

Object-Oriented Programming

- OOP uses a number of techniques to achieve reusability and adaptability (able to be modified; adjusting quickly) including:

abstraction, encapsulation, inheritance, and polymorphism

Defining Object **Composition**

- Objects can be composed of other objects.
- Objects can be part of other objects.
- This relationship between objects is known as *aggregation*.



A PC may be an object.



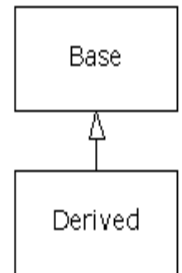
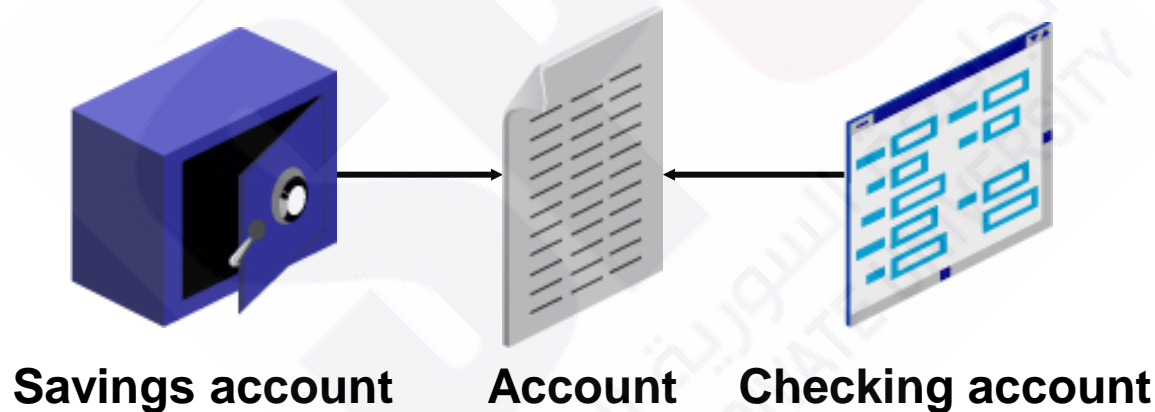
A PC may have a keyboard, mouse, and network card, all of which may be objects.



A PC may have a CD drive, which may be an object.

What Is Inheritance?

- There may be a commonality between different classes.
- Define the common properties in a superclass.



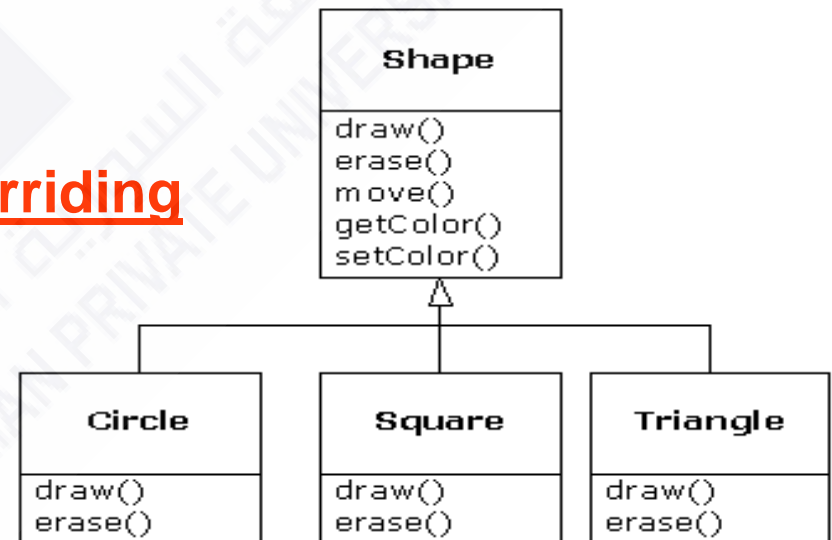
- The subclasses use inheritance to include those properties.

Using the “Is-a-Kind-of” Relationship

You have two ways to differentiate your new derived class from the original base class it inherits from.

The first is: you simply add new functions to the derived class.

The second way is: to change the behavior of an existing base-class function. This is referred to as overriding that function.

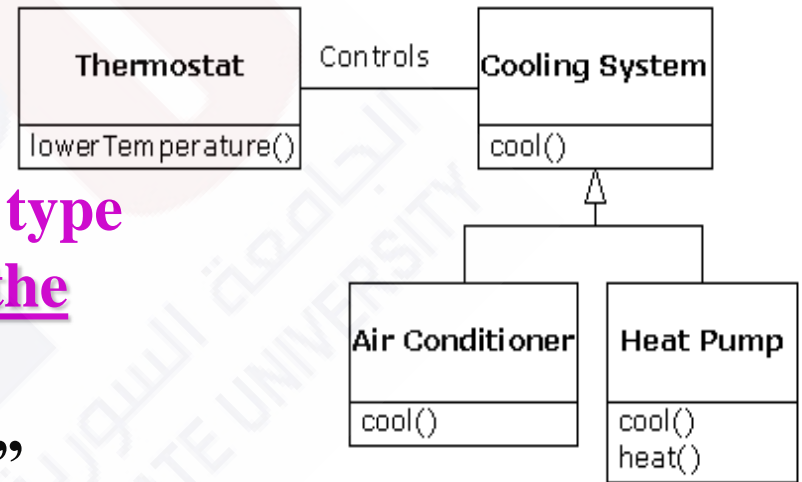


Is-a vs. is-like-a relationships

Is-a

override *only* base-class functions

The derived type is *exactly* the same type as the base class since it has exactly the same interface.



“a circle *is a* shape.”

is-like-a

Add new interface elements to a derived type, thus extending the interface and creating a new type.

heat pump *is-like-an* air conditioner

What Is Polymorphism?

Polymorphism refers to:

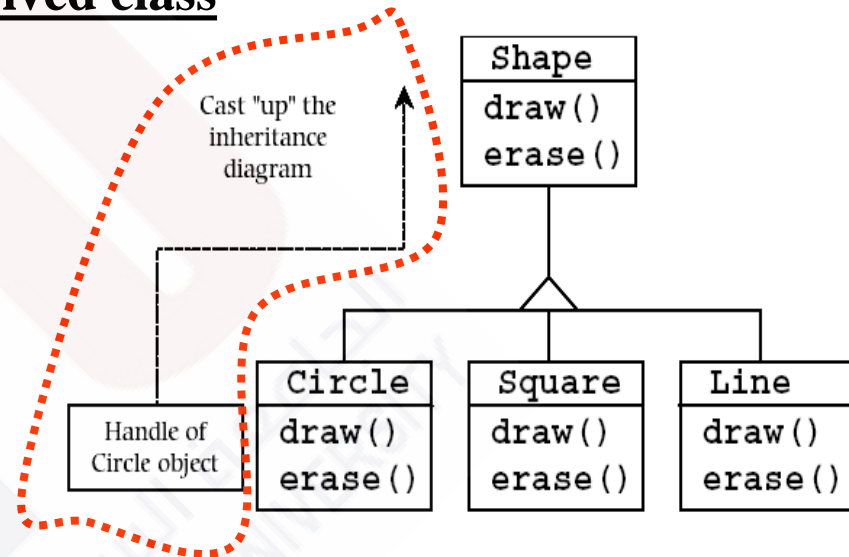
- **Many forms of the same operation**
- **The ability to request an operation with the same meaning to different objects. However, each object implements the operation in a unique way.**
- **The principles of inheritance and object substitution.**



Load passengers

One of the most important things you do with such a family of classes is to treat an object of a derived class as an object of the base class.

This is important because it means you can write a single piece of code that ignores the specific details of type and talks just to the base class.



```
void doStuff(Shape s)
{
    s.erase();
    // ...
    s.draw();
}
```

```
Circle c = new Circle();
Triangle t = new Triangle();
Line l = new Line();

doStuff(c);
doStuff(t);
doStuff(l);
```

We call this process of treating a derived type as though it were its base type *upcasting*

Specifying Inheritance in Java

- Inheritance is achieved by specifying which superclass the subclass extends.

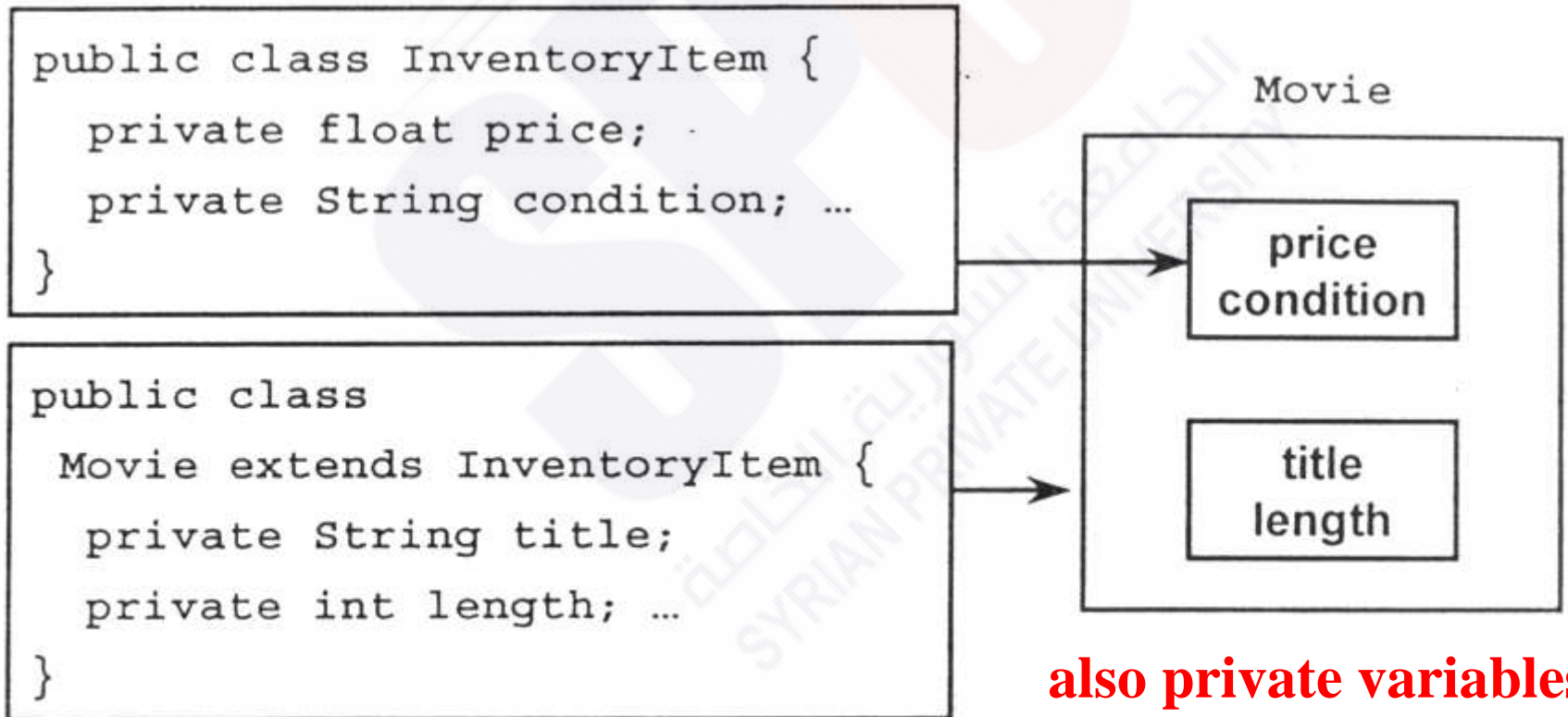
```
public class InventoryItem {  
    ...  
}
```

```
public class Movie extends InventoryItem {  
    ...  
}
```

- `Movie` inherits all the variables and methods of `InventoryItem`.

What Does a Subclass Object Look Like?

A subclass inherits all the instance variables of its superclass.

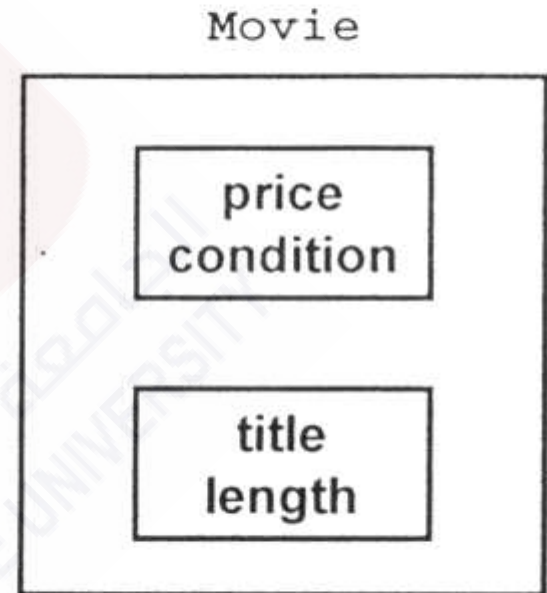


**also private variables
was inherited.**

Default Initialization

- What happens when a subclass object is created?

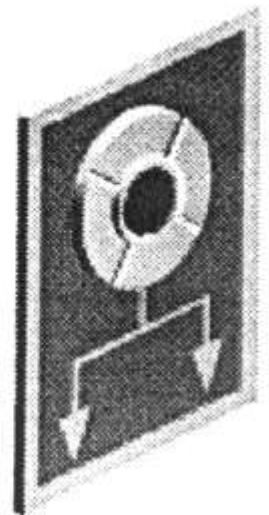
```
Movie movie1 = new Movie();
```



- If no constructors are defined:
 - First, the default no-arg constructor is called in the superclass.
 - Then, the default no-arg constructor is called in the subclass.

The super Reference

- Refers to the base, top-level class
- Is useful for calling base class constructors
- Must be the first line in the derived class constructor
- Can be used to call any base class methods



The super Reference Example

```
public class InventoryItem {  
    InventoryItem(String cond) {  
        System.out.println("InventoryItem");  
        ...  
    }  
}  
  
class Movie extends InventoryItem {  
    Movie(String title) {  
        super(title);  
        ...  
        System.out.println("Movie");  
    }  
}
```

Base class
constructor

Calls base
class
constructor

Using Superclass Constructors

Use `super()` to call a superclass constructor:

```
public class InventoryItem {
```

```
→ InventoryItem(float p, String cond) {
```

```
    price = p;
```

```
    condition = cond;
```

```
    }
```

```
    ... public class Movie extends InventoryItem {
```

```
        Movie(String t, float p, String cond) {
```

```
            super(p, cond);
```

```
            title = t;
```

```
        } ...
```


Overriding Superclass Methods

- A subclass inherits all the methods of its superclass.
- The subclass can override a method with its own specialized version.
 - The subclass method must have the same signature and semantics as the superclass method.

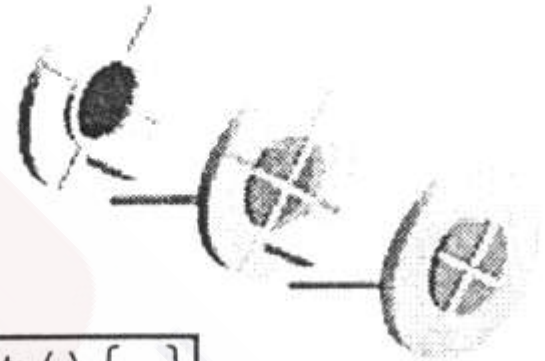
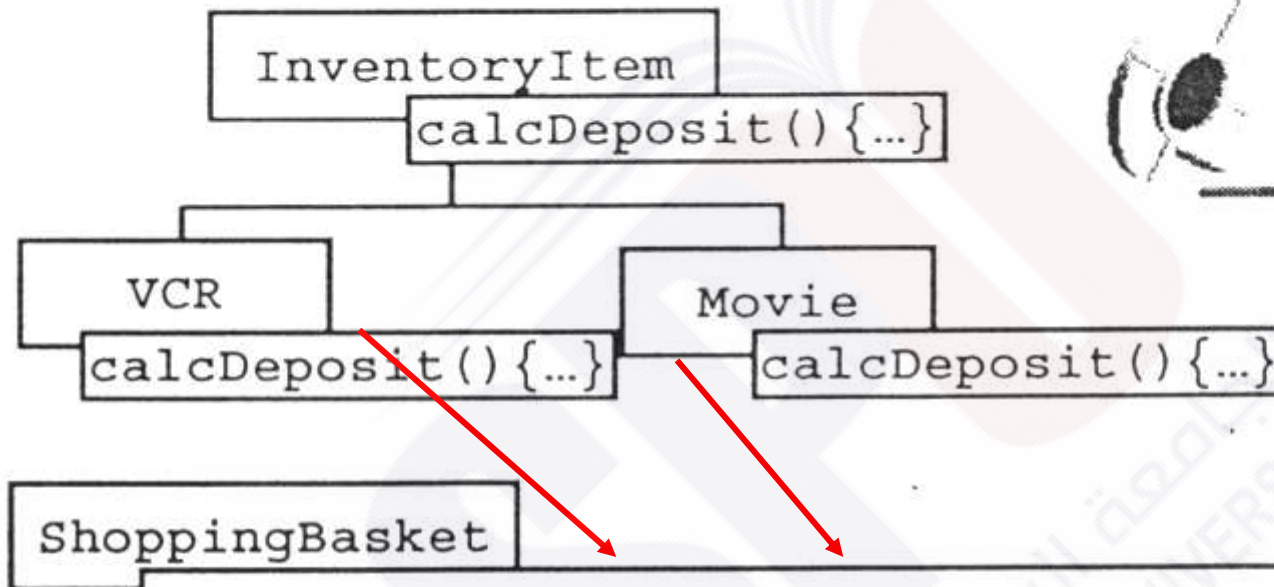
```
public class InventoryItem {  
    public float calcDeposit(int custId) {  
        if ...  
        return ...  
    }  
}  
  
public class Vcr extends InventoryItem {  
    public float calcDeposit(int custId) {  
        if ...  
        return itemDeposit;  
    }  
}
```

Invoking Superclass Methods

- If a subclass overrides a method, then it can still call the original superclass method.
- Use `super.method()` to call a superclass method from the subclass.

```
public class InventoryItem {  
    public float calcDeposit(int custId) {  
        if  
        re  
    }  
}  
  
public class Vcr extends InventoryItem {  
    public float calcDeposit(int custId) {  
        itemDeposit = super.calcDeposit(custId);  
        return (itemDeposit + vcrDeposit);  
    }  
}
```

Using Polymorphism for Acme Video



```
void addItem(InventoryItem item) {
    // this method is called each time
    // the clerk scans in a new item
    float deposit = item.calcDeposit();
    .....
    ...
}
```


Using the instanceof Operator

- You can determine the true type of an object by using an instanceof operator. (**At run time**)
- An object reference can be downcast to the correct type, if necessary.

```
public void aMethod(InventoryItem i) {  
    ...  
    if (i instanceof Vcr)  
        ((Vcr)i).playTestTape();  
}
```

```

class Cleanser {
    private String s = new String("Cleanser");
    public void append (String a) { s += a; }
    public void dilute()      { append(" dilute(111)"); }
    public void apply ()      { append(" apply(2222)"); }
    public void scrub ()      { append(" scrub(333)"); }
    public String toString ()  { return s; }
    public static void main (String[] args) {
        Cleanser x = new Cleanser();      x.dilute();  x.apply();  x.scrub();  System.out.println(x);
    } }

public class Detergent extends Cleanser {
    public void scrub() {
        append (" BM.scrub()");
        super.scrub();
    }

    public void foam() { append(" foam()"); }
    public static void main (String[] args) {      System.out.println("-----");
        Detergent x = new Detergent();
        x.dilute();      x.apply();      x.scrub();  x.foam();
        System.out.println(x);
        System.out.println("Testing base class:");
        Cleanser.main(args);
    } }

```

```
class Art {  
    Art() {  
        System.out.println("Art constructor");  
    }  
}  
  
class Drawing extends Art {  
    Drawing() {  
        System.out.println("Drawing constructor");  
    }  
}  
  
public class Cartoon extends Drawing {  
  
    public Cartoon() {  
        System.out.println("Cartoon constructor");  
    }  
  
    public static void main(String[] args) {  
        Cartoon x = new Cartoon();  
    }  
}
```

```
class Game {  
    Game (int i){  
        System.out.println("Game constructor");  
    }  
}  
  
class BoardGame extends Game {  
    BoardGame(int i) {  
        super(i);  
        System.out.println("BoardGame constructor");  
    }  
}  
  
public class Chess extends BoardGame {  
    Chess() {  
        super(11);  
        System.out.println("Chess constructor");  
    }  
    public static void main(String[] args) {  
        Chess x = new Chess();  
    }  
}
```

```
class Homer {  
    char doh(char c) {  
        System.out.println("doh(char)");  
        return 'd';  
    }  
    float doh(float f) {  
        System.out.println("doh(float)");  
        return 1.0f;  
    } }  
class Milhouse {}  
class Bart extends Homer {  
    void doh(Milhouse m) {  
        System.out.println("doh(Milhouse)");  
    } }  
public class Hide {  
    public static void main(String[] args) {  
        Bart b = new Bart();  
        b.doh(1);    b.doh('x');    b.doh(1.0f);  
        b.doh(new Milhouse());  
    } }
```

```

class Engine {    /// Composition
    public void start() {}    public void rev() {}    public void stop() {}
}
class Wheel {
    public void inflate(int psi) {}
}
class Window {
    public void rollup() {}    public void rolldown() {}
}
class Door {
    public Window window = new Window();
    public void open() {}    public void close() {}
}
public class Car {
    public Engine engine = new Engine();    public Wheel[] wheel = new Wheel[4];
    public Door    left = new Door(), right = new Door(); // 2-door
    public Car() {
        for(int i = 0; i < 4; i++)    wheel[i] = new Wheel();    }
    public static void main(String[] args) {
        Car car = new Car();
        car.left.window.rollup();    car.wheel[0].inflate(72);
    } }

```

```

import java.util.*;

class Shape { void draw() {} void erase() {} }

class Circle extends Shape {
    void draw() { System.out.println("Circle.draw()"); } void erase() { System.out.println("Circle.erase()"); }
}

class Square extends Shape {
    void draw() { System.out.println("Square.draw()"); } void erase() { System.out.println("Square.erase()"); }
}

class Triangle extends Shape {
    void draw() { System.out.println("Triangle.draw()"); } void erase() { System.out.println("Triangle.erase()"); }
}

class RandomShapeGenerator {
    private Random rand = new Random();

    public Shape next() {
        switch (rand.nextInt(3)) {
            default:
                case 0: return new Circle(); case 1: return new Square(); case 2: return new Triangle(); }}}

public class Shapes {
    private static RandomShapeGenerator gen = new RandomShapeGenerator();

    public static void main(String[] args) {
        Shape[] s = new Shape[9];
        for(int i = 0; i < s.length; i++) s[i] = gen.next();
        for(int i = 0; i < s.length; i++) s[i].draw(); } }

```

```

import java.util.*;

class Shape {
    void draw() {}    void erase() {}    }

class Circle extends Shape {
    void draw() {    System.out.println("Circle.draw()"); }    void erase() {    System.out.println("Circle.erase()"); } }

class Square extends Shape {
    void draw() {    System.out.println("Square.draw()"); }    void erase() {    System.out.println("Square.erase()"); } }

class Triangle extends Shape {
    void draw() {    System.out.println("Triangle.draw()"); }    void erase() {    System.out.println("Triangle.erase()"); } }

class RandomShapeGenerator {
    private Random rand = new Random();

    public Shape next() {
        switch ( rand.nextInt(3) ) {
            default:
                case 0: return new Circle();    case 1: return new Square();    case 2: return new Triangle();
        } } }

public class Shapes {
    private static RandomShapeGenerator    gen    =    new RandomShapeGenerator();

    public static void main(String[] args) {
        Object[] s = new Shape[9];

        for(int i = 0; i < s.length; i++)
            s[i] = gen.next();
        for(int i = 0; i < s.length; i++)
            s[i].draw();
    } }

```



```

import java.util.*;

class Shape {
    void draw() {}    void erase() {}    }

class Circle extends Shape {
    void draw() {    System.out.println("Circle.draw()"); }    void erase() {    System.out.println("Circle.erase()"); }    }

class Square extends Shape {
    void draw() {    System.out.println("Square.draw()"); }    void erase() {    System.out.println("Square.erase()"); }    }

class Triangle extends Shape {
    void draw() {    System.out.println("Triangle.draw()"); }    void erase() {    System.out.println("Triangle.erase()"); }    }

class RandomShapeGenerator {
    private Random rand = new Random();
    public Shape next() {
        switch(rand.nextInt(3)) {
            default:
                case 0: return new Circle(); case 1: return new Square(); case 2: return new Triangle();        }    }
}

public class Shapes {
    private static RandomShapeGenerator gen = new RandomShapeGenerator();
    public static void main(String[] args) {
        Object[] s = new Shape[9];
        for(int i = 0; i < s.length; i++)
            s[i] = gen.next();
        for(int i = 0; i < s.length; i++) {
            if ( s[i] instanceof Circle )    ((Circle)    s[i]).draw();
            if ( s[i] instanceof Square )    ((Square)    s[i]).draw();
            if ( s[i] instanceof Triangle ) ((Triangle) s[i]).draw();    }    }
}

```

```

import java.util.*;
class Shape {
    void draw() {}    void erase() {}    }
class Circle extends Shape {
    void draw() { System.out.println("Circle.draw()"); }    void erase() { System.out.println("Circle.erase()"); } }
class Square extends Shape {
    void draw() { System.out.println("Square.draw()"); }    void erase() { System.out.println("Square.erase()"); } }
class Triangle extends Shape {
    void draw() { System.out.println("Triangle.draw()"); }    void erase() { System.out.println("Triangle.erase()"); } }

```

```

class RandomShapeGenerator {

```

```

    private Random rand = new Random();

```

```

    public Shape next() {

```

```

        switch(rand.nextInt(3)) {

```

```

            default:

```

```

                case 0: return new Circle();

```

```

                case 1: return new Square();

```

```

                case 2: return

```

```

new Triangle();    }    }    }

```

```

public class Shapes {

```

```

    private static RandomShapeGenerator gen = new RandomShapeGenerator();

```

```

    public static void main(String[] args) {

```

```

        Object[] s = new Shape[9];

```

```

        for(int i = 0; i < s.length; i++)

```

```

            s[i] = gen.next();

```

```

        for(int i = 0; i < s.length; i++) {

```

```

            if ( s[i] instanceof Shape )    ((Shape )    s[i]).draw(); }

```

```

        } }

```