

Lecture 11: Inheritance and Polymorphism - 1

Prof. Chen-Hsiang (Jones) Yu, Ph.D. College of Engineering

Materials are edited by Prof. Jones Yu from

Liang, Y. Daniel. Introduction to Java Programming, Comprehensive Version, 12th edition, Pearson, 2019.

Motivations

- Suppose you will define classes to model circles, rectangles, and triangles.
- These classes have many common features.
- What is the best way to design these classes to avoid redundancy?
- The answer is to use inheritance.

Outline

- Inheritance
- Polymorphism

Inheritance

Superclasses and Subclasses

Superclass

-color: String -filled: boolean -dateCreated: java.util.Date +GeometricObject() +GeometricObject(color: String, filled: boolean) +getColor(): String +setColor(color: String): void +isFilled(): boolean +setFilled(filled: boolean): void +getDateCreated(): java.util.Date +toString(): String

The color of the object (default: white).

Indicates whether the object is filled with a color (default: false).

The date when the object was created.

Creates a GeometricObject.

 $\label{lem:condition} Creates\ a\ {\tt GeometricObject}\ with\ the\ specified\ color\ and\ filled$

values.

Returns the color.

Sets a new color.

Returns the filled property.

Sets a new filled property.

Returns the dateCreated.

Returns a string representation of this object.

Circle

-radius: double

+Circle()

+Circle(radius: double)

+Circle(radius: double, color: String,

filled: boolean)

+getRadius(): double

+setRadius(radius: double): void

+getArea(): double

+getPerimeter(): double

+getDiameter(): double

+printCircle(): void

Rectangle

-width: double
-height: double

+Rectangle()

+Rectangle(width: double, height: double)

+Rectangle(width: double, height: double

color: String, filled: boolean)

+getWidth(): double

+setWidth(width: double): void

+getHeight(): double

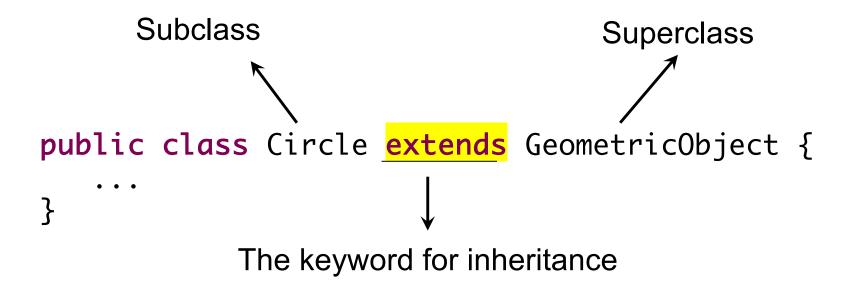
+setHeight(height: double): void

+getArea(): double

+getPerimeter(): double

Subclass

Syntax



Is Superclass's Constructor Inherited?

- No. They are not inherited.
- Unlike properties and methods, a superclass's constructors are not inherited in the subclass.
- They can only be invoked from the subclasses' constructors, using the keyword super, i.e., super() or super(parameter, ...).
- If the keyword super is not explicitly used, the superclass's no-arg constructor is automatically invoked.

Superclass's Constructor Is Always Invoked

- A constructor may invoke (1) an overloaded constructor;
 or (2) its superclass's constructor.
- If none of them is invoked explicitly, the compiler puts super() as the first statement in the constructor. For example, if you have a class A with following constructors:

```
public A() {
    super();
    }

public A(double d) {
    // some statements
}

public A(double d) {
    super();
    // some statements
}
public A(double d) {
    super();
    // some statements
}
```

Using the Keyword super

- The keyword super refers to the superclass.
- This keyword can be used in two ways:
 - » To call a superclass constructor
 - super() or super(parameter1,...)
 - » To call a superclass method
 - super.methodName()

Caution

- You must use the keyword super to call the superclass constructor.
- Invoking a superclass constructor's name in a subclass causes a syntax error.
- Java requires the statement that uses the keyword super appear first in the constructor.

Constructor Chaining

- Constructing an instance (object) of a class invokes all the superclasses' constructors along the inheritance chain.
- This is known as constructor chaining.

Exercise - What is the output? (Create Faculty.java)

```
public class Faculty extends Employee {
  public static void main(String[] args) {
    new Faculty();
  public Faculty() {
    System.out.println("(4) Faculty's no-arg constructor is invoked");
class Employee extends Person {
  public Employee() {
    this ("(2) Invoke Employee's overloaded constructor");
    System.out.println("(3) Employee's no-arg constructor is invoked");
  }
  public Employee(String s) {
    System.out.println(s);
class Person {
  public Person() {
    System.out.println("(1) Person's no-arg constructor is invoked");
```

Exercise - Result

- (1) Performs Person's tasks
- (2) Invoke Employee's overloaded constructor
- (3) Performs Employee's tasks
- (4) Performs Faculty's tasks

```
public class Faculty extends Employee {
 public static void main(String[] args) {
    new Faculty();
  }
 public Faculty() {
    System.out.println("(4) Faculty's no-arg constructor is invoked");
  }
}
class Employee extends Person {
 public Employee() {
    this("(2) Invoke Employee's overloaded constructor");
    System.out.println("(3) Employee's no-arg constructor is invoked");
  }
 public Employee(String s) {
    System.out.println(s);
class Person {
 public Person() {
    System.out.println("(1) Person's no-arg constructor is invoked");
  }
}
```

```
public class Faculty extends Employee {
 public static void main(String[] args)
                                                     1. Start from the main method
    new Faculty();
 public Faculty() {
    System.out.println("(4) Faculty's no-arg constructor is invoked");
  }
class Employee extends Person {
 public Employee() {
    this("(2) Invoke Employee's overloaded constructor");
    System.out.println("(3) Employee's no-arg constructor is invoked");
  }
 public Employee(String s) {
    System.out.println(s);
class Person {
 public Person() {
    System.out.println("(1) Person's no-arg constructor is invoked");
  }
}
```

```
public class Faculty extends Employee {
  public static void main(String[] args) {
    new Faculty();
                                                   2. Invoke Faculty constructor
  public Faculty() {
    System.out.println("(4) Faculty's no-arg constructor is invoked");
class Employee extends Person {
 public Employee() {
    this("(2) Invoke Employee's overloaded constructor");
    System.out.println("(3) Employee's no-arg constructor is invoked");
  }
 public Employee(String s) {
    System.out.println(s);
class Person {
 public Person() {
    System.out.println("(1) Person's no-arg constructor is invoked");
  }
}
```

```
public class Faculty extends Employee {
  public static void main(String[] args) {
    new Faculty();
  public Faculty() {
    System.out.println("(4) Faculty's no-arg constructor is invoked");
                                        3. Invoke Employee's no-arg constructor
class Employee extends Person {
 public Employee() {
    this("(2) Invoke Employee's overloaded constructor");
    System.out.println("(3) Employee's no-arg constructor is invoked");
  }
 public Employee(String s) {
    System.out.println(s);
class Person {
 public Person() {
    System.out.println("(1) Person's no-arg constructor is invoked");
```

```
public class Faculty extends Employee {
  public static void main(String[] args) {
    new Faculty();
  public Faculty() {
    System.out.println("(4) Faculty's no-arg constructor is invoked");
class Employee extends Person {
 public Employee() {
    this("(2) Invoke Employee's overloaded constructor");
    System.out.println("(3) Employee's no-arg constructor is invoked");
  }
 public Employee(String s) {
    System.out.println(s);
                      4. Invoke Person() constructor
class Person {
 public Person() {
    System.out.println("(1) Person's no-arg constructor is invoked");
```

```
public class Faculty extends Employee {
  public static void main(String[] args) {
    new Faculty();
  public Faculty() {
    System.out.println("(4) Faculty's no-arg constructor is invoked");
class Employee extends Person {
 public Employee() {
    this("(2) Invoke Employee's overloaded constructor");
    System.out.println("(3) Employee's no-arg constructor is invoked");
 public Employee(String s) {
    System.out.println(s);
                                                   5. Execute println
class Person {
 public Person() {
   System.out.println("(1) Person's no-arg constructor is invoked");
}
```

```
public class Faculty extends Employee {
  public static void main(String[] args) {
    new Faculty();
  public Faculty() {
    System.out.println("(4) Faculty's no-arg constructor is invoked");
                                          6. Invoke Employee(String) constructor
class Employee extends Person {
 public Employee() {
   this("(2) Invoke Employee's overloaded constructor");
    System.out.println("(3) Employee's no-arg constructor is invoked");
  public Employee(String s) {
    System.out.println(s);
class Person {
 public Person() {
    System.out.println("(1) Person's no-arg constructor is invoked");
```

```
public class Faculty extends Employee {
  public static void main(String[] args) {
    new Faculty();
  public Faculty() {
    System.out.println("(4) Faculty's no-arg constructor is invoked");
class Employee extends Person {
 public Employee() {
   this("(2) Invoke Employee's overloaded constructor");
    System.out.println("(3) Employee's no-arg constructor is invoked");
 public Employee(String s) {
   System.out.println(s);
                               7. Execute println
class Person {
 public Person() {
    System.out.println("(1) Person's no-arg constructor is invoked");
```

```
public class Faculty extends Employee {
  public static void main(String[] args) {
    new Faculty();
  public Faculty() {
    System.out.println("(4) Faculty's no-arg constructor is invoked");
class Employee extends Person {
 public Employee() {
    this("(2) Invoke Employee's overloaded constructor");
    System.out.println("(3) Employee's no-arg constructor is invoked");
                                               8. Execute println
 public Employee(String s) {
    System.out.println(s);
class Person {
 public Person() {
    System.out.println("(1) Person's no-arg constructor is invoked");
  }
}
```

```
public class Faculty extends Employee {
  public static void main(String[] args) {
    new Faculty();
 public Faculty() {
    System.out.println("(4) Faculty's no-arg constructor is invoked");
                                                        9. Execute println
class Employee extends Person {
 public Employee() {
    this("(2) Invoke Employee's overloaded constructor");
    System.out.println("(3) Employee's no-arg constructor is invoked");
  }
 public Employee(String s) {
    System.out.println(s);
class Person {
 public Person() {
    System.out.println("(1) Person's no-arg constructor is invoked");
  }
```

The Impact of a Superclass without no-arg Constructor

Find out the errors in the program:

```
public class Apple extends Fruit {
}

class Fruit {
  public Fruit(String name) {
    System.out.println("Fruit's constructor is invoked");
  }
}
```

Exercise

Create a Java file (Inheritance_1.java) with following content. What is the output?

```
class NU{
    public NU(){
        System.out.println("This is NU no-arg constructor.");
    }
}

class MGEN extends NU{
    public MGEN(){
        System.out.println("This is MEGN no-arg constructor.");
    }
}

public class Inheritance_1 {
    public static void main(String[] args){
        System.out.println("Before MGEN object is created.");
        MGEN mgen = new MGEN();
        System.out.println("After MGEN object is created.");
    }
}
```

Answer

Before MGEN object is created.

This is NU no-arg constructor.

This is MGEN no-arg constructor.

After MGEN object is created.