



# CHƯƠNG 4. LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG VỚI JAVA





# NỘI DUNG

## LẬP TRÌNH HĐT

*Chương 1: Tổng quan về lập trình HĐT*

*Chương 2: Giới thiệu về lập trình Java*

*Chương 3: Lập trình Java cơ sở*

*Chương 4: Lập trình hướng đối tượng với Java*

*Chương 5: Xử lý ngoại lệ*

- ❖ Đối tượng & Lớp
- ❖ Kế thừa
- ❖ Đa hình
- ❖ Gói (Packages)
- ❖ Giao diện (Interface)
- ❖ Một số lớp tiện ích trong Java



# Đối tượng và lớp

**VD: Đối tượng**

**Một chiếc xe hơi CIVIC:**

Hãng: HONDA

Màu: đen

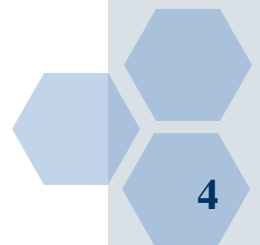
Dung tích xilanh: 2.4

Giá: 30000\$

Chạy theo hướng xác định

Dừng lại

Tự động báo động





# Trừu tượng hoá đối tượng thành lớp

**Một chiếc CIVIC**

**Hãng: HONDA**

**Màu: đen**

**Dung tích xilanh: 2**

**Giá: 30000\$**

**Chạy theo hướng x**

**Dừng lại**

**Tự động báo động**

**- Tên đối tượng: CIVIC**

**- Các đặc điểm:**

Hãng: HONDA

Màu: đen

Dung tích xilanh: 2.4

Giá: 30000\$

**- Các tính năng:**

Chạy theo hướng xác định

Dừng lại

Tự động báo động



# Trình tượng hoá đối tượng thành lớp

- **Tên đối tượng:** CIVIC
- **Các đặc điểm:**
  - Hãng: HONDA
  - Màu: đen
  - Dung tích xilanh: 2.4
  - Giá: 30000\$
- **Các tính năng:**
  - Chạy theo hướng xác định
  - Dừng lại
  - Tự động báo động

- **Tên chung:** Xe hơi
- **Các đặc điểm chung:**
  - Hãng sản xuất:
  - Màu xe:
  - Dung tích xilanh:
  - Giá:
- **Các tính năng chung:**
  - Chạy theo hướng xác định
  - Dừng lại



# Lớp và đối tượng

Lớp	Đối tượng
<ul style="list-style-type: none"><li>- Lớp là mô hình khái niệm, mô tả các thực thể.</li></ul>	<ul style="list-style-type: none"><li>- Đối tượng là sự vật thật, là thực thể thực sự.</li></ul>
<ul style="list-style-type: none"><li>- Lớp như một bản mẫu, định nghĩa các thuộc tính và phương thức chung của các đối tượng.</li></ul>	<ul style="list-style-type: none"><li>- Đối tượng là một thể hiện của một lớp, dữ liệu của các đối tượng khác nhau là khác nhau.</li></ul>
<ul style="list-style-type: none"><li>- Một lớp là sự trừu tượng hoá của một tập các đối tượng.</li></ul>	<ul style="list-style-type: none"><li>- Mỗi đối tượng có một lớp xác định dữ liệu và hành vi của nó.</li></ul>



# Khai báo lớp

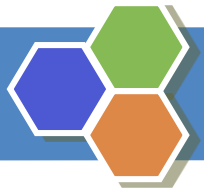
```
public class Car{  
    ...  
}
```

Khai báo phạm vi

Từ khóa cho  
khai báo lớp

Khai báo tên lớp





# Khai báo lớp

```
[Modifier] class <ClassName>
{
    <kiểu dữ liệu> <field_1>;
    <kiểu dữ liệu> <field_2>;
    constructor; // Hàm khởi tạo
    method_1;
    method_2;
    ....
}
```

- ❖ Không có modifier: Mặc định là friendly, cho phép các đối tượng thuộc các class cùng package truy cập.





# Phạm vi truy cập

## ❖ **private:**

- Chỉ truy cập được từ trong lớp khai báo.

## ❖ **protected:**

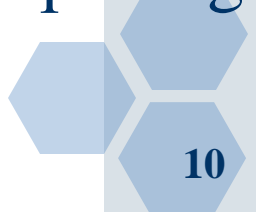
- Truy cập được từ trong lớp khai báo, lớp con của lớp khai báo và các lớp cùng gói với lớp khai báo.

## ❖ **public:**

- Truy cập được từ mọi nơi.

## ❖ **Mặc định:**

- Truy cập được từ trong lớp khai báo và các lớp cùng gói với lớp khai báo





# Phạm vi truy cập

	Cùng gói, cùng lớp	Cùng gói, khác lớp	Lớp con cùng gói với lớp cha	Lớp con khác gói với lớp cha	Khác gói, khác lớp
public	✓	✓	✓	✓	✓
protected	✓	✓	✓	✓	
default (friendly)	✓	✓			
private	✓				



# Ví dụ

**td1.java**

```
1 public class TD1
2 {   public static void main(String args[])
3     {
4     }
5 }
6
```

**Task List**

	description	resource
	class TD1 is public, should be declared in a file named TD1.java	td1.java

**Error**

**Lớp public thì tên file.java phải trùng với tên lớp**  
**Vì khi bên ngoài truy cập, nhìn tên file là biết tên lớp**

**Nếu không là lớp public,**  
**tên lớp và tên file có thể khác nhau**  
**Vì lớp này chỉ dùng trong một gói (package)**



# Khai báo đối tượng

❖ Có ba bước khi tạo một đối tượng từ một lớp:

- **Khai báo:** Một khai báo biến với một tên biến với một loại đối tượng.
- **Cài đặt:** Từ khóa new được sử dụng để tạo đối tượng
- **Khởi tạo:** Từ khóa new được theo sau bởi một lời gọi một constructor. Gọi hàm này khởi tạo đối tượng mới.

❖ Cú pháp

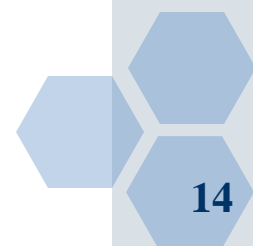
- **<Tên lớp><Tên đối tượng> = new <Hàm khởi tạo>**



# Khai báo đối tượng

```
public class Xecon{

    public Xecon(String ten){
        // Contructor nay co mot tham so la ten.
        System.out.println("Ten xe la :" + ten );
    }
    public static void main(String []args){
        // Lenh sau se tao mot doi tuong la Xecuatoi
        Xecon Xecuatoi = new Xecon( "Toyota" );
    }
}
```

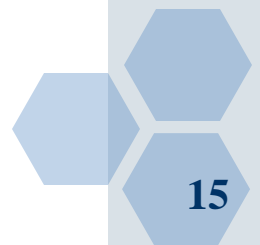




# Một số từ khoá

## ❖ Từ khoá final:

- Áp dụng cho lớp, phương thức, biến.
- Lớp final: là lớp không thể có lớp con
  - **public final class NoChild{...}**
- Biến final: là biến không thể thay đổi khi đã gán giá trị
  - **private final int MAX=100;**
- Phương thức final: là phương thức không thể nạp chồng
  - **public final void NoOverride();**

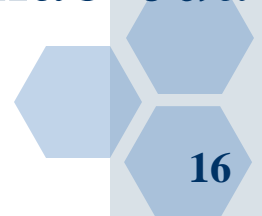




# Một số từ khoá

## ❖ Từ khoá static

- Được dùng với phương thức và biến.
- Biến static: là biến chung cho mọi đối tượng của lớp, nó được truy cập qua đối tượng của lớp hoặc qua tên lớp.
  - **private static char TAB='\t';**
- Phương thức static: là phương thức chỉ được phép truy cập tới các biến static của lớp, nó có thể gọi ngay cả khi chưa có đối tượng nào của lớp.
  - **public static void Welcome() {....}**

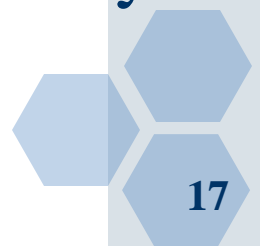






# Thuộc tính của lớp

- ❖ Thuộc tính của lớp là một biến có kiểu dữ liệu bất kỳ, nó có thể là một biến có kiểu là chính lớp đó.
- ❖ Khi khai báo các thành phần của lớp (thuộc tính và phương thức) có thể dùng một trong các từ khoá private, protected, public để giới hạn sự truy cập đến thành phần đó.
- ❖ Các thuộc tính nên để là private để đảm bảo tính giấu kín. Khi đó, để bên ngoài phạm vi của lớp có thể truy cập được đến thành phần private này ta phải tạo ra các phương thức get và set.





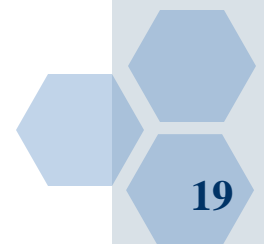
# Phương thức (hàm) của lớp

- ❖ Các phương thức thường khai báo là public, để chúng có thể truy cập từ bất kỳ nơi nào.
- ❖ Trong một chương trình chỉ có 1 lớp được khai báo là public, và tên lớp public này phải trùng với tên của tệp kể cả chữ hoa, chữ thường.



# Khai báo thuộc tính cho lớp

```
public class Car{                // lớp xe ô tô
    private String productora;   // tên nhà sản xuất
    private String color;        // màu xe
    private float capacity;      // dung tích xilanh
    private int price;           // giá xe
}
```





# Phương thức (hàm) của lớp

## ❖ Cú pháp:

[<Phạm vi hoặc thuộc tính kiểm soát truy cập>]<Kiểu trả lại><Tên hàm>([<Danh sách tham biến biến hình thức>])

{

<Nội dung của hàm>

}

Kiểu trả về: Nguyên thủy, kiểu lớp hoặc không có giá trị trả về

Danh sách tham biến biến hình thức: Gồm 1 dãy các tham biến cách nhau dấu phẩy

```
public class Employee{  
    public String fullname;  
    public double salary;
```

Trường



```
    public void input(){  
        Scanner scanner = new Scanner(System.in);  
        System.out.print(" >> Full Name: ");  
        this.fullname = scanner.nextLine();  
  
        System.out.print(" >> Salary: ");  
        this.salary = scanner.nextDouble();  
    }
```

Phương thức

```
    public void output(){  
        System.out.println(this.fullname);  
        System.out.println(this.salary);  
    }  
}
```

Lớp Employee có 2 thuộc tính là fullname và salary và 2 phương thức là input() và output()



# Ví dụ về khai báo thuộc tính

```
public class Rectangle
{
    private float length;
    private float width;

    public Rectangle()
    {
        length = 0;
        width = 0;
    }

    public Rectangle(float l, float w)
    {
        length = l;
        width = w;
    }

    public float area()
    {
        return length * width;
    }
}
```

```
Rectangle recA = new Rectangle(3, 4);
```

```
Rectangle recB = new Rectangle(6.4F, 4.7F);
```

recA

length = 3.0  
width = 4.0

recB

length = 6.4  
width = 4.7



# Ví dụ về khai báo thuộc tính

```
public class Circle
{
    public float radius = 0;
    public static float PI = 3.14F;

    public Circle(float r)
    {
        this.radius = r;
    }
}
```

```
Console
<terminated> Circle [Java Applicatio
C1 radius = 10.3
C1 PI = 3.14

C2 radius = 5.7
C2 PI = 3.14

PI = 3.14
C1 radius = 16.2
C1 PI = 7.8

C2 radius = 5.7
C2 PI = 7.8
```

```
public static void main(String[] args)
{
    Circle c1 = new Circle(10.3F);
    Circle c2 = new Circle(5.7F);

    System.out.println("C1 radius = " + c1.radius);
    System.out.println("C1 PI = " + c1.PI);

    System.out.println("\nC2 radius = " + c2.radius);
    System.out.println("C2 PI = " + c2.PI);

    System.out.println("\nPI = " + Circle.PI);

    c1.radius = 16.2F;
    c1.PI = 7.8F;

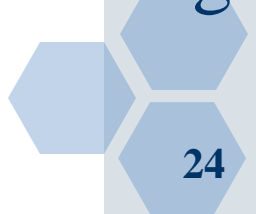
    System.out.println("C1 radius = " + c1.radius);
    System.out.println("C1 PI = " + c1.PI);

    System.out.println("\nC2 radius = " + c2.radius);
    System.out.println("C2 PI = " + c2.PI);
}
```



# Hàm khởi tạo (Constructor)

- ❖ Hàm khởi tạo là một phương thức đặc biệt của lớp.
- ❖ Phải có cùng tên với lớp và được gọi đến với từ khoá new.
- ❖ Dùng để gọi tự động khi khởi tạo 1 thể hiện (đối tượng) của lớp, có thể dùng để khởi gán những giá trị mặc định.
- ❖ Không có giá trị trả về, có thể không có , có một hoặc nhiều hơn một tham số.
- ❖ Nếu một lớp không có constructor thì java sẽ cung cấp cho lớp một constructor mặc định.







# Hàm khởi tạo (Constructor)

- ❖ Những thuộc tính, biến của lớp sẽ được khởi tạo bởi giá trị mặc định (số: là giá trị 0, kiểu logic giá trị false, kiểu đối tượng giá trị null,...).
- ❖ Một lớp có thể có nhiều hơn một constructor.
- ❖ Phương thức constructor có thể bị nạp chồng (overload)

```
public class Rectangle
{
    private float length;
    private float width;

    public Rectangle()
    {
        length = 0;
        width = 0;
    }

    public Rectangle(float l, float w)
    {
        length = l;
        width = w;
    }

    public float area()
    {
        return length * width;
    }
}
```



# Hàm khởi tạo (Constructor)

```
public class Car{                                // lớp xe ô tô
    private String producer; // tên nhà sản xuất
    private String color;    // màu xe
    private float capacity;  // dung tích xilanh
    private int price;       // giá xe

    public Car(){
        producer = "";
        color = "";
        capacity = 0.0;
        price = 0;
    }
}
```



# Gọi hàm tạo từ hàm tạo

- ❖ Trường hợp lớp có nhiều hàm tạo, khi muốn gọi 1 hàm tạo này từ bên trong một hàm tạo khác, để tránh phải viết lặp mã ta sử dụng từ khoá this.
- ❖ Cú pháp: this (danh sách đối số).
- ❖ Chú ý: Khi gọi 1 hàm tạo bên trong 1 hàm tạo khác thì lời gọi hàm tạo phải là lệnh đầu tiên trong thân phương thức, nếu không sẽ bị báo lỗi.



# Gọi hàm tạo từ hàm tạo

```
public class Test {  
    public Test ()  
    {  
        System.out.println("hàm tạo không đối");  
    }  
    public Test ( int i)  
    {  
        this(); // gọi đến hàm tạo không đối của chính lớp này  
    }  
  
    public static void main(String[] args) {  
        TestPassByValue thu=new TestPassByValue(10);  
    }  
}
```



# Gọi hàm tạo từ hàm tạo

❖ **Chú ý: Bên trong hàm tạo ta chỉ có thể gọi tối đa 1 hàm tạo.**

❖ **VD:**

```
public class TestPassByValue {  
    public TestPassByValue() {  
        System.out.println("Day la ham tao khong doi");  
    }  
    public TestPassByValue(int i) {  
        System.out.println("Day la ham tao doi so nguyen");  
    }  
    public TestPassByValue(String s) {  
        this();// không thể gọi hai hàm tạo trở lên bên trong một hàm tạo  
        this(10);  
        System.out.println("Day la ham tao doi so xau");  
    }  
}
```



# Gọi hàm tạo từ hàm tạo

```
public static void main(String[] args) {  
    TestPassByValue thu = new TestPassByValue();  
    TestPassByValue thu1 = new TestPassByValue("Hello World");  
}
```



## Hàm khởi tạo (Constructor)

- ❖ **VD2: Tạo lớp sinh viên với các thuộc tính: maSV, hoTen, diemTb và hai phương thức nhập (nhập thông tin cho 1 sinh viên), xuất (xuất thông tin 1 sinh viên lên màn hình).**
  - Tạo constructor không đối
  - Tạo constructor có 3 tham số truyền vào.
- ❖ **Viết chương trình trong main() khai 1 đối tượng sinh viên gọi đến hàm khởi tạo ko tham số sau đó gọi phương thức nhập,xuất;**
- ❖ **Hàm khởi tạo có đối số sau đó gọi phương thức xuất.**



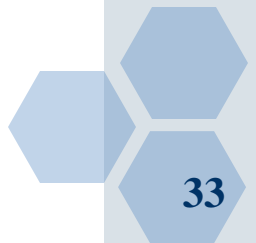
- ❖ **VD3: Tạo lớp nhân viên với các thuộc tính: maNV, hoTen, phòng ban, hệ số lương và hai phương thức nhập (nhập thông tin cho 1 nhân viên), xuất (xuất thông tin 1 nhân viên lên màn hình).**
  - Tạo constructor không đối
  - Tạo constructor có 4 tham số truyền vào.
- ❖ **Viết chương trình trong main() khai 1 đối tượng nhân viên gọi đến hàm khởi tạo ko tham số sau đó gọi phương thức nhập,xuất;**
- ❖ **Hàm khởi tạo có đối số sau đó gọi phương thức xuất.**





# Hàm khởi tạo (Constructor)

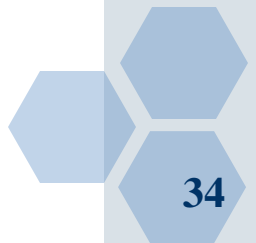
```
class SinhVien {  
    private String hoTen;  
    private int namSinh;  
    private float diemTb;  
    private String lop;  
    public SinhVien() {  
    }  
}
```





## Hàm khởi tạo (Constructor)

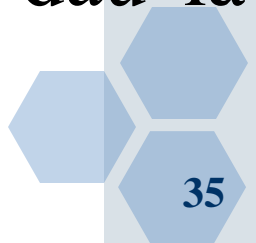
```
public SinhVien(String hoTen, int namSinh,  
String lop) {  
    this.hoTen = hoTen;  
    this.namSinh = namSinh;  
    this.lop = “DH2C4”;  
    System.out.println(“Họ tên: ” + this.hoTen  
+ ” Năm sinh: ” + this.namSinh + “Lớp” +  
this.lop);  
}  
}
```





# Biến this

- ❖ Biến this là một biến ẩn tồn tại trong tất cả các lớp của java.
- ❖ Đại diện cho đối tượng, dùng để truy xuất một thành phần của đối tượng `this.tênThànhPhần`
- ❖ Khi tham số trùng với tên thuộc tính thì nhờ từ khoá this chúng ta có thể phân biệt rõ đâu là thuộc tính, đâu là tham số.





# Biến this

vongtron.java \* SuDungVongTron.java |

```
1 class VONGTRON
2 { protected int x,y,r;
3   int getX() { return x; }
4   public void setX (int xx) { x=xx;}
5   public int getY() { return y; }
6   public void setY (int yy) { y=yy;}
7   public int getR() { return r; }
8   public void setR (int rr) { if (rr>=0) r=rr;}
9   public double getArea () { return Math.PI * r *r; }
10  private double getPerimeter () { return 2*Math.PI * r; }
11  public void OutData()
12  { System.out.println(" " + x + ", " + y + ", " + r);
13  }
14  public void setData (int x, int y, int r)
15  {   this.
16  }
17 }
```

◆ setData (int, int, int)	void
◆ setR (int)	void
◆ setX (int)	void
◆ setY (int)	void
◆ toString ()	String
◆ wait (long, int)	void
◆ wait (long)	void
◆ wait ()	void
◆ x	int
◆ y	int



# Biến this

vongtron.java | SuDungVongTron.java |

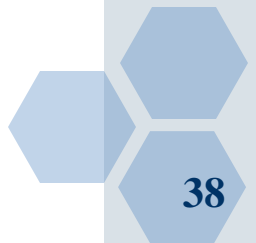
```
1 class WONGTRON
2 { protected int x,y,r;
3   int getX() { return x; }
4   public void setX (int xx) { x=xx;}
5   public int getY() { return y; }
6   public void setY (int yy) { y=yy;}
7   public int getR() { return r; }
8   public void setR (int rr) { if (rr>=0) r=rr;}
9   public double getArea () { return Math.PI * r *r; }
10  private double getPerimeter () { return 2*Math.PI * r; }
11  public void OutData()
12  { System.out.println(" " + x + ", " + y + ", " + r);
13  }
14  public void setData (int x, int y, int r)
15  { this.x=x; this.y=y; this.r=r;
16  }
17 }
```

Truy cập thành phần  
qua từ khóa this

Truy cập thành phần  
không qua từ khóa  
this



```
Class RGBColor{  
    int red, green, blue;  
    RGBColor (int red, int green, int blue){  
        this.red=red;  
        this.green=green;  
        this.blue=blue;  
    }  
}
```





# VD: gọi hành vi trong constructor

Box.java

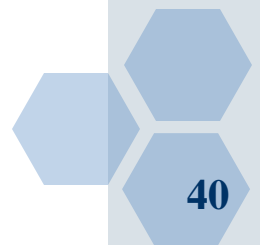
```
1 // constructor demo
2 class Box
3 {   int x,y,z;
4   Box ()
5   {   x=y=z=0;
6       this.outData();
7   }
8   Box (int x, int y, int z)
9   {   this.x=x; this.y=y; this.z=z;
10      this.outData();
11  }
12 void outData()
13 {   System.out.println (" " + x + ", " + y + ", " + z);
14 }
15 public static void main (String args[])
16 {   Box b1= new Box();
17     Box b2= new Box(5,6,7);
18 }
19 }
```

```
C:\Program Files\Xinox Software\J...
0, 0, 0
5, 6, 7
Press any key to continue...
```



# Phương thức (Hành vi- Methods)

- ❖ Phương thức thể hiện
- ❖ Gọi phương thức và truyền tham số kiểu tham trị
- ❖ Gọi phương thức và truyền tham số kiểu tham chiếu
- ❖ Phương thức tĩnh
- ❖ Phương thức tham số biến







# Phương thức của lớp

```
public class Car{                                // lớp xe ô tô
    private String producer; // tên nhà sản xuất
    private String color;    // màu xe
    private float capacity;  // dung tích xilanh
    private int price;       // giá xe

    public void setProducer(String producer){
        this.producer = producer;
    }

    public String getProducer(){
        return producer;
    }
}
```



# Định nghĩa các phương thức get/set

- ❖ Các thuộc tính *private* của một lớp hoặc sẽ không thể truy xuất, hoặc chỉ được phép cho đọc (thông qua phương thức *get*), hoặc chỉ cho ghi (thông qua phương thức *set*) từ các lớp bên ngoài



# Định nghĩa các phương thức get/set

## ❖ Cách khai báo:

- Với *set* nên đặt tên phương thức là *setTênThuộcTính()*.
- Với *get* thì nên đặt tên phương thức là *getTênThuộcTính()*



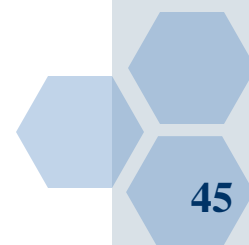
# Định nghĩa các phương thức get/set

```
1 package tenpackage;
2 public class TenLop{
3     //Khai báo các thuộc tính
4     private KieuDuLieu thuocTinh1; . . .
5     private KieuDuLieu thuocTinhN;
6     //Các phương thức khởi tạo đối tượng
7     ...
8     //Các phương thức get/set
9     public KieuDuLieu getThuocTinh1() {
10         return thuocTinh1;
11     }
12     public void setThuocTinh1(KieuDuLieu thuocTinh1) {
13         this.thuocTinh1 = thuocTinh1;
14     } . . .
15     public KieuDuLieu getThuocTinhN() {
16         return thuocTinhN;
17     }
18     public void setThuocTinhN(KieuDuLieu thuocTinhN) {
19         this.thuocTinhN = thuocTinhN;
20     }
21 }
```



# Định nghĩa các phương thức get/set

```
1  public class PhanSo{
2      //Khai báo các thuộc tính
3      private int tuSo;  private int mauSo;
4      //Phương thức khởi tạo đối tượng
5      . . .
6      //Các phương thức get/set
7      public int getTuSo() {
8          return tuSo;
9      }
10     public void setTuSo(int tuSo) {
11         this.tuSo = tuSo;
12     }
13     public int getMauSo() {
14         return mauSo;
15     }
16     public void setMauSo(int mauSo) {
17         if (mauSo != 0) {
18             this.mauSo = mauSo;
19         }
20     }
21 }
```





# Định nghĩa các phương thức get/set

```
1 package bt1;
2
3 public class Main {
4
5     public static void main(String[] args) {
6
7         PhanSo ps1=new PhanSo();
8         System.out.println(ps1.getTuSo()
9                             + "/" + ps1.getMauSo());
10        ps1.setTuSo(3);
11        ps1.setMauSo(4);
12        System.out.println(ps1.getTuSo()
13                            + "/" + ps1.getMauSo());
14
15    }
16 }
```



# Định nghĩa các phương thức get/set

```
1  public class MangPhanSo {
2      //Khai báo thuộc tính
3      private PhanSo[] arr;
4      //Phương thức khởi tạo mặc định
5      . . .
6      public PhanSo[] getArr() {
7          return arr;
8      }
9
10     public void setArr(PhanSo ... arr) {
11         this.arr = new PhanSo[arr.length];
12         for (int i = 0; i < arr.length; i++) {
13             this.arr[i] = new PhanSo(arr[i]);
14         }
15     }
16     . . .
17 }
```



# Định nghĩa các phương thức get/set

```
1      . . .
2      public void set(int index, PhanSo value) {
3          if (index >= 0 && index < this.arr.length) {
4              //this.arr[index] = new PhanSo(value);
5              this.arr[index]=value;
6          }
7      }
8
9      public PhanSo get(int index) {
10         PhanSo ps = null;
11         if (index >= 0 && index < this.arr.length) {
12             //ps=new PhanSo(this.arr[index]);
13             ps = this.arr[index];
14         }
15         return ps;
16     }
17 }
```





# Định nghĩa các phương thức get/set

```
1 public static void main(String[] args) {
2     MangPhanSo mps = new MangPhanSo(
3         new PhanSo(1, 3),
4         new PhanSo(4, 3),
5         new PhanSo(5, 9));
6     //Sử dụng get
7     PhanSo[] arr = mps.getArr();
8     int index = 1;
9     PhanSo ps = mps.get(index);
10    //Sử dụng set
11    PhanSo[] arr2 = new PhanSo[2];
12    arr2[0] = new PhanSo(2, 4);
13    arr2[1] = new PhanSo(3, 7);
14    mps.setArr(arr2);
15    index=0;
16    mps.set(index, new PhanSo(8, 5));
17 }
```



# Định nghĩa các phương thức get/set

```
1 package btl;
2 public class MangSoNguyen {
3     //Khai báo thuộc tính
4     private int[] arr;
5     //Phương thức khởi tạo mặc định
6     . . .
7     public int[] getArr() {
8         return arr;
9     }
10    public void setArr(int ... arr) {
11        this.arr = new int[arr.length];
12        for (int i = 0; i < arr.length; i++) {
13            this.arr[i] = rr[i];
14        }
15    }
16    . . .
17 }
```



# Định nghĩa các phương thức get/set

```
1      . . .
2      public void set(int index, int value) {
3          if (index >= 0 && index < this.arr.length) {
4              this.arr[index]=value;
5          }
6      }
7
8      public Integer get(int index) {
9          Integer value = null;
10         if (index >= 0 && index < this.arr.length) {
11             value = this.arr[index];
12         }
13         return value;
14     }
15 }
```



# Định nghĩa các phương thức get/set

```
1 public static void main(String[] args) {
2     MangSoNguyen mps = new MangSoNguyen( 3, 9, 5);
3     //Sử dụng get
4     int[] arr = mps.getArr();
5     int index = 1;
6     Integer value = mps.get(index);
7     //Sử dụng set
8     int[] arr2 = {9,6,8};
9     mps.setArr(arr2);
10    index=0;
11    mps.set(index, 7);
12 }
```



# Phương thức nhập xuất console

```
1 public class PhanSo{
2     //Khai báo các thuộc tính
3     private int tuSo;
4     private int mauSo;
5     . . .
6     public void nhap(String tieuDe) {
7         System.out.println(tieuDe);
8         Scanner scan = new Scanner(System.in);
9         System.out.print("Tù số:");
10        this.tuSo = Integer.parseInt(scan.nextLine());
11        System.out.print("Mẫu số:");
12        this.mauSo = Integer.parseInt(scan.nextLine());
13    }
14
15    public void xuat(String tieuDe){
16        System.out.println(tieuDe);
17        String str=String.format("%d/%d",this.tuSo, this.mauSo);
18        System.out.println(str);
19    }
20 }
21 }
```



# Phương thức nhập xuất console

```
1 package bt1;
2
3 public class Main {
4
5     public static void main(String[] args) {
6
7         PhanSo ps1=new PhanSo();
8         ps1.nhap("Nhập phân số 1");
9         ps1.xuat("Xuất phân số 1");
10
11         PhanSo ps2=new PhanSo();
12         ps2.nhap("Nhập phân số 2");
13         ps2.xuat("Xuất phân số 2");
14
15     }
16 }
```



# Phương thức nhập xuất console

```
1 package btl;
2 public class MangPhanSo {
3     //Khai báo thuộc tính
4     private PhanSo[] arr;
5     . . .
6     public void nhap(String tieuDe){
7         Scanner scan=new Scanner(System.in);
8         System.out.println(tieuDe);
9         System.out.print("Số lượng phần tử:");
10        int n=Integer.parseInt(scan.nextLine());
11        this.arr=new PhanSo[n];
12        for(int i=0; i<this.arr.length; i++){
13            this.arr[i] = new PhanSo();
14            this.arr[i].nhap("Phần số thứ "+i);
15        }
16    }
17    . . .
18 }
```



# Phương thức nhập xuất console

```
1      . . .
2      public void xuat(String tieuDe){
3          System.out.println(tieuDe);
4          int n = this.arr.length;
5          System.out.println("Số lượng phần tử: " + n);
6          for(int i=0; i<this.arr.length; i++){
7              this.arr[i].xuat("Phần số thứ "+i);
8          }
9      }
10 }
```





# Phương thức nhập xuất console

```
1 public static void main(String[] args) {  
2  
3     MangPhanSo mps=new MangPhanSo();  
4     mps.nhap("Nhập mảng phân số");  
5     mps.xuat("Xuất mảng phân số");  
6  
7 }
```



# Phương thức nhập xuất console

```
1 package btl;
2 public class MangSoNguyen {
3     //Khai báo thuộc tính
4     private int [] arr;
5     . . .
6     public void nhap(String tieuDe){
7         Scanner scan=new Scanner(System.in);
8         System.out.println(tieuDe);
9         System.out.print("Số lượng phần tử:");
10        int n=Integer.parseInt(scan.nextLine());
11        this.arr=new int[n];
12        for(int i=0; i<this.arr.length; i++){
13            System.out.print("Phần tử thứ "+ i + ":");
14            this.arr[i]=Integer.parseInt(scan.nextLine())
15        }
16    }
17    . . .
}
```



# Phương thức nhập xuất console

```
1      . . .
2      public void xuat(String tieuDe) {
3          System.out.println(tieuDe);
4          int n = this.arr.length;
5          System.out.println("Số lượng phần tử: " + n);
6          for(int i=0; i<this.arr.length; i++){
7              System.out.print("Phần số thứ "+i+ ":");
8              System.out.println(this.arr[i]);
9          }
10     }
11 }
```



# Phương thức nhập xuất console

```
1 public static void main(String[] args) {  
2  
3     MangSoNguyen msn=new MangSoNguyen();  
4     msn.nhap("Nhập mảng số nguyên");  
5     msn.xuat("Xuất mảng số nguyên");  
6  
7 }
```



# Phương thức thể hiện (Instance Method)

- ❖ Là hàm định nghĩa trong lớp.
- ❖ Định nghĩa hành vi của đối tượng
- ❖ Cung cấp cách thức truy xuất tới các dữ liệu riêng của đối tượng.
- ❖ Truy xuất thông qua tên đối tượng

```
public class Rectangle
{
    private float length;
    private float width;

    public Rectangle()
    {
        length = 0;
        width = 0;
    }

    public Rectangle(float l, float w)
    {
        length = l;
        width = w;
    }

    public float area()
    {
        return length * width;
    }
}
```



# Gọi phương thức

## ❖ Cú pháp:

<Tham chiếu đối tượng>.<Tên hàm>(<ds ts thực>)

<Tên lớp>.<Tên hàm tĩnh>(<ds tham số thực>)

<Tên hàm>(<ds tham số thực>)

## ❖ Ví dụ

hinhTron.ve();// hinhTron là đối tượng của lớp HinhTron

int i = java.lang.Math.abs(-4);//Gọi đầy đủ của lớp Math

int j = Math.abs(-4);//Gọi theo tên của lớp Math

someMethod(ofValue);//Đối tượng hoặc lớp không tường minh

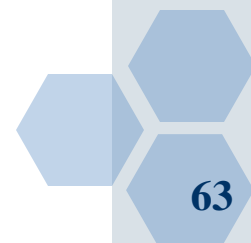
## ❖ Lưu ý: Trong Java, mọi tham biến đều được truyền theo tham trị





# Gọi PT và truyền tham số kiểu tham trị

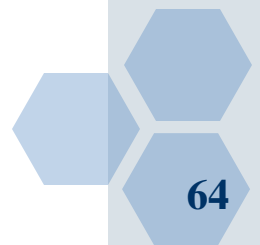
- ❖ Các giá trị từ phương thức gọi sẽ được truyền như đối số tới phương thức được gọi.
- ❖ Bất kỳ sự thay đổi nào của đối số trong phương thức được gọi đều không ảnh hưởng tới các giá trị được truyền từ phương thức gọi.
- ❖ Các biến có giá trị kiểu nguyên thủy sẽ được truyền theo kiểu này.





## Ví dụ

```
package lop;
import java.util.Scanner;
public class PassByValue {
    private int a = 50;
    public void doubleValue(int a)
    {
        a = a*2;
    }
    public static void main(String[] args) {
        PassByValue obj = new PassByValue();
        System.out.println("Before pass to method: " +obj.a);
        obj.doubleValue(2);
        System.out.println("After pass to method: " +obj.a);
    }
}
```

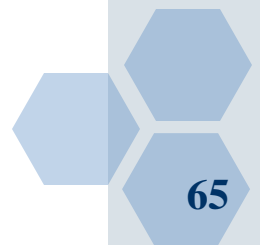






# Gọi PT và truyền tham số kiểu tham biến

- ❖ Sự thay đổi giá trị trong phương thức được gọi sẽ ảnh hưởng tới giá trị truyền từ phương thức gọi.
- ❖ Khi các tham chiếu được truyền như đối số tới phương thức được gọi, các giá trị của đối số có thể thay đổi nhưng tham chiếu sẽ không thay đổi.





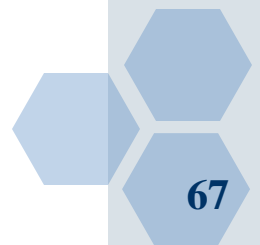
# Ví dụ

```
package lop;
import java.util.Scanner;
public class PassByReference {
    int a = 50;
    public void doubleValue(PassByReference ob)
    {
        ob.a = ob.a*2;
    }
    public static void main(String[] args) {
        PassByReference ob1 = new PassByReference();
        System.out.println("Before pass to method: " +ob1.a);
        ob1.doubleValue(ob1);
        System.out.println("After pass to method: " +ob1.a);
    }
}
```



# Phương thức tĩnh (Static Methods)

- ❖ Là những phương thức được gọi thông qua tên Lớp (không cần đối tượng).
- ❖ Khai báo phương thức thêm từ khoá static.
- ❖ Chỉ có thể truy xuất một cách trực tiếp tới các biến tĩnh (static) và các phương thức tĩnh khác của lớp.
- ❖ Không thể truy xuất đến các phương thức và biến không tĩnh (non-static)





# Ví dụ phương thức tĩnh

```
public class Calculator
{
    public static int add(int a, int b)
    {
        return (a + b);
    }

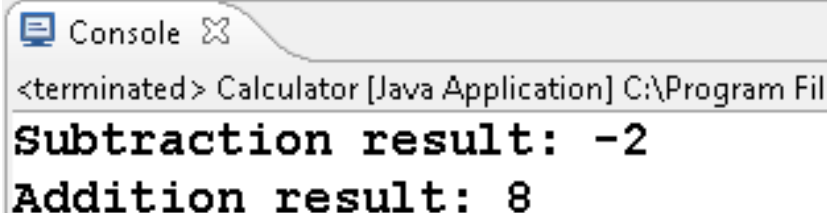
    public int sub(int a, int b)
    {
        return (a - b);
    }
}
```

```
public static void main(String[] args)
{
    Calculator cal = new Calculator();
    int s = cal.sub(3, 5);

    System.out.println("Subtraction result: " + s);

    //calling static method
    int a = Calculator.add(3, 5);

    System.out.println("Addition result: " + a);
}
```

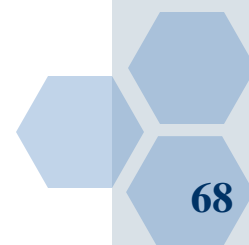


Console X

<terminated> Calculator [Java Application] C:\Program Fil

**Subtraction result: -2**

**Addition result: 8**





# Việc sử dụng phương thức tĩnh

- ❖ Khi phương thức không truy xuất tới các trạng thái của đối tượng.
- ❖ Khi phương thức chỉ quan tâm đến các biến tĩnh.



# Phương thức tham số biến

❖ Cho phép gọi phương thức với số tham số thay đổi.

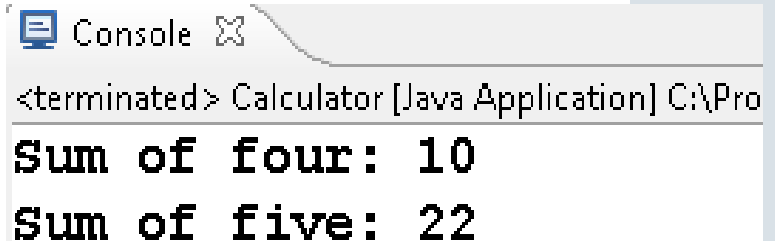
```
public class Calculator
{
    public int add(int... a)
    {
        int result = 0;

        for (int i = 0; i < a.length; i++)
        {
            result += a[i];
        }

        return result;
    }
}
```

```
public static void main(String[] args)
{
    Calculator cal = new Calculator();

    System.out.println("Sum of four: " + cal.add(1,2,3,4));
    System.out.println("Sum of five: " + cal.add(5,2,5,3,7));
}
```



Console

<terminated> Calculator [Java Application] C:\Pro

Sum of four: 10

Sum of five: 22



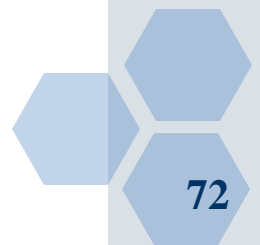
# Phương thức nạp chồng (Overload)

- ❖ **Overload:** Là kỹ thuật cho phép xây dựng các hành vi trùng tên nhưng khác chữ ký (signature) trong cùng 1 lớp.
- ❖ **Chữ ký bao gồm:**
  - Số lượng các tham số.
  - Thứ tự các kiểu của các tham số.



# Phương thức nạp chồng (Overload)

- ❖ Hàm tạo cũng có thể được nạp chồng.
- ❖ Tùy theo ta gọi đối số thế nào thì nó sẽ gọi hàm tương ứng.
- ❖ Nạp chồng là hình thức đa hình trong quá trình biên dịch.







# Nạp chồng hàm khởi tạo

```
public class Car{                                // lớp xe ô tô
    private String producer; // tên nhà sản xuất
    private String color;    // màu xe
    private float capacity;  // dung tích xilanh
    private int price;       // giá xe

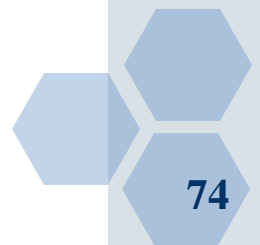
    public Car(){
        producer = "";
        color = "";
        capacity = 0.0;
        price = 0;
    }

    public Car(String v1, String v2, float v3, int v4){
        producer = v1;
        color = v2;
        capacity = v3;
        price = v4;
    }
}
```



# Ví dụ về phương thức nạp chồng

```
public class OverloadingOrder {  
    static void print(String s, int i) {  
        System.out.println(  
            "String: " + s +  
            ", int: " + i);  
    }  
    static void print(int i, String s) {  
        System.out.println(  
            "int: " + i +  
            ", String: " + s);  
    }  
    public static void main(String[] args) {  
        print("String first", 11);  
        print(99, "Int first");  
    }  
} // !:~
```





## Ví dụ về phương thức nạp chồng

**BT: Tính diện tích hình vuông, diện tích hình chữ nhật, diện tích hình tam giác.**

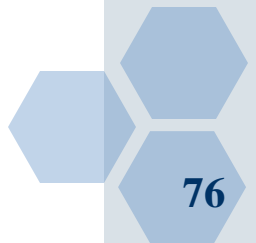
**Gợi ý:**

- **Tạo ra 3 phương thức đều có tên là `diệnTich` sao cho:**
  - Nếu có 1 tham số truyền vào, nó sẽ tự hiểu là cần tính diện tích hình vuông.
  - Nếu có 2 tham số truyền vào, nó sẽ tự hiểu là cần tính diện tích hình chữ nhật.
  - Nếu có 3 tham số truyền vào, nó sẽ tự hiểu là cần tính diện tích hình tam giác.



# Ví dụ về phương thức nạp chồng

```
class Hình {  
  
    public float dienTich(float a) {  
        return (float) a * a;  
    }  
  
    public float dienTich(float a, float b) {  
        return (float) a * b;  
    }  
  
    public double dienTich(float a, float b, float c) {  
        float p;  
        p = (float) (a + b + c) / 2;  
        return Math.sqrt(p * (p - a) * (p - b) * (p - c));  
    }  
}
```





# Ví dụ về phương thức nạp chồng

```
public class DienTichHinh {  
  
    public static void main(String[] args) {  
        Hinh h = new Hinh();  
        System.out.println("Diện tích hình vuông có cạnh 5  
: " + h.dienTich(5));  
        System.out.println("Diện tích hình chữ nhật có 2  
cạnh là 4 và 5 : " + h.dienTich(4, 5));  
        System.out.println("Diện tích hình tam giác có 3 cạ  
nh là 3, 4, 5 : " + h.dienTich(3, 4, 5));  
    }  
}
```



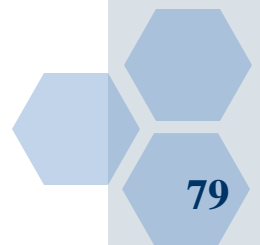
# LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG VỚI JAVA

- ❖ Đối tượng & Lớp
- ❖ Kế thừa
- ❖ Đa hình
- ❖ Gói (Packages)
- ❖ Giao diện (Interface)
- ❖ Một số lớp tiện ích trong Java



# Kế thừa (Inheritance)

- ❖ Sự kế thừa được sử dụng khi muốn tạo 1 lớp mới từ một lớp đã biết.
- ❖ Khi đó, tất cả các thuộc tính và phương thức của lớp cũ đều trở thành thuộc tính và phương thức của lớp mới.
- ❖ Lớp cũ được gọi là lớp cha (lớp cơ sở), lớp mới được gọi là lớp con (lớp dẫn xuất).





# Lớp cơ sở và lớp dẫn xuất

- ❖ Một lớp được xây dựng thông qua kế thừa từ một lớp khác gọi là lớp dẫn xuất (hay gọi là lớp con).
- ❖ Lớp dùng để xây dựng lớp dẫn xuất được gọi là lớp cơ sở (hay gọi là lớp cha).
- ❖ Lớp dẫn xuất ngoài các thành phần của riêng nó, nó còn được kế thừa tất cả các thành phần của lớp cha.





# Khai báo lớp kế thừa

❖ Được thực hiện bởi từ khoá **extends**:

❖ Cú pháp:

```
<Modifier> <tên lớp con>extends <tên lớp cha>
{
    // phần thân của lớp con
}
```



# Khai báo lớp kế thừa

```
1 package tenpackage;  
2 public class TenLopCha {  
3     //Khai báo các thuộc tính  
4     //Khai báo các phương thức  
5 }
```

```
1 package tenpackage;  
2 public class TenLopCon extends TenLopCha {  
3     //Khai báo các thuộc tính  
4     //Khai báo các phương thức  
5 }
```

- ❖ Object là lớp cơ sở nhất trong Java.
- ❖ Trường hợp một lớp không khai báo kế thừa từ bất kỳ lớp nào thì lớp Object chính là lớp cha của nó.



# Khai báo kế thừa

```
1 package quanly;
2
3 public class GiangVien {
4     //Khai báo các thuộc tính
5
6     //Khai báo các phương thức
7 }
```

```
1 package bt1;
2
3 public class GiangVienCoHuu extends GiangVien{
4     //Khai báo các thuộc tính
5
6     //Khai báo các phương thức
7 }
```

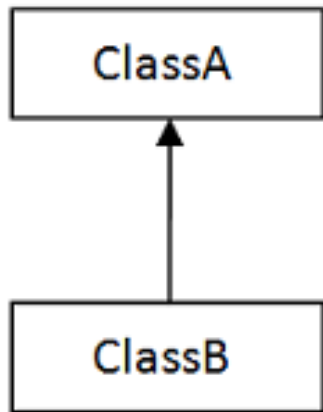


# Kế thừa các thuộc tính và phương thức

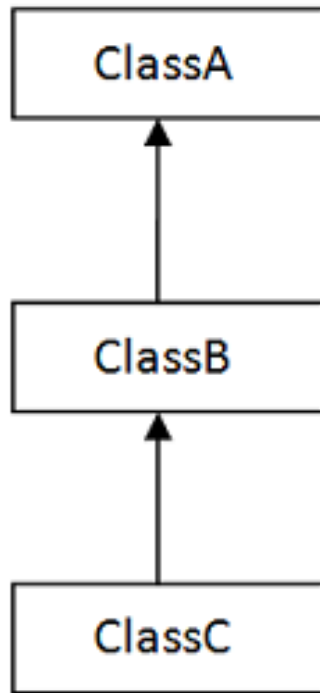
- ❖ Thuộc tính của lớp cơ sở được kế thừa trong lớp dẫn xuất.
- ❖ Tuy nhiên, trong lớp dẫn xuất không thể truy cập vào các thành phần private, package của lớp cơ sở.
- ❖ Lớp dẫn xuất kế thừa tất cả các phương thức của lớp cơ sở trừ:
  - Phương thức khởi tạo
  - Phương thức final



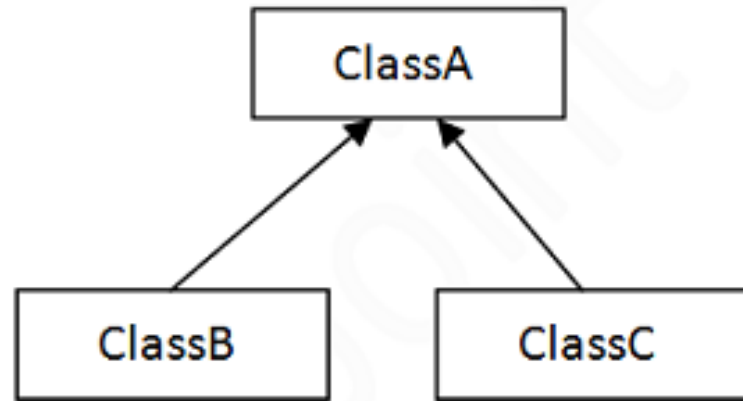
# Các kiểu kế thừa trong Java



1. Đơn kế thừa



2. Kế thừa nhiều tầng



3. Kế thừa thứ bậc



# Ví dụ về đơn kế thừa

```
class Animal {  
    void eat() {  
        System.out.println("eating...");  
    }  
}  
  
class Dog extends Animal {  
    void bark() {  
        System.out.println("barking...");  
    }  
}  
  
public class TestInheritance1 {  
    public static void main(String args[]) {  
        Dog d = new Dog();  
        d.bark();  
        d.eat();  
    }  
}
```



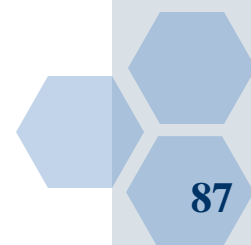
# Ví dụ về kế thừa nhiều tầng

```
class Animal {
    void eat() {
        System.out.println("eating...");
    }
}

class Dog extends Animal {
    void bark() {
        System.out.println("barking...");
    }
}

class BabyDog extends Dog {
    void weep() {
        System.out.println("weeping...");
    }
}

public class TestInheritance2 {
    public static void main(String args[]) {
        BabyDog d = new BabyDog();
        d.weep();
        d.bark();
        d.eat();
    }
}
```





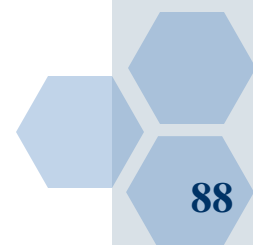
# Ví dụ về kế thừa thứ bậc

```
class Animal {
    void eat() {
        System.out.println("eating...");
    }
}

class Dog extends Animal {
    void bark() {
        System.out.println("barking...");
    }
}

class Cat extends Animal {
    void meow() {
        System.out.println("meowing...");
    }
}

public class TestInheritance3 {
    public static void main(String args[]) {
        Cat c = new Cat();
        c.meow();
        c.eat();
        // c.bark(); // compile error
    }
}
```







# Một số từ khoá thông dụng

- Chỉ định phương thức khởi tạo của lớp cha

`super() , super(...)`

- Cài đặt lại phương thức của lớp cha

`@Override`

`public KiểuDuLieu tenPhuongthuc(...)`

Lưu ý: phương thức `static` không được phép `Override`

- Gọi phương thức của lớp cha

`super.tenPhuongThuc(...)`

- Kiểm tra thể hiện của đối tượng

– `instanceOf`



# Khởi đầu lớp cơ sở

- ❖ Thực hiện khởi đầu cho lớp cơ sở bằng cách gọi hàm khởi tạo của lớp cơ sở bên trong hàm khởi tạo của lớp dẫn xuất. (nếu không làm thì java tự động thực hiện).
- ❖ Để có thể gọi hàm khởi tạo của lớp cơ sở ta sử dụng từ khoá super.



# Lớp cơ sở

❖ Sử dụng từ khoá **super** để gọi hàm khởi tạo của lớp cơ sở 1 cách tường minh.

```
class B
{
    public B ()
    {
        System.out.println ( "Ham tao của lop co so" );
    }
}
public class A extends B
{
    public A ()
    {
        super();// gọi tạo của lớp cơ sở một cách tường minh
        System.out.println ( "Ham tao của lop dan xuat" );
    }
}
public static void main ( String arg[] )
{
    A thu = new A ();
}
}
```

Kết quả chạy chương trình như sau:  
Ham tao của lop co so  
Ham tao của lop dan xuat



# Lớp cơ sở

- ❖ Gọi hàm khởi tạo của lớp cơ sở không tường minh.

```
class B
{
    public B ()
    {      System.out.println ( "Ham tao của lop co so" );}
}
public class A extends B
{
    public A () {// không gọi hàm tạo của lớp cơ sở tường minh
        System.out.println ( "Ham tao của lop dan xuat" );
    }
}
public static void main ( String arg[] )
{
    A thu = new A ();
}
```

Kết quả chạy chương trình như sau:  
Ham tao của lop co so  
Ham tao của lop dan xuat



## ❖ *Chú ý:*

- Nếu gọi tường minh hàm khởi tạo của lớp cơ sở thì lời gọi này phải là lệnh đầu tiên.
- Chỉ có thể gọi tối đa 1 hàm tạo của lớp cơ sở bên trong hàm tạo của lớp dẫn xuất.



# Lớp cơ sở

- ❖ **Ví dụ:** Khai báo và khởi tạo lớp Person gồm có tên và tuổi, địa chỉ. Các phương thức: Constructor 3 tham số, Input, Output.
- ❖ Xây dựng lớp Nhân viên (Employee) được kế thừa từ lớp Person, có thêm lương. Các phương thức: Constructor có 4 tham số và gọi đến hàm tạo lớp cơ sở, Input (gọi đến Input của lớp cơ sở), Output (gọi đến Output của lớp cơ sở).



## Ví dụ

```
public class Person {  
    public String name;  
    public int age;  
    // Khởi tạo  
    public Person (String name, int age)  
    {  
        this.name=name;  
        this.age=age;  
    }  
    public void show()  
    {  
        System.out.println("Họ tên"+name +", Tuổi:"+age);  
    }  
}
```



## Ví dụ

**class Employee extends Person**

**{**

**public float salary;**

**// Phương thức khởi tạo**

**public Employee(String name, int age, float salary)**

**{**

**super(name, age);**

**this.salary=salary;**

**}**

**// Khai báo nạp chồng**

**public void show()**

**{**

**System.out.println("Họ tên:"+name + ", có lương là:"+salary);**

**}**

**}**





## Ví dụ

```
class Employee2  
{  
    public static void main(String[] args) {  
        Employee ep=new  
Employee('Hoa',22,3000f);  
        ep.show();  
    }  
}
```





# Ghi đè phương thức (Override)

- ❖ Hiện tượng trong lớp cơ sở và lớp dẫn xuất có 2 phương thức giống hệt nhau (cả tên và bộ tham số) gọi là ghi đè phương thức.
- ❖ Trong lớp dẫn xuất nếu có hiện tượng ghi đè thì phương thức bị ghi đè của lớp cơ sở sẽ bị ẩn đi.
- ❖ Chú ý: Nếu phương thức của lớp cơ sở bị nạp chồng thì nó không thể bị ghi đè ở lớp dẫn xuất.



# Ghi đè phương thức (Override)

- ❖ Override chỉ xảy ra giữa các lớp có quan hệ kế thừa.
- ❖ Một phương thức final hoặc static không thể được Override.
- ❖ Khi đối tượng thuộc lớp con gọi phương thức thì sẽ chọn lựa và chạy theo phương thức trong lớp con. Nếu lớp con không có phương thức đó thì mới lên tìm đến lớp cha để chạy,
- ❖ Ghi đè là hình thức đa hình trong quá trình thực thi.

# Ví dụ

OverLoad\_Override.java

```
1 // OverLoad_Override.java
2 class MyArray1 // calculating area
3 { int MaxN, a[], n;
4   MyArray1 ( int [] ar)
5   { n=ar.length;
6     MaxN= 2* n; // bo nho co the chua gap 2 lan mang ban dau
7     a= new int [MaxN];
8     for (int i=0; i<n; i++) a[i]=ar[i];
9   }
10  void Output()
11  { for (int i=0; i<n; i++)
12    System.out.print(i<n-1? a[i] + ", " : a[i] + "\n");
13  }
14  void Add (int x) // add to the last of array
15  { if (n==MaxN) System.out.println("Array is full!");
16    else
17      { a[n]=x; // add to the last
18        n++;
19      }
20  }
21  void Add (int x, int i)
22  { if (n==MaxN) System.out.println("Array is full!");
23    else
24      { for (int j=n; j>i; j--) a[j]=a[j-1];
25        a[i]=x;
26        n++;
27      }
28  }
29 }
30 class MyArray2 extends MyArray1
31 { MyArray2(int ar[])
32 { super (ar);
33 }
34 void Add(int x) // add to the first of array
35 { if (n==MaxN) System.out.println("Array is full!");
36   else
37     { for (int i=n; i>0; i--) a[i]=a[i-1];
38       a[0]=x;
39       n++;
40     }
41 }
42 }
```

## Ví dụ về overload và override

**overloading method:**  
cùng tên,  
khác tham số,  
cùng lớp

**overriding method:**  
cùng tên,  
cùng tham số,  
ở hai lớp cha con



# Ví dụ về Overload và Override

```
43 class Problem
44 { public static void main(String args[])
45 { int a1[] = { 1,2,3,4,5 };
46   MyArray1 obj1= new MyArray1(a1);
47   obj1.Output();
48   obj1.Add(6);
49   obj1.Output();
50   obj1.Add(7,3);
51   obj1.Output();
52   int a2[] = { 7, 8, 9, 0 };
53   MyArray2 obj2 = new MyArray2(a2);
54   obj2.Output();
55   obj2.Add(10);
56   obj2.Output();
57 }
58 }
59 }
```

thêm

đôi

thêm đầu

```
C:\Program Files\Xinox Software\JC...
1, 2, 3, 4, 5
1, 2, 3, 4, 5, 6
1, 2, 3, 7, 4, 5, 6
7, 8, 9, 0
10, 7, 8, 9, 0
Press any key to continue...
```

# LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG VỚI JAVA

- ❖ Đối tượng & Lớp
- ❖ Kế thừa
- ❖ Đa hình
- ❖ Gói (Packages)
- ❖ Giao diện (Interface)
- ❖ Một số lớp tiện ích trong Java



# Đa hình (Polymorphism)

- ❖ **Đa hình: nhiều hình thức thực hiện một hành vi, nhiều kiểu tồn tại của một đối tượng**
- ❖ **Đa hình trong lập trình:**
  - Đa hình phương thức: nạp chồng phương thức, ghi đè phương thức.
  - Đa hình đối tượng
    - Nhìn nhận đối tượng theo nhiều kiểu khác nhau.  
VD: 1 SV là cán bộ lớp thì có thể nhìn nhận theo 2 góc nhìn: SV hoặc cán bộ.
    - Các đối tượng khác nhau giải nghĩa thông điệp theo cách thức khác nhau.

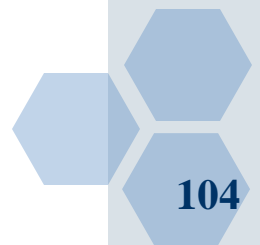


# Đa hình (Polymorphism)

## ❖ VD:

- Lớp A có hành vi M().
- Lớp B là con của lớp A, trong lớp B viết hành vi M().
- Có biến đối tượng obj.
- Tại thời điểm t1: obj chỉ đến một thực thể A. Khi đó obj.M() sẽ cho một phản ứng.
- Tại thời điểm t2: obj chỉ đến một thực thể B. Khi đó obj.M() sẽ cho một phản ứng khác.

❖ **Tính đa hình có được là nhờ kỹ thuật override hành vi giữa 2 lớp cha con.**







## Các bước tạo đa hình:

- ❖ B1: Xây dựng lớp cơ sở.
- ❖ B2: Xây dựng lớp dẫn xuất từ lớp cơ sở vừa tạo.  
Trong lớp dẫn xuất này ta sẽ ghi đè các phương thức của lớp cơ sở.
- ❖ B3: Thực hiện downcasting thông qua lớp cơ sở để thực hiện hành vi đa hình.



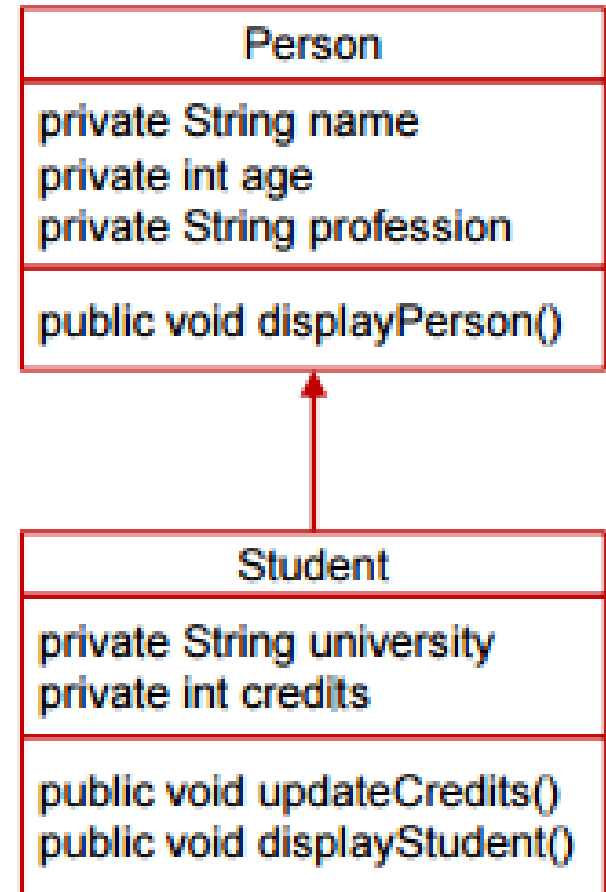
# Upcasting và Downcasting

❖ **Upcasting:** đối tượng lớp con được nhìn nhận như đối tượng lớp cha.

- Tự động ép kiểu

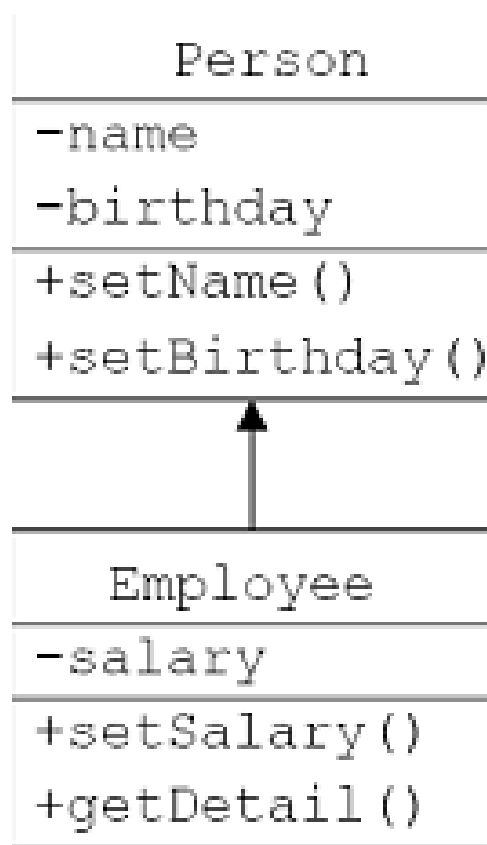
❖ **Downcasting:** đối tượng lớp cha được nhìn nhận như đối tượng lớp con

- Phải ép kiểu.





## Ví dụ



```
public class Test1 {  
    public static void main(String arg[]) {  
        Person p;  
        Employee e = new Employee();  
        p = e;  
        p.setName("Hoa");  
        p.setSalary(350000); // compile error  
    }  
}
```



## Ví dụ

```
class Manager extends Employee {
    Employee assistant;
    // ...
    public void setAssistant(Employee e) {
        assistant = e;
    }
    // ...
}

public class Test2 {
    public static void main(String arg[]){
        Manager junior, senior;
        // ...
        senior.setAssistant(junior);
    }
}
```





## Ví dụ

```
public class Test3 {  
    String static teamInfo(Person p1, Person p2) {  
        return "Leader: " + p1.getName() +  
            ", member: " + p2.getName();  
    }  
  
    public static void main(String arg[]){  
        Employee e1, e2;  
        Manager m1, m2;  
        // ...  
        System.out.println(teamInfo(e1, e2));  
        System.out.println(teamInfo(m1, m2));  
        System.out.println(teamInfo(m1, e2));  
    }  
}
```



# Downcasting

- ❖ Là hiện tượng một đối tượng ở lớp cha tham trỏ đến một đối tượng của lớp con.
- ❖ Không tự động chuyển đổi kiểu nên ta phải ép kiểu tường minh cho nó.



## Ví dụ

```
public class Test2 {  
    public static void main(String arg[]) {  
        Employee e = new Employee();  
        Person p = e; // up casting  
        Employee ee = (Employee) p; // down casting  
        Manager m = (Manager) ee; // run-time error  
  
        Person p2 = new Manager();  
        Employee e2 = (Employee) p2;  
    }  
}
```



# Ví dụ DT lớp cha nhưng cụ thể lại là lớp con

LopCon\_1.java

```
1 // LopCon_1.java
2 // Bien doi tuong la lop cha nhưng cu the lai la lop con
3 class Father
4 { void Out()
5   { System.out.println("I am a father!");
6   }
7 }
8 class Son extends Father
9 { void Out()
10  { System.out.println("I am a son!");
11  }
12 }
13 class Demo
14 { public static void main (String args[])
15   { Son aSon = new Son();
16     aSon.Out();
17     Father aFather = new Father();
18     aFather.Out();
19     Father obj = new Son();
20     obj.Out();
21   }
22 }
```

Biến đối tượng là tham khảo nên hoàn toàn có thể khai báo biến là lớp cha nhưng khởi tạo biến là đối tượng thuộc lớp con.

Tính đa hình

```
C:\PROGRA~1\XINOS~1\JCRE...
I am a son!
I am a father!
I am a son!
Press any key to continue...
```





# Lớp lồng nhau

**class *EnclosingClass*{ // Lớp bao bên ngoài**

**...**

**static class *StaticNestedClass* { // Lớp lồng tĩnh**

**...**

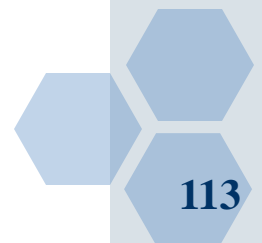
**}**

**class *InnerClass* { // Lớp lồng phi tĩnh hay lớp  
nội bộ**

**...**

**}**

**}**





# Lớp có dữ liệu là một ĐT của lớp ngoài

Chương trình  
xuất hóa đơn

```
C:\Program Files\Xinox Software\JCreatorV3... - □ X
Hoa đơn số:A001, ngày:12/09/2006
Khách hàng:Hoang Le Thuy, 23 Le Thanh Ton
Chi tiết hóa đơn:

TV Sony 21, 1, 3200000,3200000
DVD Toshiba, 1, 1300000,1300000
IBM Laptop, 2, 13200000,26400000

Tong hoa đơn:30900000
Press any key to continue...
```

BaoGop.java

```
1 // BaoGop.java
2 class KháchHang
3 { String Ten, DiaChi;
4   void SetData( String t, String dc) { Ten=t; DiaChi=dc; }
5   void OutData() { System.out.println(Ten + ", " + DiaChi); }
6 }
7 class Hang
8 { String Ten;
9   int SoLuong;
10  long DonGia;
11  void setData(String t, int sl, long dg)
12  { Ten=t; SoLuong=sl; DonGia=dg; }
13  void OutData()
14  { System.out.println(Ten + ", " + SoLuong + ", " + DonGia + ", " + SoLuong*DonGia); }
15 }
```



# Lớp có dữ liệu là một ĐT của lớp ngoài

BaoGop.java

```
16 class HoaDon
17 { String So, Ngay;
18   KháchHang Kh;
19   Hang [] ds;
20   int MaxN=0; // so mat hang
21   int n=0;
22   void SetData(String So, String ng, KháchHang Kh, int maxn)
23   { this.So=So; Ngay=ng; this.Kh=Kh; MaxN=maxn;
24     ds= new Hang[MaxN];
25     for (int i=0;i<n;i++) ds[i]=new Hang();
26   }
27   void Add ( Hang h)
28   { if (n==MaxN) System.out.println("Danh sach day!");
29     else ds[n++]=h;
30   }
31   long TongHoaDon()
32   { long S=0;
33     for (int i=0;i<n;i++) S+= ds[i].SoLuong* ds[i].DonGia;
34     return S;
35   }
36   void XuatHoaDon()
37   { System.out.println("Hoa don so:" + So + ", ngay:" + Ngay);
38     System.out.print("Khach hang:");
39     Kh.OutData();
40     System.out.println("Chi tiet hoa don:");
41     System.out.println("_____");
42     for (int i=0;i<n;i++) ds[i].OutData();
43     System.out.println();
44     System.out.println("_____");
45     System.out.println("Tong hoa don:" + TongHoaDon());
46   }
```



# Lớp có dữ liệu là một ĐT của lớp ngoài

BaoGop.java

```
47 public static void main(String args[])
48 {   HoaDon hd= new HoaDon();
49     KhachHang Kh= new KhachHang();
50     Kh.SetData("Hoang Le Thuy", "23 Le Thanh Ton");
51     hd.SetData("A001", "12/09/2006", Kh, 3);
52     Hang h= new Hang();
53     h.setData("TV Sony 21", 1, 3200000);
54     hd.Add(h);
55     h= new Hang();
56     h.setData("DVD Toshiba", 1, 1300000);
57     hd.Add(h);
58     h= new Hang();
59     h.setData("IBM Laptop", 2, 13200000);
60     hd.Add(h);
61     hd.XuatHoaDon();
62 }
63 }
```

```
C:\Program Files\Xinox Software\JCreatorV3...
Hoa don so:A001, ngay:12/09/2006
Khach hang:Hoang Le Thuy, 23 Le Thanh Ton
Chi tiet hoa don:

TV Sony 21, 1, 3200000,3200000
DVD Toshiba, 1, 1300000,1300000
IBM Laptop, 2, 13200000,26400000

Tong hoa don:30900000
Press any key to continue...
```



# Lớp trong – Inner/nested class

- ❖ Là lớp được khai báo bên trong một lớp khác.
- ❖ Lớp trong truy xuất được lớp ngoài.
- ❖ Lớp ngoài phải có ít nhất một thành phần là instance của lớp trong và truy xuất các instance này.



# Lớp trong – Inner/nested class

## ■ Cú pháp:

```
class Outer
{
    ....
    class Inner
    {
        ...
    }
}
```

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

## Lợi ích:

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



# Lớp trong – Inner/nested class

❖ Lớp ngoài không thể truy cập trực tiếp lớp trong.

Inner1.java

```
1 // Inner class Demo
2 class Outer
3 { int m=3,n=2;
4   //Inner in_obj=new Inner();
5   Outer()
6   { System.out.println (m + ", " + n + ",k=" + k);
7   }
8   class Inner
9   { int k=3;
10    Inner()
11    { System.out.println ("Inner access Outer m=" + m + ", n=" + n);
12    }
13  }
14 }
15 class InnerOuterDemo
16 { public static void main(String args[])
17 { Outer obj=new Outer();
18 }
19 }
```

Task List

	description
✖	cannot find symbol variable k



# Lớp trong – Inner/nested class

- ❖ Lớp ngoài truy cập thành phần là ĐT thuộc lớp trong, lớp trong truy cập trực tiếp lớp ngoài

Inner1.java

```
1 // Inner class Demo
2 class Outer
3 { int m=3,n=2;
4   Inner in_obj=new Inner(); ←
5   Outer()
6   { System.out.println (m + ", " + n + ",k=" + in_obj.k);
7   }
8   class Inner
9   { int k=3;
10    Inner()
11    { System.out.println ("Inner access Outer m=" + m + ", n=" + n);
12    }
13  }
14 }
15 class InnerOuterDemo
16 { public static void main(String args[])
17   { Outer obj=new Outer();
18   }
19 }
```

```
C:\Program Files\Xinox Software\J...
Inner access Outer m=3, n=2
3, 2,k=3
Press any key to continue...
```





# Toán tử instanceof

❖ Kiểm tra một đối tượng có phải đang là thể hiện của lớp hoặc giao diện nào đó không?

❖ Cú pháp: `objectName instanceof Class`

`objectName instanceof Interface`

❖ Kết quả:

- True: đúng

---

```
if (pObj instanceof Person)
    System.out.println("pObj is a Person");
if (pObj instanceof Student)
    System.out.println("pObj is a Student");
```

---



# Toán tử instanceof

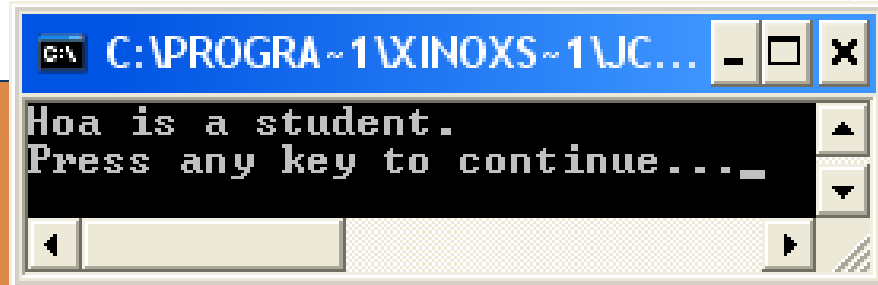
InstanceOfDemo.java

```
1 // InstanceOfDemo.java
2 import java.io.*;
3 class Student
4 {   String Name;
5     int Score1, Score2, Score3;
6     public Student(String aName, int S1, int S2, int S3)
7     {   Name= aName; Score1=S1; Score2=S2; Score3=S3;}
8     String GetName() { return Name;}
9 }
10
11 public class InstanceOfDemo
12 {   public static void main(String args[])
13     {   Student st= new Student("Hoa", 5,6,7);
14         if (st instanceof Student) System.out.println(st.GetName()+" is a student.");
15         else System.out.println("This isn't a student.");
16     }
17 }
```

Toán tử *instanceof* có 2 toán hạng  
Toán hạng trái: Một đối tượng  
Toán hạng phải: Tên 1 lớp  
Trả trị:

true: Nếu đối tượng thuộc lớp này.

false: nếu đối tượng không thuộc lớp này.

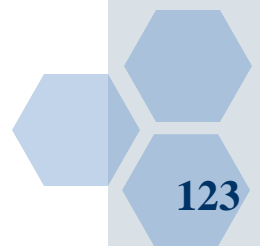




# Lớp trừu tượng (Abstract Class)

- ❖ Lớp trừu tượng: là lớp mà ta không thể tạo ra các đối tượng cho nó.
- ❖ Lớp trừu tượng thường được dùng để định nghĩa các khái niệm chung, đóng vai trò làm lớp cơ sở cho các lớp cụ thể khác.
- ❖ Đi cùng từ khoá abstract
- ❖ Cú pháp:

```
public abstract class NameClass  
{  
  
    // Thân chương trình  
  
}
```





# Lớp trừu tượng (Abstract Class)

- ❖ Lớp trừu tượng có thể chứa các phương thức trừu tượng không được định nghĩa.
- ❖ Các lớp dẫn xuất có trách nhiệm định nghĩa lại (override) các phương thức trừu tượng này.
- ❖ Sử dụng các lớp trừu tượng đóng vai trò quan trọng trong thiết kế phần mềm. Nó cho phép định nghĩa tạo ra những phần tử dùng chung trong cây thừa kế, nhưng khái quát để tạo ra các thể hiện.



# Lớp trừu tượng (Abstract Class)

## ❖ Để trở thành một lớp trừu tượng cần:

- Khai báo với từ khoá abstract.
- Chứa ít nhất một phương thức trừu tượng.

VD: `public abstract float Area();`

- Lớp con khi kế thừa phải cài đặt cụ thể cho các phương thức trừu tượng của lớp cha => Phương thức trừu tượng không thể khai báo là final hoặc static.

## ❖ Nếu một lớp có một hay nhiều phương thức trừu tượng thì nó phải là lớp trừu tượng

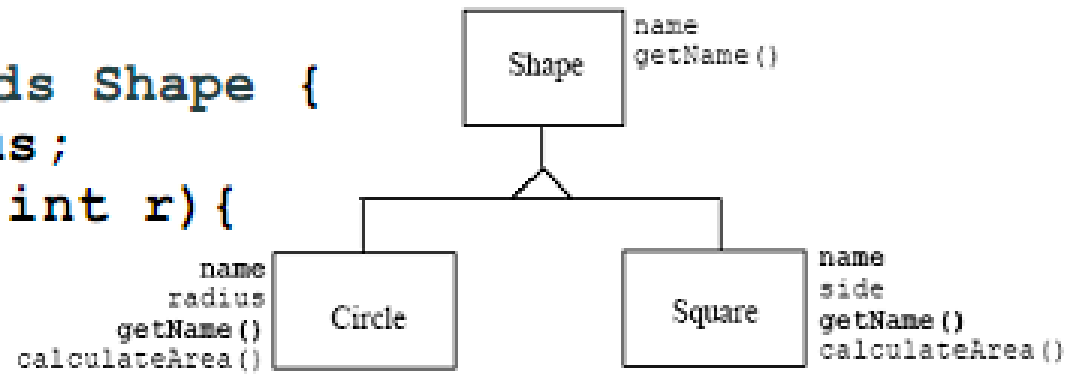




# Lớp trừu tượng (Abstract Class)

```
abstract class Shape {  
    protected String name;  
    Shape(String n) { name = n; }  
    public String getName() { return name; }  
    public abstract float calculateArea();  
}
```

```
class Circle extends Shape {  
    private int radius;  
    Circle(String n, int r) {  
        super(n);  
        radius = r;  
    }  
}
```



```
    public float calculateArea() {  
        float area = (float) (3.14 * radius * radius);  
        return area;  
    }  
}
```

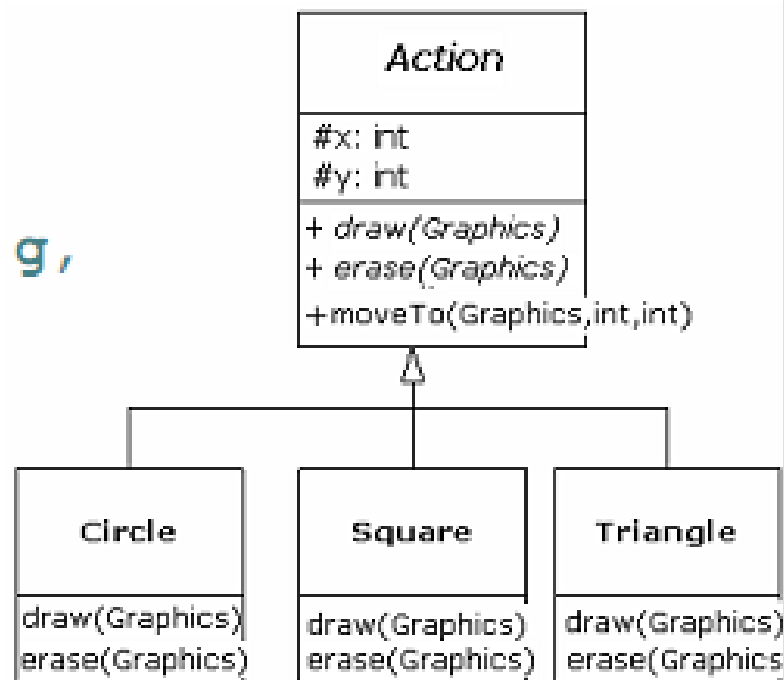
Lớp con bắt buộc phải override tất cả các phương thức abstract của lớp cha



# Lớp trừu tượng (Abstract Class)

```
import java.awt.Graphics;  
abstract class Action {  
    protected int x, y;  
    public void moveTo(Graphics g,  
        int x1, int y1) {  
        erase(g);  
        x = x1; y = y1;  
        draw(g);  
    }  
}
```

```
    abstract public void erase(Graphics g);  
    abstract public void draw(Graphics g);  
}
```





# Lớp trừu tượng (Abstract Class)

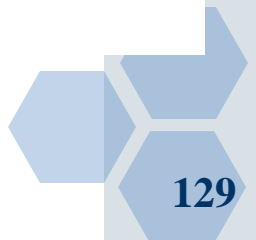
```
class Circle extends Action {
    int radius;
    public Circle(int x, int y, int r) {
        super(x, y); radius = r;
    }
    public void draw(Graphics g) {
        System.out.println("Draw circle at ("
                           + x + "," + y + ")");
        g.drawOval(x-radius, y-radius,
                   2*radius, 2*radius);
    }
    public void erase(Graphics g) {
        System.out.println("Erase circle at ("
                           + x + "," + y + ")");
        // paint the circle with background color...
    }
}
```





# Lớp trừu tượng (Abstract Class)

```
abstract class Point {  
    private int x, y;  
    public Point(int x, int y) { this.x = x; this.y = y; }  
    public void move(int dx, int dy) {  
        x += dx; y += dy;  
        plot();  
    }  
    public abstract void plot(); // phương thức trừu tượng không  
        có code thực hiện  
}
```





# Lớp trừu tượng

```
abstract class ColoredPoint extends Point {  
    int color;  
    public ColoredPoint(int x, int y, int color) { super(x, y); this.color =  
        color; }  
}
```

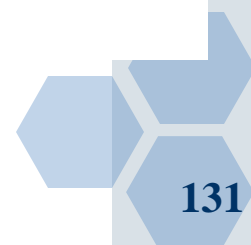
```
class SimpleColoredPoint extends ColoredPoint {  
    public SimpleColoredPoint(int x, int y, int color) {  
        super(x,y,color);  
    }  
    public void plot() { ... } // code to plot a SimplePoint  
}
```



# Lớp trừu tượng

- Lớp `ColoredPoint` không cài đặt mã cho phương thức `plot()` do đó phải khai báo: `abstract`
- Chỉ có thể tạo ra đối tượng của lớp `SimpleColoredPoint`.
- Tuy nhiên có thể:  

```
Point p = new SimpleColoredPoint(a, b, red);  
p.plot();
```



- ❖ Đối tượng & Lớp
- ❖ Kế thừa
- ❖ Đa hình
- ❖ Gói (Packages)
- ❖ Giao diện (Interface)
- ❖ Một số lớp tiện ích trong Java



# Giao diện Interface

- ❖ Java không cho phép đa kế thừa từ nhiều lớp.
- ❖ Để thực hiện đa kế thừa, java sử dụng khái niệm giao diện (interface).
- ❖ Giao diện chỉ quy định các phương thức phải có, nhưng không định nghĩa cụ thể.
- ❖ Các giao diện có thể kế thừa nhau



# Giao diện Interface

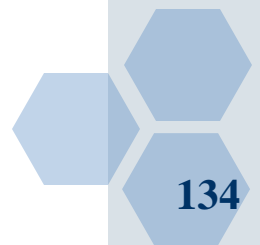
❖ Là mức trừu tượng cao hơn lớp trừu tượng

❖ Gồm:

- Phương thức trừu tượng
- Hằng số
- Mặc định là public

❖ Cú pháp:

```
Modifer interface InterfaceName{  
    // Thân interface  
}
```



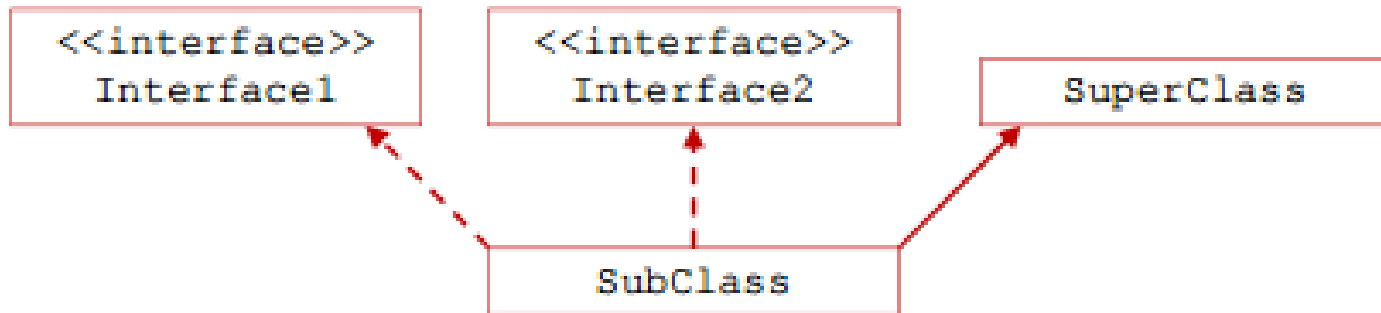


# Giao diện Interface

- ❖ **Interface luôn luôn có modifier: public interface.**
- ❖ **Nếu có các trường (thuộc tính) thì chúng đều là: public static final.**
- ❖ **Các phương thức của nó đều là phương thức trừu tượng**
  - Không có thân hàm
  - Luôn có modifier: public abstract
- ❖ **Không có hàm khởi tạo**
- ❖ **Sử dụng các từ khoá interface và implements**



# Kế thừa và triển khai giao diện đồng thời



## ❖ Cú pháp:

```
Modifier class SubClass extends SuperClass implements  
Interface1, Interface2{  
    // Thân class  
}
```

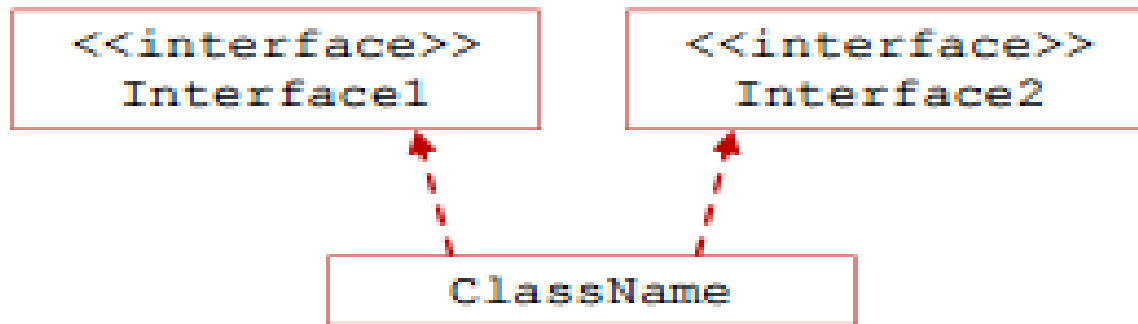
- ❖ `SubClass` kế thừa các phương thức, thuộc tính của `SuperClass` và phải định nghĩa mọi phương thức của `Interface 1` và `Interface2`





# Giao diện Interface

## ❖ Triển khai giao diện:



**Modifier class ClassName implements Interface1, Interface2{**

**// Thân class**

**}**

**ClassName phải định nghĩa mọi phương thức của Interface1 và Interface2**



## Ví dụ

```
interface KiemTra {  
    void inSo(int p);  
}  
class HienThuc implements KiemTra {  
    public void inSo(int p) {  
        System.out.println("Giá trị của p là : "+p);  
    }  
    void boSung() {  
        System.out.println("Class hiện thực giao diện có thể định nghĩa  
thêm "+ "thành viên khác hay không");  
    }  
}
```



## Ví dụ

```
class HTKhac implements KiemTra {  
    public void inSo(int p) {  
        System.out.println(“Bình phương của p là  
: “+p*p);  
    }  
}
```



# Truy xuất lớp thông qua giao diện

```
class UngDung {  
    public static void main (String args[] {  
        KiemTra c = new HienThuc(); //c chỉ biết hàm khai báo  
                                   trong giao diện  
        HienThuc d  = new HienThuc(); //d biết các hàm khai  
        báo                                           trong HienThuc  
        HTKhac e = new HTKhac(); //e biết các hàm khai báo  
                                   trong HTKhac  
        c. inSo(50);
```



# Truy xuất lớp thông qua giao diện

```
c = d; // c bây giờ tham chiếu đến đối tượng kiểu HienThuc
c.bo Sung();
c = e; // c bây giờ tham chiếu đến đối tượng kiểu HTKhac
c.inSo(50);
    }
}
```



# Truy xuất lớp thông qua giao diện

**Kết quả chương trình là :**

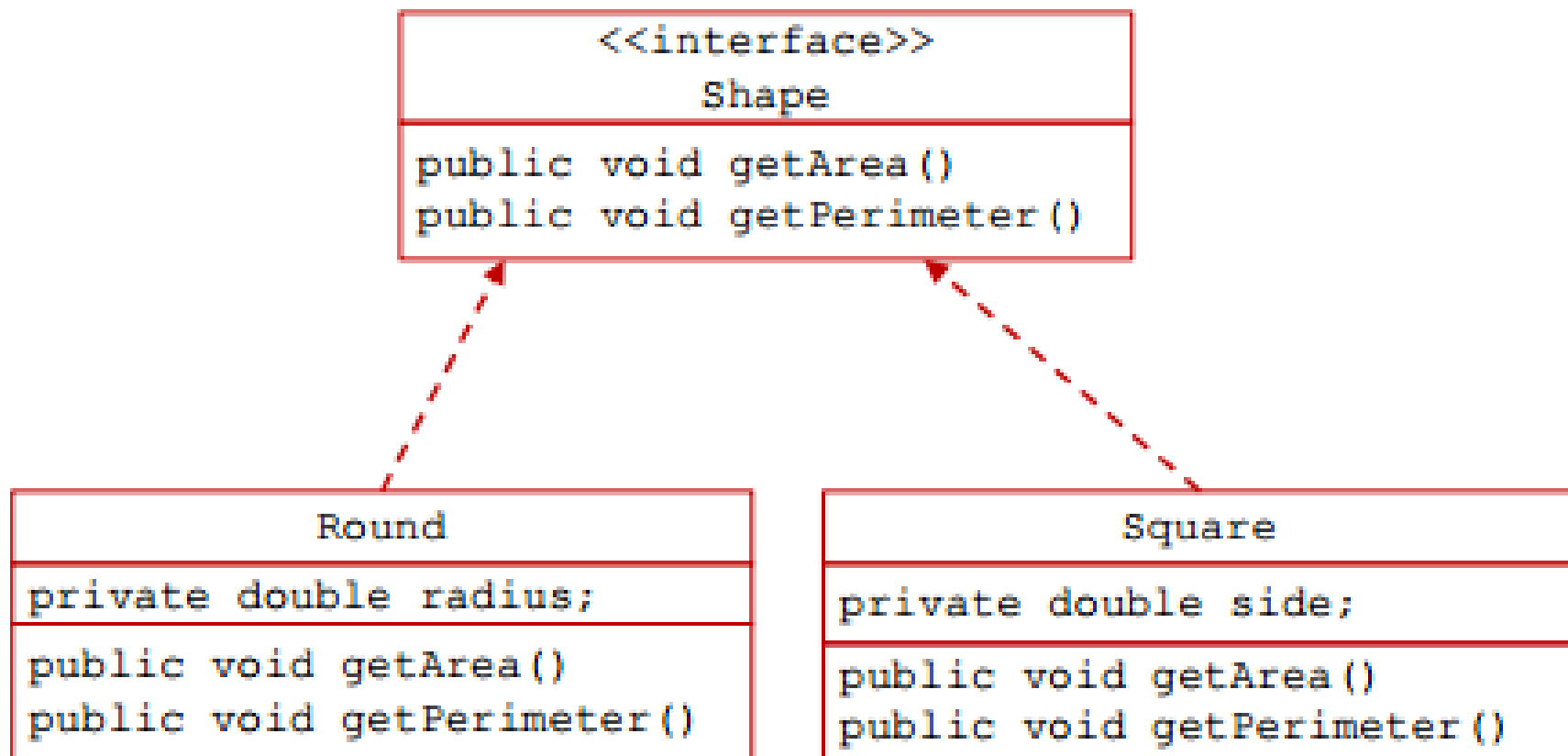
**Giá trị của p là : 50**

**Class hiện thực giao diện có thể định nghĩa thêm thành viên khác hay không**

**Bình phương của p là : 2500**



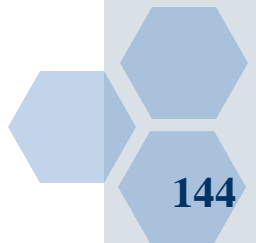
# Ví dụ về interface





# Giao diện Shape

```
package java.oop.shape;  
public interface Shape{  
    public void getArea();  
    public void getPerimeter();  
}
```







# Lớp Round

```
package java.oop.shape;
public class Round implements Shape{
    private double radius;
    // Hàm khởi tạo
    public Round(double radius){
        this.radius=radius;
    }
    public void getArea()
    {
        return Math.PI*radius*radius;
    }
    public void getPerimeter(){
        return 2*Math.PI*radius;
    }
    public double getRadius() {return this.radius;}
}
```

- ❖ Đối tượng & Lớp
- ❖ Kế thừa
- ❖ Đa hình
- ❖ Giao diện (Interface)
- ❖ Gói (Packages)
- ❖ Một số lớp tiện ích trong Java



# Package

❖ **Package dùng để đóng gói, gom nhóm chung các kiểu dữ liệu (lớp, giao diện,...) có liên hệ với nhau thành 1 khối với phạm vi truy xuất nhất định.**

❖ **Cú pháp:**

```
package <packageName>;
```

❖ **Sử dụng package:**

```
import tên_package_*;
```

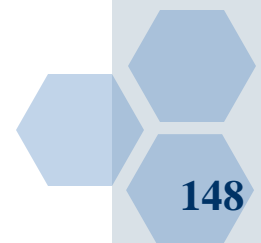
Hoặc      

```
import tên_package_.tên_class_trong_package;
```



# Package

```
package goi;
import java.io.*;
import java.util.*;
class PTB1{
    private double a,b;
    private void nhapPTB1() {
        Scanner sc=new Scanner(System.in);
        System.out.println("Nhập hệ số a = ");a=sc.nextDouble();
        System.out.println("Nhập hệ số b = ");b=sc.nextDouble();
    }
    protected void tinhPTB1(double a1, double b1){
        if(b1!=0)
            System.out.println("PT có nghiệm x = " + (-b1/a1));
        else
            System.out.println("PT vô nghiệm");
    }
}
```





# Ví dụ

```
class giaiPTB2{
    private double a,b,c;
    void nhapPTB2() {
        Scanner sc=new Scanner(System.in);
        System.out.println("Nhập hệ số a = ");a=sc.nextDouble();
        System.out.println("Nhập hệ số b = ");b=sc.nextDouble();
        System.out.println("Nhập hệ số c = ");c=sc.nextDouble();
    }
    public void tinhPTB2(){
        double delta;
        delta= b*b-4*a*c;
        if(a==0){
            PTB1 b1=new PTB1();
            b1.tinhPTB1(b, c);
        }else if(delta >=0){
            double x1=(-b+Math.sqrt(delta))/(2*a);
            double x2=(-b-Math.sqrt(delta))/(2*a);
            System.out.println("PT có 2 nghiệm phân biệt x1 = " +x1 + " và x2 = "+x2);

        }else System.out.println("PT vô nghiệm!!!");
        }
}
```



```
public class ptb2 {  
    public static void main(String[] args) {  
        PTB1 pt1=new PTB1();  
        //pt1.nhapPTB1(); // Lỗi do private  
        pt1.tinhPTB1(4, 2);  
        giaiPTB2 pt2=new giaiPTB2();  
        pt2.nhapPTB2();  
        pt2.tinhPTB2();  
    }  
}
```



# Package

Gói	Mô tả
<b>java.lang</b>	Không cần phải khai báo nhập. Gói này luôn được nhập cho bạn.
<b>java.io</b>	Bao gồm các lớp để trợ giúp cho bạn tất cả các thao tác vào ra.
<b>java.applet</b>	Bao gồm các lớp để bạn cần thực thi một applet trong trình duyệt.
<b>java.awt</b>	Các thành phần để xây dựng giao diện đồ họa (GUI).
<b>java.util</b>	Cung cấp nhiều lớp và nhiều giao diện tiện ích khác nhau như là các cấu trúc dữ liệu, lịch, ngày tháng, v.v..
<b>java.net</b>	Cung cấp các lớp và các giao diện cho việc lập trình mạng TCP/IP.

- ❖ Đối tượng & Lớp
- ❖ Kế thừa
- ❖ Đa hình
- ❖ Gói (Packages)
- ❖ Giao diện (Interface)
- ❖ Một số lớp tiện ích trong Java



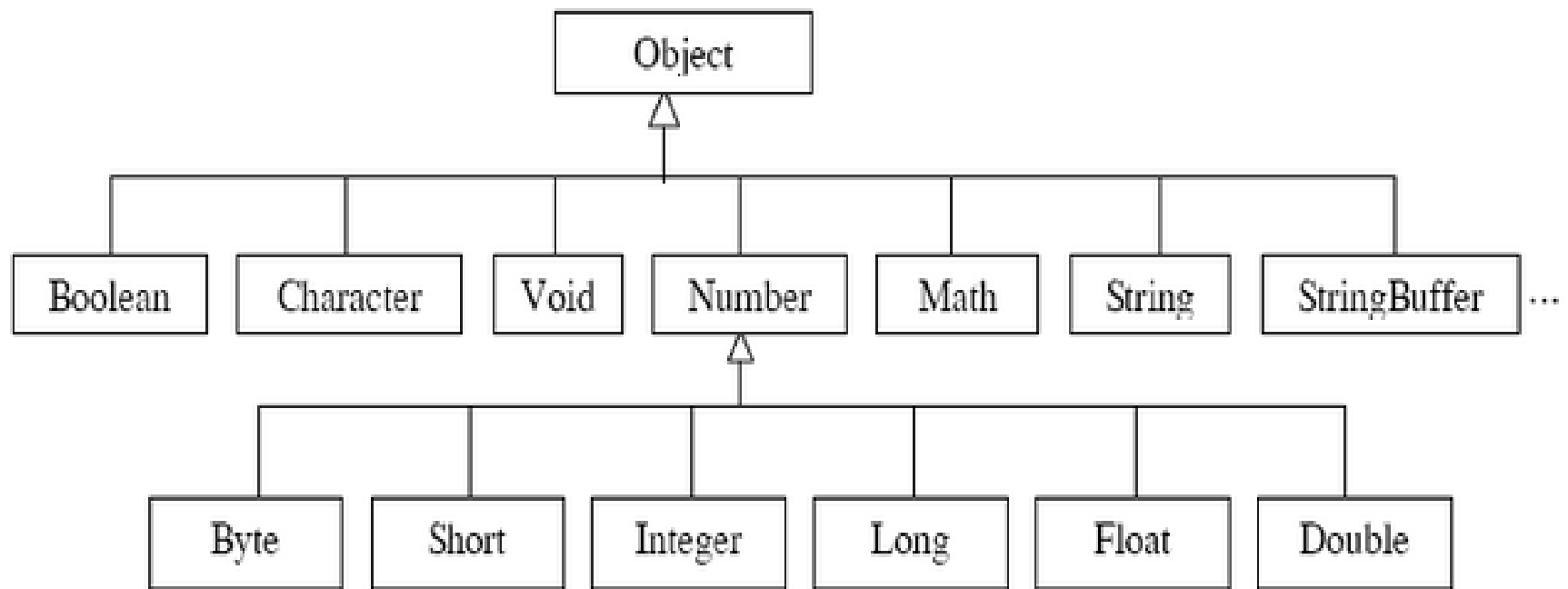


# Một số lớp tiện ích trong Java

- 1. Lớp bao gói kiểu nguyên thủy (Wrapped class)**
- 2. Lớp Math**
- 3. Lớp String**
- 4. Lớp StringBuffer.**
- 5. Lớp Object**



# Lớp bao gói kiểu nguyên thủy





# Lớp bao gói kiểu nguyên thủy

Kiểu dữ liệu	Lớp bao gói
boolean	Boolean
byte	Byte
char	Character
double	Double
float	Float
int	Integer
long	Long
short	Short

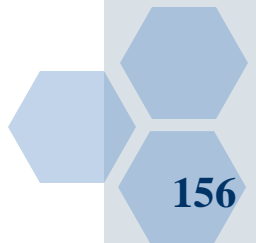


# Lớp bao gói kiểu nguyên thủy

**VD:**

*class CmdArg*

```
{  
    public static void main(String args[])  
    {  
        int tong = 0;  
        for(int i = 0; i < args.length; i++)  
            tong += Integer.parseInt(args[i]); // chuyển đổi  
                                                kiểu String sang kiểu nguyên Integer  
        System.out.println( "Tổng là: " + tong);  
    }  
}
```



❖ Khai báo: **String tênBiếnChuỗi;**

❖ VD: **String hoTen;**

❖ Khởi tạo:

- **tenBiếnChuỗi**="Hello Java";
- Hoặc **tenBiếnChuỗi**= **new String**("Chuỗi cần tạo");
- VD: **hoTen**=**new String** ("Nguyễn Văn A");



# Lớp String

## ❖ Khai báo và khởi tạo:

- Cú pháp: `String tênBiếnChuỗi="Chuỗi cần tạo";`

**Hoặc `String tênBiếnChuỗi =new String("Chuỗi cần tạo");`**

## ❖ VD:

`String monhoc="Lập trình Java";`

`String mon hoc=new String("Tin học ứng dụng");`

**❖ Chú ý: Đối tượng chuỗi có tính bất biến và không thể thay đổi nội dung khi chúng đã được tạo ra.**



# Các phương thức lớp String

**1. charAt():** trả về 1 ký tự tại 1 vị trí trong chuỗi.

Vd: `String name=new String ("Java Language");`  
`char ch=name.charAt(5); // ch chứa giá trị 'L'.`

**2. startsWith():** trả về giá trị kiểu logic (Boolean), phụ thuộc vào chuỗi có bắt đầu với chuỗi con cụ thể nào đó không?

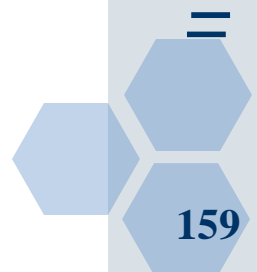
**VD:** *String strname = "Java Language";*

*boolean*

*flag*

*strname.startsWith("Java");*

=> Biến "flag" chứa giá trị true.





# Các phương thức lớp String

3. `endsWith()`: trả về giá trị kiểu logic (Boolean), phụ thuộc vào chuỗi có kết thúc với chuỗi con nào đó không?

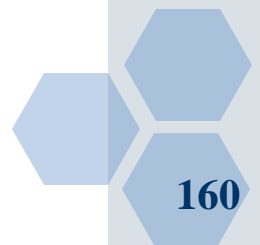
VD: *String strname = "Java Language";*  
*boolean flag = strname.endsWith("Java");*

=> Biến “flag” chứa giá trị false.

4. `copyValueOf()`: Trả về một chuỗi được rút ra từ một mảng ký tự được truyền như một đối số.

VD: *char name[] = {'L', 'a', 'n', 'g', 'u', 'a', 'g', 'e'};*  
*String subname = String.copyValueOf(name, 5, 2);*

=> biến “subname” chứa chuỗi “ag”.







# Các phương thức lớp String

**5. toCharArray():** chuyển chuỗi thành một mảng ký tự

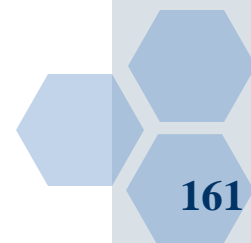
**VD:** *String text = new String("Hello World");*  
*char textArray[] = text.toCharArray( );*

**6. toUpperCase():** trả về chữ hoa của chuỗi

*String lower = new String("good morning");*  
*System.out.println("Uppercase: "+lower.toUpperCase( ));*

**7. toLowerCase():** trả về chữ thường của chuỗi

*String upper = new String("APTECH");*  
*System.out.println("Lowercase: "+upper.toLowerCase( ));*





# Lớp String

## ○ Một số phương thức quan tâm:

- `String()` → chuỗi rỗng (null string) ;
- `String(String value)` → Tạo chuỗi value.
- `length(str)` → độ dài chuỗi str
- `charAt(int index)` → ký tự ở chỉ số index
- `trim(str)` → cắt bỏ khoảng trắng thừa bên trái và phải của str
- `compareTo(String anotherStr)` → <0: nhỏ hơn, 0: bằng nhau, >0: lớn hơn
- `compareToIgnoreCase( String anotherStr);` // không phân biệt hoa thường
- `toLowerCase(str)` → chuyển thành chữ thường của str
- `toUpperCase(str)` → chuyển thành chữ in hoa cho chuỗi str
- `substring(int beginIndex)` → Tạo chuỗi con mới bắt đầu ở chỉ số beginIndex
- `substring(int beginIndex, int endIndex)` → chuỗi con từ beginIndex đến endIndex
- `equals(Object anObject)` → So sánh **giá trị** của đối tượng: true nếu bằng và ngược lại.
- toán tử `firstObjectString == secondObjectString` → So sánh **địa chỉ** đối tượng (địa chỉ reference chứa chuỗi): true nếu hai đối tượng bằng nhau và ngược lại.



# Lớp String

Ví dụ : Tạo, so sánh giá trị và địa chỉ của chuỗi

```
public class Main {  
    public static void main(String[] args) {  
        String s1 = new String("Hello!");  
        String s2 = new String("Hello!");  
        String s3 = s2;  
        String s4 = "Hello!";  
        String s5 = "Hello!";  
  
        System.out.println("s1.equals(s2): " + s1.equals(s2));  
        System.out.println("s1 == s2: " + (s1 == s2));  
  
        System.out.println("s2.equals(s3): " + s2.equals(s3));  
        System.out.println("s2 == s3: " + (s2 == s3));  
  
        System.out.println("s1.equals(s4): " + s1.equals(s4));  
        System.out.println("s1 == s4: " + (s1 == s4));  
  
        System.out.println("s4.equals(s5): " + s4.equals(s5));  
        System.out.println("s4 == s5: " + (s4 == s5));  
    }  
}
```

```
run:  
s1.equals(s2): true  
s1 == s2: false  
s2.equals(s3): true  
s2 == s3: true  
s1.equals(s4): true  
s1 == s4: false  
s4.equals(s5): true  
s4 == s5: true
```



# Lớp String

## ❖ Truyền tham số bằng chuỗi String

```
public class PassStr {  
    static void kiemTra_Str(String tsht) {  
        tsht = "Chuoi da cap nhat!";  
    }  
    public static void main(String[] args) {  
        String tst = new String("Chuoi ban dau!");  
        System.out.println("Gia tri cua tham so thuc, tst: ");  
        System.out.println("\t>> TRUOC khi ra loi goi  
            kiemTra_Str: "+tst);  
        kiemTra_Str(tst);  
        System.out.println("\t>>SAU khi kiemTra_Str hoan  
            thanh: " + tst);  
    }  
}
```





# Lớp StringBuffer

- ❖ Đối tượng String khi tạo ra thì nội dung cố định
- ❖ Đối tượng StringBuffer khi tạo ra thì có thể thay đổi được => linh hoạt hơn String.
- ❖ Sử dụng toán tử new để khởi tạo đối tượng.
  - `newStringBuffer()`: khởi tạo đối tượng StringBuffer có độ dài mặc định và không chứa ký tự nào.
  - `newStringBuffer(int length)`: có độ dài length.
  - `newStringBuffer(String s)`: chứa chuỗi s và có độ dài  $16 + s.length()$ .
- ❖ Lớp String Buffer cung cấp các phương thức khác nhau để thao tác một đối tượng dạng chuỗi.
- ❖ Sử dụng phương thức `toString()` để đổi StringBuffer sang String.





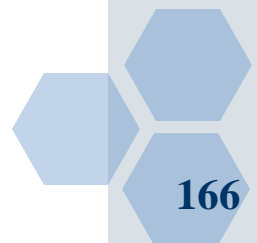


# Lớp StringBuffer

## *class StringBufferDemo*

```
{
    public static void main(String args[])
    {
        StringBuffer s1 = new StringBuffer();
        StringBuffer s2 = new StringBuffer(20);
        StringBuffer s3 = new StringBuffer( "StringBuffer");

        System.out.println( "s3 = "+ s3);
        System.out.println(s2.length()); //chứa 0
        System.out.println(s3.length()); //chứa 12
        System.out.println(s1.capacity()); //chứa 16
        System.out.println(s2.capacity()); //chứa 20
        System.out.println(s3.capacity()); //chứa 28
    }
}
```





# Lớp StringBuffer

❖ **Phương thức `length()` và `capacity()` của `StringBuffer` là hoàn toàn khác nhau.**

- `length()`: số các ký tự mà đối tượng thực chứa.
- `capacity()`: tổng dung lượng của 1 đối tượng (mặc định là 16) và số ký tự trong đối tượng `StringBuffer`.



# Các phương thức StringBuffer

- ❖ **append():** nối thêm 1 chuỗi hoặc một mảng ký tự vào cuối cùng của đối tượng StringBuffer.

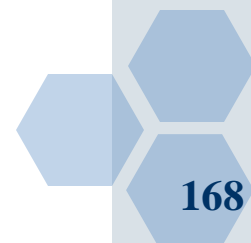
VD: *StringBuffer s1 = new StringBuffer("Good");*  
*s1.append("evening");*

=> Giá trị trong s1 bây giờ là "goodevening".

- ❖ **insert():** có 2 tham số, tham số đầu là vị trí chèn và tham số thứ 2 có thể là 1 chuỗi, 1 ký tự, 1 số được chèn vào. Điều kiện:  $0 \leq \text{vị trí chèn} \leq \text{chiều dài đối tượng StringBuffer}$  (đối số phải chuyển sang chuỗi mới được chèn vào).

VD: *StringBuffer str = new StringBuffer("Java sion");*  
*str.insert(1, 'b');*

=> Biến "str" chứa chuỗi "Jbava sion".







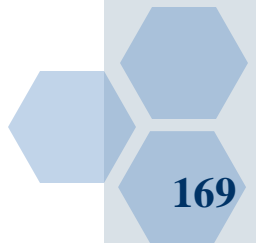
# Các phương thức StringBuffer

❖ **charAt():** trả về một giá trị ký tự trong đối tượng **StringBuffer** tại vị trí được chỉ định.

Ví dụ: *StringBuffer str = new StringBuffer("Sinh vien");*  
*char letter = str.charAt(5); //chứa "v"*

❖ **setCharAt():** dùng để thay thế ký tự trong một **StringBuffer** bằng 1 ký tự khác tại 1 vị trí chỉ định.

VD: *StringBuffer name = new StringBuffer("Jawa");*  
*name.setCharAt(2, 'v');*  
=> Biến "name" chứa "Java".





# Các phương thức StringBuffer

## ❖ **setLength():** thiết lập chiều dài của đối tượng StringBuffer

VD: *StringBuffer str = new StringBuffer(10);*  
*str.setLength(str.length() + 10);*

## ❖ **getChars():** trích ra các ký tự từ đối tượng StringBuffer và sao chép chúng vào một mảng, gồm 4 tham số:

- Chỉ số đầu: vị trí bắt đầu, từ nơi mà ký tự lấy ra.
- Chỉ số kết thúc: vị trí kết thúc.
- Mảng: Mảng đích, nơi ký tự được sao chép.
- Vị trí bắt đầu trong mảng đích: Ký tự được sao chép vào mảng đích từ vị trí này.





# Các phương thức StringBuffer

❖ VD: *StringBuffer str = new  
StringBuffer("Leopard");*

*char ch[] = new char[10];*

*str.getChars(3,6,ch,0);*

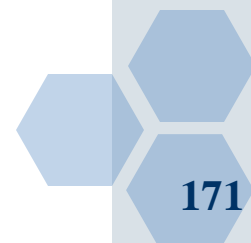
=> Bây giờ biến "ch" chứa "par"

❖ reverse(): đảo ngược nội dung của một đối tượng StringBuffer, sau đó trả về một đối tượng StringBuffer khác.

❖ VD: *StringBuffer str = new StringBuffer("devil");*

*StringBuffer strrev = str.reverse();*

=> Biến "strrev" chứa "lived".





# Lớp StringBuffer

```
public class PassStrBuffer {  
    static void kiemTra_Str(StringBuffer tsht) {  
        System.out.println("\t\tTruoc khi cap nhat, tsht= "+ tsht);  
        tsht.delete(0,tsht.length()).append("Chuoi ban dau da thay doi!");  
        System.out.println("\t\tSau khi da cap nhat, tsht= "+ tsht);  
    }  
    public static void main(String[] args) {  
        StringBuffer tst = new StringBuffer("Chuoi ban dau!...");  
        System.out.println("Gia tri cua tham so thuc, tst: ");  
        System.out.println("\t>>TRUOC khi ra loi goi kiemTra_Str: "+tst);  
        kiemTra_Str(tst);  
        System.out.println("\t>>SAU khi kiemTra_Str hoan thanh: "+tst);  
    }  
}
```



# Bài tập quản lý giảng viên

- ❖ Một trung tâm tin học cần quản lý giảng viên cơ hữu và giảng viên thỉnh giảng.
- ❖ Giảng viên cơ hữu ký hợp đồng lao động lớn hơn 1 năm được hưởng thu nhập hàng tháng bao gồm lương thỏa thuận cố định và lương cộng thêm trong trường hợp vượt giờ quy định trong tháng (số giờ quy định là 40 giờ).
- ❖ Giảng viên tham gia giảng dạy thỉnh giảng ký hợp đồng lao động theo từng lớp học được hưởng thu nhập hàng tháng theo số giờ lên lớp. Biết rằng mỗi giờ dạy có giá 200.000 VNĐ





# Bài tập quản lý giảng viên

- ❖ Thông tin giảng viên cơ hữu: tên giảng viên, email, địa chỉ, điện thoại, số giờ giảng dạy trong tháng, lương thỏa thuận và số giờ quy định chung trong tháng.
- ❖ Thông tin giảng viên thỉnh giảng: tên giảng viên, email, địa chỉ, điện thoại, cơ quan làm việc, số giờ giảng dạy trong tháng.



# Bài tập quản lý giảng viên

- ❖ Hãy xây dựng chương trình cho phép nhân viên trong trung tâm thực hiện các chức năng sau:
  - Nhập thông tin của các giảng viên
  - Xuất danh sách toàn bộ giảng viên
  - Xuất danh sách giảng viên cơ hữu
  - Xuất danh sách giảng viên thỉnh giảng
  - Tính tổng số tiền lương của toàn bộ giảng viên
  - Tìm loại giảng viên có tổng lương cao nhất