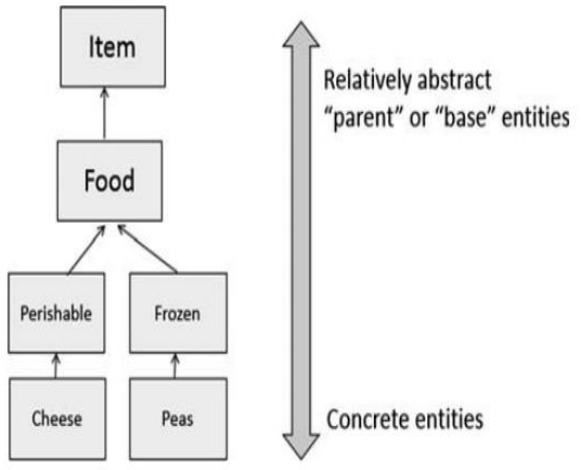
Inheritance

Objectives

- Study concepts: superclass, subclass
- Understand common relationships
- Functions in inheritance
- Using an “instanceof” operator
- Casting

Derived and Super Classes



- Object-oriented languages implement reusability of coding structure through inheritance
- It refers to the relationship between classes where one class inherits the entire structure of another class
- The root of our design is a relatively abstract entity, and we build upon that entity to produce progressively more concrete entities
- the higher-level entities are **“parent”, “base” or “super”** classes
- the lower-level ones built from them are **“child”, "derived" or “sub”** classes.

Object-Oriented Relationships

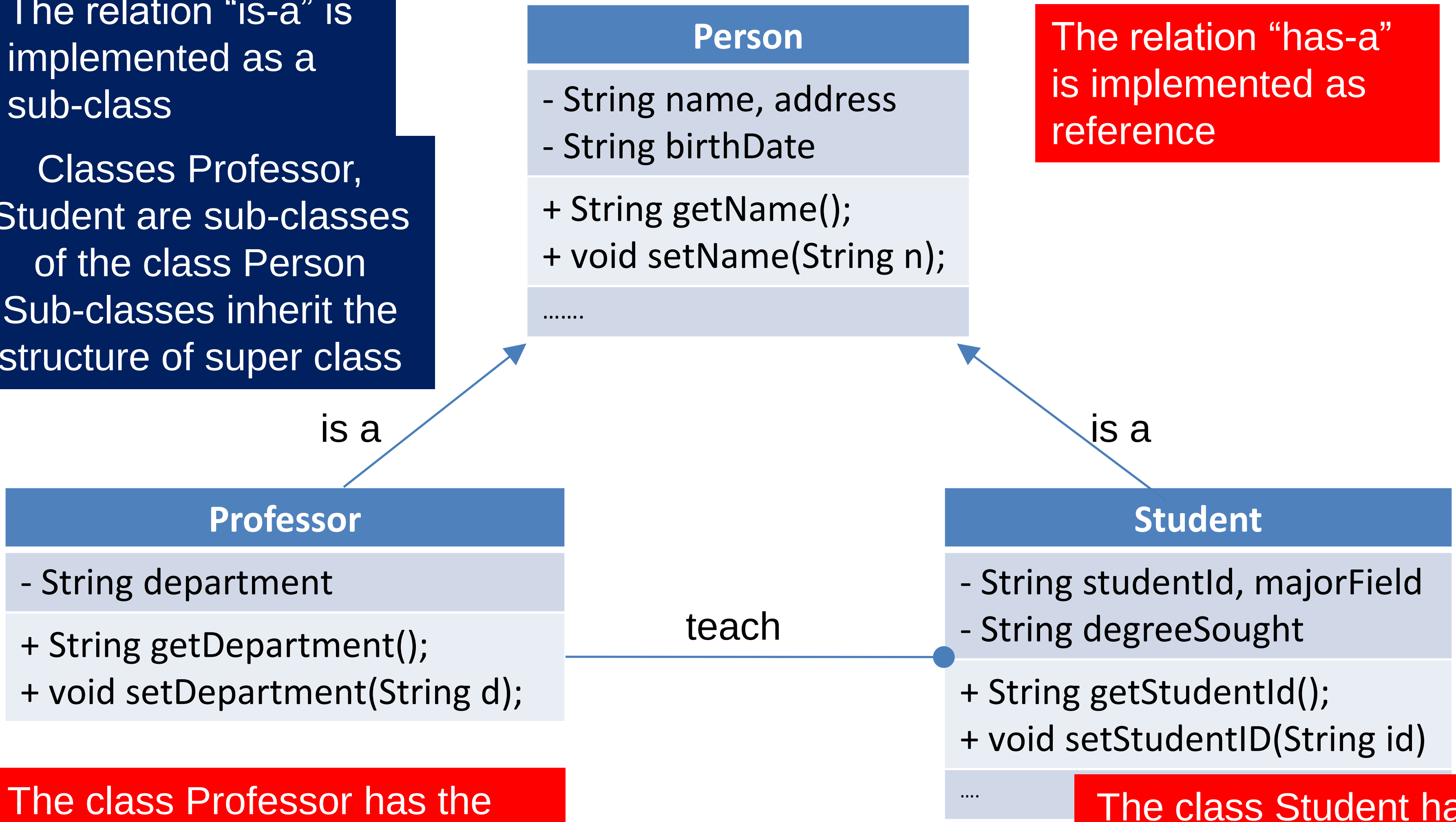
- common relationships in classes:
  - “is-a/ a kind of”
  - “has-a”
- Examples:
  - Student is a person
  - “A home is a house that has a family and a pet.”
  - An invoice contains some products and a product can be contained in some invoices

Object-Oriented Relationships…

The relation “is-a” is implemented as a sub-class

Classes Professor, Student are sub-classes of the class Person
Sub-classes inherit the structure of super class

The relation “has-a” is implemented as reference

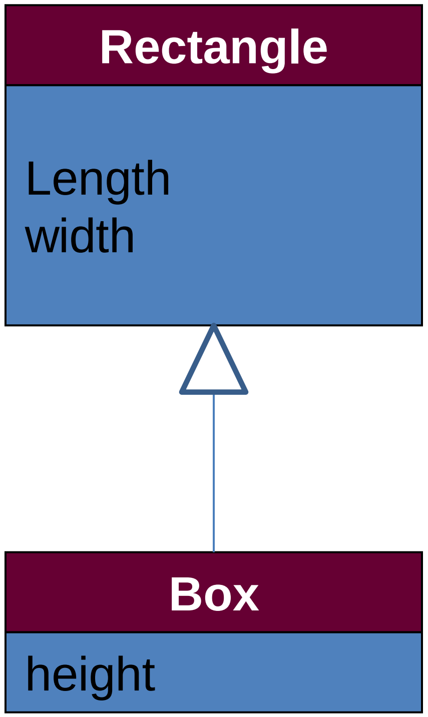


The class Professor has the field Student[] students

The class Student has the field Professor pr

Inheritance…

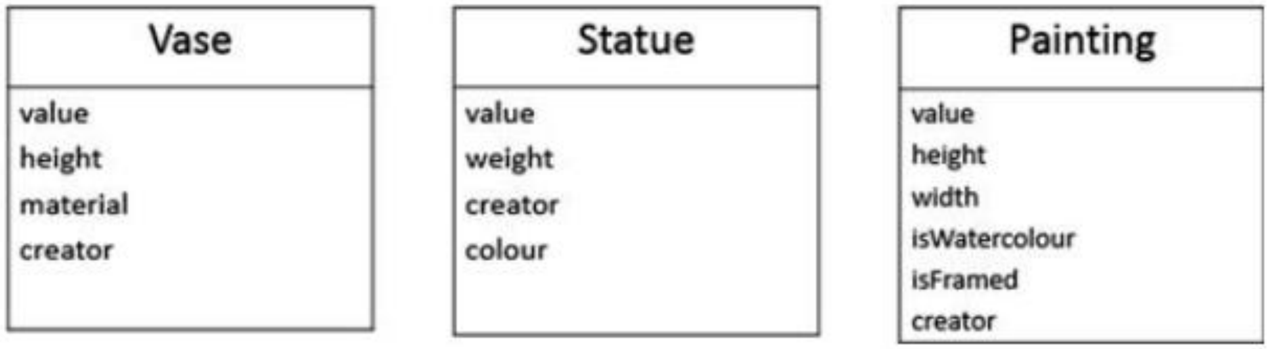
- **How to construct a class hierarchy? → Intersection**
  - Rectangle< length, width>
  - Box < length, width, height>

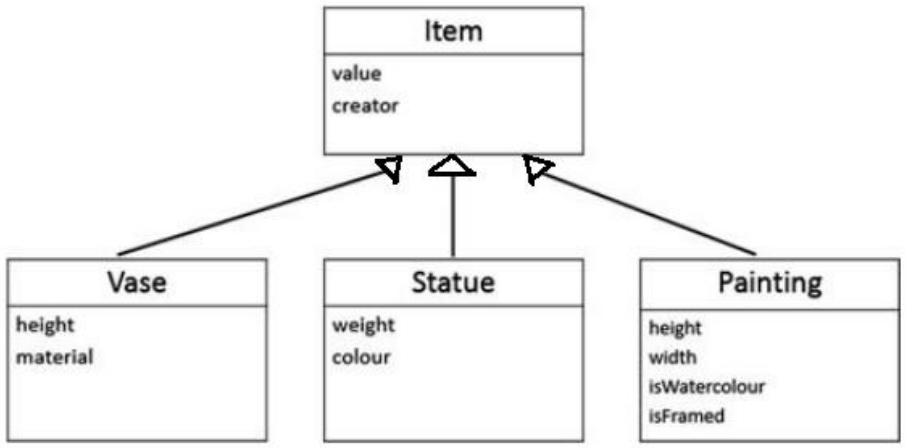


Inheritance…

- **How to construct a class hierarchy? → Intersection**

Consider a shop that sells antiques items, namely **vases**, **statues and paintings**





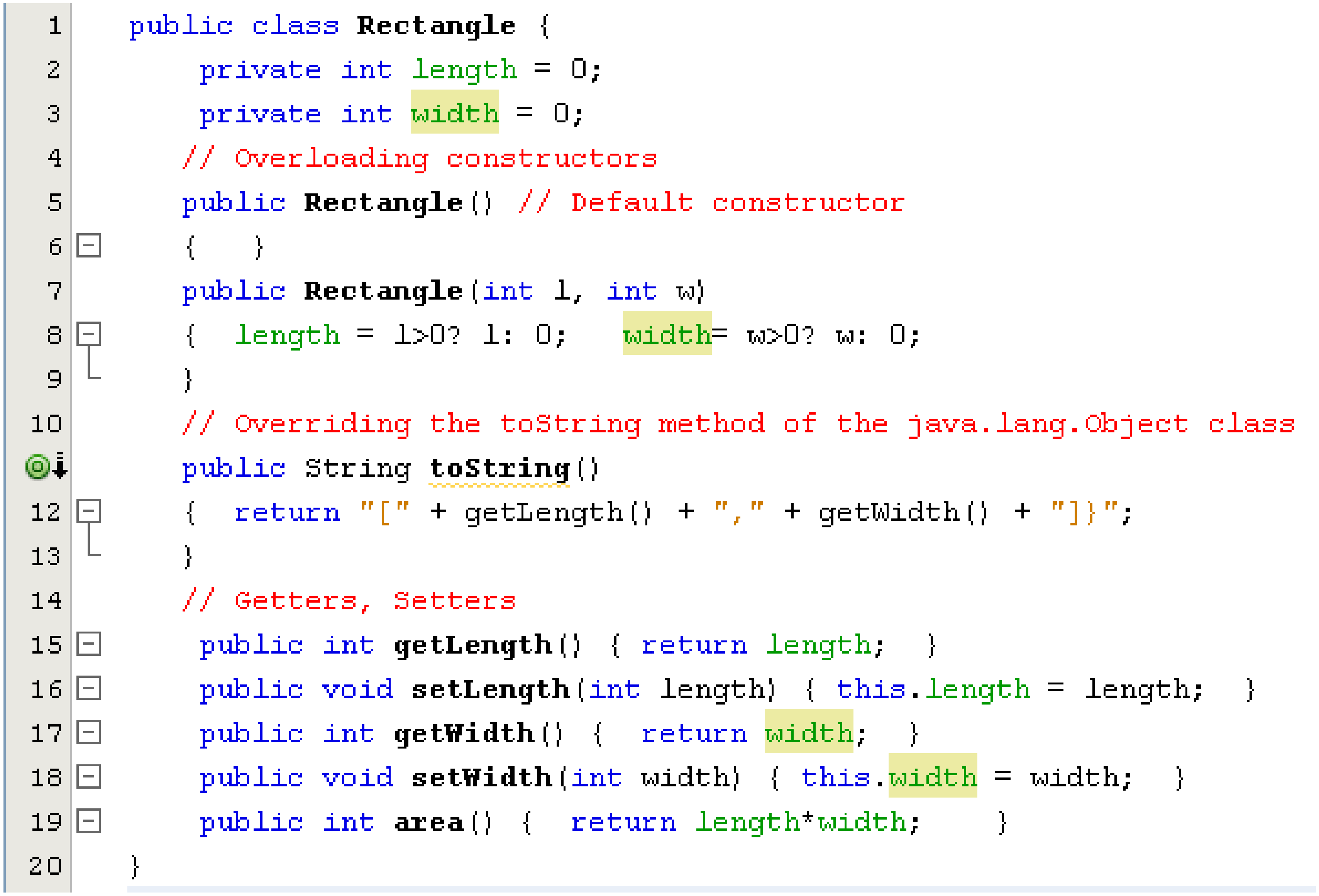
Inheritance

- There are some sub-classes from one super class ➔ An inheritance is a relationship where objects **share a common structure**: the structure of one object is a sub-structure of another object.
- The <u>**extends**</u> keyword is used to create sub-class.
- A class can be directly derived from only one class *(Java is a single-inherited OOP language)*.
- If a class does not have any superclass, then it is implicitly derived from Object class.
- Unlike other members, constructor **cannot be inherited** *(constructor of super class can not initialize sub-class objects)*

“super” Keyword

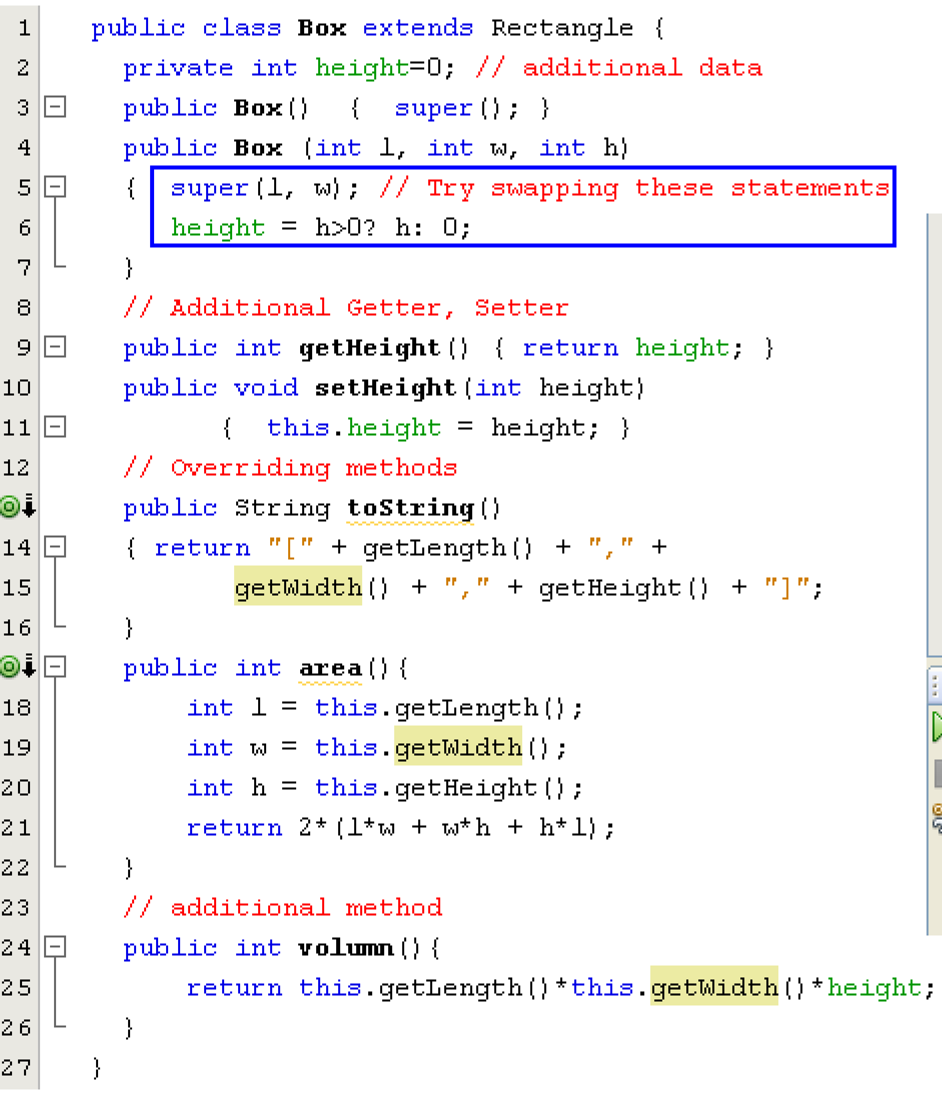
- Constructors Are **Not** Inherited
- super(...) for Constructor Reuse
  - super(arguments); *//invoke a superclass constructor*
  - Subclass constructor **must invoke super class constructor**
  - The call ***must* be the *first* statement in the subclass constructor**
- **Note**: If a constructor *does not explicitly invoke a superclass constructor*, the Java compiler *automatically inserts a call to the no-argument constructor of the superclass*. If the super class does not have a no-argument constructor, you will get a compile-time error.

Inheritance...

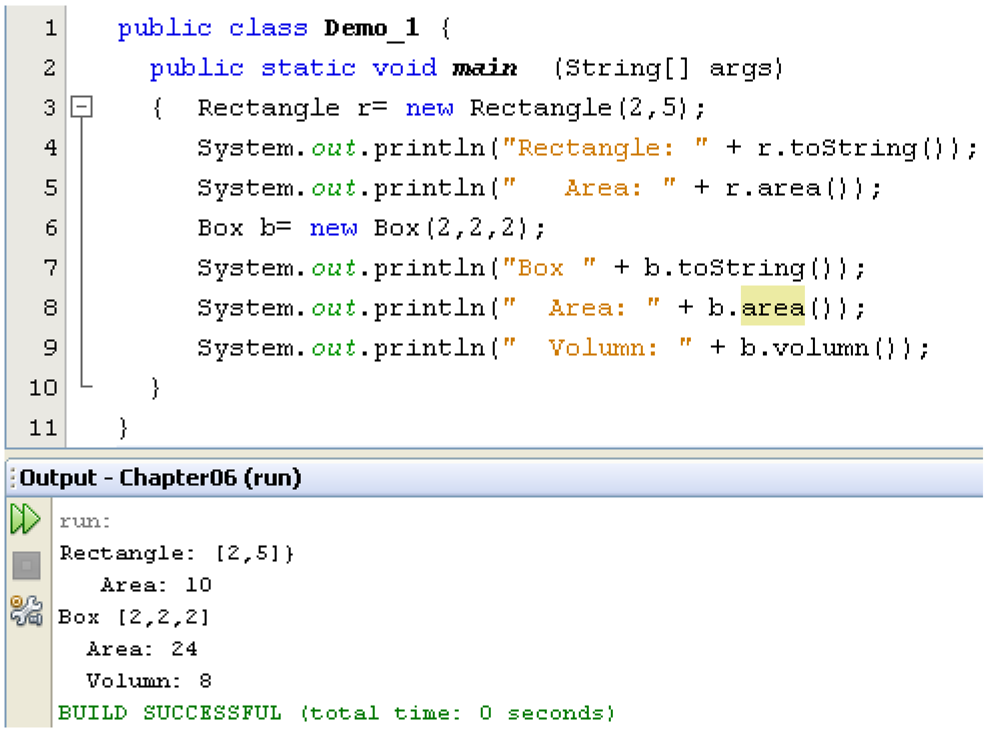


```
public class Rectangle {
    private int length = 0;
    private int width = 0;
   // Overloading constructors
   public Rectangle() // Default constructor
   {    }
   public Rectangle(int l, int w)
   {  length = l>0? l: 0;    width= w>0? w: 0;
   }
   // Overriding the toString method of the java.lang.Object class
   public String toString()
   {  return "[" + getLength() + "," + getWidth() + "]}";
   }
   // Getters, Setters
    public int getLength() { return length;   }
    public void setLength(int length) { this.length = length;   }
    public int getWidth() {  return width;   }
    public void setWidth(int width) { this.width = width;   }
    public int area() {  return length*width;     }
}
```

Inheritance…



```java
public class Box extends Rectangle {
  private int height=0; // additional data
  public Box()  {  super(); }
  public Box (int l, int w, int h)
  { super(l, w); // Try swapping these statements
     height = h>0? h: 0;
  }
  // Additional Getter, Setter
  public int getHeight() { return height; }
  public void setHeight(int height)
        {  this.height = height; }
  // Overriding methods
  public String toString()
  { return "[" + getLength() + "," +
          getWidth() + "," + getHeight() + "]";
  }
  public int area(){
      int l = this.getLength();
      int w = this.getWidth();
      int h = this.getHeight();
      return 2*(l*w + w*h + h*l);
  }
  // additional method
  public int volumn(){
      return this.getLength()*this.getWidth()*height;
  }
}
```



```java
public class Demo_1 {
  public static void main  (String[] args)
  {  Rectangle r= new Rectangle(2,5);
     System.out.println("Rectangle: " + r.toString());
     System.out.println("   Area: " + r.area());
     Box b= new Box(2,2,2);
     System.out.println("Box " + b.toString());
     System.out.println("  Area: " + b.area());
     System.out.println("  Volumn: " + b.volumn());
  }
}
```

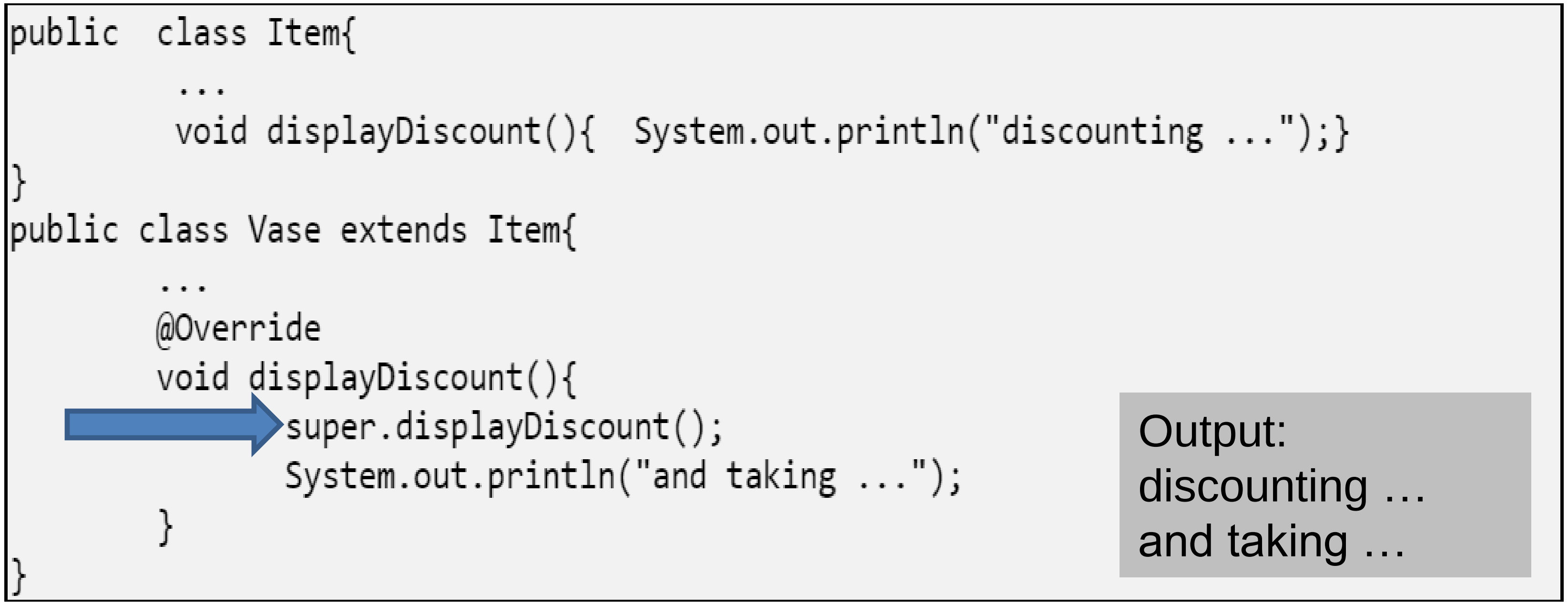
Output - Chapter06 (run)

```
run:
Rectangle: [2,5])
   Area: 10
Box [2,2,2]
  Area: 24
  Volumn: 8
BUILD SUCCESSFUL (total time: 0 seconds)
```

Functions in inheritance

- A derived class inherits from superclass is limited to the normal member functions of the superclass.
- We use the Java keyword **super** as the qualifier for calling a superclass 's method:
  - ***super.methodName(arguments);***
  - To invoke the version of method methodName that was defined by our superclass.
- **Hiding a method**: Re-implementing a static method implemented in super class

Functions in inheritance



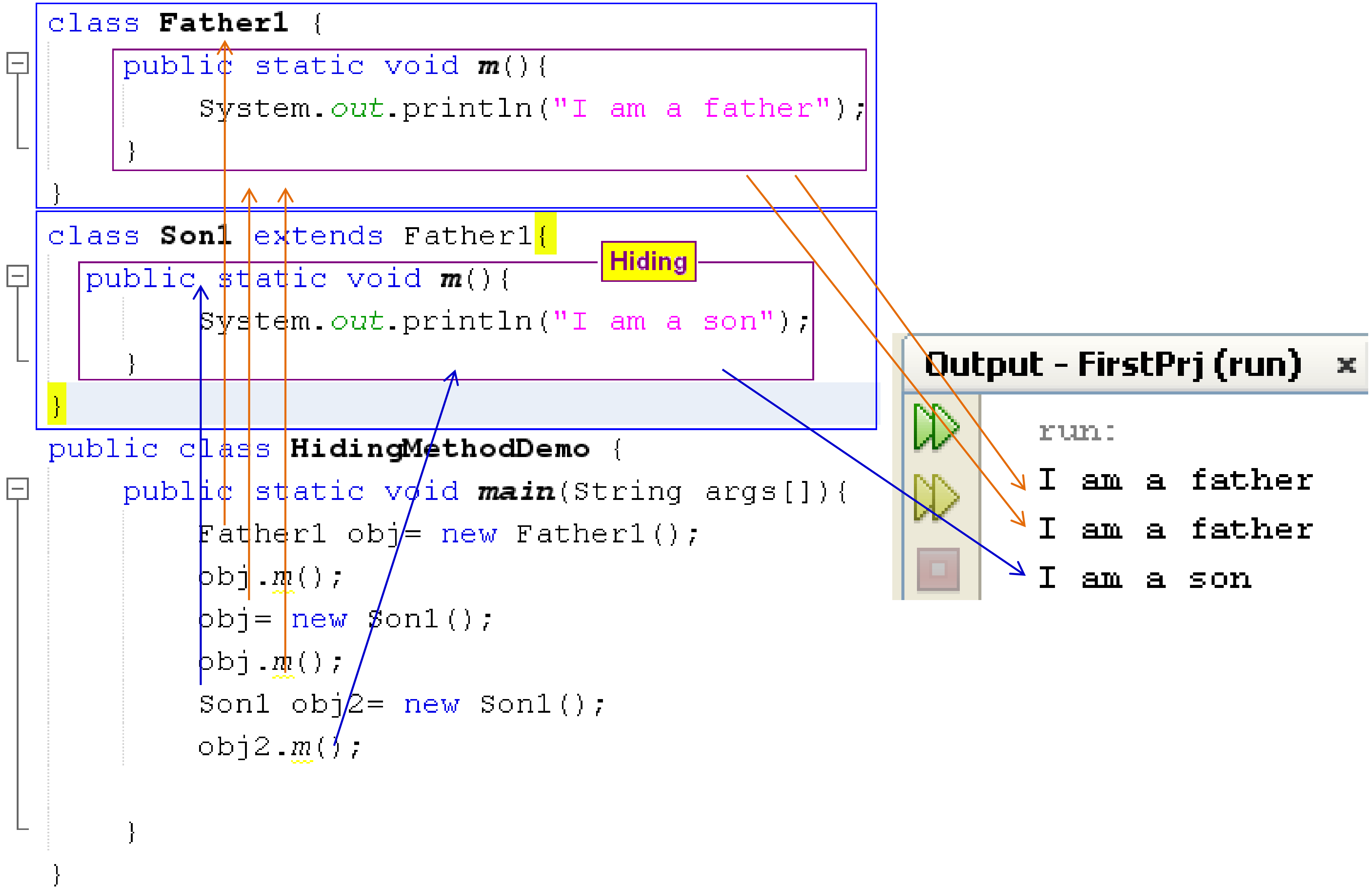
```
public  class Item{
         ...
         void displayDiscount(){  System.out.println("discounting ...");}
}
public class Vase extends Item{
        ...
        @Override
        void displayDiscount(){
              super.displayDiscount();
              System.out.println("and taking ...");
        }
}
```

Output:
discounting …
and taking …

- The "displayDiscount" method has the same signature (name, plus the number and the type of its parameters) and return type as in the superclass. It is called overriding the superclass's method=> We will learn override method in the next topic

- The *"displayDiscount"*) that was defined by our superclass. We use the **"super"** keyword

Functions in inheritance: Hiding Method



```
class Father1 {
    public static void m(){
        System.out.println("I am a father");
    }
}
class Son1 extends Father1{
    public static void m(){
        System.out.println("I am a son");
    }
}
public class HidingMethodDemo {
    public static void main(String args[]){
        Father1 obj= new Father1();
        obj.m();
        obj= new Son1();
        obj.m();
        Son1 obj2= new Son1();
        obj2.m();

    }
}
```

```
run:
I am a father
I am a father
I am a son
```

Using an "instanceof" operator

- Dynamic and Static type
  - dynamic type: A reference variable that has the type of the superclass can store the address of the object of sub class. It is called to be *dynamic type*, the type that is has at runtime.

    Rectangle obj1 = new Box();
  - Static type: The type that it has when first declared. Static type checking is enforced by the compiler.

    Box obj2 = new Box();
- *"Instanceof" operator:* It checks whether the reference of an object belongs to the provided type or not, the instanceof operator will return true or false.

```
If ( obj1  instanceof  Box)
      System.out.println(" obj1 is pointing to the Box object");
```

Casting

- A variable that has the type of the superclass only calls methods of the superclass. To call methods of the subclass we must *cast explicitly*
- *for example,*

```
Rectangle obj = new Box();
((Box)obj).setHeight(300);
```

Summary

- Object-oriented languages implement reusability of coding structure through inheritance
- A derived class does not by default inherit the constructor of a super class
- Constructors in an inheritance hierarchy execute in order from the super class to the derived class
- Using the instanceof keyword if we need to check the type of the reference variable.
- Check the type of the reference variable before casting it explicitly.