LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Bài 4: Lập trình HĐT với Java

Trinh Thi Van Anh – PTIT

Nội dung

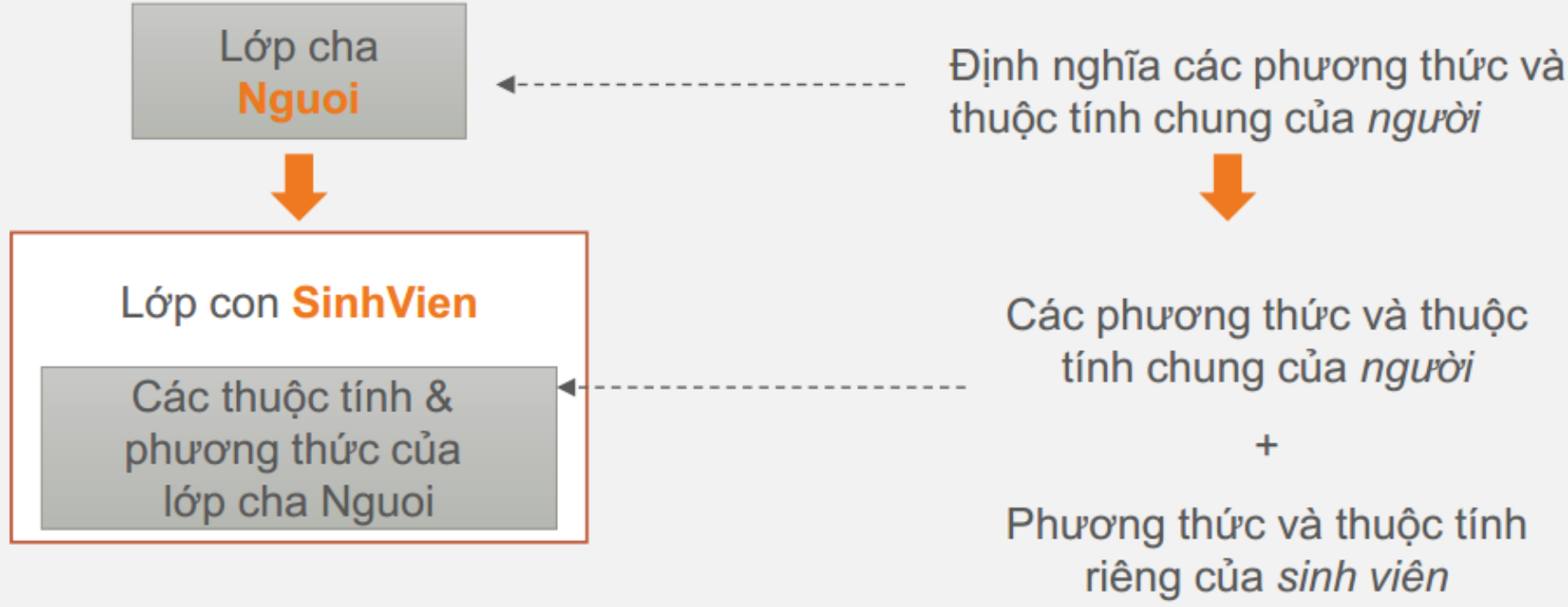
- Khái niệm kế thừa
- Biểu diễn quan hệ kế thừa trong biểu đồ lớp
- Nguyên lý kế thừa
- Khởi tạo và hủy bỏ đối tượng lớp con
- Kế thừa kép (Multi-Inheritance)
- Các lớp trừu tượng (Abtract Classes)
- Interface
- Upcasting và downcasting

Khái niệm kế thừa (Inheritance)

- Kế thừa? Xây dựng các lớp mới có sẵn các đặc tính của lớp cũ, đồng thời chia sẻ hay mở rộng các đặc tính sẵn có
- Bản chất: phát triển lớp mới dựa trên các lớp đã có
- Ví dụ
  - Lớp Người có các thuộc tính như tên, tuổi, chiều cao, cân nặng…; các phương thức như ăn, ngủ, chơi…
  - Lớp Sinh Viên thừa kế từ lớp Người, thừa kế được các thuộc tính tên, tuổi, chiều cao, cân nặng…; các phương thức ăn, ngủ, chơi…
  - Bổ sung thêm các thuộc tính như mã số sinh viên, số tín chỉ tích lũy…, các phương thức như tham dự lớp học, thi…

Khái niệm

- Lớp cũ: Lớp cha (parent, superclass), lớp cơ sở (base class)
- Lớp mới: Lớp con (child, subclass), lớp dẫn xuất (derived class)
- Ví dụ: SinhVien thừa kế (dẫn xuất) từ lớp Nguoi



Mối quan hệ kế thừa

- Cả GiaoVien và SinhVien đều có quan hệ là (is-a) với lớp Nguoi
- Cả giáo viên và sinh viên đều có một số hành vi thông thường của con người
- Lớp con
  - Là một loại (is-a-kind-of) của lớp cha
  - Kế thừa các thành phần dữ liệu và các hành vi của lớp cha
  - Chi tiết hóa cho phù hợp với mục đích sử dụng mới
    - Extension: Thêm các thuộc tính/hành vi mới
    - Redefinition (Method Overriding): Chỉnh sửa lại các hành vi kế thừa từ lớp cha

Cú pháp

- Cú pháp:

```
<Lớp con> extends <Lớp cha>
```

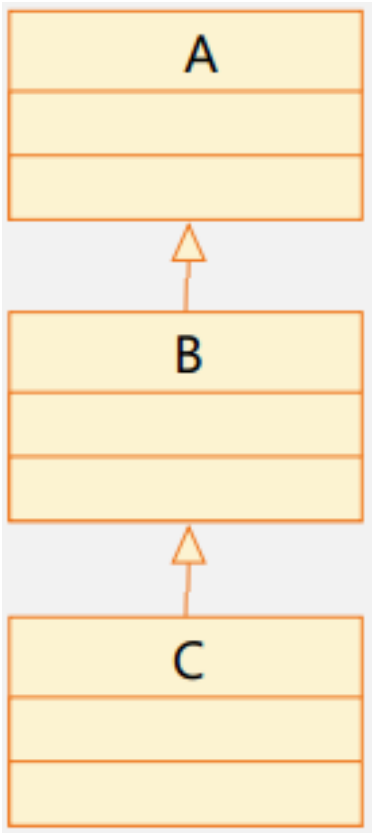
- Ví dụ

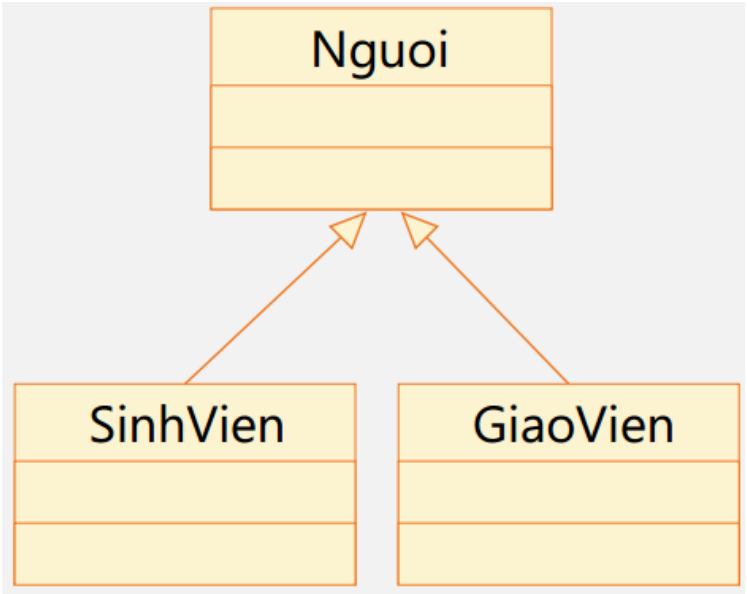
```
class Nguoi {
   String name;
   int age;}
class SinhVien extends Nguoi {
    int studentId;}
```

- Lớp con mở rộng các đặc tính của lớp cha
- Bản chất kế thừa: Là một kỹ thuật tái sử dụng mã nguồn thông qua lớp

Biểu diễn quan hệ kế thừa trong biểu đồ lớp

- Dùng mũi tên với tam giác rỗng ở đầu
- Cấu trúc phân cấp hình cây, biểu diễn mối quan hệ kế thừa giữa các lớp.
- Dẫn xuất trực tiếp: B dẫn xuất trực tiếp từ A
- Dẫn xuất gián tiếp: C dẫn xuất gián tiếp từ A





Lớp Object

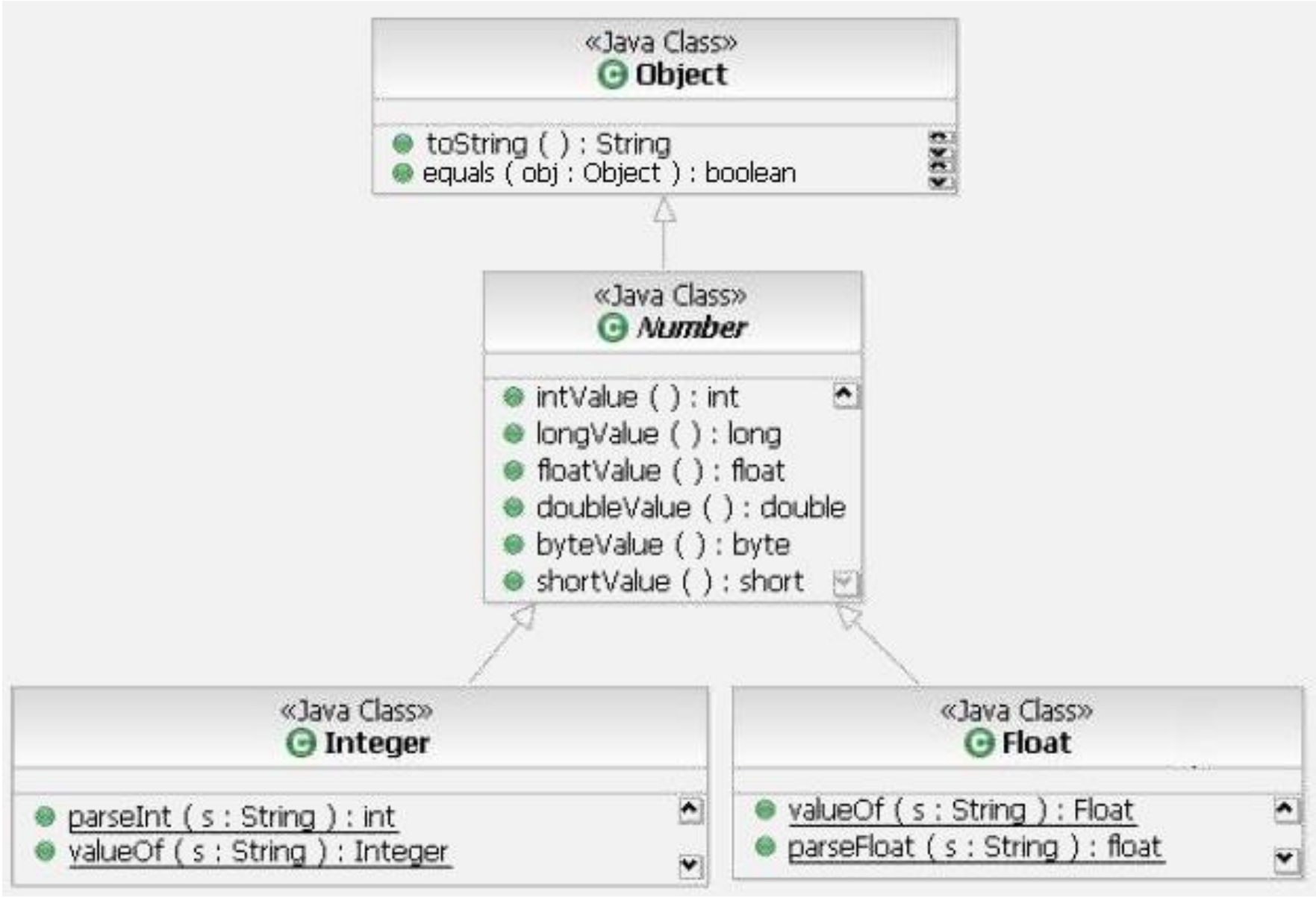
- Lớp Object là lớp gốc trên cùng của tất cả các cây phân cấp kế thừa
  - Nếu một lớp không được định nghĩa là lớp con của một lớp khác thì mặc định nó là lớp con trực tiếp của lớp Object.
- Được định nghĩa trong package chuẩn java.lang

- Mọi lớp trong Java đều được mặc định thừa kế lớp Object dù không khai báo dung từ khóa extends.

`public class SinhVien {…}` Tương đương với

```
public class SinhVien extends Object {…}
```

- Một số phương thức trong lớp Object
- `public String toString()`: trình bày object như là 1 chuỗi
- `public boolean equals(Object obj)`: dùng để so sánh 2 đối tượng
- `public int hashCode()`: trả về 1 mã băm dùng trong việc xác định đối tượng trong 1 tập hợp.
- `Class getClass()`: trả lại tên lớp của đối tượng hiện thời.


«Java Class»
Object
toString ( ) : String
equals (obj : Object) : boolean
«Java Class»
Number
intValue ( ) : int
longValue ( ) : long
floatValue ( ) : float
doubleValue ( ) : double
byteValue ( ) : byte
shortValue ( ) : short
«Java Class»
Integer
parseInt (s : String) : int
valueOf (s : String) : Integer
«Java Class»
Float
valueOf (s : String) : Float
parseFloat (s : String) : float

Nguyên lý kế thừa

- Lớp con có thể thừa kế được gì từ lớp cha?
  - Kế thừa được các thành viên được khai báo là **public** và **protected** của lớp cha.
  - **Không** Goi được các thành viên **private**.
- Thành viên **protected** trong lớp cha được truy cập trong:
  - Các thành viên lớp cha
  - Các thành viên lớp con
  - Các thành viên các lớp cùng thuộc 1 package với lớp cha

	public	protected	*mặc định*	private
Cùng lớp	✔	✔	✔	✔
Lớp bất kỳ cùng gói	✔	✔	✔	✘
Lớp con khác gói	✔	✔	✘	✘
Lớp bất kỳ khác gói	✔	✘	✘	✘

- Các phương thức **không** được phép kế thừa:
  - Các phương thức khởi tạo và hủy
    - Làm nhiệm vụ khởi đầu và gỡ bỏ các đối tượng
    - Chúng chỉ biết cách làm việc với từng lớp cụ thể
  - Toán tử gán (=)
    - Làm nhiệm vụ giống như phương thức khởi tạo

Câu hỏi

- Nếu Thuộc tính được khai báo private, lớp con có được kế thừa không?
  - không. Lớp con kế thừa tất cả trừ những gì ghi đè.
- Nếu phương thức được khai báo private, thì lớp con có đươc gọi không?
  - không! Chỉ những code bên trong cùng lớp mới gọi được phương thức private.
- Muốn gọi thì phải như thế nào?
  - Sử dụng `protected`

Ví dụ (ViDu) - modeldiem

```
public class Diem {
    private int x,y;
    public Diem(int x,int y){
        this.x=x;  this.y=y;}
    …..//getter & setter }
public class TuGiac {
    protected Diem d1, d2, d3, d4;
    public void setD1(Diem d1) {this.d1=d1;}
     public Diem getD1(){return d1;}
    @Override
    public String toString(){
      return
  d1.getX()+"\t"+d1.getY()+"\t"+d2.getX()+"\t"+d2.ge
  tY()+
  "\t"+d3.getX()+"\t"+d3.getY()+"\t"+d4.getX()+"\t"+
  d4.getY(); }}
```

Sử dụng các thành phần protected của lớp cha trong lớp con

```
public class HinhVuong extends TuGiac{
  public HinhVuong(){
      d1 = new Diem(0,0); d2 = new Diem(0,1);
      d3 = new Diem(1,0); d4 = new Diem(1,1);
    }
}
public class Main {
    public static void main(String args[]){
        HinhVuong hv = new HinhVuong();
        System.out.println(hv.toString());
    }
}
```

Gọi phương thức public của lớp cha trong đối tượng lớp con

Vidu2

```
public class Person {
    private String name;
    private String bithday;
    public Person() {
    }  // getter & setter }
public class Employee extends Person {
    private double salary;
    public double getSalary() { return salary; }
    public void setSalary(double salary){
        this.salary = salary;    }
    @Override
    public String toString() {
 //return name + "\t" + birthday + "\t" + salary;
       return super.getName() + "\t" +
  super.getBithday() + "\t" + salary;
    }}
```

```
public class Main {
    public static void main(String
  args[]){
        Employee e = new Employee();
        e.setName("To Ngoc Van");
        e.setBithday("3/4/1994");
        e.setSalary(4.4);
  System.out.println(e.toString());
    }
}
```

Cùng gói và Khác gói

```
package vidu2;
public class Person1 {
    String name;  String bithday;
    public Person1(){} // getter và setter …}
package vidu2.person;
import vidu2.Person1;
public class Employee1 extends Person1 {
    private double salary;
    public double getSalary() {return salary;    }
    public void setSalary(double salary) {
        this.salary = salary; }
    public String toString() {
   //return name + "\t" + bithday + "\t" + salary;
     return super.getName() + "\t" +
  super.getBithday() + "\t" + salary;
    }}
```

```
package vidu2;
public class Person2 {
    protected String name;
    protected String bithday;
    public Person2() {}}
package vidu2.person;
import vidu2.Person2;
public class Emploee2 extends Person2 {
    private double salary;
    public double getSalary() {return salary;}
    public void setSalary(double salary) {
        this.salary = salary;}
    public String toString() {
        return name + "\t" + bithday + "\t" +
  salary; // ERROR???
    }}
```

Khởi tạo và hủy bỏ đối tượng

- Khởi tạo đối tượng:
  - Lớp cha được khởi tạo trước lớp con.
  - Các phương thức khởi tạo của lớp con luôn gọi phương thức khởi tạo của lớp cha ở câu lệnh đầu tiên
    - Tự động gọi (ngầm định - implicit): Khi lớp cha CÓ phương thức khởi tạo mặc định (hoặc ngầm định)
    - Nếu không có lời gọi tương minh đến hàm khởi tạo của lớp cha tại lớp con, trình biên dịch sẽ tự động chèn lời gọi tới hàm dựng mặc nhiên (implicity) hoặc hàm khởi tạo không tham số (explicity) của lớp cha trước khi thực thi đoạn code khác trong hàm khởi tạo lớp con.
    - Gọi trực tiếp (tường minh - explicit) (Sử dụng từ khóa **super**)
- Hủy bỏ đối tượng:
  - Ngược lại so với khởi tạo đối tượng

Gọi phương thức của lớp cha

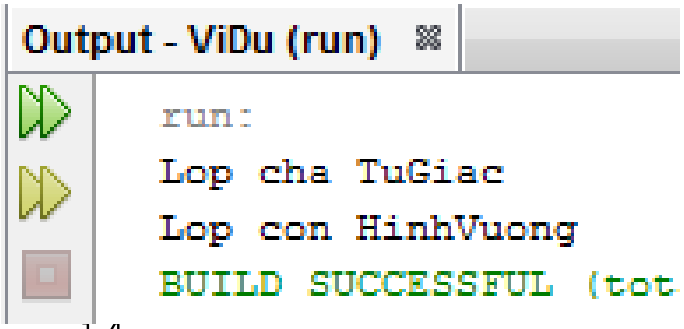
- Tái sử dụng các đoạn mã của lớp cha trong lớp con
- Gọi phương thức khởi tạo

```
super(danh sách tham số);
```

  - Bắt buộc nếu lớp cha không có phương thức khởi tạo mặc định
  - Phải được khai báo tại dòng lệnh đầu tiên trong phương thức khởi tạo của lớp con
- Gọi các phương thức của lớp cha

```
super.tênPt(danh sách tham số);
```

Ví dụ




```
1. public class TuGiac {
     protected Diem d1, d2, d3, d4;
     public TuGiac() {
         System.out.println("Lop cha TuGiac");
     }……}
2. public class HinhVuong extends TuGiac{
   public HinhVuong(){ // Tu dong goi TuGiac()
      System.out.println("Lop con HinhVuong");
     }}
3. public class Main {
     public static void main(String args[]){
         HinhVuong hv = new HinhVuong();
     }}
```

```
public class TuGiac {
protected Diem d1, d2;
protected Diem d3, d4;
public TuGiac(Diem d1, Diem d2, Diem d3, Diem d4) {
System.out.println("Lop cha TuGiac(d1, d2, d3, d4)");
this.d1 = d1; this.d2 = d2;
this.d3 = d3; this.d4 = d4;
}}
public class HinhVuong extends TuGiac {
public HinhVuong() { // lỗi
System.out.println("Lop con HinhVuong()");
}}
public class Test {
public static void main(String arg[]) {
HinhVuong hv = new HinhVuong();
}}
```

```
public class TuGiac {
protected Diem d1, d2, d3, d4;
public TuGiac(Diem d1, Diem d2, Diem d3, Diem d4) {
System.out.println("Lop cha TuGiac(d1, d2, d3, d4)");
this.d1 = d1; this.d2 = d2;
this.d3 = d3; this.d4 = d4;
}}
public class HinhVuong extends TuGiac {
public HinhVuong() {
super(new Diem(0,0), new Diem(0,1),
new Diem(1,1),new Diem(1,0));
System.out.println("Lop con HinhVuong()");
}}
public class Test {
public static void main(String arg[]) {
HinhVuong hv = new HinhVuong();
}}
```

```
public class TuGiac {
protected Diem d1, d2, d3, d4;
public TuGiac(Diem d1, Diem d2, Diem d3, Diem d4) {
System.out.println("Lop cha TuGiac(d1, d2, d3, d4)");
this.d1 = d1; this.d2 = d2;
this.d3 = d3; this.d4 = d4;}}
public class HinhVuong extends TuGiac {
public HinhVuong(Diem d1, Diem d2, Diem d3, Diem d4){
super(d1, d2, d3, d4);
System.out.println("Lop con HinhVuong(d1, d2, d3,
  d4)");}}
public class Test {
public static void main(String arg[]) {
HinhVuong hv = new HinhVuong(
(new Diem(0,0), new Diem(0,1),
new Diem(1,1),new Diem(1,0)));
}}
```

Câu hỏi

```
public class Foo {
    public void method1() {
        System.out.println("foo 1");
    }

    public void method2() {
        System.out.println("foo 2");
    }

    public String toString() {
        return "foo";
    }
}

public class Bar extends Foo {
    public void method2() {
        System.out.println("bar 2");
    }
}
```

```
public class Baz extends Foo {
    public void method1() {
        System.out.println("baz 1");
    }

    public String toString() {
        return "baz";
    }
}

public class Mumble extends Baz {
    public void method2() {
        System.out.println("mumble
  2");
    }
}
```

output ?

```
Foo[] pity = { new Baz(),
        new Bar(),
                new Mumble(),
        new Foo() };

for (int i = 0; i < pity.length; i++) {
    System.out.println(pity[i]);
    pity[i].method1();
    pity[i].method2();
    System.out.println();
}
```

output

```
baz
baz 1
foo 2

foo
foo 1
bar 2

baz
baz 1
mumble 2

foo
foo 1
foo 2
```

Variable Shadowing

- Chỉ nên áp dụng **đa hình (nạp chồng)** cho phương thức trong Java
- ***Không*** *được với trường (thuộc tính)!*

```
public class A {
  int x = 1;
  int method() { return 1; }
}
public class B extends A {
  int x = 2;
  int method() { return 2; }
}
```

```
A a1 = new A();
A a2 = new B();
B b1 = (B)a2;

System.out.println(a1.method());
// prints 1
System.out.println(a2.method());
// prints 2
System.out.println(a1.x);
// prints 1
System.out.println(a2.x);
???
System.out.println(b1.x);
???
```

Phương thức finalize()

- Java **không** có phương thức huỷ bỏ đối tượng. (destructor). Java có các trình dọn dẹp cài đặt sẵn (garbage. collection system), còn gọi là bộ thu gom rác. Phương thức này được gọi là **finalize()**, và nó có thể được sử dụng để bảo đảm rằng đã hoàn toàn kết thúc một đối tượng.
- Ví dụ:

```
ObjectDemo cal = new ObjectDemo();
System.out.println(""+cal.getTime());
System.out.println("Finalizing...");
cal.finalize();
System.out.println("Finalized.");
```

Bài toán(GiaoDich)

Xây dựng chương trình quản lý danh sách các giao dịch. Hệ thống gồm 2 loại giao dịch:

- Giao dịch vàng: Mã giao dịch, ngày giao dịch (ngày, tháng, năm), đơn giá, số lượng, **loại vàng**.

  Thành tiền được tính như sau:

  thành tiền = số lượng * đơn giá.

- Giao dịch tiền tệ: Mã giao dịch, ngày giao dịch (ngày, tháng, năm), Đơn giá, số lượng, **tỉ giá, loại tiền tệ** có 3 loại: tiền Việt Nam, tiền USD, tiền Euro. Thành tiền được tính như sau:
  - Nếu là tiền USD hoặc Euro thì: thành tiền = số lượng * đơn giá* tỉ giá
  - Nếu là tiền VN thì: thành tiền = số lượng * đơn giá

Thực hiện bài toán

- Thực hiện các yêu cầu sau:
  - 1. Them Giao Dich Vang
  - 2. Them Giao Dich Tien Te
  - 3. Hien Thi Danh Sach Giao Dich
  - 4. Xem Tong So Luong cua cac Giao Dich
  - 5. Hien Thi Danh Sach Giao Dich theo ngay
  - 6. Hien Thi DS Giao Dich co chua ngay
  - 7. Hien Thi DS Giao Dich theo Tu nam den Nam
  - 0. Thoat

Tạo lớp cha (GiaoDich)

```
public class GiaoDich{
    protected int MaGD,SoLuong;
    protected String NgayGD;
    protected double DonGia, ThanhTien;
    protected GiaoDich(){
        this.MaGD=0;
        this.NgayGD="";
        this.DonGia=0;
        this.SoLuong=0;
        this.ThanhTien=0;
    }
```

```
protected GiaoDich(int magd){
        this.magd=magd;
        this.ngaygd="";
        this.dongia=0;
        this.soluong=0;
        this.thanhtien=0;
    }
protected GiaoDich(int ma, int sl, String
  ngay, double dgia){
        this.magd=ma;
        this.ngaygd=ngay;
        this.dongia=dgia;
        this.soluong=sl;
        this.thanhtien=0;
    }
```

```
// Các getter và setter cho các thuộc
  tính
public void setThanhtien(double
  thanhtien){
        this.thanhtien = thanhtien;
    }
public void setThanhtien(){
        thanhtien = dongia*soluong;
    }
public String toString() {
    return magd + "\t" + soluong
     + "\t" + ngaygd + "\t" + dongia;
    }
```

```
// chỉ cần khi ta gọi phương thức
  contains
public boolean equals(Object o) {
  if (this == o)
     return true;
  if (o == null)
     return false;
//   if (getClass() != o.getClass())
//        return false;
  GiaoDich other = (GiaoDich) o;
        if (magd != other.magd)
     return false;
  return true;
    }
```

Tạo lớp con (GiaoDichVang)

```
class GiaoDichVang extends GiaoDich {
    private String loaivang;
    public GiaoDichVang(){
        super();
        this.loaivang="";
    }
public GiaoDichVang(int ma, int sl,
  String ngay, double dgia,
        String lvang){
        super(ma, sl, ngay, dgia);
        this.loaivang=lvang;
    }
```

```
// không cần cài đặt các phương thức getter và
   setter để lấy các thuộc tính của lớp cha nữa
// chỉ cài đặt các phương thức getter và
   setter của thuộc tính lớp mình
public String getLoaivang() {
        return loaivang;
    }
public void setLoaivang(String loaivang) {
        this.loaivang = loaivang;
    }
// viết đè lên phương thức toString() của lớp
   cha nếu thấy cần
public String toString(){
      return super.toString() + "\t" +
        loaivang + "\t\t" + getThanhtien();
    }
```

Tạo lớp con(GiaoDichTienTe)

```
class GiaoDichTienTe extends GiaoDich{
    private double tigia;
    private int loaitiente;
    public GiaoDichTienTe(){
        super();
        this.tigia=0;
        this.loaitiente=0;
    }
public GiaoDichTienTe(int ma, int sl,
     String ngay, double dgia, double tgia,
  int loai){
        super(ma, sl, ngay, dgia);
        this.tigia=tgia;
        this.loaitiente=loai;
    }
```

```
public double getTigia() {
        return tigia;
    }
public void setTigia(double tigia) {
        this.tigia = tigia;
    }
public int getLoaitiente() {
        return loaitiente;
    }
public void setLoaitiente(int loaitiente) {
        this.loaitiente = loaitiente;
    }
public void setThanhien(){
        if(loaitiente==1)
   this.setThanhtien(this.getDongia()*this.getSoluong(
   ));
        else
   this.setThanhtien(this.getDongia()*this.getSoluong(
   )*this.tigia);
    }
```

```
public String toString(){
    String temp;
    if(loaitiente==1)
       temp=" VND";
    else if(loaitiente==2)
          temp=" USD";
        else
          temp=" EURO";
    return super.toString() + "\t" +
    tigia + temp + "\t\t" +
  getThanhtien();
     }
```

Tạo lớp quản lý chung(ListGD)

```
public class ListGD {
   private ArrayList<GiaoDich> list;
   private static int tonggdv,
  tonggdtt;
   public ListGD(){
      list = new
  ArrayList<GiaoDich>();
          tonggdv=0;
          tonggdtt=0;
      }
```

```
private  GiaoDich themGD(){
    Scanner in=new Scanner(System.in);
    System.out.println("Nhap thong tin GD ");
    System.out.println("Ma  GD (nhap so): ");
    int magd = in.nextInt();
    GiaoDich gd = new GiaoDich(magd);
    if(list.contains(gd))
       return null;
    else{
       String ngaygd="";
       while(true){
         System.out.println("Ngay Giao
  Dich(dd-mm-yyyy): ");
         ngaygd = in.next();
```

```
if(ngaygd.matches("\\d{2}-\\d{2}-\\d{4}$"))
      break;
            }
gd.setNgaygd(ngaygd);
System.out.println("Don Gia: ");
double dongia = in.nextDouble();
gd.setDongia(dongia);
System.out.println("So Luong: ");
int soluong = in.nextInt();
gd.setSoluong(soluong);
return gd;
      }
  }
```

```
public boolean themGDVang(){
  Scanner in=new Scanner(System.in);
  GiaoDich gd = themGD();
  if(gd == null)
    return false;
  else{
    System.out.println("Loai Vang: ");
    String loaivang = in.next();
    GiaoDichVang gdv = new
  GiaoDichVang(gd.getMagd(),
  gd.getSoluong(), gd.getNgaygd(),
  gd.getDongia(), loaivang);
    gdv.setThanhtien();
    tonggdv += gdv.getSoluong();
  return list.add(gdv);
            }
    }
```

```
public boolean themGDTT(){
    Scanner in=new Scanner(System.in);
    GiaoDich gd = themGD();
    if(gd == null)
      return false;
    else{
      System.out.println("Loai Tien Te
   (nhap 1:vnd, 2:ngoai te): ");
      int loaitiente=in.nextInt();
      double tigia;
      if(loaitiente==1)
         tigia=1;
      else{
        System.out.println("Ti Gia: ");
        tigia=in.nextDouble();
             }
```

```
GiaoDichTienTe gdtt = new
  GiaoDichTienTe(gd.getMagd(),
  gd.getSoluong(), gd.getNgaygd(),
  gd.getDongia(), tigia,
  loaitiente);
   tonggdtt += gdtt.getSoluong();
   gdtt.setThanhtien();
   return list.add(gdtt);
      }
      }
```

```
public void hienthiDS(){
   System.out.println("==DANH SACH GIAO
   DICH ==");
   System.out.println("MaGD So Luong  Ngay
   GD  Don Gia Loai vang/tygia TTien");
   double tt=0;
   for (GiaoDich gd: list){
     System.out.println(gd.toString());
     tt += gd.getThanhtien();
         }
   System.out.println("==================");
   System.out.println("Tong tien:
   "+Double.toString(tt));
     }
```

```
public void getByDate(String ngaygd){
   System.out.println("==DANH SACH GIAO
   DICH Ngay "+ngaygd+"==========");
   System.out.println("MaGD So Luong
   Ngay GD  Don Gia Loai vang/tygia
   TTien");
   for (GiaoDich gd: list)
     if(gd.getNgaygd().equals(ngaygd))
   System.out.println(gd.toString());
   System.out.println("==============");
     }
```

```
public void getContainDate(String
  ngaygd){
   System.out.println("=====DANH SACH
   GIAO DICH Ngay "+ngaygd+"==========");
   System.out.println("MaGD So Luong
   Ngay GD  Don Gia Loai vang/tygia
   TTien");
   for (GiaoDich gd: list)
   if(gd.getNgaygd().indexOf(ngaygd)>=0)
   System.out.println(gd.toString());
   System.out.println("================");
      }
```

```
public int tongGGVang(){
        return tonggdv;    }
public int tongGGTienTe(){
        return tonggdtt;    }
public void getBySoLuong(int from,int
  to){
  System.out.println("=====DANH SACH GIAO
  DICH Ngay =========");
  System.out.println("MaGD So Luong  Ngay
  GD  Don Gia Loai vang/tygia TTien");
  for (GiaoDich gd: list)
    if((gd.getSoluong()>=from) &&
  (gd.getSoluong()<=to))
  System.out.println(gd.toString());
  System.out.println("==================
  ================");
    }
```

```
public void getByYear(int from,int to){
   System.out.println("=====DANH SACH
  GIAO DICH Ngay =========");
   System.out.println("MaGD So Luong
  Ngay GD  Don Gia Loai vang/tygia
  TTien");
   for (GiaoDich gd: list)
  if((Integer.parseInt((gd.getNgaygd().sub
  string(6)))>=from)
                        &&
  Integer.parseInt((gd.getNgaygd().substri
  ng(6)))<=to)
  System.out.println(gd.toString());
  System.out.println("==================
  ================");
     }
```

```
public void sapXepByNgay(){
  Collections.sort(list, new
  Comparator<GiaoDich>() {
@Override
  public int compare(GiaoDich g1,
  GiaoDich g2) {
    return
  ((g1.getNgaygd().substring(6)+g1.getNgay
  gd().substring(3,5)+g1.getNgaygd().subst
  ring(0,2)).compareTo((g2.getNgaygd().sub
  string(6)+g2.getNgaygd().substring(3,5)+
  g2.getNgaygd().substring(0,2))));
            }
        });
  }
```

Làm thêm???

- Tìm kiếm (tìm gần đúng) theo riêng từng tiêu chí: mã giao dịch, ngày giao dịch,…
- Tìm kiếm theo khoảng xác định của từng tiêu chí: Từ tháng… đến tháng (ngày giao dịch), từ số lượng… đến số lượng (số lượng giao dịch)….
- Tìm 1 số trường (tìm gần đúng)
- Sắp xếp theo: ngày giao dịch, đơn giá, số lượng giao dịch, theo tổng tiền…..
- Đưa ra với tính toán theo tiêu chí

Kế thừa kép (Multi-Inheritance)

- Java **không** cho phép đa kế thừa từ **nhiều lớp** cha/cơ sở
  - Đảm bảo tính dễ hiểu
  - Hạn chế xung đột
- Có thể cài đặt đồng thời nhiều giao diện

Lớp trừu tượng (abstract class)

- Có thể tạo ra các lớp cơ sở để tái sử dụng mà không muốn tạo ra đối tượng thực của lớp ????
- Các lớp Point, Circle, Rectangle chung nhau khái niệm cùng là hình vẽ (Shape)
- → Giải pháp là khái báo lớp trừu tượng
- Lớp trừu tượng được xem như khung làm việc chung cung cấp các hành vi (behavior) cho các lớp khác.
- Không thể tạo đối tượng từ lớp trừu tượng
- Có thể thừa kế từ lớp trừu tượng
- Các lớp con phải cài đặt các phương thức trừu tượng được khai báo trong lớp trừu tượng (lớp cha). **Nếu không nó cũng sẽ trở thành lớp trừu tượng**

"abstract" Methods

- Khai báo lớp trừu tượng bằng cách sử dụng từ khóa **abstract** trước từ khóa **class**.
- Phương thức trừu tượng: Là những phương thức **chỉ** có **khai báo** mà **không** có **phần cài đặt**.
- Có từ khóa "abstract" trong phần khai báo phương thức
- Phần khai báo sẽ **không** có cặp ngoặc ({}) và được kết thúc bởi dấu **;** (semicolon)

Ví dụ

- Giả sử ta có lớp NhanVien (hoten, diachi,hsluong, luong). Nhưng khi tính lương cho các đối tượng nhân viên khác nhau thì khác nhau.

```
public abstract class NhanVien {
    private String hoten,diachi;
    private double hsluong,luong;
    //… getter and setter
    //hoten,diachi,hsluong
    public void setLuong(double luong){
       this.luong = luong;
     }
    public abstract double getLuong();
}
```

Một số qui tắc

- Tất cả lớp con của NhanVien cần có mức lương, nhưng **các lớp con phải tự định nghĩa mức lương của mình**.
- Định nghĩa phương thức getLuong() là *trừu tượng* - **abstract**
- **Không** viết code nhưng phần absract**:**
  - Khai báo phương thức trừu tượng không có code, chỉ cung cấp đinh nghĩa, tương tự như đánh dấu.
  - Sau đó lớp con phải định nghĩa.
- **Nếu lớp có một phương thức trừu tượng,** hoặc nó thừa kế một phương thức trừu tượng, nhưng không ghi đè, thì lớp đó phải được khai báo trừu tượng.

- Nếu lớp là trừu tượng, nó **không** được có **hàm khởi tạo**.

```
public static void main(String [] args) {
    NhanVien nv;// NO ERROR:reference is ok
    nv = new NhanVien();// ERROR! No
  constructor
  }
}
```

- Không đối tượng NhanVien nào được khởi tạo
- Nhưng vẫn khai báo tham chiếu NhanVien được

Thừa kế từ lớp abstract

```
public class NhanVienGiaoVu extends
    NhanVien {
   public double getLuong() {
      return 22*hsluong*1.2+2350000;
   }
   public static void main(String [] args){
     NhanVien nv;     // Fine, no problem
     nv = new NhanVienGiaoVu(); // Also fine
    (polymorphism)
     //nv = new NhanVien(); // ERROR!  No
    constructor!
   }
}
```

Interface (Completely abstract)

- Là 1 lớp bao gồm **hằng** (hằng là khai báo với từ khóa **final** và **khởi tạo giá trị**) và **các phương thức được khai báo nhưng chưa được định nghĩa** (phần thân hàm rỗng).
- Interface chứa các hành vi chuẩn (standard behavior) được thực hiện bởi bất kỳ lớp nào trong hệ thống phân cấp lớp (Nghĩa là các phương thức trong interface có thể được thực hiện bởi các lớp không liên quan gì tới nó)
- Đặc tả cho các bản cài đặt khác nhau (**implements**)

- Định nghĩa 1 kiểu chỉ chứa "định nghĩa hằng" và phương thức trừu tượng (phương thức ko có từ khóa abstract nhưng đc **ngầm hiểu là phương thức abstract**)
- Một lớp cài đặt (**implements**) interface **bắt buộc** phải : hoặc cài đặt tất cả các phương thức đã được khai báo trong interface , hoặc khai báo là lớp trừu tượng.

Chú ý

- Một lớp thi hành các hàm trong interface qua từ khóa : **implements**
- Một lớp có thể implements **nhiều** interface (java không có tính đa thừa kế nên sẽ được thay thế tính **đa thừa kế bằng interface**)
- Dù không khai báo từ khóa abstract trong các hàm của interface, nhưng **các hàm trong interface đều được coi là abstract**.
- Các **biến** trong interface bắt buộc phải là : **public hoặc static hoặc final**
- Không được dùng **private** hay **protected** để khai báo các biến hoặc phương thức trong inteface , vì các phương thức này sẽ **được ghi đè** trong lớp thực thi nó.
- Các **phương thức** trong interface **không thể** là **final**.

Ví dụ (dongvat)

```
public interface DongVatDuoiNuoc {
    public void move();}
public interface DongVatTrenCan {
    public void move();}
public class Ca implements DongVatDuoiNuoc {
    public void move() {
        System.out.println("Swim");
    }}
public class Meo implements DongVatTrenCan {
    public void move() {
        System.out.println("Run");
    }
}
```

```
public class CaSau implements
  DongVatTrenCan, DongVatDuoiNuoc {
     private String ten;
     private int tuoi;
     private static boolean foundTen =
  false;
public CaSau(){
     }
public CaSau(String ten, int tuoi) {
          this.ten = ten;
          this.tuoi = tuoi;
     }
```

```
public void move() {
        System.out.println("Run and swim");
    }
public static boolean isFoundTen() {
        return foundTen;
    }
public String getTen() {
        return ten;
    }
public void setTen(String ten) {
        this.ten = ten;
    }
public int getTuoi() {
        return tuoi;
    }
public void setTuoi(int tuoi) {
        this.tuoi = tuoi;
    }
```

```
public void inputCaSau() {
    Scanner scanner = new
  Scanner(System.in);
    System.out.print("Nhap ten: ");
    setTen(scanner.nextLine());
    System.out.print("Nhap tuoi: ");
    setTuoi(scanner.nextInt());
     }
public void outputCaSau() {
    System.out.println("Ten: " +
  getTen());
    System.out.println("Tuoi: " +
  getTuoi());
}
```

```
public void timTen(String ten) {
   if
   (this.ten.equalsIgnoreCase(ten)) {
      foundTen = true;
      System.out.println("Ten
  Tuoi");
  System.out.println(toString());
         }
    }
public String toString(){
    return ten+"\t"+tuoi;
    }
```

Xây dựng Menu

- 1. Nhap thong tin dan ca sau
- 2. Hien thi thong tin dan ca sau
- 3. Tim kiem ca sau theo ten
- 4. Sap xep dan ca sau theo do tuoi tang dan
- 5. Sap xep dan ca sau theo do tuoi giam dan
- ……..
- 0. Thoat chuong trinh

Sử dụng interface

- Tại sao dùng interface? dùng interface như thế nào?
  - Xây dựng khung cho tập các đối tượng
  - Sử dụng Interface tạo hành động chung cho đối tượng.
  - Truyền nhận dữ liệu, thông điệp giữa các class
  - ….

Xây dựng khung

- Ví dụ List: add, remove, get, set, ….
- Ví dụ interface for sql: Connection, Statement, ….

Tạo hành động chung cho đối tượng

- Giả sử ta có các môn Toán, Lý, Hóa, Sinh,... mỗi môn là một class, ta có một mảng các cuốn sách đó và cần mang tới trường.
- Để tạo ra được mảng sach chúng ta không còn cách nào khác là đặt nó là mảng Object và khi gọi các phương thức mangSachToiTruong() sẽ không thực hiện được.
- Bây giờ **interface** sẽ giúp chúng ta giải quyết vấn đề này. Trước tiên chúng ta tạo một interface **Sach** có phương thức là **mangSachToiTruong();**

file name: TruongHoc.java

```java
class Toan {
    public void mangSachToiTruong() {
        System.out.println("Sach Toan duoc mang toi truong");
    }
}

class Ly {
    public void mangSachToiTruong() {
        System.out.println("Sach Ly nhe");
    }
}

class Hoa {
    public void mangSachToiTruong() {
        System.out.println("Sach Hoa ok");
    }
}

class Sinh {
    public void mangSachToiTruong() {
        System.out.println("Sach Sinh day roi");
    }
}

public class TruongHoc {
    public static void main(String[] args) {
        Object[] sach = { new Toan(), new Ly(), new Hoa(), new Sinh() };
        for (Object s : sach) {
            s.mangSachToiTruong(); // loi roi nhe
        }
    }
}
```

```java
class Toan implements Sach {
    @Override
    public void mangSachToiTruong() {
        System.out.println("Sach Toan duoc mang toi truong");
    }
}

class Ly implements Sach {
    @Override
    public void mangSachToiTruong() {
        System.out.println("Sach Ly nhe");
    }
}

class Hoa implements Sach {
    @Override
    public void mangSachToiTruong() {
        System.out.println("Sach Hoa ok");
    }
}

class Sinh implements Sach {
    @Override
    public void mangSachToiTruong() {
        System.out.println("Sach Sinh day roi");
    }
}

public class TruongHoc {
    public static void main(String[] args) {
        Sach[] sach = { new Toan(), new Ly(), new Hoa(), new Sinh() };
        for (Sach s : sach) {
            s.mangSachToiTruong();
        }
    }
}
```

file name: Sach.java

```java
public interface Sach {
    public void mangSachToiTruong();
}
```

Truyền nhận dữ liệu, thông điệp giữa các class

- Giả sử có 2 class: **GiaoDien** và **Download**. class **GiaoDien** có một nút, khi click vào nút thì gọi phương thức download nằm trong class **Download** để download file, và trong quá trình download, class Download phải phản hồi lại cho class **GiaoDien** biết là download được bao nhiêu phần trăm rồi.

Cách 1

- Tạo lớp **Download** có chứa đối tượng **GiaoDien** để cập nhật phần trăm download

```
public class Download {
   private GiaoDien giaoDien;
   public Download(GiaoDien giaoDien) {
      this.giaoDien = giaoDien;     }
   public void download() {
      for (int i = 0; i < 100; i++) {
       System.out.println("Dang
   download...");
       giaoDien.capNhatGiaoDien(i);
         }
     System.out.println("Ket thuc download");
     }}
```

```
public class GiaoDien {
  private Download download;
  public GiaoDien() {
     download = new Download(this);     }
  private void nhanDownload() {
     download.download();     }
  public void capNhatGiaoDien(int
  phanTramDownload) {
     System.out.println("class giao dien
  cap nhat duoc: " + phanTramDownload
                + "%");
    }
public static void main(String[] args) {
      GiaoDien giaoDien = new GiaoDien();
        giaoDien.nhanDownload();  }}
```

Hạn chế

- Ta thấy thì việc cập nhật giao diện tỏ ra đơn giản khi mà chúng ta chỉ cần truyền chính đối tượng giao diện sang lớp Download là xong.
- Tuy nhiên nhiều vấn đề xảy ra khi mà đối tượng **GiaoDien** được truyền sang, lớp **DownLoad** được phép gọi một số phương thức của lớp GiaoDien mà nó không cần, phương thức cập nhật giao diện bắt buộc phải cho phép gọi ở lớp Download thì mới có thể gọi được mà ở đây chúng ta để là public chứ không thể là private.
- Giả sử như lớp chúng ta phải thông qua một lớp X nào đó ở giữa thì phương thức download của lớp Download mới được gọi đến và như vậy chúng ta phải truyền GiaoDien sang X, rồi X truyền sang Download, như vậy rất phức tạp…

Cách 2: Sử dụng Interface

- Chúng ta tạo thêm một interface tên là **CapNhat** có chứa phương thức **capNhatGiaoDien()**; sau đó cho **GiaoDien** implements

```
public interface CapNhat {
    public void capNhatGiaoDien(int
  giaTri);
}
```

```
public class Download {
   private CapNhat capNhat;
   public void addDownloadListener(CapNhat
   capNhat) {
      this.capNhat = capNhat;
     }
   public void download() {
      for (int i = 0; i < 100; i++) {
         System.out.println("Dang
   download...");
         capNhat.capNhatGiaoDien(i);
         }
     System.out.println("Ket thuc
   download");
     }
}
```

```
public class GiaoDien implements CapNhat {
    private Download download;
    public GiaoDien() {
        download = new Download();
        download.addDownloadListener(this);
    }
    private void nhanDownload() {
        download.download();
    }
    @Override
    public void capNhatGiaoDien(int giaTri) {
        System.out.println("class giao dien
  cap nhat duoc: " + giaTri + "%");
    }
    public static void main(String[] args) {
        GiaoDien giaoDien = new GiaoDien();
        giaoDien.nhanDownload(); }}
```

Nhận xét

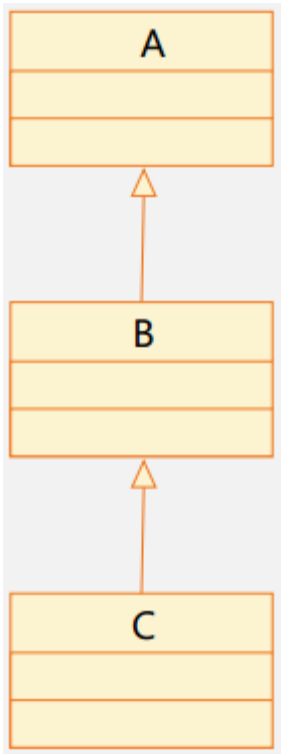
- Có thể thấy rằng cách 2 này khắc phục được các nhược điển của cách 1. Lớp **DownLoad** không cần truyền đối tượng **GiaoDien** mà chỉ cần một interface **CapNhat**, do đó các phương thức, thuộc tính của **GiaoDien** không "bị lộ" hay có thể được gọi bên **Download**. Nếu phải thông qua một lớp X nào đó rồi download mới được thực hiện thì cũng không cần truyền đối tượng **GiaoDien** qua những lớp trung gian đó mà vẫn cập nhật được giao diện trong quá trình download. Hoặc giả sử có nhiều giao diện cần được cập nhật trong quá trình download thì với cách 1 sẽ khó khăn hơn nhiều vì cần truyền mọi giao diện đó vào lớp **DownLoad**…

Interface vs Abstract class

	Interface	Abstract class
Multiple inheritance	Một class có thể hiện thực nhiều interface. (tạm coi là thừa kế)	Không hỗ trợ đa thừa kế
Default implementation	Không thể định nghĩa code xử lý, chỉ có thể khai báo.	Có thể định nghĩa thân phương thức, property.
Access Modifiers	Mọi phương thức, property đều mặc định là public.	Có thể xác định modifier.
Adding functionality	Mọi phương thức, property của interface cần được hiện thực trong class.	Không cần thiết.
Fields and Constants	Không	Có

Chuyển đổi kiểu dữ liệu tham chiếu

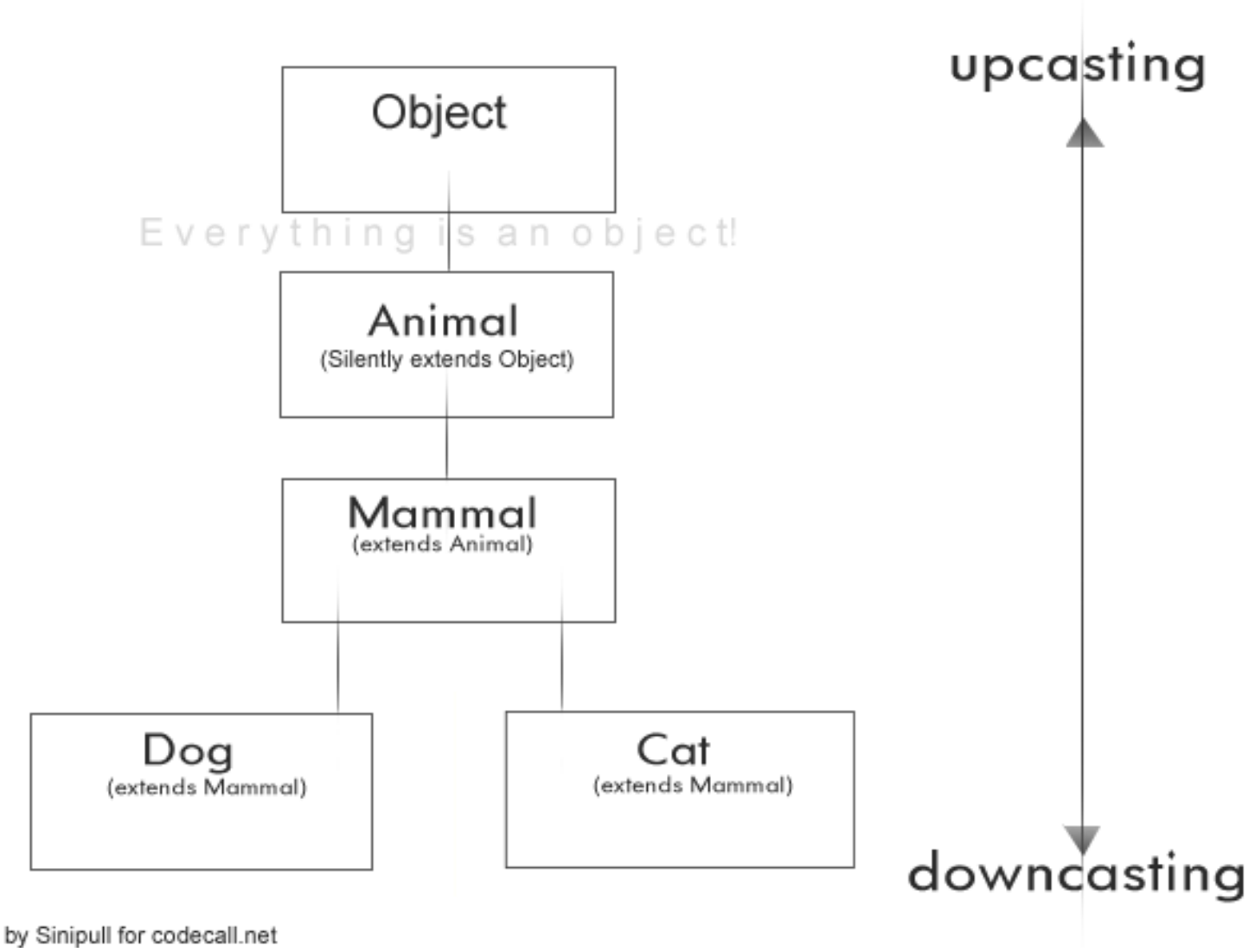
- Kiểu dữ liệu tham chiếu có thể được chuyển đổi kiểu khi
  - Kiểu dữ liệu tham chiếu (lớp) tương thích, nằm trên cùng một cây phân cấp kế thừa
- Hai cách chuyển đổi
  - Up-casting
  - Down-casting
- Up casting
  - Đi lên trên cây phân cấp thừa kế (moving up the inheritance hierarchy)
  - Up casting là khả năng nhìn nhận đối tượng thuộc lớp dẫn xuất như là một đối tượng thuộc lớp cơ sở.
  - Tự động chuyển đổi kiểu



Down-casting

- Down casting: đi xuống cây phân cấp thừa kế (move back down the inheritance hierarchy)
- Down casting là khả năng nhìn nhận một đối tượng thuộc lớp cơ sở như một đối tượng thuộc lớp dẫn xuất.
- Không tự động chuyển đổi kiểu. Phải ép kiểu.

Ví dụ


upcasting
Object
Everything is an object!
Animal
(Silently extends Object)
Mammal
(extends Animal)
Dog
(extends Mammal)
Cat
(extends Mammal)
downcasting
by Sinipull for codecall.net

Code

```
public interface Animal {
    public void eat();
    public void move();
    public void sleep();
}
public class Mammal implements Animal {
    public void eat() {
        System.out.println("Eating...");    }
    public void move() {
        System.out.println("Moving...");    }
    public void sleep() {
        System.out.println("Sleeping...");
    }}
```

```
public class Cat extends Mammal {
    public void meow() {
        System.out.println("Meow Meow!");
    } }
public class Dog extends Mammal {
    public void bark() {
        System.out.println("Gow gow!");
    }
    public void eat() {
        System.out.println("Dog is eating...");
    }}
public class MammalTrainer {
    public void teach(Mammal maml) {
        maml.move();
        maml.eat();
    }}
```

- `Mammal ma = new Dog();`
- `ma.eat();`
- Dòng 1 được gọi là implicit casting up-casting. nó là hợp lệ vì đối tượng của lớp con cũng là 1 thể hiện của lớp cha.
- `Dog dog = new Dog();`
- `Cat cat = new Cat();`
- `MammalTrainer trainer = new MammalTrainer();`
- `trainer.teach(dog);`
- `trainer.teach(cat);`

downcasting

- `Mammal ma1 = new Cat();`
- `Cat cat1 = ma1;`
- `Cat cat1 = (Cat) ma1;`
- Dòng 2 gặp lỗi biên dịch bởi vì kiểu lớp cha Mammal không phải là thể hiện của lớp con Cat => lệnh gán bị lỗi. Mặc dù ta thấy mam1 thực sự là đối tượng của lớp Cat nhưng trình biên dịch không đủ thông minh để nhận ra điều đó. Để khắc phục nó ta dùng explicit casting như dòng 3 để thông báo với trình biên dịch rằng mam1 là đối tượng thuộc lớp Cat. Khi ép kiểu biến thuộc lớp con thành biến lớp cha gọi là upcasting. Ngược lại khi ép kiểu biến lớp cha thành biến lớp con gọi là downcasting.

Toán tử instanceof

- Kiểm tra xem một đối tượng có phải là thể hiện của một lớp nào đó không

```
public class Employee extends Person {}
public class Student extends Person {}
public class Test{
  public doSomething(Person e) {
  if (e instanceof Employee) {...
   } else if (e instanceof Student) {...
    } else {...
    }
}
}
```

- Bởi vì **Mammal** downcast có thể là Cat, Dog… nên ta sử dụng toán tử **instanceof** để kiểm tra một đối tượng có hay không thuộc về một lớp nào đó. Nếu kết quả trả về là true thì downcasting là cho phép, ngược lại kết quả là false thì không cho phép.

```
Mammal mam1 = new Cat();
Cat cat1 = (Cat) mam1;
Mammal mam2 = new Dog();
Dog dog = (Dog) mam2;
if (mam1 instanceof Cat) {
    Cat cat2 = (Cat) mam1;
    cat2.meow();
} else if (mam2 instanceof Dog) {
            Dog dog1 = (Dog) mam2;
            dog1.bark();  }
```