Inheritance

Objectives

- Study concepts: superclass, subclass
- Understand the “is-a” relationship
- Functions in inheritance
- Using an “instanceof” operator

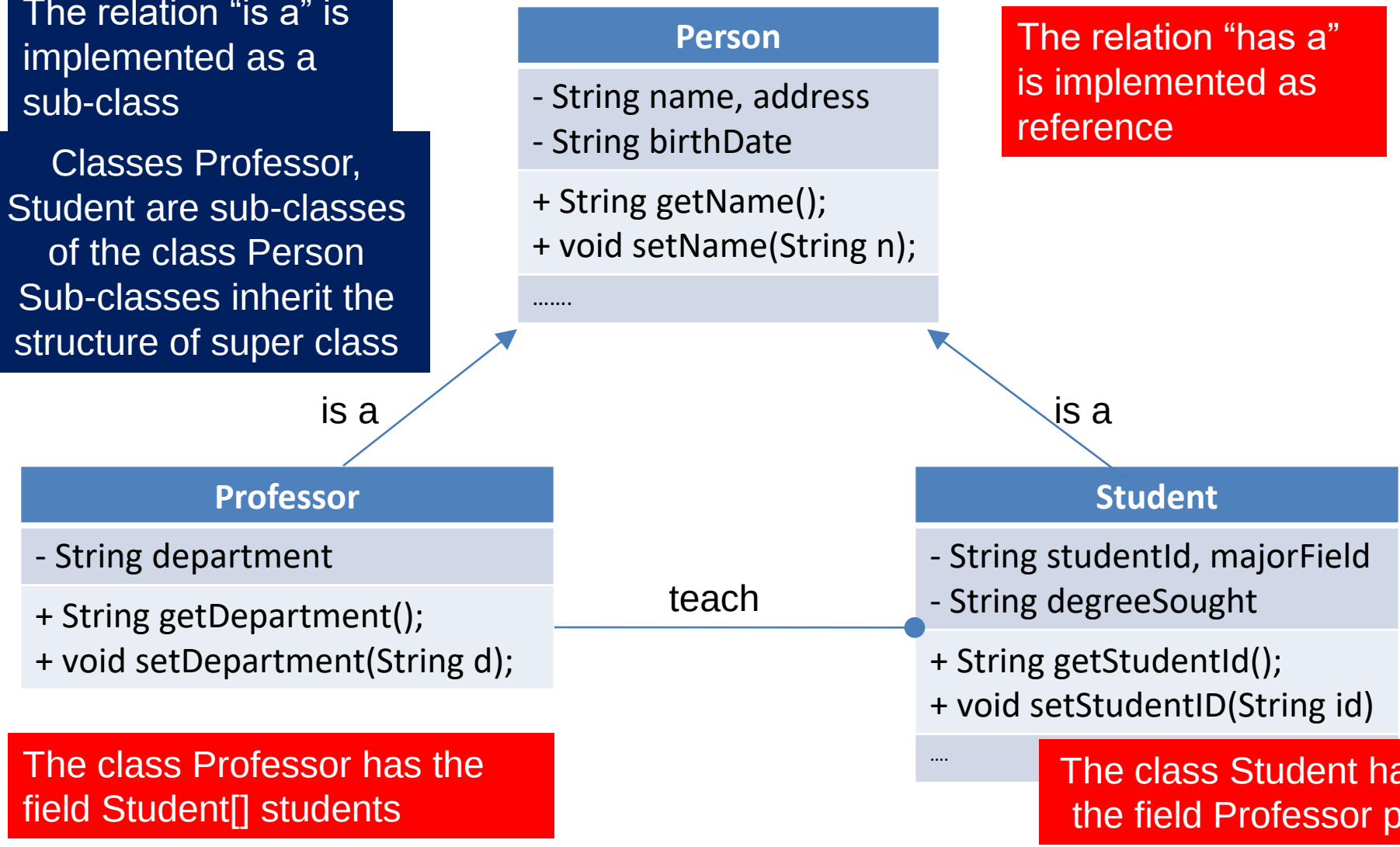
Implementing Object-Oriented Relationships

- 3 common relations in classes:
  - “is a/ a kind of”
  - “has a”
  - association
- 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

Implementing 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

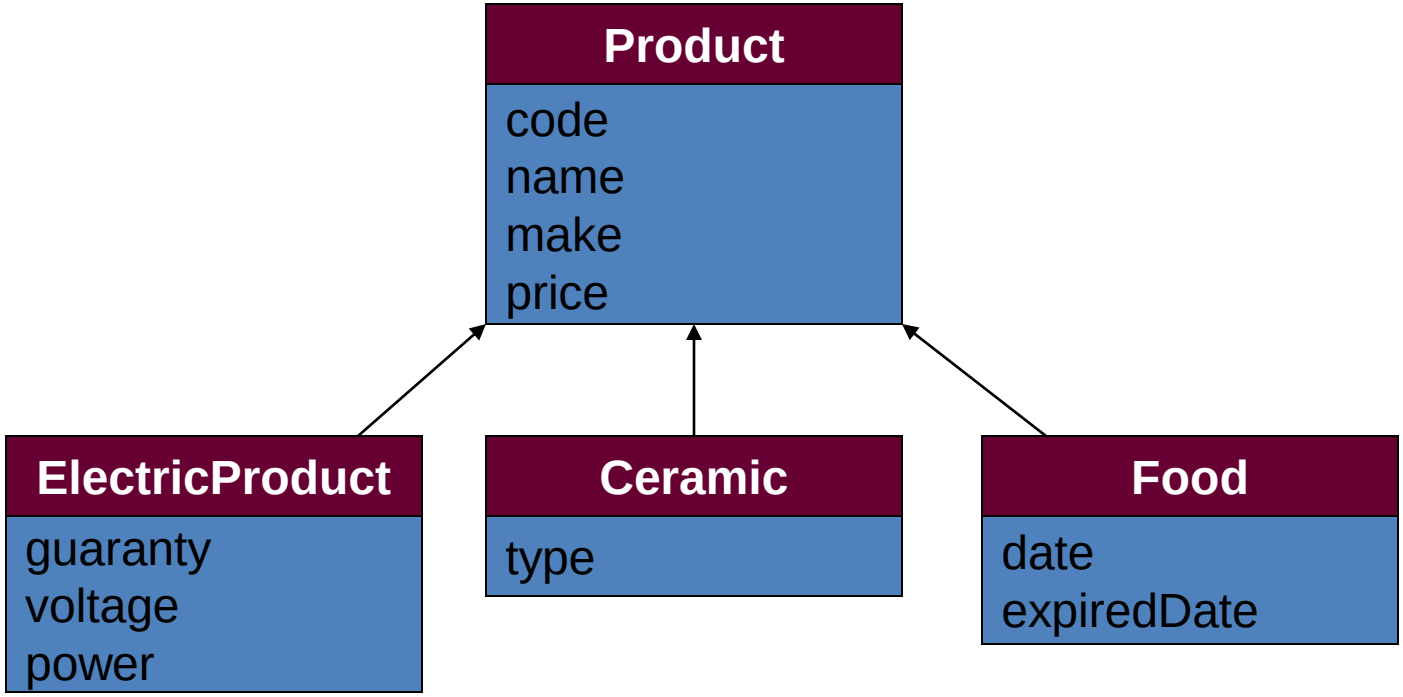
Inheritance

- 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 extends 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)

Inheritance…

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

- Electric Products< code, name, make, price, guaranty, voltage, power>
- Ceramic Products < code, name, make, price, type >
- Food Products < code, name, make, price , date, expiredDate >



Inheritance…: "super" Keyword

- Constructors Are **Not** Inherited
- super(...) for Constructor Reuse
  - super(arguments); *//invoke a superclass constructor*
  - The call ***must*** **be the *first* statement in the subclass constructor**
- Replacing the Default Parameterless Constructor

Inheritance…: "super" Keyword

- We use the Java keyword super as the qualifier for a method call:
  super. methodName(arguments);
- Whenever we wish to invoke the version of method methodName that was defined by our superclass.
- **super()** is used to access the superclass's constructor. And It must be the first statement in the constructor of the subclass.

Inheritance…

```
1   public class Rectangle {
2        private int length = 0;
3        private int width = 0;
4       // Overloading constructors
5       public Rectangle() // Default constructor
6       {    }
7       public Rectangle(int l, int w)
8       {  length = l>0? l: 0;    width= w>0? w: 0;
9       }
10      // Overriding the toString method of the java.lang.Object class
        public String toString()
12      {  return "[" + getLength() + "," + getWidth() + "]}";
13      }
14      // Getters, Setters
15       public int getLength() { return length;  }
16       public void setLength(int length) { this.length = length;  }
17       public int getWidth() {  return width;  }
18       public void setWidth(int width) { this.width = width;  }
19       public int area() {  return length*width;    }
20  }
```

Inheritance…

```
1  public class Box extends Rectangle {
2    private int height=0; // additional data
3    public Box()   {  super(); }
4    public Box (int l, int w, int h)
5    {  super(l, w); // Try swapping these statements
6       height = h>0? h: 0;
7    }
8    // Additional Getter, Setter
9    public int getHeight() { return height; }
10   public void setHeight(int height)
11         {  this.height = height; }
12   // Overriding methods
     public String toString()
14   { return "[" + getLength() + "," +
15         getWidth() + "," + getHeight() + "]";
16   }
     public int area(){
18       int l = this.getLength();
19       int w = this.getWidth();
20       int h = this.getHeight();
21       return 2*(l*w + w*h + h*l);
22   }
23   // additional method
24   public int volumn(){
25       return this.getLength()*this.getWidth()*height;
26   }
27 }
```

```
1  public class Demo_1 {
2    public static void main  (String[] args)
3    {  Rectangle r= new Rectangle(2,5);
4       System.out.println("Rectangle: " + r.toString());
5       System.out.println("   Area: " + r.area());
6       Box b= new Box(2,2,2);
7       System.out.println("Box " + b.toString());
8       System.out.println("  Area: " + b.area());
9       System.out.println("  Volumn: " + b.volumn());
10   }
11 }
```

Output - Chapter06 (run)

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

Overriding and Hiding Methods (1)

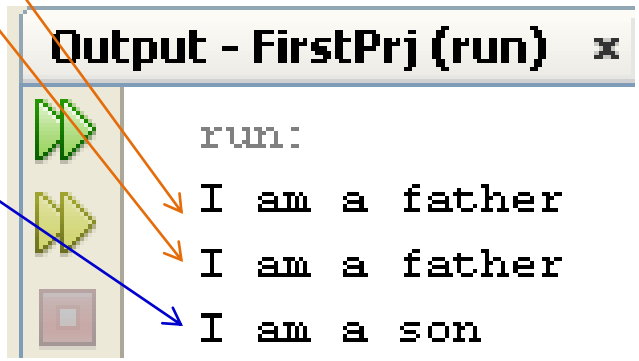
- **Overriding a method**: An instance method in a subclass with the same signature (name, plus the number and the type of its parameters) and return type as an instance method in the superclass overrides the superclass's method.
  - Use the **@Override** annotation that instructs the compiler that you intend to override a method in the superclass (you may not use it because overriding is the default in Java).
- **Hiding a method**: Re-implementing a static method implemented in super class

Overriding and Hiding Methods (2)

```
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();

    }
}
```

Hiding



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);
```

Test

```
1  /* What is the output of the following program */
2  class Study_1A{
3      void M() { System.out.println("A");}
4  }
5  class Study_1B extends Study_1A{
6      void M() { System.out.println("B"); }
7  }
8  class Study_1C{
9      void M() { System.out.println("C"); }
10 }
11 public class Study_1 {
12   public static void main(String[] args)   {
13     Study_1A obj= new Study_1A();
14     obj.M();
15     obj=new Study_1B();
16     obj.M();
17     obj= new Study_1C();
18     obj.M();
19   }
20 }
```

a) ABC

b) AAC

c) ABA

d) Compile-time error

Study_1A and Study_1C are inconvertible

Test

```
/* What is the output of the following program */
class Study_1A{
   void M() { System.out.println("A");}
}
class Study_1B extends Study_1A{
   void M() { System.out.println("B"); }
}
class Study_1C{
   void M() { System.out.println("C"); }
}
public class Study_1 {
  public static void main(String[] args)  {
    Object obj= new Study_1A();
    obj.M();
    obj=new Study_1B();
    obj.M();
    obj= new Study_1C();
    obj.M();
  }
}
```

a) ABC

b) AAC

c) ABA

d) Compile-time error

The java.lang.Object class does not have the M() method

Test

```
/* What is the output of the following program */
class Study_1A{
   void M() { System.out.print("A");}
}
class Study_1B extends Study_1A{
   void M() { System.out.print("B"); }
}
class Study_1C{
   void M() { System.out.print("C"); }
}
public class Study_1 {
  public static void main(String[] args)  {
    Study_1A obj= new Study_1A();
    obj.M();
    obj=new Study_1B();
    obj.M();
    Object obj2= new Study_1C();
    ((Study_1A)obj2).M();
  }
}
```

a) ABC

b) AAA

c) ABA

d) None of the others

AB and a ClassCastException

Test

```
/* What is the output of the following program */
class Study_1A{
   void M() { System.out.print("A");}
}
class Study_1B extends Study_1A{
   void M() { System.out.print("B"); }
}
class Study_1C extends Study_1B {
   void M() { System.out.print("C"); }
}
public class Study_1 {
  public static void main(String[] args)  {
    Study_1A obj= new Study_1A();
    obj.M();
    obj=new Study_1B();
    obj.M();
    obj= new Study_1C();
    obj.M();
  }
}
```

a) AAA

b) ACB

c) None of the others

d) ABC

Test

```
/* What is the output of the following program */
class Study_1A{
   void M() { System.out.print("A");}
}
class Study_1B extends Study_1A{
   void M() { System.out.print("B"); }
}
class Study_1C extends Study_1B {
   void M() { System.out.print("C"); }
}
public class Study_1 {
  public static void main(String[] args)  {
    Study_1C obj= new Study_1C();
    obj.M();
    obj=new Study_1B();
    obj.M();
    obj= new Study_1A();
    obj.M();
  }
}
```

a) ABC

b) AAA

c) ABA

d) None of the others

Compile-time error
(Type conformity violation)

Test

```
public class Study_2 {
   static int N =10;
   int x = 120;
   static{
        N = 50;
        System.out.print("A");
   }
   public void M(){
        System.out.print(x);
   }
   public static void main(String [] args){
        Study_2 obj = new Study_2();
        obj.M();
   }
}
```

a) 120

b) 120A

c) None of the others

d) A120

Test

```
public class Study_2 {
   static int N =10;
   int x = 120;
   static{
        N = 7;
        System.out.print("A" + N );
        x = 500;
   }
   public void M(){
        System.out.print( x );
   }
   public static void main(String [] args){
        Study_2 obj = new Study_2();
        obj.M();
   }
}
```

a) A7500

b) 500A7

c) 500

d) None of the others

Compile-time error
(static code can not access instance variables)

Test

```
public class Study_2 {
   static int N = 2;
   int x = 10;
   static{
        N = 5;
        int y = 7;
        System.out.print("A" + (N + y) );
   }
   public void M(){
        System.out.print( x + y );
   }
   public static void main(String [] args){
        Study_2 obj = new Study_2();
        obj.M();
   }
}
```

a) A1210

b) 10A12

c) 17

d) None of the others

Compile-time error
(The y variable is out of scope)

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.