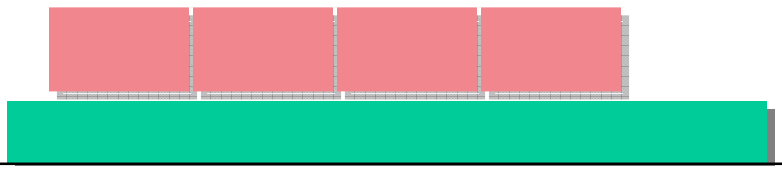
Bài giảng

LẬP TRÌNH JAVA

Lê Đình Thanh
Bộ môn Mạng và Truyền thông Máy tính
Khoa Công nghệ Thông tin
Trường Đại học Công nghệ, ĐHQGHN

Bài 1

Cơ bản về Java



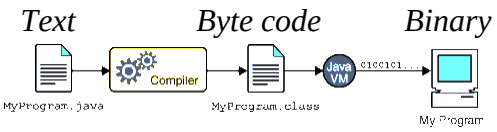
Nội dung

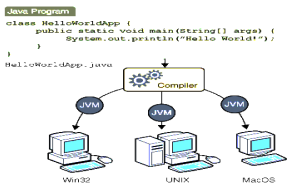
- Lập trình với Java
- Môi trường
- Các phiên bản
- Kiểu dữ liệu
- Biến, mảng
- Toán tử
- Khối lệnh

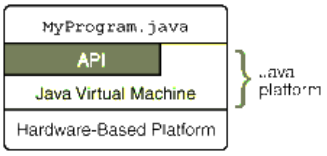
Lập trình với Java

Java là ngôn ngữ lập trình hướng đối tượng *được phát triển bởi Sun, nay thuộc Oracle*

- Mạnh mẽ
- Phân tán
- Đa luồng
- Độc lập nền
- Khả chuyển
- An ninh cao
- Hiệu năng cao



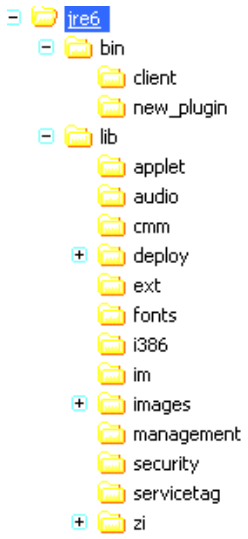




Môi trường

- **JRE** (Java Runtime Environment) bao gồm
  - Máy Java ảo (JVM – Java Virtual Machine),
  - Thư viện các lớp thực thi
  - Bộ khởi động ứng dụng Java

  cần thiết để chạy ứng dụng viết bằng java.



Môi trường

- **JDK** (Java Development Kit) bao gồm
  - Các công cụ để phát triển ứng dụng Java như công cụ biên dịch, gỡ lỗi, chạy ứng dụng hay công cụ viết tài liệu, công cụ triệu gọi từ xa, …
  - JRE
  - Thư viện cần cho các công cụ phát triển
  - Các tệp tiêu đề C (.h) để lập trình mã native
  - Các chương trình mẫu sử dụng Java API
  - Các applet và ứng dụng mẫu
  - Mã nguồn của JDK

Môi trường

- **IDE** (Integrated Development Environment)
  - Là môi trường tích hợp cho phép lập trình, dịch, gỡ lỗi, kiểm thử các ứng dụng
  - Liên kết sử dụng JDK
  - Cung cấp trình soạn thảo, công cụ thiết kế trực quan, công cụ quản lý mã nguồn, tích hợp CSDL, …

Các phiên bản

Java SE: cho phát triển ứng dụng Desktop
Java EE: cho phát triển ứng dụng phía Server
Java ME: cho phát triển ứng dụng trên thiết bị di động, cầm tay
JavaFX: cho phát triển ứng dụng web

Kiểu dữ liệu

- **byte**: nguyên có dấu, 8-bit
- **short**: nguyên có dấu, 16-bit
- **int**: nguyên có dấu, 32-bit
- **long**: nguyên có dấu, 64-bit
- **float**: thực, 32-bit
- **fouble**: thực, 64-bit
- **char**: 16-bit Unicode
- **boolean**: true/false
- **String**: xâu ký tự (lớp)

Biến

- **Biến thể hiện**: Trường không tĩnh
- **Biến lớp**: Trường tĩnh (static)
- **Biến cục bộ**: Được khai báo, sử dụng trong phương thức
- **Tham số**: Đầu vào của phương thức

- Tên: bao gồm chữ cái, số, _, $, không bắt đầu bằng số, phân biệt chữ hoa-thường.
- Khai báo: **Kiểu_dữ_liệu Tên_biến** [= Giá_trị] [, …] **;**

Mảng

- **Khai báo**:

  Kiểu[] tên_mảng;

  Kiểu[] tên_mảng = new Kiểu[số_phần_tử];

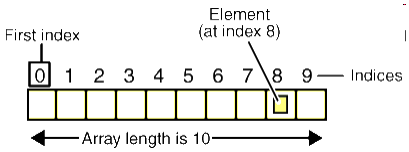
- **Truy cập phần tử**:

  Tên_mảng[chỉ_số];

- Ví dụ:

  int[] a, float b[] = float[100];

  a = int[500];



Toán tử

Toán tử	**Cú pháp**
Hậu tố	*expr*++ *expr*--
Một ngôi	++*expr* --*expr* +*expr* -*expr* ~ !
Nhân, chia, dư	* / %
Cộng trừ	+ -
Dịch bit	<< >> >>>
Quan hệ	< > <= >= instanceof
Bằng nhau	== !=
Và bít	&
Hoặc loại trừ (XOR) bít	^
Hoặc bít	|
Và logic	&&
Hoặc logic	||
Ba ngôi	? :
Gán	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Ghi chú

- // ghi chú trên một dòng
- /* ghi chú
- trên
- nhiều dòng */

Khối lệnh

- Tập các lệnh được đặt giữa { và }
- Ví dụ:

Cấu trúc điều khiển

- Tuần tự: lệnh viết trước được thực hiện trước
- Rẽ nhánh:
  - if
  - switch
- Lặp:
  - for
  - while, do-while

if

if (điều_kiện) lệnh/khối_lệnh
[else lệnh_khác/khối_lệnh_khác]

Ví dụ:

```
if (testscore >= 90) { grade = 'A'; }
else if (testscore >= 80) { grade = 'B'; }
else if (testscore >= 70) { grade = 'C'; }
else if (testscore >= 60) { grade = 'D'; }
else { grade = 'F'; }
```

switch

switch (biểu_thức)
{
    case giá_trị_1: các lệnh 1
    case giá_trị_2: các lệnh 2
    …
    [default: các lệnh]
}

- Biểu thức và các giá trị có kiểu byte, short, int hoặc char
- Lệnh break sẽ đưa chương trình thoát khỏi switch

switch

```
switch (month) {
    case 1:
    case 3:
    case 5:
    case 7:
    case 8:
    case 10:
    case 12: numDays = 31; break;
    case 4:
    case 6:
    case 9:
    case 11: numDays = 30; break;
    case 2: if ( ((year % 4 == 0) && !(year % 100 == 0)) || (year % 400 == 0) )
              numDays = 29; else numDays = 28; break;
    default: System.out.println("Invalid month."); break;
}
```

while

while (điều_kiện) lệnh/khối_lệnh

Ví dụ:

```
int count = 1;
while (count < 11)
{
        System.out.println("Count is: " +
    count);
        count++;
}
```

Chú ý: các lệnh của while cần đưa điều_kiện dần chuyển thành false nếu không vòng lặp sẽ không dừng.

do-while

do lệnh/khối_lệnh while (điều_kiện);

Ví dụ:

```
int count = 1;
    do
    {
        System.out.println("Count is: " + count);
        count++;
    } while (count <= 11);
```

Chú ý: các lệnh của do-while cần đưa điều_kiện dần chuyển thành false nếu không vòng lặp sẽ không dừng.

for

for (khởi_tạo; điều_kiện; lệnh) lệnh/khối lệnh

```
for(int i=1; i<11; i++)
{
      System.out.println("Count is: " + i);
}
```

Chú ý: lệnh sau điều_kiện thường được dùng để đưa điều_kiện dần chuyển thành false

Khởi_tạo hay lệnh sau điều_kiện có thể là dãy các lệnh cách nhau bởi dấu phẩy (,)

Thay đổi hoạt động lặp

Trong các cấu trúc lặp (for, while, do-while), sử dụng

break để kết thúc lặp

continue để bỏ qua vòng lặp hiện tại

Ví dụ:

```
int i; boolean foundIt = false;
for (i = 0; i < arrayOfInts.length; i++) {
if (arrayOfInts[i] == searchfor) { foundIt = true; break; } }

String searchMe = "peter piper picked a peck of pickled peppers";
int max = searchMe.length(); int numPs = 0;
for (int i = 0; i < max; i++) {if (searchMe.charAt(i) != 'p') continue; numPs++; }
      System.out.println("Found " + numPs + " p's in the string.");
```

Một số ví dụ

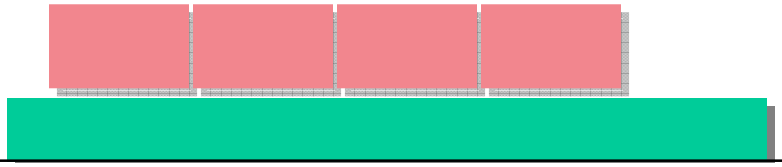
- **Xem một số chương trình mẫu**

Thực hành

- **Cài đặt JDK, NetBeans IDE**

Tiếp theo

Lớp và đối tượng



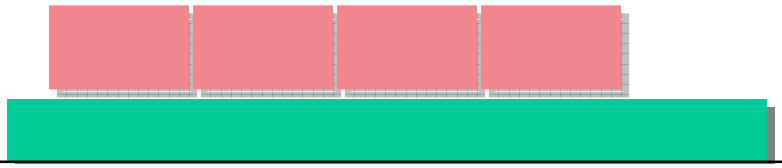
Bài giảng

LẬP TRÌNH JAVA

Lê Đình Thanh
Bộ môn Mạng và Truyền thông Máy tính
Khoa Công nghệ Thông tin
Trường Đại học Công nghệ, ĐHQGHN

Bài 2

Lớp và đối tượng



Nội dung

- Khai báo lớp
- Khai báo trường
- Khai báo phương thức
  - Chồng phương thức
  - Phương thức tạo
  - Tham số
- Khai báo đối tượng
- Sử dụng đối tượng
- Đối tượng this
- Lớp lồng nhau
- Gói lớp
- Phạm vi truy cập
- Kiểu liệt kê

Khai báo lớp

```
[public/private] class TênLớp {
        //các trường (field),
        //phương thức tạo (constructor)
        //phương thức khác (method)
}
```

Khai báo trường

Phạm_vi_truy_cập Kiểu tên_trường;

```
public class Bicycle {
    private int cadence;
    private int gear;
    private int speed;
}
```

Phạm vi truy cập có thể là: public/private/…

Khai báo phương thức

Phạm_vi_truy_cập Kiểu tên_pt (danh_sách_tham_số)
{
        các lệnh
        return biểu_thức nếu Kiểu khác void
}

```
public class Bicycle {
       public int speedUp(int increment)
       { speed += increment; return speed; }
    }
```

Phạm vi truy cập có thể là: public/private/…

Chồng phương thức

Một lớp có thể có nhiều phương thức cùng (chồng - overload) tên

- Các phương thức chồng nhau phải khác nhau về số lượng/kiểu tham số.

```
public class DataArtist {
    public void draw(String s) { ... }
    public void draw(int i) { ... }
    public void draw(double f) { ... }
    public void draw(int i, double f) { ... }
}
```


Phương thức tạo

Phương thức tạo là một phương thức đặc biệt

- Có tên trùng tên lớp
- Không có kiểu (ngầm định kiểu là lớp)
- Được gọi ngầm định khi đối tượng của lớp được tạo

```
public class Bicycle {
        public int cadence;
        public int gear;
        public int speed;
        public Bicycle(int sCadence, int sSpeed, int sGear)
        { gear = sGear; cadence = sCadence; speed = sSpeed; }
```

Tham số của phương thức

Một phương thức có thể có 0, 1 hoặc nhiều tham số
Tham số của phương thức có thể là biến hoặc mảng

```
public void moveCircle(Circle circle, int deltaX, int deltaY)
{…}

public Polygon polygonFrom(Point [] corners) {…}
public Polygon polygonFrom(Point... corners) {
    int numberOfSides = corners.length;
    …
}
```


Khai báo đối tượng

TênLớp tên_đối_tượng = new Phương_thức_tạo(tham_số)

```
Point originOne = new Point(23, 94);
Rectangle rectOne = new Rectangle(originOne, 100, 200);
Rectangle rectTwo;                    //rectTwo == null
rectTwo = new Rectangle(50, 100);   //rectTwo != null
```

Chú ý: Khi chưa gọi phương thức tạo và gán kết quả cho đối tượng, đối tượng chỉ mới có tên và đang bằng null

Sử dụng đối tượng

Với trường và phương thức thuộc phạm vi truy cập

- objectReference.fieldName;
- objectReference.methodName(argumentList);

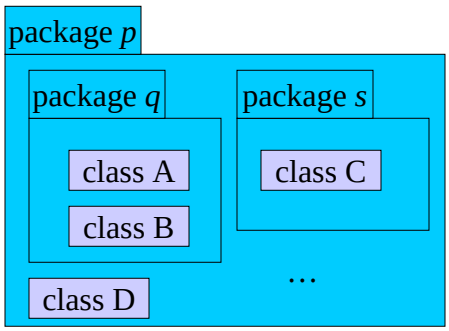
```
Bicycle b = new Bicycle(12, 5, 6);
    b.speedUp(5) ;
```

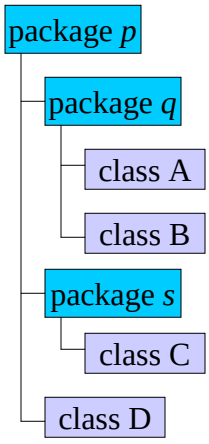

this

Trong phương thức của lớp, từ khóa this có nghĩa "đối tượng hiện tại" hoặc phương thức tạo

```
public Rectangle() { this(0, 0, 0, 0); }
public Rectangle(int width, int height) { this(0, 0, width,
    height); }
public Rectangle(int x, int y, int width, int height) {
    this.x = x; this.y = y; this.width = width; this.height =
    height;
}
```

Nhóm các lớp theo gói





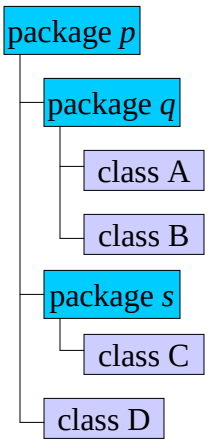
Các lớp được tổ chức quản lý thành các gói (package)
Giữa các gói có quan hệ "chứa": Gói *p* chứa gói *q*
Giữa gói và lớp có quan hệ "chứa": Gói *p* chứa lớp C
Quan hệ "chứa" giữa các gói và giữa gói với lớp tạo thành cây

Tên gói và tên lớp

- Tên gói = Tên_gói_cha.Tên_gói_con
- Tên lớp = Tên_gói.Tên_lớp

p.q *//gói q*
p.D *//lớp D*
p.q.A *//lớp A*

Khi không có sự nhập nhằng, có thể không cần dùng Tên_gói. trước tên lớp
D tương đương p.D

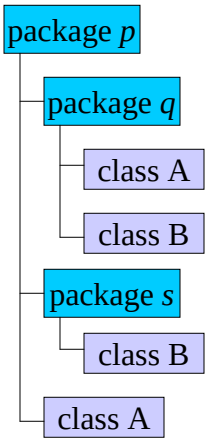


Tên gói và tên lớp

Các gói cùng gói cha, các lớp cùng gói không được trùng tên nhau

Các gói khác gói cha, các lớp khác gói có thể trùng tên

p.q.A khác p.A



Sử dụng lớp trong gói

- import TenGoi.TenLop;
- import TenGoi.*; //tất cả các lớp trong gói

- TenGoi.TenLop tenDoituong = new TenGoi.TenLop();

Phạm vi truy cập/tính khả kiến

- Các lớp và những trường, phương thức của lớp được bảo vệ trước truy cập trái phép.
- Phạm vi truy cập lớp:
  - **public**: có thể được truy cập bởi lớp bất kỳ khác
  - **private**: chỉ có thể truy cập bởi các lớp cùng gói (mặc định)
- Phạm vi truy cập phương thức và trường:
  - **public**: có thể truy cập từ lớp bất kỳ thuộc phạm vi truy cập của lớp chứa
  - **private**: không được truy cập từ bất kỳ lớp nào khác
  - **protected**: có thể truy cập bởi các lớp cùng gói và các lớp con

Các lớp lồng nhau

- Một lớp có thể được chứa trong/thành viên của một lớp khác
- Lớp trong, với tư cách thành viên của lớp chứa, có thể truy cập mọi thành phần khác của lớp chứa, được che bởi lớp chứa
- Lớp trong có thể tĩnh (static) hoặc không
  - Tĩnh: OuterClass.StaticNestedClass nestedObject = new OuterClass.StaticNestedClass();
  - Không tĩnh: OuterClass.InnerClass innerObject = outerObject.new InnerClass();

```
class OuterClass {
    class InnerClass { ... }
}
```

Các lớp lồng nhau-Ví dụ

```
public class EvenArray {
        private final static int SIZE = 15;
        private int[] arrayOfInts = new int[SIZE];
        public EvenArray() { for (int i = 0; i < SIZE; i++) { arrayOfInts[i] = i; } }
        public void printEven() {
                InnerEvenIterator iterator = this.new InnerEvenIterator();
                while (iterator.hasNext()) { System.out.println(iterator.getNext() + " "); }
         }
        private class InnerEvenIterator {
                private int next = 0;
                public boolean hasNext() {return (next <= SIZE - 1); }
                public int getNext() {
                        int retValue = arrayOfInts[next];
                        next += 2; return retValue;  }
        }
public static void main(String s[]) {
        EvenArray ds = new EvenArray();
        ds.printEven(); }
}
```


Kiểu liệt kê

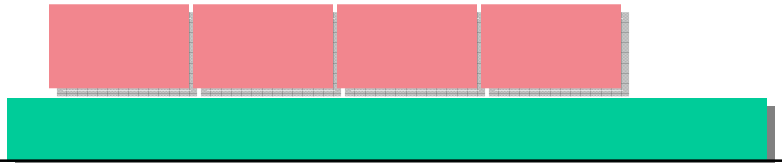
- Liệt kê là một kiểu dữ liệu chỉ bao gồm hữu hạn các giá trị hằng

```
public enum Day { SUNDAY, MONDAY, TUESDAY,
      WEDNESDAY, THURSDAY, FRIDAY,
      SATURDAY }
Day day;
switch (day) {
    case MONDAY: System.out.println("Mondays are
      bad."); break;   …}
```

- Tất cả kiểu liệt kê được ngầm định kế thừa từ java.lang.Enum.

Tiếp theo

Giao diện và kế thừa



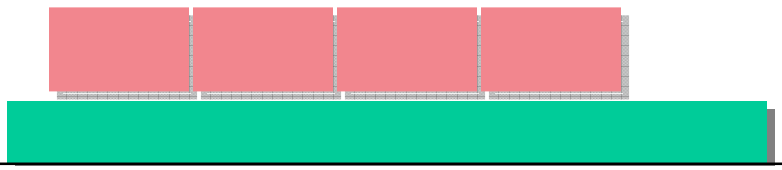
Bài giảng

LẬP TRÌNH JAVA

Lê Đình Thanh
Bộ môn Mạng và Truyền thông Máy tính
Khoa Công nghệ Thông tin
Trường Đại học Công nghệ, ĐHQGHN

Bài 3

Giao diện, kế thừa



Nội dung

- Giao diện
- Kế thừa
- Đa hình
- Lớp ảo, phương thức ảo

Giao diện

- Giao diện là phương tiện để giao tiếp
- Để "lắp ghép" các môđun phần mềm được với nhau, các môđun phải "khớp" với nhau về giao diện.
- Tại sao cần giao diện?
  - Mỗi nhóm phát triển một số mô đun, không phải quan tâm đến mã bên trong của các mô đun do nhóm khác phát triển, chỉ cần các nhóm thông nhất với nhau về giao diện
  - Phát triển các môđun xử lý chuyên nghiệp và bán cho các nhà phát triển phần mềm, ví dụ thư viện đồ họa.

Giao diện trong Java

- Được định nghĩa tương tự lớp, nhưng chỉ bao gồm các hằng số và chữ ký phương thức. KHÔNG có thân phương thức.
- Không có thể hiện.
- Chỉ được cài đặt và mở rộng.

```
public interface OperateCar {
    // khai báo hằng, nếu có
    // chữ ký các phương thức
    int turn(Direction direction, double radius, double
      startSpeed, double endSpeed);
    int signalTurn(Direction direction, boolean signalOn);
}
```


Cài đặt giao diện

- Một lớp có thể cài đặt nhiều giao diện

```
public class OperateBMW760i implements OperateCar {
    // cài đặt thân các phương thức
    int signalTurn(Direction direction, boolean signalOn)
      {
            //bật/tắt đèn trái
            //bật/tắt đèn phải
      }
    // các phương thức khác không cài đặt phương thức
      của giao diện
}
```

Mở rộng giao diện

- Một giao diện có thể mở rộng nhiều giao diện khác

```
public interface GroupedInterface extends Interface1,
      Interface2, Interface3 {
    double E = 2.718282;
    void doSomething (int i, double x);
    int doSomethingElse(String s);
}
```


Sử dụng giao diện

- Giao diện được sử dụng như một kiểu
- Các đối tượng gán cho biến giao diện phải thuộc lớp cài đặt giao diện

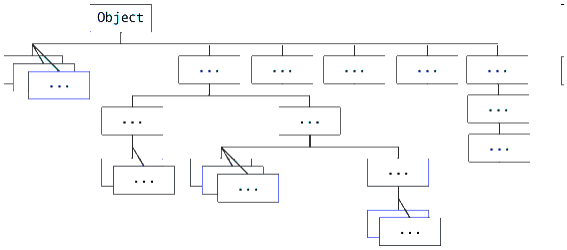
```
public interface I {}
public class A implements I {}
public class B {}
A a = new A();
B b = new B();
I i = (I) a; //đúng
I i2 = (I) b; //sai
```

Kế thừa

•Một lớp có thể được dẫn xuất (derived) từ lớp khác, và do vậy kế thừa (inherit) mọi trường và phương thức của lớp đó

- Lớp được kế thừa được gọi là lớp cha (superclass, parent class) hoặc lớp cơ sở (base class)
- Lớp kế thừa được gọi là lớp con (subclass, child class) hoặc lớp dẫn xuất (derived class) hoặc lớp mở rộng (extended class)

•Trong java, tất cả các lớp đều được dẫn xuất từ Object. Quan hệ kế thừa tạo thành cây hay cấu trúc phân cấp.



Kế thừa và tính khả kiến

- Kế thừa và tính khả kiến
  - Nếu lớp cha và lớp con ở trong cùng gói, lớp con kế thừa MỌI trường/phương thức public, protected và package-private của lớp cha
  - Nếu lớp cha và lớp con ở các gói khác nhau, lớp con kế thừa các trường/phương thức public và protected.
  - Lớp con không kế thừa các trường/phương thức private từ lớp cha

Chồng/che trường/phương thức

- Trường lớp con cùng tên trường của lớp cha sẽ **che** trường của lớp cha.
- Phương thức tĩnh của lớp con có chữ ký trùng phương thức tĩnh của lớp cha sẽ **che** phương thức lớp cha
- Phương thức không tĩnh của lớp con có chữ ký trùng phương thức không tĩnh của lớp cha sẽ **chồng** phương thức lớp cha

Chuyển kiểu trong sử dụng đối tượng

- Có thể sử dụng đối tượng lớp cha để tham chiếu đến một đối tượng lớp con (chuyển kiểu ngầm định)
  - Object obj = new MountainBike();
  - Bicycle byc = new MountainBike();
- Không được sử dụng đối tượng lớp con để tham chiếu đến đối tượng lớp cha
  - MountainBike mbike = new Bicycle(); //sai
- Nếu muốn sử dụng đối tượng lớp con để tham chiếu đến đối tượng lớp cha đang tham chiếu đến đối tượng lớp con, phải có lệnh chuyển kiểu rõ ràng
  - MountainBike mbike2 = byc; //lỗi
  - MountainBike mbike3 = (MountainBike) byc; //ok nếu byc đang tham chiếu đến đối tượng MountainBike.

Kiểm tra kiểu của một đối tượng

- if (obj instanceof Class)

```
if (obj instanceof MountainBike) {
    MountainBike myBike = (MountainBike) obj;
}
```


Đa hình

- Đa hình (polymorphism): Nhiều dạng/kiểu khác nhau của một đối tượng.
- Chồng phương thức, chuyển kiểu ngầm định là cơ sở của đa hình
- ff

```
Bicycle bike01, bike02, bike03;
bike01 = new Bicycle(20, 10, 1);
bike02 = new MountainBike(20, 10, 5, "Dual");
bike03 = new RoadBike(40, 20, 8, 23);
bike01.printDescription();
bike02.printDescription();
bike03.printDescription();
```

super – Đại diện của lớp cha

- Từ khóa **super** được sử dụng trong cài đặt lớp con để thực hiện các việc sau:
  - **Truy cập các phương thức của lớp cha đã bị chồng**
  - Truy cập các trường của lớp cha đã bị che
  - Thay cho phương thức tạo lớp cha

```
public class Superclass {
    public void printMethod() {
        System.out.println("Printed in Superclass."); } }

public class Subclass extends Superclass {
    public void printMethod() {
        super.printMethod();
        System.out.println("Printed in Subclass"); }}

public class SubSubClass extends Subclass {
    public void printMethod() {
     //Hỏi: Gọi printMethod() của Superclass  như thế nào? }
```


super – Đại diện của lớp cha

- Từ khóa super được sử dụng trong cài đặt lớp con để thực hiện các việc sau:
  - Truy cập các phương thức của lớp cha đã bị chồng
  - Truy cập các trường của lớp cha đã bị che
  - **Thay cho phương thức tạo lớp cha**

```
public MountainBike(int startHeight, int startCadence, int
      startSpeed, int startGear) {
    super(startCadence, startSpeed, startGear);
    seatHeight = startHeight;
}
```

Nếu phương thức tạo của lớp con không gọi phương thức tạo của lớp cha một cách rõ ràng thì phương thức tạo không đối super() của lớp cha được gọi ngầm định.

final – Ngăn việc kế thừa và chồng

- Để chỉ định một phương thức không thể bị chồng ở lớp con, sử dụng từ khóa final trước chữ ký phương thức
  - Cần thiết cho phương thức không nên thay đổi mã để đảm bảo tính nhất quán
- Để chỉ một lớp không thể kế thừa, sử dụng từ khóa final trước định nghĩa lớp
  - Cần để tạo các lớp bất biến, ví dụ String

```
class ChessAlgorithm {
    enum ChessPlayer { WHITE, BLACK }
    final ChessPlayer getFirstPlayer() {
        return ChessPlayer.WHITE;
    }
}
```


Lớp ảo, phương thức ảo

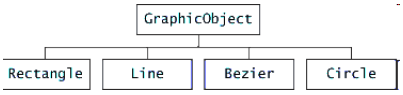
- Lớp ảo là lớp được định nghĩa với từ khóa abstract. Lớp ảo không có thể hiện.
  - Sử dụng làm lớp cha
- Phương thức ảo là phương thức được định nghĩa với từ khóa abstract và không có cài đặt
  - Cài đặt phương thức ảo được thực hiện ở lớp con với phương thức chồng
  - Nếu lớp con vẫn không cài đặt thì phương thức chồng cũng phải là ảo
  - Lớp chứa phương thức ảo phải là lớp ảo

Sử dụng lớp ảo

- Khi có nhiều lớp giống nhau ở một số trường và phương thức (cả chữ ký và thân), và có một số phương thức cùng chữ ký nhưng nhưng khác nhau về thân
  - Tạo lớp cơ sở ảo với
    - Các trường và phương thức chung được định nghĩa bình thường
    - Các phương thức giống nhau chữ ký nhưng khác thân là các phương thức ảo

Lớp ảo, phương thức ảo – Ví dụ

```
abstract class GraphicObject {
    int x, y;
    void moveTo(int newX, int newY) {x = newX; y = newY; }
    abstract void draw();
    abstract void resize();
}
class Circle extends GraphicObject {
    void draw() { ... }
    void resize() { ... }
}
class Rectangle extends GraphicObject {
    void draw() { ... }
    void resize() { ... }
}
```



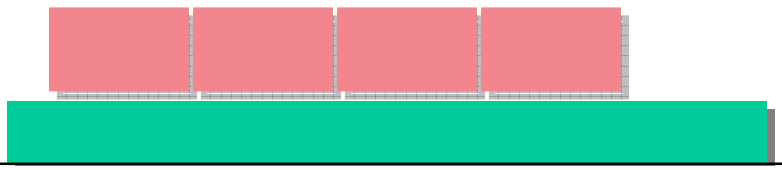
Câu hỏi kiểm tra

```
public class ClassA {
        public void methodOne(int i) { }
        public void methodTwo(int i) { }
        public static void methodThree(int i) { }
        public static void methodFour(int i) { }
}
public class ClassB extends ClassA {
        public static void methodOne(int i) { }
        public void methodTwo(int i) { }
        public void methodThree(int i) { }
        public static void methodFour(int i) { }
}
```

- Phương thức nào chồng phương thức lớp cha?
- Phương thức nào che phương thức lớp cha?
- Vấn đề gì với các phương thức khác?

Tiếp theo

Số và xâu



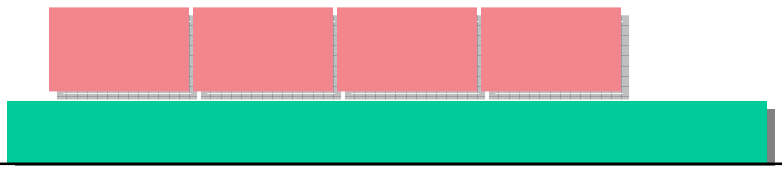
Bài giảng

LẬP TRÌNH JAVA

Lê Đình Thanh
Bộ môn Mạng và Truyền thông Máy tính
Khoa Công nghệ Thông tin
Trường Đại học Công nghệ, ĐHQGHN

Bài 4

Số và xâu



Nội dung

- Các lớp số
- Lớp Math
- Lớp String
- Lớp StringBuilder

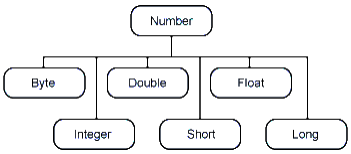
Các lớp số

- Java cung cấp các lớp bao các kiểu dữ liệu số nguyên thủy với những phương thức xử lý số một cách tiện lợi, hiệu quả
- **Boxes**: bao giá trị số nguyên thủy vào đối tượng

```
Integer x, y;
x = 12;
y = 15;
```

- **Unboxes**: lấy giá trị số nguyên thủy từ đối tượng

```
System.out.println(x+y);
```



Lớp Math

- java.lang.Math cung cấp hai hằng số E và PI cùng hàng loạt các phương thức tĩnh là các hàm số học như sin, cos, sqrt, log, pow, …
- Sử dụng:

```
import java.lang.Math;
Math.cos(angle);
```

hoặc

```
import static java.lang.Math;
cos(angle);
```


Character

- Character là lớp bao của kiểu dữ liệu ký tự, cung cấp các phương thức xử lý ký tự hiệu quả:

boolean **isLetter**(char ch) - Là chữ cái

boolean **isDigit**(char ch) – Là chữ số

boolean **isWhitespace**(char ch) – Là dấu cách

boolean **isUpperCase**(char ch) – Là chữ hoa

boolean **isLowerCase**(char ch) – Là chữ thường

char **toUpperCase**(char ch) – Chuyển thành chữ hoa

char **toLowerCase**(char ch) – Chuyển thành chữ thường

String **toString**(char ch) – Chuyển thành xâu ký tự

Các ký tự đặc biệt

- \t Tab
- \b Dấu cách
- \n Xuống dòng
- \r carriage return
- \f formfeed.
- \‘ Nháy đơn
- \“ Nháy kép
- \\ Chéo trái.

String

- Xâu ký tự: dãy các ký tự được đặt trong cặp nháy kép
  - Ví dụ: "Hello world!"
- Java cung cấp lớp String để xử lý xâu
  - String greeting = "Hello world!"; //boxed
  - System.out.print(greeting); //unboxed
  - int len = greeting.length(); //độ dài
  - greeting.**concat**(“ My name is …”); //nối xâu
    - == “Hello world!” + “ My name is …”
- String String.**format**(fmt, …); //tương tự System.out.print, chỉ khác “thiết bị ra” là một đối tượng String

Chuyển đổi xâu và số

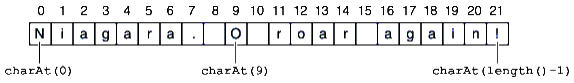
- Xâu thành số

  n = XXX.parseXXX(s); //XXX là tên lớp số như Integer, Float
- Số thành xâu

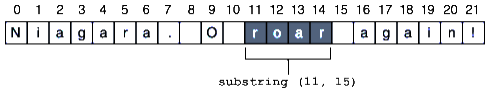
  s = n.toString();

Xử lý xâu

- String anotherPalindrome = "Niagara. O roar again!";
- char aChar = anotherPalindrome.**charAt**(9); // O



- String **substring**(int beginIndex, [int endIndex])



Xử lý xâu

- String[] split(String regex)
  String[] split(String regex, int limit)
- CharSequence subSequence(int beginIndex, int endIndex)
- String trim()
- String toLowerCase()
- String toUpperCase()

Tìm kiếm trong xâu

- int indexOf(int ch)
- int lastIndexOf(int ch)
- int indexOf(int ch, int fromIndex)
- int lastIndexOf(int ch, int fromIndex)
- int indexOf(String str)
- int lastIndexOf(String str)
- int indexOf(String str, int fromIndex)
- int lastIndexOf(String str, int fromIndex)
- boolean contains(CharSequence s)

Sửa đổi xâu

- String replace(char oldChar, char newChar)
- String replace(CharSequence target, CharSequence replacement)
- String replaceAll(String regex, String replacement)
- String replaceFirst(String regex, String replacement)

So sánh xâu

- boolean endsWith(String suffix)
- boolean startsWith(String prefix)
- boolean startsWith(String prefix, int offset)
- int compareTo(String anotherString)
- int compareToIgnoreCase(String str)
- boolean equals(Object anObject)
- boolean equalsIgnoreCase(String anotherString)
- boolean regionMatches(int toffset, String other, int ooffset, int len)
- boolean regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len)
- boolean matches(String regex)

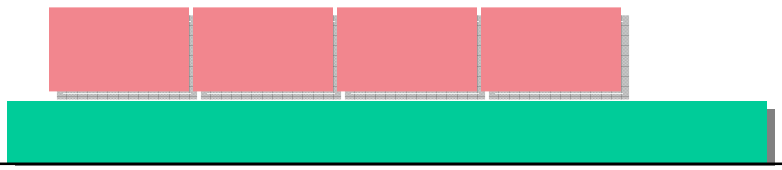
StringBuilder(*)

- Xử lý các xâu ký tự với việc lưu trữ xâu như một mảng các ký tự
- Cung cấp các phương thức xử lý xâu hiệu quả
  - append(…)
  - insert(…)
  - delete(…)
  - replace(…)
  - reverse(…)
  - setCharAt(…)
  - capacity()
  - StringBuilder(CharSequence cs)
  - StringBuilder(int initCapacity)
  - StringBuilder(String s)

(*) Tự học

Tiếp theo

Kiểu chung



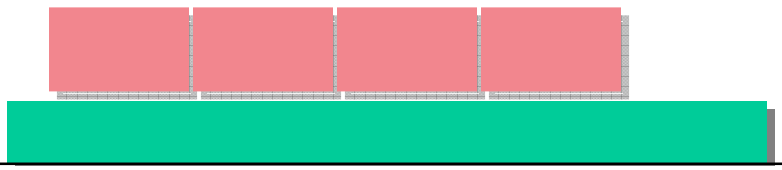
Bài giảng

LẬP TRÌNH JAVA

Lê Đình Thanh
Bộ môn Mạng và Truyền thông Máy tính
Khoa Công nghệ Thông tin
Trường Đại học Công nghệ, ĐHQGHN

Bài 5

Kiểu chung



Nội dung

- Các lỗi lập trình
- Kiểu chung
- Phương thức chung

Lỗi lập trình

- Lỗi lập trình (bug), thường xuyên gặp trong quá trình phát triển phần mềm, được chia làm hai loại:
  - Lỗi khi dịch (compile-time bugs): Các trình dịch có thể chỉ ra các lỗi này và lập trình viên phải sửa (debug) trước khi phần mềm chạy được.
  - Lỗi khi chạy (runtime bugs): Lỗi không thể phát hiện được khi dịch mà chỉ được phát hiện khi chạy.
    - Một trong những nguyên nhân gây lỗi khi chạy là sử dụng SAI KIỂU.
    - Kiểu tổng quát được đưa vào java nhằm phát hiện các lỗi sử dụng sai kiểu ngay khi dịch.

Ví dụ về lỗi khi chạy

```
public class Box {
    private Object object;
    public void add(Object object) { this.object = object; }
    public Object get() { return object; }
}
public class BoxDemo1 {
    public static void main(String[] args) {
        Box abox = new Box();
        abox.add("10");
        Integer anotherbox = (Integer) abox.get();
        System.out.println(anot
    }
 }
```

Exception in thread "main" java.lang.ClassCastException: java.lang.String cannot be cast to java.lang.Integer at BoxDemo1.main(BoxDemo1.java:6)

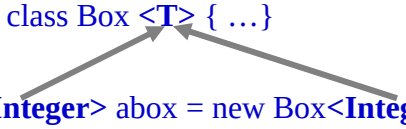
Kiểu chung – Generic Type

```
public class Box <T> {
    private T object;
    public void add(T object) { this.object = object; }
    public T get() { return object; }
}
public class BoxDemo2 {
    public static void main(String[] args) {
        Box<Integer> abox = new Box<Integer>();
        abox.add(new Integer(10));
        Integer anotherbox = abox.get();
        System.out.println(anotherbox);
    }
 }
```

Kiểu chung

- Kiểu được sử dụng như một tham số

```
public class Box <T> { …}

Box<Integer> abox = new Box<Integer>();
```



Phát hiện lỗi nhờ kiểu chung

```
public class Box <T> {
    private T object;
    public void add(T object) { this.object = object; }
    public T get() { return object; }
}
public class BoxDemo3 {
    public static void main(String[] args) {
        Box<Integer> abox = new Box<Integer>();
        abox.add("10");
        Integer anotherbox = (Integer) abox.get();
        System.out.println(anot
    }
}
```

```
BoxDemo3.java:5: add(java.lang.Integer) in
Box<java.lang.Integer> cannot be applied to
(java.lang.String) abox.add("10");
^ 1 error
```

Phương thức chung

- Là phương thức có tham số kiểu chung

```
public <U> void inspect(U u){
    System.out.println("U: " + u.getClass().getName());
}

public static <U> void fillBoxes(U u, List<Box<U>> boxes) {
    for (Box<U> box : boxes) { box.add(u); }
}
Crayon red = ...;
List<Box<Crayon>> crayonBoxes = ...;
Box.<Crayon>fillBoxes(red, crayonBoxes);  //hoặc
Box.fillBoxes(red, crayonBoxes);
```


Tham số kiểu được giới hạn

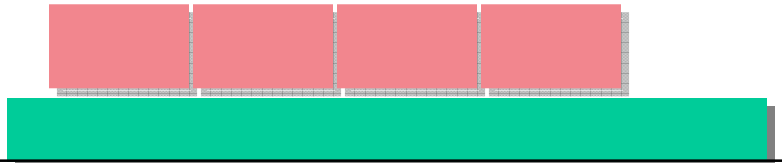
- Là phương thức có tham số kiểu chung

```
public <U extends Number [& OtherClasses]> void inspect(U
    u){
    System.out.println("U: " + u.getClass().getName());
}

abox.inspect("some text");
```

```
Box.java:21: <U>inspect(U) in
Box<java.lang.Integer> cannot be applied to
(java.lang.String) abox.inspect("some text");
^ 1 error
```

Tiếp theo

Các lớp thiết yếu



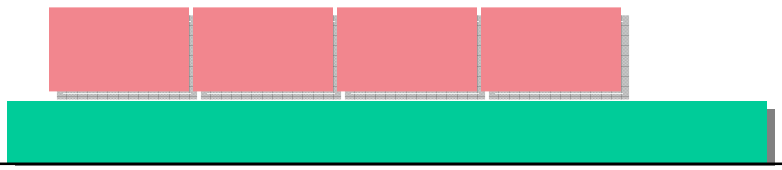
Bài giảng

LẬP TRÌNH JAVA

Lê Đình Thanh
Bộ môn Mạng và Truyền thông Máy tính
Khoa Công nghệ Thông tin
Trường Đại học Công nghệ, ĐHQGHN

Bài 6

Các lớp thiết yếu



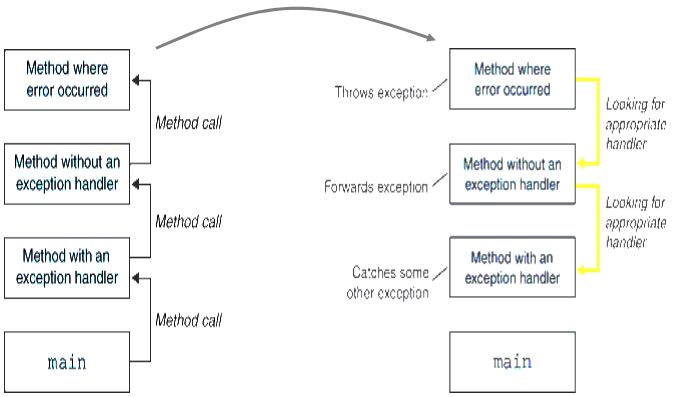
Nội dung

- Xử lý ngoại lệ
- Vào, ra cơ bản
- Xử lý đồng thời
- Môi trường
- Biểu thức chính quy

Ngoại lệ

- Ngoại lệ (exception hay exceptional event) là sự kiện xảy ra trong khi thực thi chương trình và phá vỡ luồng điều khiển của chương trình.
- Khi lỗi xuất hiện
  - Phương thức xuất hiện lỗi tạo đối tượng ngoại lệ (exception object) chứa thông tin về lỗi và gửi cho hệ thống thực thi (runtime system). Việc tạo và gửi đối tượng ngoại lệ cho hệ thống thực thi được gọi là ném (throw) ngoại lệ.
  - Hệ thống thực thi tìm ngược chuỗi triệu gọi (call stack) phương thức để tìm ra phương thức chứa đoạn mã xử lý ngoại lệ (exception handler)
  - Đoạn mã xử lý ngoại lệ bắt (catch) ngoại lệ và xử lý

Xử lý ngoại lệ



Call Stack

Searching the call stack for the exception handler

Bắt và xử lý ngoại lệ

```
try {
    //mã chương trình xử lý nghiệp vụ
}
catch (ExceptionType1 e) {
    //mã xử lý ngoại lệ nếu ngoại lệ thuộc kiểu ExceptionType1
}
[catch (ExceptionType2 e) {
    //mã xử lý ngoại lệ nếu ngoại lệ thuộc kiểu ExceptionType2
}
finally {
    //mã thực thi bất kể ngoại lệ xảy ra hay không
}]
```

Bắt và xử lý ngoại lệ

```
public void writeList() {
    PrintWriter out = null;
    try {
        System.out.println("Entering try statement");
        out = new PrintWriter(
                  new FileWriter("OutFile.txt"));
        for (int i = 0; i < SIZE; i++)
            out.println("Value at: " + i + " = "
                        +  vector.elementAt(i));

    } catch (ArrayIndexOutOfBoundsException e) {
         System.err.println("Caught "
                    + "ArrayIndexOutOfBoundsException: "
                    + e.getMessage());

    } catch (IOException e) {
         System.err.println("Caught IOException: "
                          + e.getMessage());

    } finally {
        if (out != null) {
            System.out.println("Closing PrintWriter");
            out.close();
        }
        else {
            System.out.println("PrintWriter not open");
        }
    }
}
```

Mã được thực thi

Kết quả in ra màn hình

```
Entering try statement
Closing PrintWriter
```


Bắt và xử lý ngoại lệ

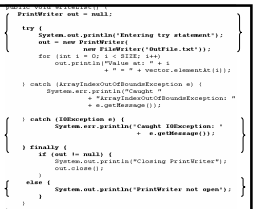
```
public void writeList() {
    PrintWriter out = null;

    try {
        System.out.println("Entering try statement");
        out = new PrintWriter(
                  new FileWriter("OutFile.txt"));
        for (int i = 0; i < SIZE; i++)
            out.println("Value at: " + i
                        + " = " + vector.elementAt(i));

    } catch (ArrayIndexOutOfBoundsException e) {
         System.err.println("Caught "
                    + "ArrayIndexOutOfBoundsException: "
                    + e.getMessage());

    } catch (IOException e) {
            System.err.println("Caught IOException: "
                             +  e.getMessage());

    } finally {
        if (out != null) {
            System.out.println("Closing PrintWriter");
            out.close();
        }
     else {
            System.out.println("PrintWriter not open");
        }
    }
}
```

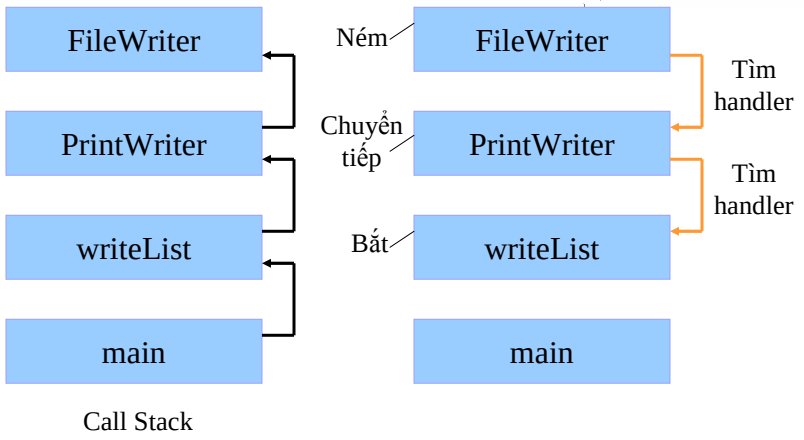
Mã được thực thi

Kết quả in ra màn hình

```
Entering try statement
Caught IOException: OutFile.txt
PrintWriter not open
```

Bắt và xử lý ngoại lệ





Ném ngoại lệ

- Mọi ngoại lệ đều phải được phát sinh ở một phương thức trước khi được chuyển tiếp và bắt ở các phương thức triệu gọi
- Một phương thức có thể phát sinh ngoại lệ và phương thức triệu gọi có thể bắt và xử lý
  - Áp dụng khi viết gói, ví dụ gói xử lý danh sách.
  - Để người viết ứng dụng có thể xử lý theo ý của họ

```
public Object pop() {
    Object obj;
    if (size == 0) { throw new EmptyStackException(); }
    obj = objectAt(size - 1);
    setObjectAt(size - 1, null);
    size--;
    return obj;
}
```

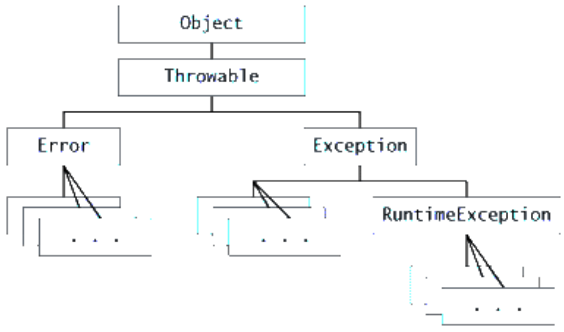
Chuyển tiếp ngoại lệ

- Một phương thức có thể không bắt và xử lý ngoại lệ mà chuyển tiếp cho phương thức đã triệu gọi nó (phương thức liền trước trong chuỗi triệu gọi)
  - Áp dụng khi viết gói, ví dụ gói xử danh sách.
  - Để người viết ứng dụng có thể xử lý theo ý của họ

```
public void writeList() throws IOException,
    ArrayIndexOutOfBoundsException {
    PrintWriter out = new PrintWriter(new
    FileWriter("OutFile.txt"));
    for (int i = 0; i < SIZE; i++) {
       out.println("Value at: " + i + " = " + vector.elementAt(i));
    }
    out.close();
}
```


Các lớp ném được

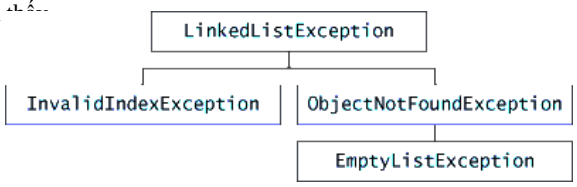
- Các lớp Error định nghĩa các sự cố nghiêm trọng, thuộc JVM. Các ứng dụng thường không ném và bắt Error
- Các lớp Exception định nghĩa các ngoại lệ khi viết gói



Định nghĩa lớp ngoại lệ

- Định nghĩa lớp ngoại lệ ứng với các ngoại lệ phát sinh trong lớp được định nghĩa
- Kế thừa từ lớp Exception

Ví dụ, lớp **LinkedList** với các phương thức
objectAt(int n) — Lấy đối tượng ở vị trí thứ n – Ném ngoại lệ InvalidIndexException nếu n < 0 hoặc n >= size.
firstObject() — Lấy đối tượng đầu tiên – Ném ngoại lệ EmptyListException khi danh sách rỗng.
indexOf(Object o) — Tìm đối tượng – Ném ngoại lệ ObjectNotFoundException nếu không tìm thấy



Lợi ích của sử dụng ngoại lệ

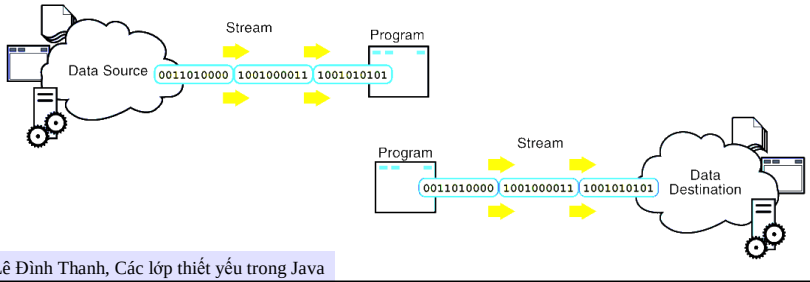
- Tách mã xử lý lỗi với các mã xử lý nghiệp vụ
- Chuyển tiếp lỗi lên các phương thức ở trên
- Phân nhóm các lỗi

Vào/ra cơ bản

- Các dòng vào/ra (I/O Streams)
- Vào/ra với tệp (File I/O)

Các dòng vào/ra

- Một dòng vào/ra (I/O stream) biểu diễn một nguồn vào/đích ra dữ liệu
  - Có thể biểu diễn nhiều nguồn/đích khác nhau: bộ đệm, tệp, thiết bị, chương trình khác.
  - Hỗ trợ nhiều kiểu dữ liệu khác nhau: byte, dữ liệu nguyên thủy, đối tượng
  - Phương thức cơ bản: read()/write(data)



Byte streams

- Biểu diễn dòng dữ liệu gồm các bytes liên tiếp
- Dòng vào: XXX**InputStream**, phương thức đọc **byte read()**
- Dòng ra: XXX**OutputStream**, phương thức ghi **write(byte)**

```
FileInputStream in = null;
FileOutputStream out = null;
try {
    in = new FileInputStream("xanadu.txt");
    out = new FileOutputStream("outagain.txt");
    int c;
    while ((c = in.read()) != -1) { out.write(c); }
} finally {
    if (in != null) { in.close(); }
    if (out != null) { out.close(); }
}
```


Character Streams

- Biểu diễn dòng các ký tự liên tiếp
- Tự động chuyển đổi các mã thành Unicode
- Dòng vào: XXX**InputReader**, phương thức đọc **int read()**
- Dòng ra: XXX**OutputWriter**, phương thức ghi **write(int)**
- **Mã ký tự được lưu ở 16 bit thấp của int**

```
FileReader in = null; FileWriter out = null;
try {
    in = new FileReader("xanadu.txt");
    out = new FileWriter("outagain.txt");
    int c;
    while ((c = in.read()) != -1) { out.write(c); }
} finally {
    if (in != null) { in.close(); }
    if (out != null) { out.close(); }
}
```

Đọc/ghi theo dòng

- Character stream thường chứa nội dung của cả văn bản: gồm nhiều dòng (lines) văn bản
- Một line là một chuỗi ký tự kết thúc bằng ký tự xuống dòng: '\r\n', '\n' hoặc '\r'.
- Đọc/ghi cả dòng văn bản là phương pháp hiệu quả hơn đọc từng ký tự

```
BufferedReader  in = null;  PrintWriter out = null;
try {
    in = new BufferedReader(new FileReader("xanadu.txt"));
    out = new PrintWriter(new FileWriter("outagain.txt"));
    String l;
    while ((l = in.readLine()) != -1) { out.println(l); }
} finally {
    if (in != null) { in.close(); }
    if (out != null) { out.close(); }
}
```


Buffered Streams

- Các dòng không đệm (unbuffered) như FileInputStream, FileReader, … làm việc trực tiếp với HĐH, các thao tác đọc ghi được quản lý trực tiếp bởi HĐH
  - Thao tác đọc: đọc từ thiết bị
  - Thao tác ghi: ghi ngay ra thiết bị
  - Không hiệu quả vì nhiều lần phải truy cập đĩa, mạng, …
- Giải pháp: Sử dụng dòng đệm (buffered) để đọc/ghi cả khối => ít lần truy cập phần cứng
  - Đọc: Khi vùng đệm rỗng, đọc từ nguồn vào đầy đệm
  - Ghi: Khi vùng đệm đầy, ghi tất cả vào đích
  - Nếu cần chủ động ghi ra đích khi chưa đầy đệm: Gọi flush()
- Các lớp: BufferedInputStream, BufferedOutputStream, BufferedReader, BufferedWriter

Scanner

- Cung cấp các phương thức để đọc từng từ tố (token) trong character stream, phân loại từ tố (số nguyên, số thập phân, …)
- Từ tố đơn thuần là xâu ký tự. Character stream là dãy liên tiếp các từ tố và các từ phân cách (separator/delimiter). Giữa hai từ phân cách liên tiếp là một từ tố. Giữa hai từ tố có thể có nhiều từ phân cách.
- Mặc định, dấu trắng (blank, tab, dấu câu, …) được dùng làm từ phân cách.
- Để kiểm tra một ký tự có phải là dấu trắng hay không **Character.isWhitespace**.
- Muốn sử dụng từ phân cách khác, sử dụng phương thức **useDelimiter(",\\s*")**

Scanner

- Đọc các từ tố

```
import java.io.*;
import java.util.Scanner;

Scanner s = null;
try {
    s = new Scanner(new BufferedReader(new
      FileReader("xanadu.txt")));
    while (s.hasNext()) { System.out.println(s.next()); }
} finally { if (s != null) { s.close(); } }
```

Scanner

- Đọc từ tố được phân loại

```
s.useLocale(Locale.US);
while (s.hasNext()) {
    if (s.hasNextDouble()) {
      sum += s.nextDouble();
    }
    else {
      s.next();
    }
}
```


Printing and Formatting

- Hai lớp cơ sở thực cung cấp phương thức print, println và format là
  - PrintWriter: một chacracter stream
  - PrintSteam: một byte stream
- **print** và **println**
  - Có một tham số có kiểu nguyên thủy
  - Biến đổi một giá trị thành xâu và in xâu

```
int i = 2; double r = Math.sqrt(i);
System.out.print("The square root of "); System.out.print(i);
System.out.print(" is ");
System.out.print(r);
System.out.println("The square root of " + i + " is " + r + ".");
```

Printing and Formatting

- **format**
  - Có một hoặc nhiều tham số bất kỳ
  - Tham số đầu tiên là xâu ký tự có thể chứa ký tự định dạng (%d, %f, %s, …)
  - Các tham số tiếp theo có kiểu nguyên thủy
  - Thay ký tự định dạng thứ i trong xâu tham số thứ nhất bằng biểu diễn giá trị tham số thứ i+1 theo định dạng được xác định bởi ký tự định dạng thứ i. Trả về xâu tham số thứ nhất đã được thay các ký tự định dạng bằng giá trị tham số phía sau.

    ```
    int i = 2; double r = Math.sqrt(i);
    System.out.format("The square root of %d is %f.%n", i, r);
    ```


Vào/ra với dòng lệnh

- Dòng vào chuẩn
  - **System.in** – là một byte stream. Để dùng như một character stream, sử dụng lớp bao InputStreamReader

    ```
    InputStreamReader cin = new
        InputStreamReader(System.in);
    ```
- Dòng ra chuẩn
  - **System.out** và **System.err** – là các byte streams nhưng có thể mô phỏng character stream
- Bàn giao tiếp
  - **Console** - sử dụng thay thế cho các dòng vào/ra chuẩn với các phương thức thuận tiện hơn: readLine, readPassword, format, …

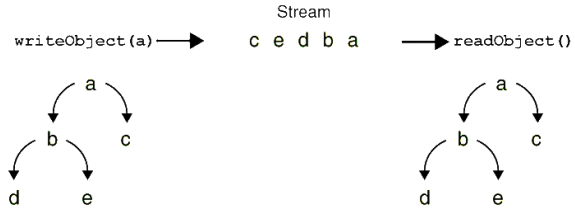
Data Stream

- Các dòng vào/ra cho các kiểu dữ liệu nguyên thủy
- Hai lớp hay dùng là DataInputStream với các phương thức readTTT, và DataOutputStream với các phương thức writeTTT, trong đó TTT là tên kiểu dữ liệu nguyên thủy

```
out = new DataOutputStream(new BufferedOutputStream(new
      FileOutputStream(dataFile)));
out.writeDouble(price);
out.writeInt(unit);
out.writeUTF(desc);
in = new DataInputStream(new BufferedInputStream(new
      FileInputStream(dataFile)));
price = in.readDouble();
unit = in.readInt();
desc = in.readUTF();
```


Object Stream

- Dòng vào/ra cho đối tượng
- Các đối tượng có thể ghi vào/đọc từ dòng phải thuộc lớp cài đặt giao diện Serializable.
- Hai lớp hay dùng là ObjectInputStream với phương thức readObject() và ObjectOutputStream với phương thức writeObject(obj)
- Khi ghi vào/đọc ra một đối tượng, các đối tượng được tham chiếu cũng được ghi/đọc theo



Object Stream

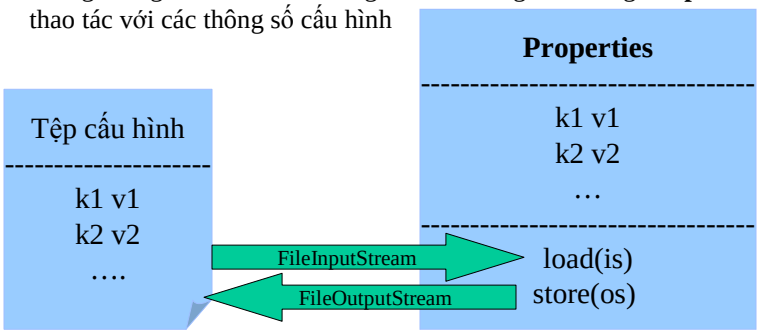
- Một dòng chỉ lưu đối tượng duy nhất một lần, nếu ghi nhiều lần thì nhiều tham chiếu được ghi
- Nếu ghi một đối tượng vào hai dòng khác nhau thì khi đọc ra được 2 đối tượng khác nhau

Môi trường nền

- Môi trường nền (platform environment) = hệ điều hành, máy ảo Java, thư viện các lớp, dữ liệu cấu hình.
- CácAPI để cấu hình môi trường nền:
  - Các tiện ích cấu hình
    - Thông số cấu hình
    - Tham số dòng lệnh
    - Biến môi trường
  - Các tiện ích hệ thống
    - Thông số hệ thống
    - Quản lý an ninh (tự đọc)
    - Một số phương thức hệ thống (tự đọc)
  - CLASS và CLASSPATH (tự đọc)

Thông số cấu hình

- Một thông số cấu hình là một cặp xâu ký tự *khóa/giá trị*, ví dụ *LastBrowsedTabs=“http://dantri.com.vn;http://uet.vnu.edu.vn”*
- Các thông số cấu hình được lưu ở một tệp cấu hình (thường ở dạng văn bản), được chương trình đọc vào khi khởi động và lưu lại trước khi kết thúc
- Trong thời gian thực thi, chương trình sử dụng đối tượng **Properties** để thao tác với các thông số cấu hình



Thông số cấu hình

- Đọc cấu hình từ tệp

```
Properties applicationProps = new Properties();
FileInputStream in = new
    FileInputStream("appProperties.config");
applicationProps.load(in);
in.close();
```

- Ghi cấu hình ra tệp

```
FileOutputStream out = new FileOutputStream("
    appProperties.config ");
applicationProps.store(out, "---Comments---");
    out.close();
```

Thông số cấu hình

- Properties cung cấp hàng loạt các phương thức để đọc và cập nhật thông số

getProperty(String key)
getProperty(String key, String default)
setProperty(String key, String value)
contains(Object value)
containsKey(Object key)
list(PrintStream s)
list(PrintWriter w)
elements()
keys()
propertyNames()
stringPropertyNames()
size()
remove(Object key)

Properties kế thừa Hashtable cài đặt một bảng băm nên các thao tác tìm theo khóa rất hiệu quả

Tham số dòng lệnh

- Khi gọi một chương trình bằng lệnh java TenCT, chúng ta có thể cung cấp các tham số cho chương trình
- Các tham số này được truyền cho **agrs** của hàm main

```
public class SomeClass {
    public static void main (String[] args) {
        for (String s: args) {
    System.out.println(s); }
    }
}
```

java SomeClass “A sunny day” “We go out”
A sunny day
We go out

Biến môi trường

- Java cung cấp phương thức ddocj các biên môi trường

```
import java.util.Map;
public class EnvMap {
    public static void main (String[] args) {
        Map<String, String> env = System.getenv();
        for (String envName : env.keySet()) {
            System.out.format("%s=%s%n", envName,
                env.get(envName));
        }
    }
}
//String value = System.getenv(env);
```


Thông số hệ thống

- Lớp System chứa một đối tượng Properties để quản lý các thông số hệ thống như tên hệ điều hành, thư mục cài đặt JRE, …
- Đọc thông số hệ thống

```
Properties System.getProperties()
String System.getProperty(tenTS);
```

- Thay đổi thông số hệ thống

```
System.setProperties(Properties  p);
```

- Hỏi: Làm như thế nào để thay đổi một thông số hệ thống?

Thông số hệ thống

- Lớp System chứa một đối tượng Properties để quản lý các thông số hệ thống như tên hệ điều hành, thư mục cài đặt JRE, …
- Đọc thông số hệ thống

  Properties System.getProperties()

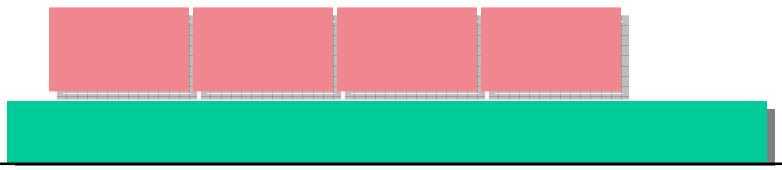
  String System.getProperty(tenTS);
- Thay đổi thông số hệ thống

  System.setProperties(Properties p);

- Hỏi: Làm như thế nào để thay đổi một thông số hệ thống?

Tiếp theo

Ứng dụng CSDL



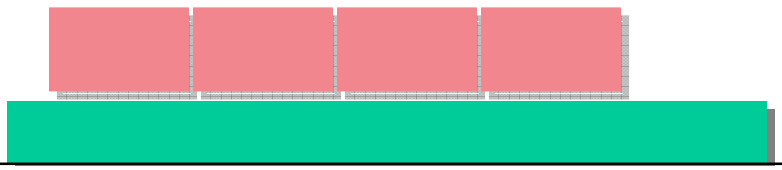
Bài giảng

LẬP TRÌNH JAVA

Lê Đình Thanh
Bộ môn Mạng và Truyền thông Máy tính
Khoa Công nghệ Thông tin
Trường Đại học Công nghệ, ĐHQGHN

Bài 7

Ứng dụng CSDL

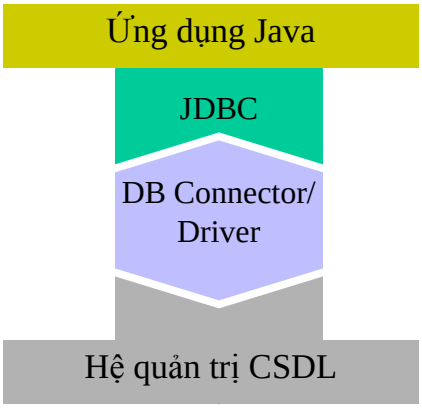


Nội dung

- Mô hình ứng dụng CSDL
- Sử dụng JavaDB

Mô hình ứng dụng CSDL

- JDBC (Java Database Connectivity) cung cấp các đối tượng logic: kết nối, lệnh, và kết quả truy vấn; được thiết kế để chạy độc lập với HQT CSDL
- Connector nhận lệnh của JDBC, dịch thành lệnh của HQT CSDL và truyền (qua mạng) cho HQT CSDL; nhận kết quả từ HQT CSDL và truyền cho JDBC
- HQT CSDL khác nhau về cách tiếp nhận và xử lý lệnh, cũng như trả kết quả. Vì vậy, ứng với mỗi HQT CSDL, các driver/connector tương ứng phải được sử dụng



Các bước

- Nhập gói sql
- Chọn driver
- Kết nối CSDL
- Thao tác CSDL
  - Tạo đối tượng lệnh
  - Thực hiện cập nhật
  - Tạo đối tượng kết quả
  - Thực hiện truy vấn
  - Sử dụng kết quả truy vấn
- Đóng kết nối

Nhập gói sql

- Gói java.sql bao gồm các lớp cho thao tác CSDL, vì vậy cần nhập các lớp trong gói này vào ứng dụng CSDL

```
import java.sql.*; //nhập tất cả các lớp
//hoặc chỉ nhập các lớp được dùng
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.ResultSet;
```

Chọn driver

- Class.forName(driverName);

HQT CSDL	driverName
MySQL	com.mysql.jdbc.Driver
SQL Server	com.microsoft.sqlserver.jdbc.SQLServerDriver
Derby	org.apache.derby.jdbc.EmbeddedDriver
JavaDB	org.apache.derby.jdbc.EmbeddedDriver
Oracle	sql.oracle.OracleDriver
ODBC	sun.jdbc.odbc.JdbcOdbcDriver
PostgreSQL	org.postgresql.Driver

Chọn driver

- Nếu driver được cung cấp ở dạng thư viện (.jar), chúng ta phải nhập thư viện này vào chương trình.
  - Trong cửa sổ *Projects*, bấm phải chuột vào *Libraries* của dự án đang phát triển, chọn *Add JAR/Folder …*, chọn tệp .jar cần nhập rồi chọn *Open*.
- Ví dụ, nếu sử dụng JavaDB, chúng ta cần *derby* driver, do vậy phải nhập *derby.jar* hay *derbyclient.jar* trong thư mục *Program Files\Sun\JavaDB\lib*

Kết nối CSDL

- Connection conn = DriverManager.getConnection(dburl, username, password);

trong đó dburl là địa chỉ và tên CSDL, có dạng "jdbc:xxx://IP:port//db", xxx phụ thuộc vào HQT CSDL, db là tên CSDL

HQT CSDL	xxx
MySQL	mysql
Derby, JavaDB	derby

Cập nhật dữ liệu

- Statement stmt = conn.createStatement();
- int nor = stmt.executeUpdate(sql);

sql là lệnh update, insert, delete, alter, … của SQL
Hàm trả về số bản ghi được cập nhật.

Ví dụ:

```
stmt.executeUpdate("update Sinhvien set
  hoten=\'Trần Nguyên\' where masv=\'1\'");
```

Truy vấn dữ liệu

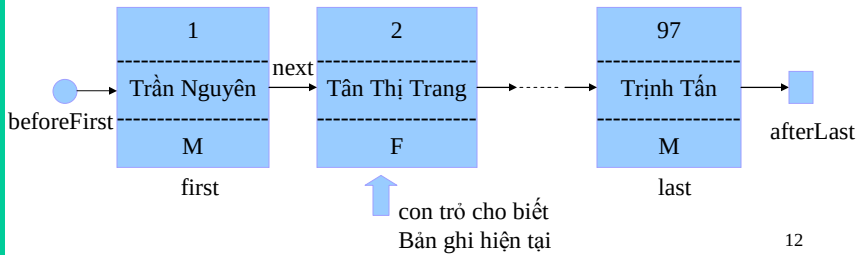
- Statement stmt = conn.createStatement();
- ResultSet rs = stmt.executeQuery("select … ");

Ví dụ:

ResultSet rs = stmt.executeQuery("select * from Sinhvien where gioitinh=\'M\'");

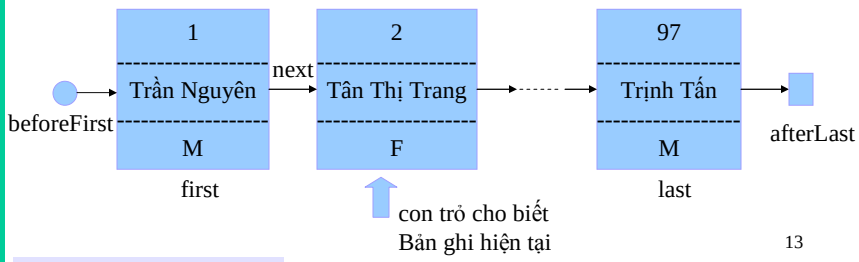
Truy vấn dữ liệu (tiếp)

- ResultSet
  - lưu kết quả truy vấn dưới dạng danh sách các bản ghi với một con trỏ di chuyển được ban đầu trỏ về đầu danh sách (trước bản ghi đầu tiên)
  - cung cấp các phương thức để duyệt danh sách – di chuyển con trỏ
    - **next()** – chuyển sang bản ghi kế tiếp và trả về true nếu còn bản ghi phía sau, ngược lại trả về false
    - **previous(), fist(), last(), beforeFirst(), afterLast().**



Truy vấn dữ liệu (tiếp)

- ResultSet
  - và các phương thức **getXXX(tên_trường)** đọc giá trị từng trường của bản ghi hiện tại – đang được trỏ bởi con trỏ, với XXX là kiểu dữ liệu của trường.
  - Ví dụ
    - rs.getInt(“masv”) => 2
    - rs.getString(“hoten”) => “Tân Thị Trang”
    - rs.getChar(“gioitinh”) => ‘F’



Truy vấn dữ liệu (tiếp)

```
try {
    Statement stmt = conn.createStatement();
    ResultSet rs = stmt.executeQuery(“select … ”);
    while (rs.next())
    {
        int ma = rs.getInt(“masv”);
        String ht = rs.getString(“hoten”);
        Char gt = rs.getChar(“gioitinh”);
        System.out.println(ma + “\t” + ht  + “\t” + gt);
    }
}catch(Exception excp) {
    System.out.println(“Loi truy van”);
}
```

Các lệnh thao tác CSDL có thể gây ngoại lệ, vì vậy cần được bắt và xử lý

Lệnh được chuẩn bị trước

- PreparedStatement được dùng thay Statement để tăng tốc độ xử lý
  - Cấu trúc lệnh sql được cung cấp trước cho PreparedStatement như một hàm được dịch trước
  - Cung cấp các tham số cho PreparedStatement trước khi thực thi như gọi hàm

```
PreparedStatement pstmt = con.preparedStatement("UPDATE Sinhvien
    SET hoten = ? WHERE masv= ? ");
pstmt.setStringt(1, “Tôn Nhất Đại”);
pstmt .setString(2, 14);
pstmt.executeUpdate();
```


Sử dụng stored procedure

- CallableStatement là lớp kế thừa PreparedStatement dùng để triệu gọi các stored procedure của CSDL nhằm tăng tốc độ xử lý ở cả DB Server và chương trình

```
CallableStatement cs = conn.prepareCall("{call
    ALL_STUDENT}");
ResultSet rs = cs.executeQuery();
```

Sử dụng giao tác

- Giao tác (transaction) được sử dụng để đảm bảo tính toàn vẹn của dữ liệu
  - Một giao tác thường bao gồm nhiều lệnh
  - Khi giao tác được xác nhận (commit), tất cả các lệnh thuộc giao tác có hiệu lực
  - Nếu giao tác bị hủy bỏ (rollback), tất cả các lệnh đều không được thực hiện
  - Giao tác = tập các lệnh hoặc cùng được thực hiện hoặc cùng không được thực hiện
- Hỏi: Vì sao giao tác để đảm bảo tính toàn vẹn của dữ liệu? Cho ví dụ.

Sử dụng giao tác

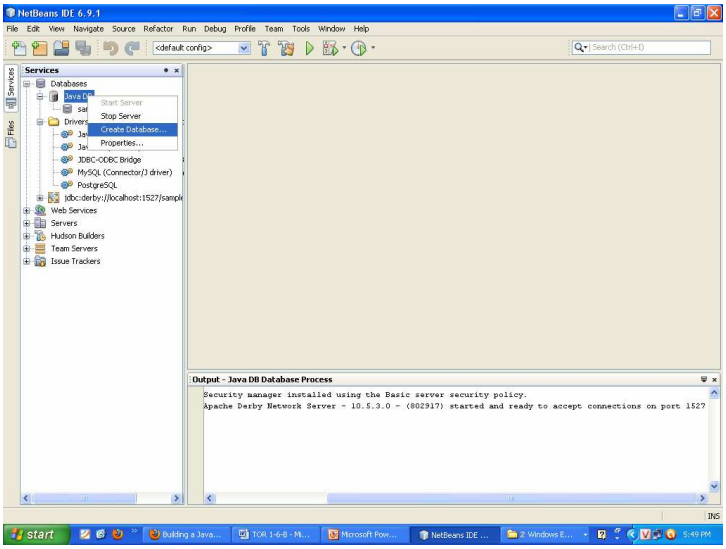
```
conn.setAutoCommit(false);
PreparedStatement updateSales = con.prepareStatement("UPDATE
    COFFEES SET SALES = ? WHERE COF_NAME LIKE ?");
updateSales.setInt(1, 50);
updateSales.setString(2, "Colombian");
updateSales.executeUpdate();
PreparedStatement updateTotal = con.prepareStatement("UPDATE
    COFFEES SET TOTAL = TOTAL + ? WHERE COF_NAME LIKE
    ?");
updateTotal.setInt(1, 50);
updateTotal.setString(2, "Colombian");
updateTotal.executeUpdate();
conn.commit();
conn.setAutoCommit(true);
```

Hỏi: Cơ chế thực hiện giao tác?

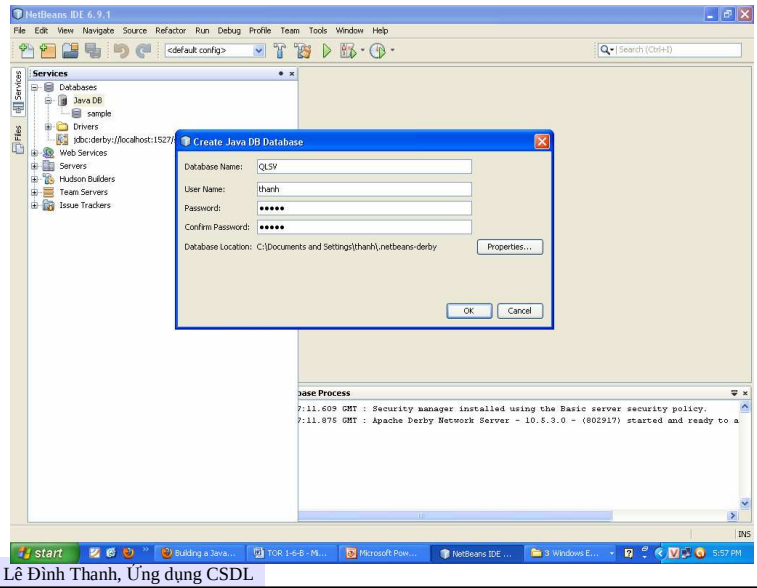
Sử dụng giao tác

```
con.setAutoCommit(false);
Statement stmt = conn.createStatement();
int rows = stmt.executeUpdate("INSERT INTO TAB1
    (COL1) VALUES " + "(?FIRST?)");
Savepoint svpt1 = conn.setSavepoint("SAVEPOINT_1");
rows = stmt.executeUpdate("INSERT INTO TAB1 (COL1) "
    +                          "VALUES (?SECOND?)");
...
conn.rollback(svpt1);
….
con.commit();
con.setAutoCommit(true);
```

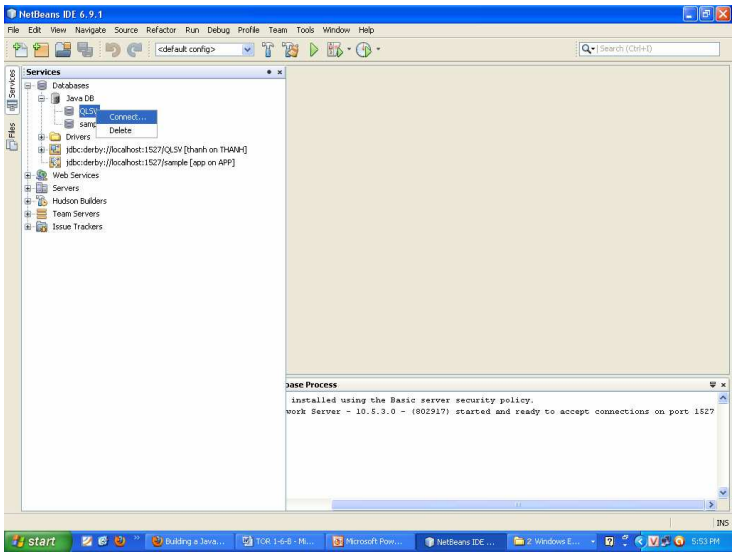

Tạo CSDL JavaDB



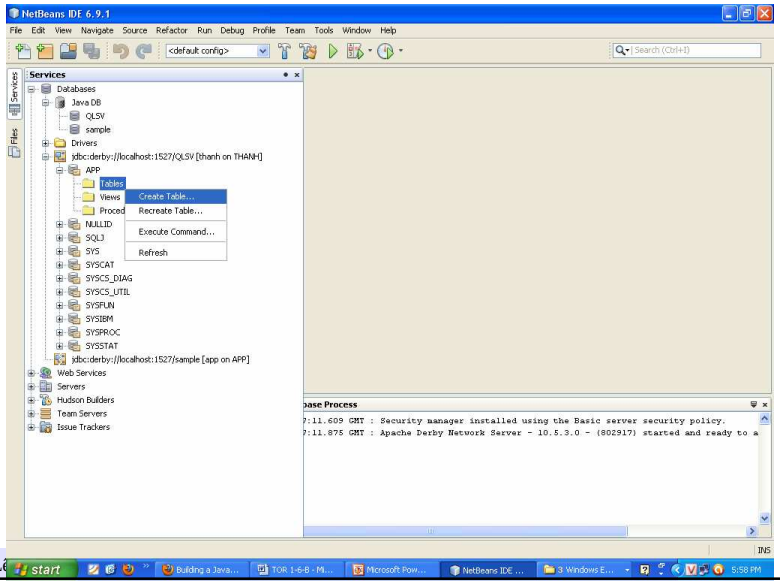
Tạo CSDL JavaDB (tiếp)



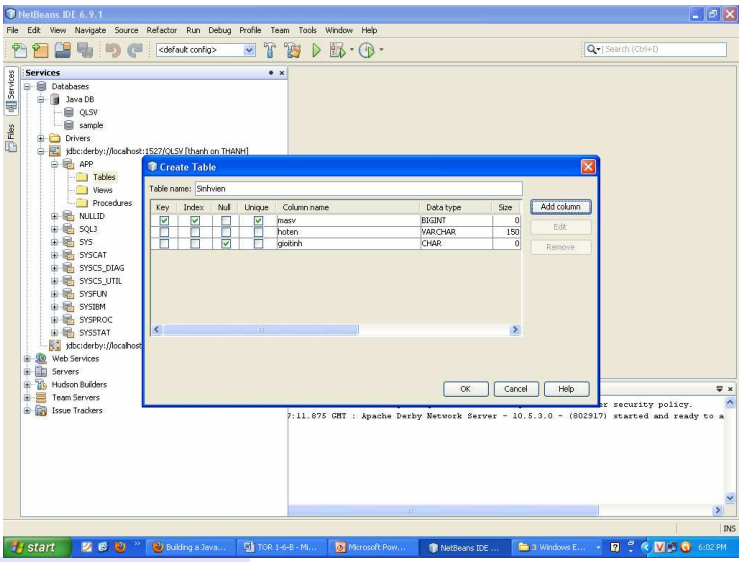
Kết nối CSDL JavaDB



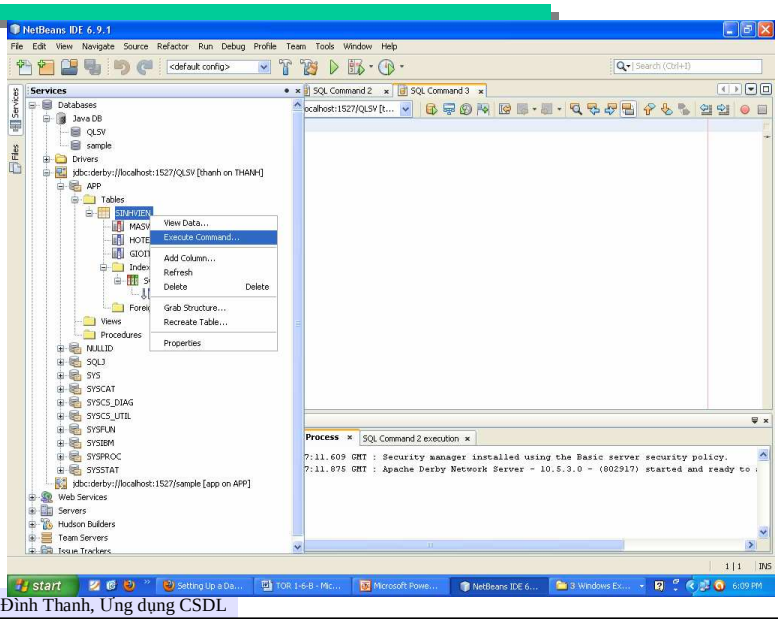
Tạo bảng



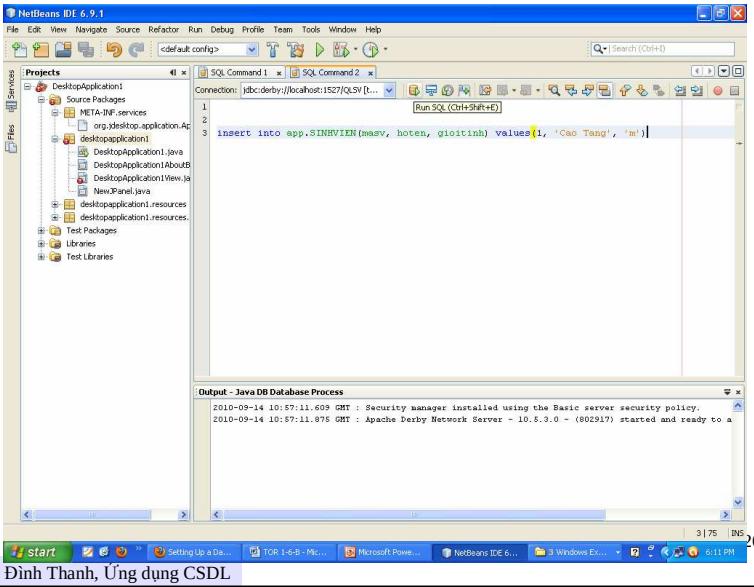
Tạo bảng (tiếp)



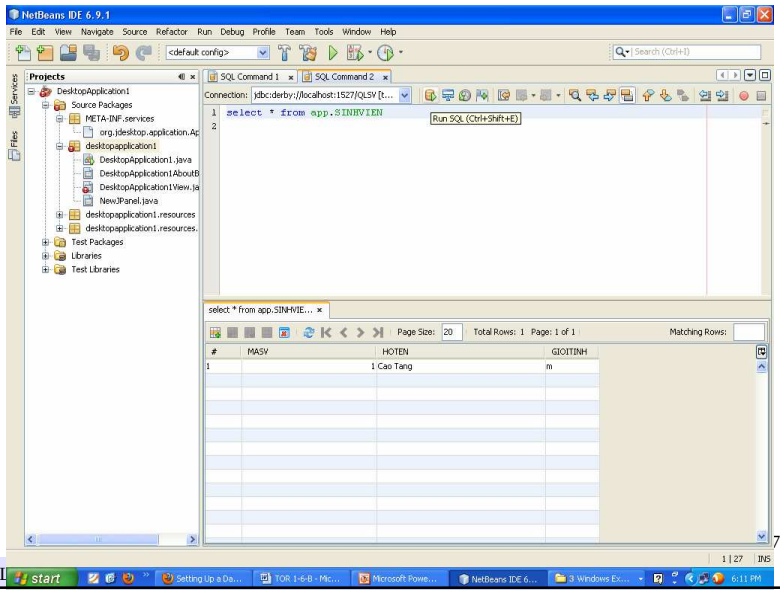
Chạy lệnh SQL



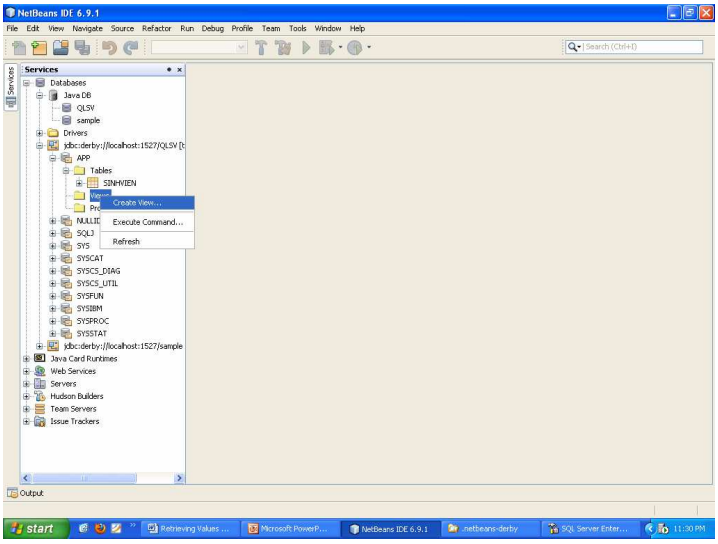
Chạy lệnh SQL (tiếp)



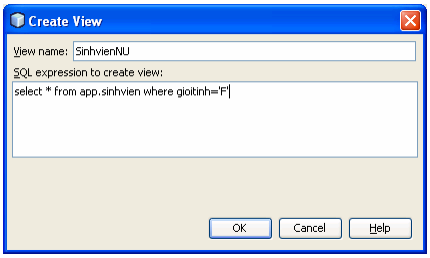
Chạy lệnh SQL (tiếp)



Tạo view



Tạo view (tiếp)



Tiếp theo

Tạo giao diện đồ họa

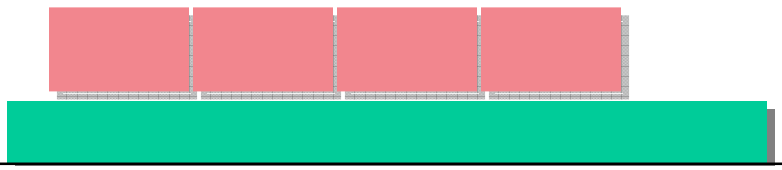
Bài giảng

LẬP TRÌNH JAVA

Lê Đình Thanh
Bộ môn Mạng và Truyền thông Máy tính
Khoa Công nghệ Thông tin
Trường Đại học Công nghệ, ĐHQGHN

Bài 8

Tạo giao diện đồ họa



Nội dung

- Java Foundation Classes
- Swing

Java Foundation Classes (JFC)

JFC

- Các lớp nền tảng của Java (JFC) bao gồm các lớp thành phần để xây dựng giao diện người dùng, được chia thành các nhóm sau:
  - **Swing**: các thành phần như nút bấm, hộp chọn, họp soạn thảo, danh sách, cây, … để phát triển giao diện đồ họa
  - **Look-and-Feel**: Tùy biến theme
  - **Java 2D**: để đồ họa 2 chiều
  - **Internationalization**: thay đổi ngôn ngữ cho người dùng sử dụng ngôn ngữ khác nhau
  - **Accessibility**: cung các các công nghệ hỗ trợ truy cập khác như đọc màn hình, …

Swing

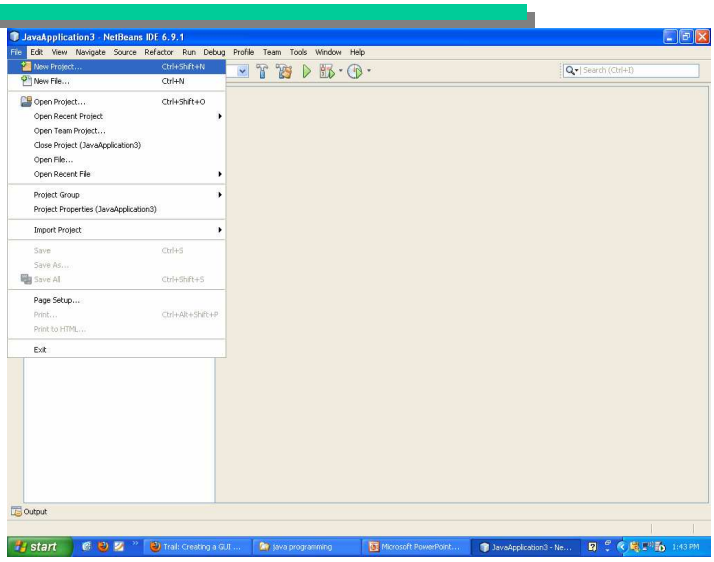
Các bước tạo ứng dụng swing

```
import javax.swing.*;  //Nhập các lớp swing
public class HelloWorldSwing {
    private static void main(String[] args) {
        //Tạo và thiết lập cửa sổ/đối tượng chứa.
        JFrame frame = new JFrame("HelloWorldSwing");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        //Thêm các thành phần vào cửa sổ.
        JLabel label = new JLabel("Hello World");
        frame.getContentPane().add(label);
        //Thêm sự kiện và mã xử lý cho các thành phần trên cửa sổ
        //Hiển thị cửa sổ
        frame.pack();
        frame.setVisible(true);
    }
}
```

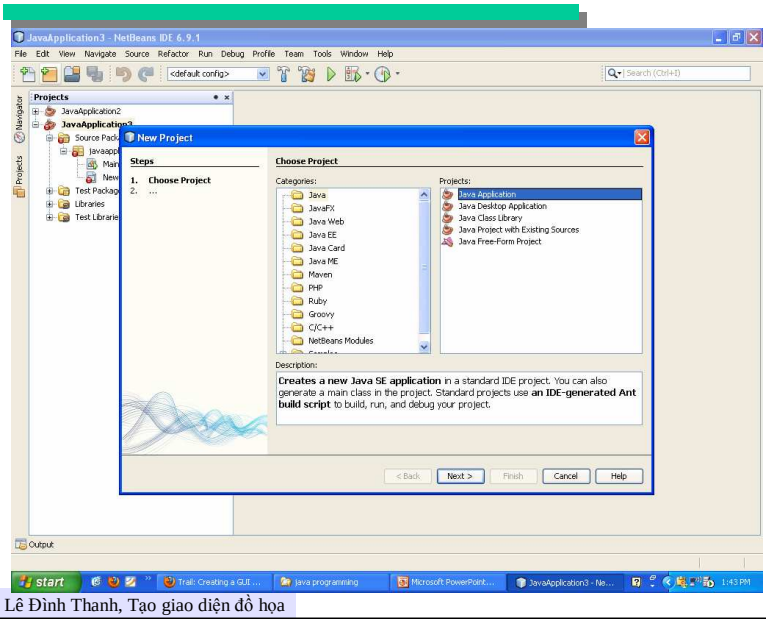

Tạo ứng dụng swing bằng NetBeens

- Tạo và thiết lập cửa sổ/đối tượng chứa
- Thêm các thành phần vào cửa sổ
- Thêm sự kiện và mã xử lý cho các thành phần trên cửa sổ
- Hiển thị cửa sổ

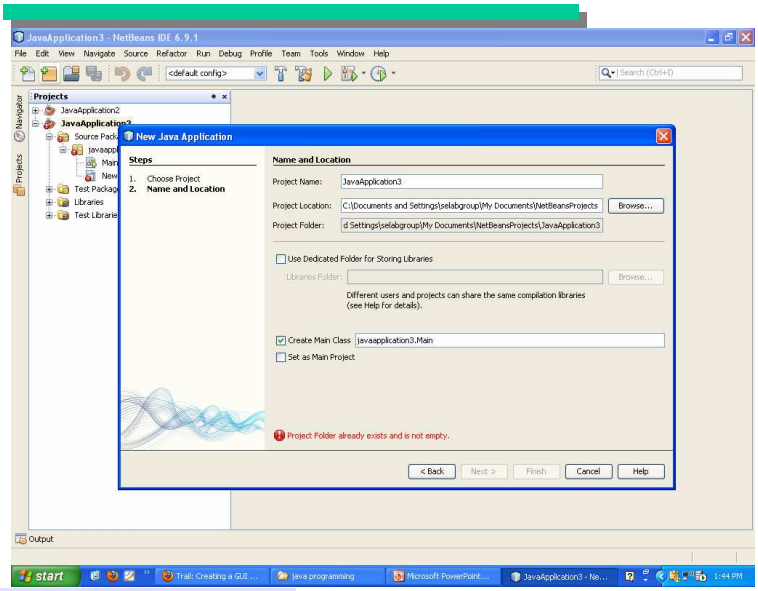
Tạo ứng dụng swing bằng NetBeens



Tạo ứng dụng swing bằng NetBeens

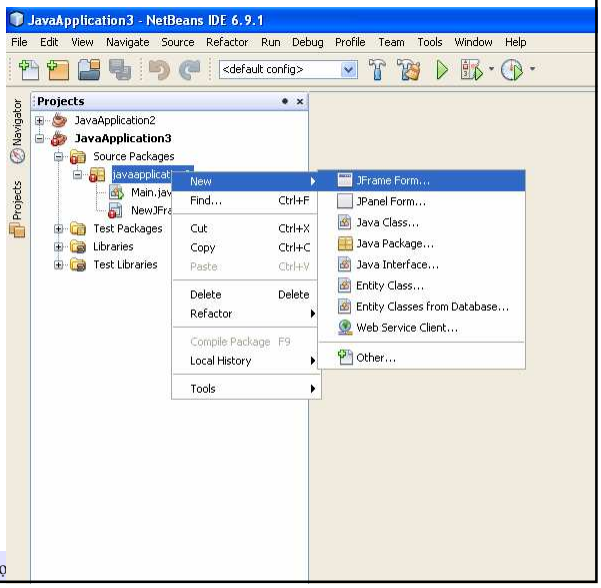


Tạo ứng dụng swing bằng NetBeens



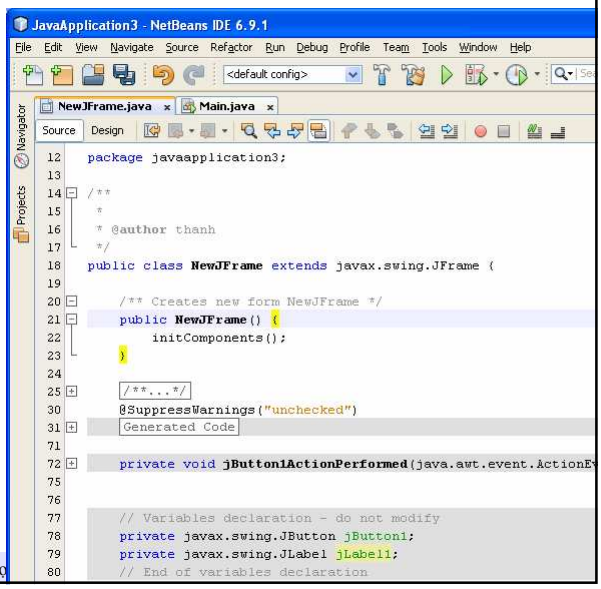
Tạo ứng dụng swing bằng NetBeens

- Tạo và thiết lập cửa sổ/đối tượng chứa
- Thêm các thành phần vào cửa sổ
- Thêm sự kiện và mã xử lý cho các thành phần trên cửa sổ
- Hiển thị cửa sổ



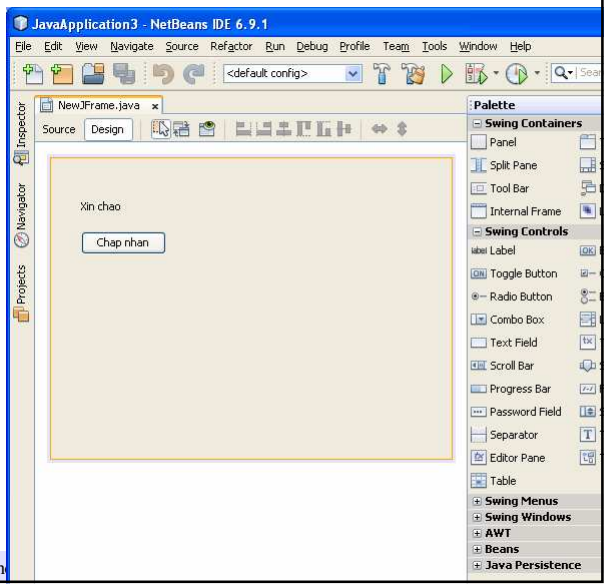
Tạo ứng dụng swing bằng NetBeens

- Tạo và thiết lập cửa sổ/đối tượng chứa
- Thêm các thành phần vào cửa sổ
- Thêm sự kiện và mã xử lý cho các thành phần trên cửa sổ
- Hiển thị cửa sổ



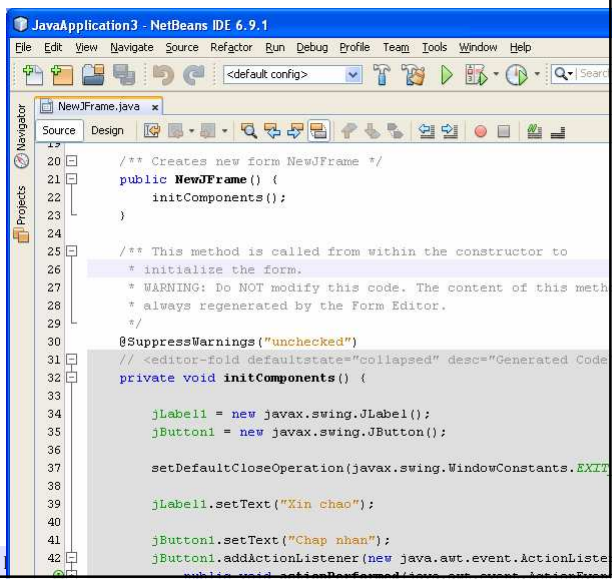
Tạo ứng dụng swing bằng NetBeens

- Tạo và thiết lập cửa sổ/đối tượng chứa
- Thêm các thành phần vào cửa sổ
- Thêm sự kiện và mã xử lý cho các thành phần trên cửa sổ
- Hiển thị cửa sổ



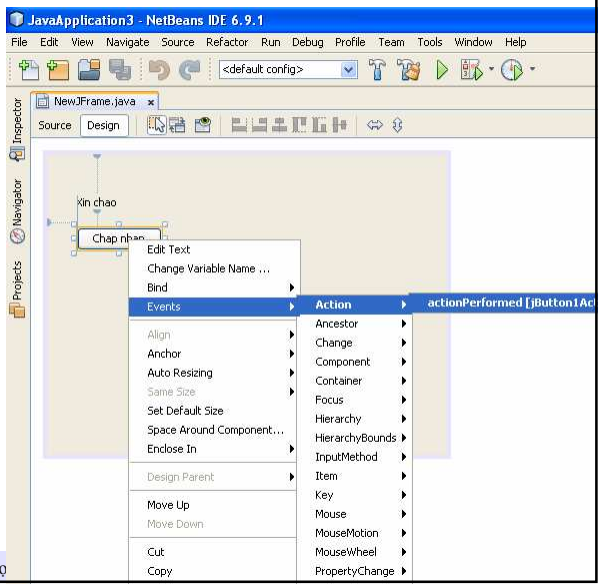
Tạo ứng dụng swing bằng NetBeens

- Tạo và thiết lập cửa sổ/đối tượng chứa
- Thêm các thành phần vào cửa sổ
- Thêm sự kiện và mã xử lý cho các thành phần trên cửa sổ
- Hiển thị cửa sổ



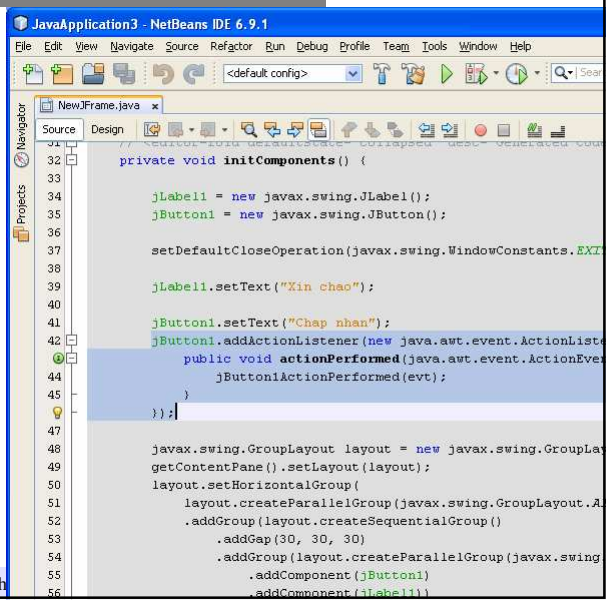
Tạo ứng dụng swing bằng NetBeens

- Tạo và thiết lập cửa sổ/đối tượng chứa
- Thêm các thành phần vào cửa sổ
- Thêm sự kiện và mã xử lý cho các thành phần trên cửa sổ
- Hiển thị cửa sổ



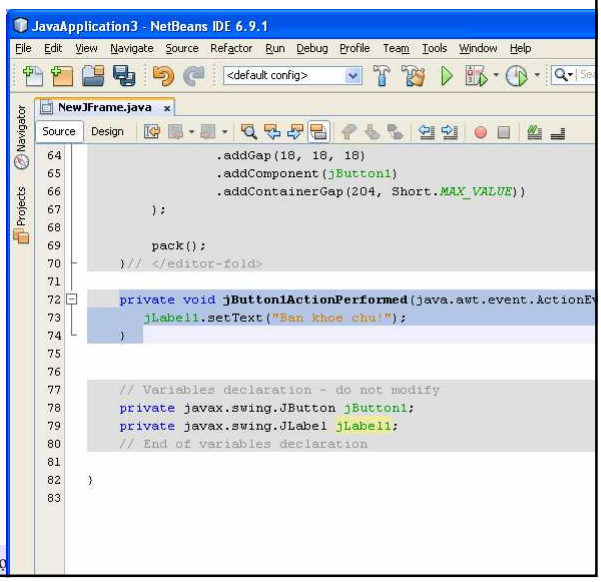
Tạo ứng dụng swing bằng NetBeens

- Tạo và thiết lập cửa sổ/đối tượng chứa
- Thêm các thành phần vào cửa sổ
- Thêm sự kiện và mã xử lý cho các thành phần trên cửa sổ
- Hiển thị cửa sổ



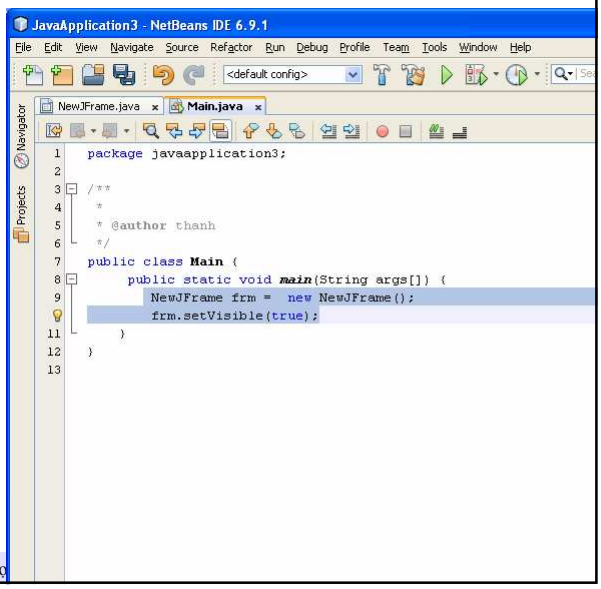
Tạo ứng dụng swing bằng NetBeens

- Tạo và thiết lập cửa sổ/đối tượng chứa
- Thêm các thành phần vào cửa sổ
- Thêm sự kiện và mã xử lý cho các thành phần trên cửa sổ
- Hiển thị cửa sổ



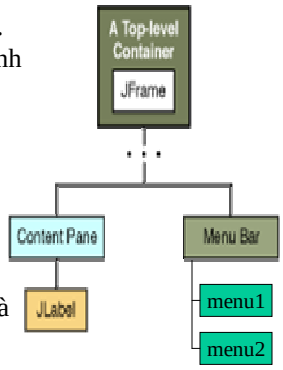
Tạo ứng dụng swing bằng NetBeens

- Tạo và thiết lập cửa sổ/đối tượng chứa
- Thêm các thành phần vào cửa sổ
- Thêm sự kiện và mã xử lý cho các thành phần trên cửa sổ
- Hiển thị cửa sổ



Một vài lưu ý

- Có 03 loại cửa sổ là khung (JFrame) (cửa sổ chính), hộp thoại (JDialog) và JApplet (applet). Chức năng chính của cửa sổ là để chứa các thành phần giao diện khác
- Cửa sổ không thể chứa cửa sổ khác (top-level)
- Các thành phần khác muốn hiển thị được phải được đặt trên cửa sổ
- Một số thành phần có thể làm vật chứa, ví dụ một toolbar chứa các button, textbox, …
- Một thành phần (trừ cửa sổ) được chứa trong và chỉ trong một thành phần khác
  - Quan hệ giữa các thành phần trên một giao diện tạo thành một cây
- Các thành phần đều được kế thừa từ JComponent



Sử dụng frame

- Frame là cửa sổ chính. Một ứng dụng GUI thường có ít nhất một frame.
- Frame có thanh tiêu đề (title) và viền (border) cùng một số nút thay đổi kích thước, đóng cửa sổ. Mằm trong viền và dưới thanh tiêu đề là vùng nội dung. Các đối tượng giao diện như nút bấm, ô văn bản, nút chọn, … được đặt vào vùng nội dung

Các thành phần giao diện

- Mỗi thành phần giao diện
  - được định nghĩa bởi một lớp
  - có các phương thức để thiết lập/đọc giá trị các thuộc tính
  - có các sự kiện liên quan bàn phím, chuột
  - Ghi chú: Các hàm xử lý sự kiện cho các điều khiển được thêm bằng cách kích chuột phải vào điều khiển, chọn Events… rồi chọn sự kiện.

JLabel

- Chức năng: Hiển thị văn bản
- Các phương thức chính
  - setText(String t): Đặt nội dung cho nhãn
  - String getText(): Đọc nội dung nhãn

JTextField

- Chức năng: Nhập một dòng văn bản
- Các phương thức chính
  - setText(String t): Đặt nội dung cho ô chữ
  - String getText(): Đọc nội dung ô chữ
- Sự kiện chính
  - keyReleased: nhả phím
  - focusLost: hết focus

JPassword

- Chức năng: Nhập mật khẩu
- Các phương thức chính
  - char [] pwd.getPassword(): Đặt nội dung password
- Sự kiện chính
  - keyReleased: nhả phím
  - focusLost: hết focus

JTextArea

- Chức năng: Nhập nhiều dòng văn bản
- Các phương thức chính
  - setText(String): Đặt nội dung văn bản
  - String getText(): Đọc nội dung văn bản
  - setLineWrap(boolean): Cho phép hiển thị một dòng văn bản trên nhiều hàng (gấp hàng)
  - setWrapStyleWord(boolean): Có được bẻ từ khi gấp hàng hay không
- Sự kiện chính
  - keyReleased: nhả phím

JTextPane

- Chức năng: Hiển thị nội dung văn bản có định dạng và cho nhập nhiều dòng văn bản
- Chứa bên trong đối tượng StyledDocument
  - getStyledDocument(): trả lại đối tượng StyledDocument được chứa bên trong
- StyledDocument
  - Chứa danh sách các styles
  - Chứa nội dung. Nộ dung được chia thành nhiều đoạn, mỗi đoạn áp dụng một style
  - …

JTextPane

- StyledDocument
  - …
  - Lấy kiểu mặc định:
    - Style kmd = StyleContext.getDefaultStyleContext(). getStyle(StyleContext.DEFAULT_STYLE);
  - Thêm kiểu:
    - Style k1 = doc.addStyle("kieu1", kmc);
    - Style k2 = doc.addStyle("kieu2", k1);
  - Xác định thuộc tính từng kiểu:
    - StyleConstants.setFontFamily(k1, "SansSerif");
    - StyleConstants.setItalic(k1, true);
    - StyleConstants.setFontSize(k2, 30);
  - Chèn nội dung có kiểu
    - doc.insertString(doc.getLength(), “Văn bản có kiểu 1", doc.getStyle("kieu1"));
    - doc.insertString(doc.getLength(), “Văn bản có kiểu 2", doc.getStyle("kieu2"));

JEditorPane

- Chức năng: Hiển thị nội dung một tài liệu HTML và cho nhập nhiều dòng văn bản
- Phương thức:
  - setText(String): Đặt nội dung văn bản
  - String getText(): Đọc nội dung văn bản
  - setPage(URL): Load tài liệu HTML

```
java.net.URL url = APPView.class.getResource("thu.html");
 try {
            jEditorPane1.setPage(url);
      } catch (Exception e) {
            System.err.println("Attempted to read a bad URL: " +
       helpURL);
      }
```


JCheckBox

- Các phương thức chính
  - boolean isSelected(): Kiểm tra nút được tích hay không
- Sự kiện chính
  - mouseClicked: kích chuột, thay đổi trạng thái tích

JRadioButton

- Cần tạo nhóm cho các radio button để chỉ được chọn một trong cả nhóm
  - Tạo JGroupButton rồi đặt thuộc tính groupButton cho JRadioButton
- Các phương thức chính
  - boolean isSelected(): Kiểm tra nút được tích hay không
- Sự kiện chính
  - mouseClicked: kích chuột, thay đổi trạng thái tích

JComboBox

- Cho chọn một phần tử trong danh sách thả xuống
- Các phương thức chính
  - JComboBox(String []): Tạo đối tượng combo với danh sách các mục chọn trong String[]
  - setSelectedIndex(i): Chọn mục i
  - int getSelectedIndex(): Trả lại chỉ mục của mục được chọn
  - Object getSelectedValue(): Trả lại mục được chọn
  - setEditable(boolean): Cho biên soạn hay không
  - setModel(Model): Đặt model
  - getModel(): Lấy model
- Sự kiện chính
  - actionPerformed: thay đổi mục chọn

JList

- Cho một hoặc nhiều phần tử trong danh sách
- Các phương thức chính
  - JList(DefaultListModel): Tạo đối tượng list với danh sách các mục chọn trong model
  - setLayoutOrientation(JList.___): Đặt hướng sắp xếp danh sách
  - setVisibleRowCount(int): Đặt số hàng nhìn thấy
  - setSelectedIndex(i): Chọn mục I
  - setSelectedIndices(int[]): Chọn các mục
  - setSelectionInterval(index, size): Chọn các mục liên tục trong khoảng
  - ensureIndexIsVisible(i): Điều chỉnh hiển thị để mục I được nhìn thấy
  - setSelectionMode(ListSelectionModel.MODE): Đặt kiểu chọn các mục
  - int getSelectedIndex(): Trả về chỉ mục của mục được chọn
  - int[] getSelectedIndices(); Trả về chỉ mục của các mục được chọn
  - Object getSelectedValue(): Trả về giá trị của mục được chọn
  - Object[] getSelectedValues(): Trả về giá trị của các mục được chọn
  - ListModel:
    - addElement(obj), add(index, obj): Thêm mục
    - remove(index): Xóa mục
- Sự kiện chính
  - actionPerformed: thay đổi mục chọn
  - mouseClicked: Kích chuột

JTable

- Trình bày dữ liệu theo bảng
- Nội dung của bảng được quản lý bởi TableModel. TableModel bao gồm
  - Một mảng String[] chứa tiêu đề của các cột
  - Một mảng Object[][] chứa các đối tượng là nội dung của các ô. Số cột của mảng hai chiều này phải bằng số cột (số phần tử của mảng chứa tiêu đề)
  - Các phương thức để lấy số hàng, số cột, tên cột, đối tượng ở một ô, xác định các ô có thể biên tập được
- Để tạo bảng
  - Tạo TableModel
  - Tạo đối tượng JTable sử dụng TableModel đã tạo
    - JTable tạo hàng tiêu đề với số ô bằng số phần tử của mảng tiêu đề và lần lượt có các tiêu đề của mảng tiêu đề
    - Với mỗi phần tử [i][j] trong mảng đối tượng
      - JTable sử dụng Renderer để hiển thị đối tượng tại ô (i, j)
      - JTable sử dụng Editor để cập nhật đối tượng

TableModel và JTable

```
class MyTableModel extends AbstractTableModel {
    private String[] a = {"First Name",  "Last Name", "Sport",  "# of Years",  "Vegetarian",};
    private Object[][] data2 = {
        {"Kathy", "Smith",  "Snowboarding", new Integer(5), new Boolean(false)},
        {"John", "Doe",  "Rowing", new Integer(3), new Boolean(true)},
        {"Sue", "Black", "Knitting", new Integer(2), new Boolean(false)},
        {"Jane", "White", "Speed reading", new Integer(20), new Boolean(true)},
        {"Joe", "Brown",  "Pool", new Integer(10), new Boolean(false)}
    };
…
}
```

Renderer cho String | Renderer cho Integer | Renderer cho Boolean

Editor cho String | Editor cho Integer | Editor cho Boolean

Renderer và Editor mặc định

```
class MyTableModel extends AbstractTableModel {
…
    public Class getColumnClass(int c) {
        return getValueAt(0, c).getClass();
    }
}
```

JTable sẽ sử dụng các Renderer/Editor mặc định cho từng lớp

Renderer cho String | Renderer cho Integer | Renderer cho Boolean

Editor cho String | Editor cho Integer | Editor cho Boolean

Định nghĩa và sử dụng Renderer

Định nghĩa

```
DefaultTableCellRenderer renderer = new DefaultTableCellRenderer();
    renderer.setBackground(Color.red);
```

hoặc

```
public class MyRenderer  implements TableCellRenderer {}
MyRenderer renderer = new MyRenderer();
renderer.set___();
```

Sử dụng

cho từng cột

```
table.getColumnModel().getColumn(3).setCellRenderer(renderer);
```

hoặc cho cả bảng

```
table.setDefaultRenderer(Class.class, renderer);
```


Ví dụ sử dụng Renderer

```
DefaultTableCellRenderer renderer = new
    DefaultTableCellRenderer();
renderer.setForeground(Color.red);

    table.getColumnModel().getColumn(1).setCellRenderer(re
    nderer);
```

Ví dụ sử dụng Renderer

```
table.setDefaultRenderer(Color.class, new ColorRenderer());

public class ColorRenderer extends JLabel
                    implements TableCellRenderer {
   public ColorRenderer() {
      setOpaque(true);
   }
   public Component getTableCellRendererComponent(
                        JTable table, Object obj,
                        boolean isSelected, boolean hasFocus,
                        int row, int column) {
      setBackground((Color)obj);
      return this;
   }
}
```


Định nghĩa và sử dụng Editor

Định nghĩa

```
DefaultTableCellEditor editor = new DefaultTableCellEditor(object);
```

hoặc

```
public class MyEditor  extends AbstractCellEditor implements
      TableCellEditor {}
MyEditor editor = new MyEditor();
```

Sử dụng

cho từng cột

```
table.getColumnModel().getColumn(3).setCellEditor(editor);
```

hoặc cho cả bảng

```
table.setDefaultEditor(Class.class, editor);
```

Ví dụ sử dụng Editor

```
JComboBox comboBox = new JComboBox();
comboBox.addItem("1");
comboBox.addItem("2");
comboBox.addItem("3");
comboBox.addItem("4");
DefaultCellEditor editor =  new DefaultCellEditor(comboBox);

    table.getColumnModel().getColumn(1).setCellEditor(edito
    r);
```

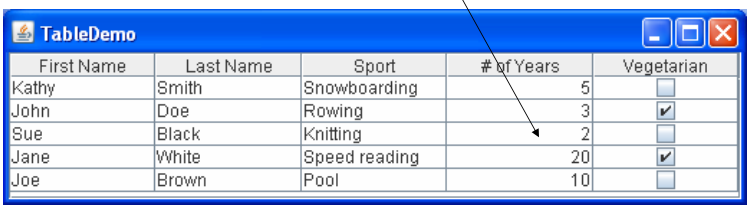

Ví dụ sử dụng Editor

```
table.setDefaultEditor(Color.class, new ColorEditor());
public class ColorEditor extends AbstractCellEditor
                implements TableCellEditor,  ActionListener {
…
 public Object getCellEditorValue() {
      return currentColor;
   }

public Component getTableCellEditorComponent(JTable table,
                                Object value,
                                boolean isSelected,
                                int row,
                                int column) {
      currentColor = (Color)value;
      return button;
   }
}
```

Cập nhật ô

```
class MyTableModel extends AbstractTableModel {
…
        public boolean isCellEditable(int row, int col) {
            …
        }
}
```

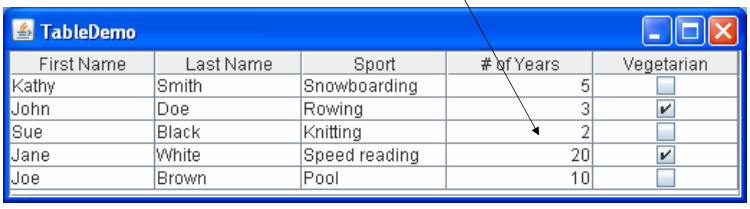
Biên tập được đối tượng trong ô (row, col) hay không tùy thuộc vào isCellEditable(int row, int col) trả về true hay false



Cập nhật ô

```
class MyTableModel extends AbstractTableModel {
…
        public void setValueAt(Object value, int row, int col) {
            data[row][col] = value;
            fireTableCellUpdated(row, col);
        }
}
```

Lưu cập nhật được nếu phương thức setValueAt được cài đặt



Sắp xếp các hàng theo giá trị cột

```
table.setAutoCreateRowSorter(true);
```


Tạo bộ lọc

```
//Tạo bảng có sorter.
        MyTableModel model = new MyTableModel();
        sorter = new TableRowSorter<MyTableModel>(model);
        table.setRowSorter(sorter);
//Đặt bộ lọc
RowFilter<MyTableModel, Object> rf = null;
        //If current expression doesn't parse, don't update.
        try {
            rf = RowFilter.regexFilter("xâu con", col);
        } catch (java.util.regex.PatternSyntaxException e) {
            return;
        }
        sorter.setRowFilter(rf);
```

Đặt chế độ chọn ô

Chế độ chọn table.setSelectionMode(mode);
Chọn một mục
ListSelectionModel.SINGLE_SELECTION
Một khoảng
ListSelectionModel.SINGLE_INTERVAL_SELECTION
Nhiều khoảng
ListSelectionModel.MULTIPLE_INTERVAL_SELECTION
Mục chọn
Cả hàng
table.setRowSelectionAllowed(true);
table.setColumnSelectionAllowed(false);
Cả cột
table.setRowSelectionAllowed(false);
table.setColumnSelectionAllowed(true);
Từng ô
table.setCellSelectionEnabled(true);
Ghi chú: Nếu chọn hành thì không được chọn cột. Chọn ô chỉ KHÔNG áp dụng cho nhiều khoảng

Một số phương thức khác

```
class MyTableModel extends AbstractTableModel {
    …
  public int getColumnCount() {
    return a.length;
  }

  public int getRowCount() {
    return data2.length;
  }

  public String getColumnName(int col) {
    return a[col];
  }

  public Object getValueAt(int row, int col) {
    return data2[row][col];
  }
}
```

JFrame và JDialog

- Đặt look and feel:

  JFrame.setDefaultLookAndFeelDecorated(true);

  JDialog.setDefaultLookAndFeelDecorated(true);
- Hiển thị hộp thoại

  JOptionPane.showConfirmDialog(parent, text, title, JOptionPane._____OPTION);

  Các hộp thoại hiển thị modal và trả về giá trị nút được bấm

JMenu

- JMenuBar – Thanh menu để chứa các menu
- JMenu – Thực đơn (được đặt trong JMenuBar)
- Các khoản thực đơn (được đặt trong JMenu)
  - JMenuItem – Một khoản thường
  - JRadioButtonMenuItem – Một khoản có nút radio
  - JCheckBoxMenuItem – Một khoản có nút check
  - Separator – Phân cách các khoản
  - Memu chứa menu con
- Sự kiện trên các khoản thực đơn
  - actionPerformed
- Kiểm tra radio hay checkbox được chọn trên menu item
  - bool isSelected()

JPopupMenu

- Menu được hiển thị khi kích chuột phải, còn gọi là menu ngữ cảnh
- Cần xử lý các sự kiện MousePressed và MouseReleased trên đối tượng hiển thị popup menu

```
private void mainPanelMousePressed(java.awt.event.MouseEvent evt) {
    maybeShowPopup(evt);
}

private void mainPanelMouseReleased(java.awt.event.MouseEvent evt) {
    maybeShowPopup(evt);
}
private void maybeShowPopup(java.awt.event.MouseEvent e) {
    if (e.isPopupTrigger()) {
        jPopupMenu1.show(e.getComponent(),
                e.getX(), e.getY());
    }
}
```

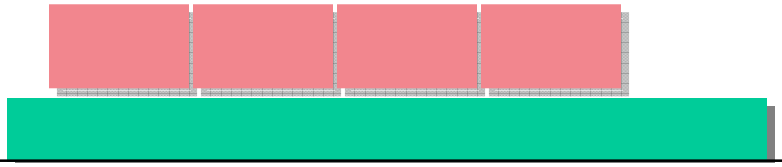

JPopupMenu

- Thêm actionPerformed cho các khoản của popup menu

```
menuItem.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jMenuItem1ActionPerformed(evt);
        }
    });

private void jMenuItem1ActionPerformed(java.awt.event.ActionEvent
evt) {
}
```

Tiếp theo

Lập trình socket



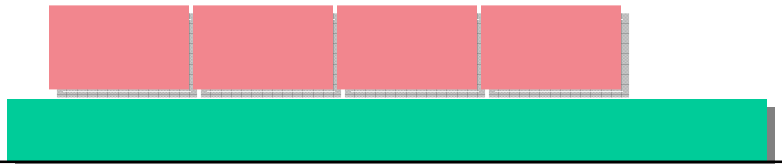
Bài giảng

LẬP TRÌNH JAVA

Lê Đình Thanh
Bộ môn Mạng và Truyền thông Máy tính
Khoa Công nghệ Thông tin
Trường Đại học Công nghệ, ĐHQGHN

Bài 9

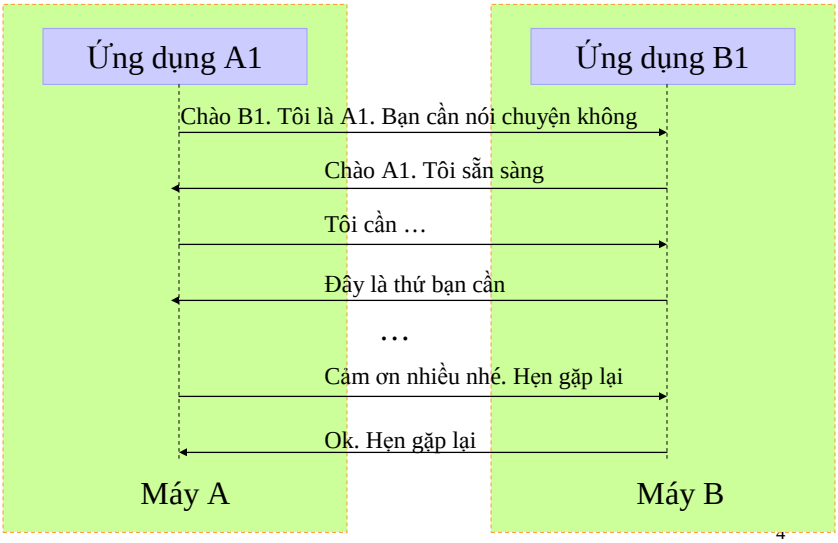
Lập trình socket



Nội dung

- Truyền thông giữa các ứng dụng
- Socket
- Lập trình socket
- Đa luồng

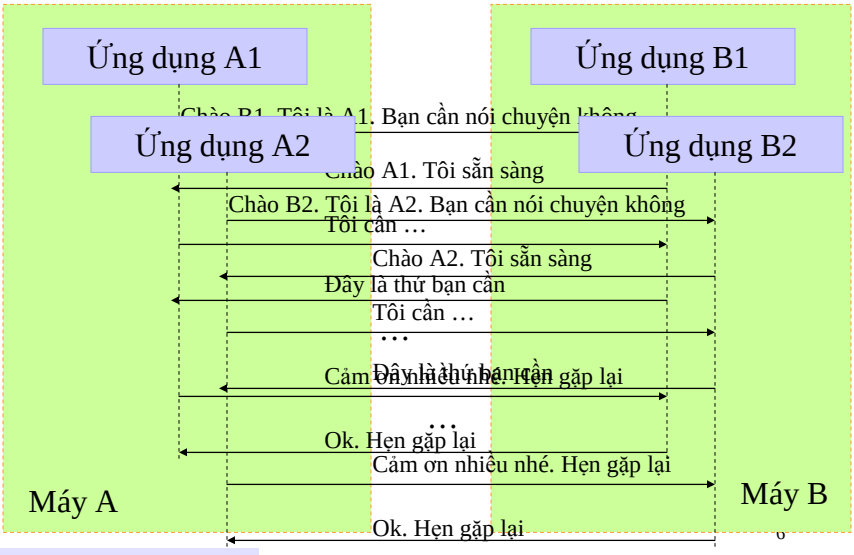
Truyền thông giữa các ứng dụng



Truyền thông giữa các ứng dụng

- Một ứng dụng (B1) phải chạy trước, chờ xem có ứng dụng khác cần giao tiếp không
- Ứng dụng cần giao tiếp (A1) thiết lập kết nối đến ứng dụng lắng nghe (B1)
- Hai bên bàn thảo về một việc gì đó (theo giao thức)
- A1/B1 muốn kết thúc sẽ chào tạm biệt trước, B1/A1 sẽ tạm biệt lại

Truyền thông giữa các ứng dụng



Truyền thông giữa các ứng dụng

- Nhiều ứng dụng có thể chạy đồng thời trên một máy, sử dụng cùng địa chỉ IP → **Làm thế nào để phân biệt được gói tin đến gửi cho ứng dụng nào**?
- Giải pháp: Mỗi ứng dụng sử dụng một mã riêng biệt là một số nguyên không âm được gọi là cổng (port).
  - Khi ứng dụng bên ngoài cần gửi cho ứng dụng B1, nó phải gửi cả IP của máy B và mã nhận diện của B1 trong gói dữ liệu.
  - Khi ứng dụng bên ngoài cần gửi cho ứng dụng B2, nó phải gửi cả IP của máy B và mã nhận diện của B2 trong gói dữ liệu.
  - …
- **IP:port** - Địa chỉ/định danh/**socket** của ứng dụng

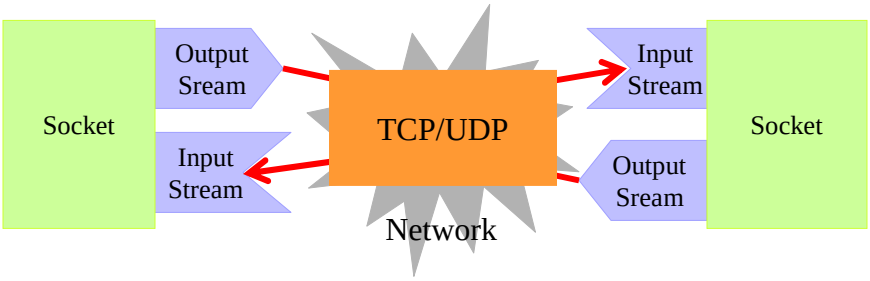
Socket

- **IP:port** - Địa chỉ/định danh/**socket** của ứng dụng
- Khi ứng dụng sử dụng cổng nào, nó sẽ đăng ký với hệ điều hành
- Nếu cổng đã được sử dụng, hệ điều hành sẽ báo là không thể sử dụng được, ứng dụng sẽ dừng chạy hoặc xin cổng khác.
- Trong lập trình java:
  - Ứng dụng lắng nghe sử dụng ServerSocket và Socket
  - Ứng dụng khởi động kết nối sử dụng Socket

Lập trình socket

Đối tượng Socket

- Mỗi đối tượng Socket có:
  - Một dòng ra: lưu dữ liệu sẽ gửi cho đối tác nhưng chưa được gửi
  - Các phương thức gửi dữ liệu = đưa dữ liệu ra dòng ra, đẩy dữ liệu cho tầng giao vận (gửi)
  - Một dòng vào: lưu dữ liệu do đối tác gửi đến
  - Các phương thức nhận dữ liệu = đọc dữ liệu từ dòng vào



ServerSocket

```
//Tạo server socket lắng nghe trên cổng 8080
    ServerSocket ssoc = new ServerSocket(8080);
//Chấp nhận kết nối
    Socket soc = ssoc.accept();
```

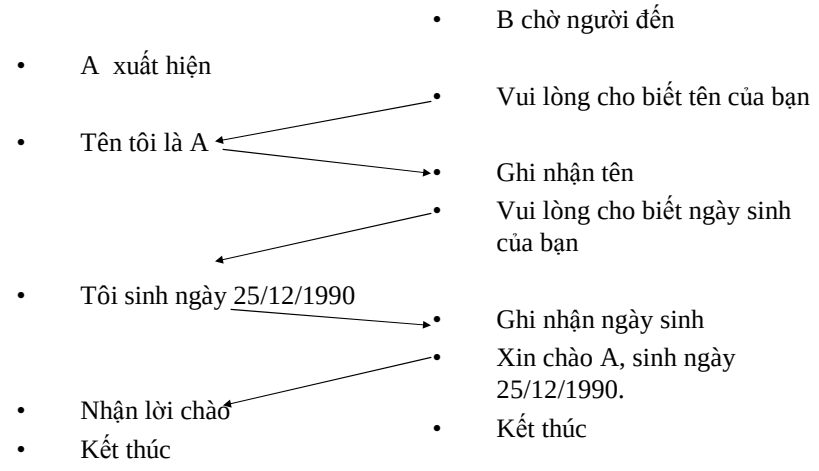

Socket

```
//Tạo kết nối đến server socket lắng nghe trên cổng 8080 ở máy
    có IP 113.202.99.8
      Socket soc = new Socket("113.202.99.8", 8080);
//Tham chiếu đến dòng vào và dòng ra của socket
    InputStream sin = soc.getInputStream();
      OutputStream sout = soc.getOutputStream();
      DataInputStream in = new DataInputStream(sin);
      DataOutputStream out = new DataOutputStream(sout);
//Gửi dữ liệu cho đối tác
    out.writeUTF("Nội dung");
      out.flush();
//Đọc dữ liệu do đối tác gửi đến
    String text = in.readUTF();
```

Giao thức tầng ứng dụng

- Một ứng dụng (B1) phải chạy trước, chờ xem có ứng dụng khác cần giao tiếp không
- Ứng dụng cần giao tiếp (A1) thiết lập kết nối đến ứng dụng lắng nghe (B1)
- **Hai bên bàn thảo về một việc gì đó (theo giao thức)**
- A1/B1 cần kết thúc sẽ chào tạm biệt trước, B1/A1 sẽ tạm biệt lại
- **Giao thức cho ứng dụng cho biết thứ tự, kiểu nội dung các bước gửi/nhận của mỗi bên**

Ví dụ: Giao thức xưng danh



Phía ghi danh (B)

```
//lang nghe tren cong 8080
ServerSocket ssoc = new ServerSocket(8080);
System.out.print("Running");
//chap nhan ket noi
Socket soc = ssoc.accept();
//tham chieu den cac dong vao va dong ra
InputStream sin = soc.getInputStream();
OutputStream sout = soc.getOutputStream();
DataInputStream in = new DataInputStream(sin);
DataOutputStream out = new DataOutputStream(sout);
////////////////////////////   thuc hien giao thuc xung danh    ////////////////////////////
//yeu cau cho biet ten
out.writeUTF("Your name: ");
out.flush();
//ghi nhan ten
String name = in.readUTF();
//yeu cau cho biet ngay sinh
out.writeUTF("Your DOB: ");
out.flush();
//ghi nhan ngay sinh
String dob = in.readUTF();
//thong bao da ghi nhan xong
out.writeUTF("Welcome " + name + " whose DOB is " + dob + "\n");
out.flush();
//ket thuc
in.close();
out.close();
soc.close();
```


Phía xưng danh (A)

```
//thiet lap ket noi
Socket soc = new Socket("127.0.0.1", 8080);
//tham chieu den cac dong vao va dong ra
InputStream sin = soc.getInputStream();
OutputStream sout = soc.getOutputStream();
DataInputStream in = new DataInputStream(sin);
DataOutputStream out = new DataOutputStream(sout);
////////////////////////////    thuc hien giao thuc xung danh  ////////////////////////////
//doc yeu cau (cho biet ten)
String text = in.readUTF();
System.out.print(text);
//mo mot input stream cho ban phim
Scanner scanner = new Scanner(new BufferedReader(new InputStreamReader(System.in)));
scanner.useDelimiter("\\n");
//nhap ten tu ban phim
String name = scanner.next();
//gui ten
out.writeUTF(name);
out.flush();
//doc yeu cau nhap ngay sinh
text = in.readUTF();
System.out.print(text);
//nhap ngay sinh tu ban phim
String dob = scanner.next();
//gui ngay sinh
out.writeUTF(dob);
out.flush();
//doc thong bao da ghi nhan
text = in.readUTF();
System.out.print(text);
//ket thuc
```

Đa luồng (Multi-threading)

Đa luồng

- Một server có thể phải giao tiếp đồng thời với nhiều clients
- Với mỗi client, server phải chấp nhận một kết nối và thực hiện giao thức ứng dụng trên một luồng
- → server phải tao nhiều luồng xử lý đồng thời, mỗi luồng thực hiện một kết nối đến một client

Phía ghi danh – Tạo luồng

```
public class MyThread extends Thread
{
    private Socket soc;

    public MyThread(Socket socket)
    {
        super("MyThread");
        this.soc = socket;
    }

    …
```


Phía ghi danh – Tạo luồng

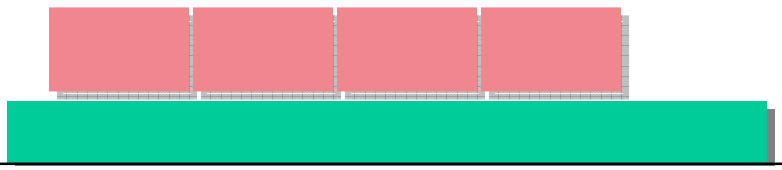
```
    public void run()
    {
     try {
          //tham chieu den cac dong vao va dong ra
          InputStream sin = soc.getInputStream();
          OutputStream sout = soc.getOutputStream();
          DataInputStream in = new DataInputStream(sin);
          DataOutputStream out = new DataOutputStream(sout);
          ////////////// thuc hien giao thuc xung danh //////////////////////
          //yeu cau cho biet ten
          out.writeUTF("Your name: ");
          out.flush();
          //ghi nhan ten
          String name = in.readUTF();
          //yeu cau cho biet ngay sinh
          out.writeUTF("Your DOB: ");
          out.flush();
          //ghi nhan ngay sinh
          String dob = in.readUTF();
          //thong bao da ghi nhan xong
          out.writeUTF("Welcome " + name + " whose DOB is " + dob + "\n");
          out.flush();
          //ket thuc
          in.close();
          out.close();
          soc.close();
     }
```

Phía ghi danh – Chương trình chính

```
public static void main(String[] args) {
        try
        {
           java.net.ServerSocket ssoc = new ServerSocket(8080);
           System.out.print("Running");
           int count = 0;
           while (true)
           {
              Socket soc = ssoc.accept();
              MyThread newthread = new MyThread(soc);
              newthread.start();
              count ++;
              System.out.println("\nClient: " + count);
           }
        }
        catch(Exception e)
        {
           System.out.print(e);
        }
    }
```


Tiếp theo

Triển khai ứng dụng



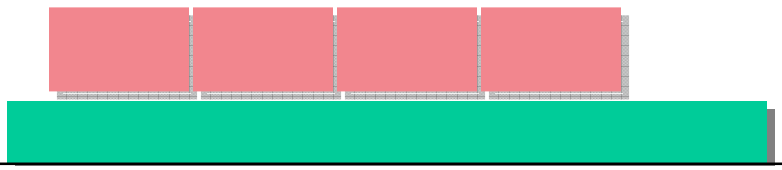
Bài giảng

LẬP TRÌNH JAVA

Lê Đình Thanh
Bộ môn Mạng và Truyền thông Máy tính
Khoa Công nghệ Thông tin
Trường Đại học Công nghệ, ĐHQGHN

Bài 9

Triển khai ứng dụng



Nội dung

- Tệp jar
- Java Network Launch Protocol (JNLP)
- Java Web Start
- Applet

Tệp jar

Đóng gói chương trình với tệp jar

- Để triển khai ứng dụng được viết bằng java, người ta thường đóng gói các tệp chương trình (.class) cùng các tệp tài nguyên cần cho chương trình dưới dạng tệp JAR (Java Archive)
- Chương trình **jar.exe** trong **jdk/bin** cung cấp các chức năng tạo lập tệp jar
  - Tạo tệp jar: **jar cf *jar-file input-file(s)***
  - Xem nội dung tệp jar: **jar tf *jar-file***
  - Giải nén tệp jar: ***jar xf jar-file***
  - Giải nén một vài tệp trong tệp jar: **jar xf *jar-file archived-file(s)***

Chạy chương trình với tệp jar

- Dòng lệnh (cần Main-class manifest)

  java -jar *app.jar*
- Sử dụng Java Web Start
- Chạy applet

jar manifest

- Tệp manifest là một tệp văn bản mô tả về các tệp được đóng gói trong tệp jar
- Khi tạp tệp jar, tệp manifest được tạo tự động và gói vào jar với nội dung

  Manifest-Version: 1.0

  Created-By: 1.6.0 (Sun Microsystems Inc.)
- Sửa nội dung tệp manifest

jar cf**m** *jar-file* ***manifest-addition-file*** *input-file(s)*

trong đó ***manifest-addition-file*** là tệp văn bản chứa nội dung được đưa thêm vào manifest

Thiết lập Main-class manifest

- Sửa nội dung tệp manifest

jar cf**m** *jar-file* ***manifest-addition-file*** *input-file(s)*

trong đó ***manifest-addition-file*** là tệp văn bản chứa nội dung

Main-Class: MyPackage.MyClass

Thiết lập thông tin phiên bản

- Sửa nội dung tệp manifest

  jar cf**m** *jar-file* ***manifest-addition-file*** *input-file(s)*

  trong đó ***manifest-addition-file*** là tệp văn bản chứa nội dung về

  Name:

  Specification-Title:

  Specification-Version:

  Specification-Vendor:

  Implementation-Title:

  Implementation-Version:

  Implementation-Vendor:

Java Network Launch Protocol (JNLP)

JNLP

- Để triển khai ứng dụng qua mạng, java sử dụng JNLP
- Các bước triển khai
  - Tạo tệp.jnlp (dạng XML) mô tả về tệp .jar chứa mã chương trình và các tài nguyên cần thiết
  - Tạo liên kết trên trên trang web trỏ tới tệp .jnlp, Java Web Start sẽ download và chạy chương trình được mô tả trong tệp .jnlp
  - Sử dụng applet với mô tả .jnlp

Nội dung tệp jnlp cho ứng dụng

```
<?xml version="1.0" encoding="UTF-8"?>
<jnlp spec="1.0+“>
    <information>
        <title>Some Title</title>
        <vendor>Some Team</vendor>
    </information>
    <resources>
        <j2se version="1.6+"
          href="http://java.sun.com/products/autodl/j2se"/>
        <jar href="DynamicTreeDemo.jar" main="true" />
    </resources>
    <application-desc name="Application Name" main-
        class=“PackageName.ClassName" width="300" height="300">
    </application-desc>
</jnlp>
```

Nội dung tệp jnlp cho applet

```
<?xml version="1.0" encoding="UTF-8"?>
<jnlp spec="1.0+">
    <information>
        <title>Dynamic Tree Demo</title>
        <vendor>Dynamic Team</vendor>
    </information>
    <resources>
        <j2se version="1.6+"
          href="http://java.sun.com/products/autodl/j2se" />
        <jar href="DynamicTreeDemo.jar" main="true" />
    </resources>
    <applet-desc name="Applet Name" main-
        class="components.DynamicTreeApplet" width="300"
        height="300">
    </applet-desc>
</jnlp>
```


Java Web Start

Java Web Start

- Java Web Start cung cấp khả năng triển khai ứng dụng java một cách đơn giản, hiệu quả bằng cách download và chạy
  - Client: Cần cài J2SE
  - Server: Cần cho **application/x-java-jnlp-file JNLP** trong MIME type.

Chạy ứng dụng Java Web Start

- Sử dụng tiện ích triển khai ứng dụng

```
<script src="http://www.java.com/js/deployJava.js"></script>
 <script>
    var dir = location.href.substring(0,
      location.href.lastIndexOf('/')+1);
    var url = dir + "webstart-desc.jnlp";
    deployJava.launchButtonPNG  = “button.gif”;
    deployJava.createWebStartLaunchButton(url, '1.6.0');
</script>
```

- Hoặc tạo link đến tệp jnlp

```
<a href="/some/path/Notepad.jnlp">Launch Notepad
Application</a>
```

Có thể download deployJava.js để triển khai tại máy chủ của mình

Applet

Applet

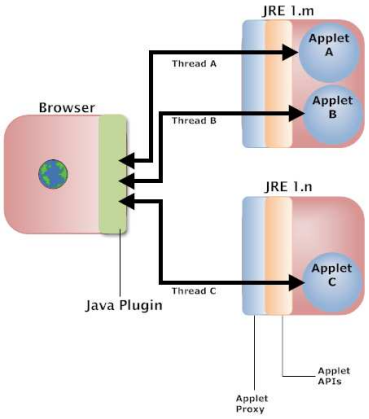
- Applet là ứng dụng java có thể nhúng vào trang web và hiển thị trên trình duyệt
- Một applet chỉ chứa một lớp được kế thừa từ java.applet.Applet hoặc javax.swing.JApplet với các phương thức
  - init() – được gọi khi applet được load lên trang web
  - start() – bắt đầu chạy
  - stop() – không chạy nữa
  - destroy() – được gọi trước khi applet được unload

Tạo applet

```
import javax.swing.JApplet;
import javax.swing.SwingUtilities;
import javax.swing.JLabel;
public class HelloWorld extends JApplet {
public void init() {
    try {
        SwingUtilities.invokeAndWait(
            new Runnable() {
                public void run() {
                JLabel lbl = new JLabel("Hello World"); add(lbl);
                }
            }
        );
    } catch (Exception e) { System.err.println("createGUI didn't complete
            successfully"); } } }
```


Nhúng applet vào trang web

- Applets có thể được chúng vào trang web và tương tác với các đối tượng tài liệu của trang web thông qua javascript
- Để applet chạy được, trình duyệt phải được cài Java Plug-in
  - Browser gửi đoạn mã nhúng applet cho Java Plug-in. Java Plug-in chịu trách nhiệm gọi và thực thi applet



Nhúng applet vào trang web

- Sử dụng thẻ <applet>

```
<applet code=AppletClassName.class
  archive="JarFileName.jar" width=width height=height>
  </applet>
```

hoặc

```
<applet code = 'AppletName' jnlp_href = ‘applet-
  desc.jnlp', width = 300, height = 300 />
```

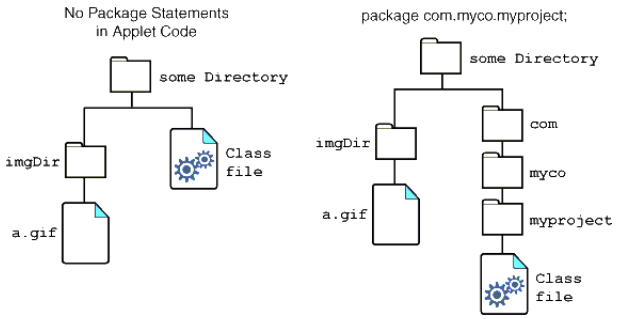
- Sử dụng tiện ích nhúng applet

```
<script src="http://www.java.com/js/deployJava.js"></script>
<script>
        var attributes = { code:'AppletName', width:300,
          height:300} ;
        var parameters = {jnlp_href: ‘applet-desc.jnlp'} ;
        deployJava.runApplet(attributes, parameters, '1.6');
</script>
```


Url của applet

- getCodeBase() - URL của applet
- getDocumentBase() – url của trang web chứa applet
- Ví dụ:

```
Image image = getImage(getCodeBase(), "imgDir/a.gif");
```



Sử dụng tham số với applet

- Truyền tham số
  - Triển khai bằng thẻ applet

```
<applet>
    <PARAM NAME = parameter VALUE = value >
</applet>
```

  - Triển khai bằng jnlp

```
<applet-desc>
  < PARAM NAME = parameter VALUE = value >
</applet-desc>
```

  và

```
var parameters = {jnlp_href: 'applettakesparams.jnlp',
  paramOutsideJNLPFile: 'fooOutsideJNLP'} ;
deployJava.runApplet(attributes, parameters, 1.4);
```

- Nhận và xử lý tham số trong applet

```
paramValue = getParameter("paramName");
```


Hiển thị trạng thái và mở trang web

- Hiển thị trạng thái

```
showStatus("Nội dung hiển thị trên thanh trạng thái");
```

- Mở trang web

```
AppletContext appletContext = getAppletContext() ;
appletContext.showDocument(java.net.URL url);
appletContext.showDocument(java.net.URL url, String
    targetWindow);
```

Truy cập mã javascript từ applet

- Lấy đối tượng window

```
JSObject window = JSObject.getWindow(this);
```

- Truy cập biến javascript

```
userName  = window.getMember("userName");
```

- Cập nhật biến javascript

```
window.setMember("userName", userName);
```

- Gọi hàm javascript

```
Number age = (Number) window.eval("getAge()");
window.call("writeSummary", new Object[] {summary}) ;
```


Truy cập thành viên của applet bằng javascript

- Quy tắc: định danh của applet được dùng như một tham chiếu đến đối tượng applet
-

```
<script
src="http://www.java.com/js/deployJava.js"></script>
<script> var attributes = { id:'mathApplet',
code:'jstojava.MathApplet', width:1, height:1} ; var
parameters = {jnlp_href: 'math-applet.jnlp'} ;
deployJava.runApplet(attributes, parameters, '1.6');
</script>
```

-

```
<script language="javascript">
   mathApplet.userName = "John Doe";
   mathApplet.getGreeting();
</script>
```

Duyệt DOM

```
public void start() {
 try {
    Class c =
        Class.forName("com.sun.java.browser.plugin2.DOM");
    Method m = c.getMethod("getDocument", new Class[] {
        java.applet.Applet.class });
    HTMLDocument doc = (HTMLDocument) m.invoke(null,
        new Object[] { this });
    HTMLBodyElement body = (HTMLBodyElement)
        doc.getBody();
    dump(body, INDENT);
 } catch (Exception e) { System.out.println("New Java Plug-In
     not available");} }
```


Duyệt DOM

```
private void dump(Node root, String prefix) {
    if (root instanceof Element) {
        System.out.println(prefix + ((Element) root).getTagName() + " / " + root.getClass().getName()); }
    else if (root instanceof CharacterData) {
        String data = ((CharacterData) root).getData().trim();
        if (!data.equals("")) { System.out.println(prefix + "CharacterData: " + data); } }
    else { System.out.println(prefix + root.getClass().getName()); }
    NamedNodeMap attrs = root.getAttributes();
    if (attrs != null) {
        int len = attrs.getLength();
         for (int i = 0; i < len; i++) {
        Node attr = attrs.item(i);
        System.out.print(prefix + HALF_INDENT + "attribute " + i + ": " + attr.getNodeName());
        if (attr instanceof Attr) { System.out.print(" = " + ((Attr) attr).getValue()); } System.out.println(); }
    }
    if (root.hasChildNodes()) {
        NodeList children = root.getChildNodes();
        if (children != null) {
        int len = children.getLength();
        for (int i = 0; i < len; i++) { dump(children.item(i), prefix + INDENT); } } }
}
```

Giao tiếp giữa các applets

- Các applets có thể giao tiếp với nhau thông qua javascript
  - Applet1 ←→ javascript ←→ Applet2