



## Introduction to Inheritance

Liang, Introduction to Java Programming and Data Structures,  
Twelfth Edition, (c) 2020 Pearson Education, Inc. All rights reserved.



By: Mamoun Nawahdah (Ph.D.)  
2022



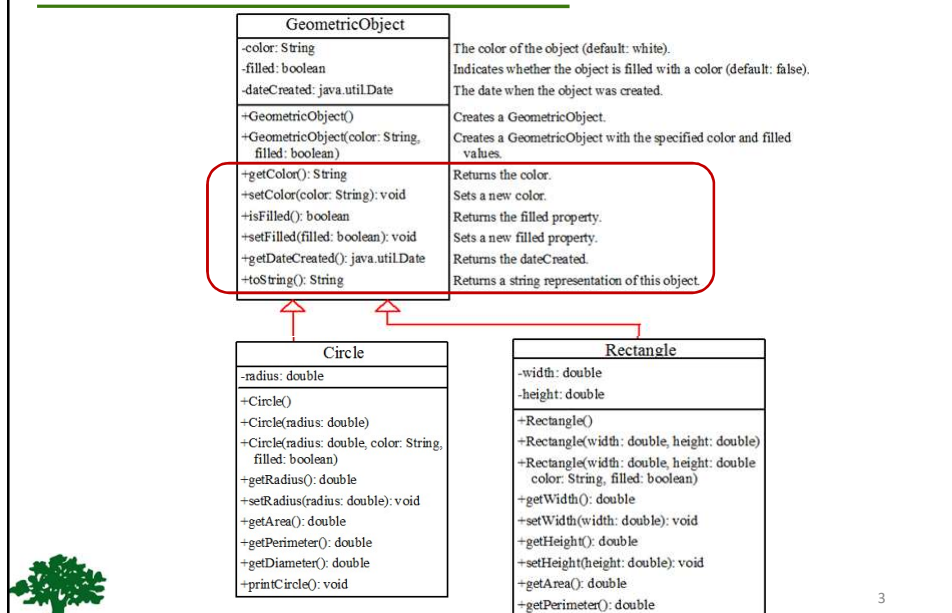
## Motivations

- ❖ 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 so to **avoid redundancy**?

The answer is to use **inheritance**



# Superclasses and Subclasses



## Superclass



```
class Picanto {
    // Key (touch)
    // :
    // Speed: 120 K/H
    // Weight: 840Kg
    // Engine: 1.0-liter Kappa II three cylinder engine
}
```



```
class GT_Line extends Picanto {
    // Speed: 160 K/H
    // Weight: 900Kg
    // Engine: 1.2-liter Kappa II four-cylinder engine
}
```

## Subclass

## Are Superclass's Constructor Inherited?

- ❖ **No**. Unlike properties and methods, a superclass's **constructors are not inherited** in the subclass.
- ❖ They are invoked **explicitly** or **implicitly**.
- ❖ Explicitly using the **super** keyword.
- ❖ They can only be invoked from the subclasses' constructors, using the keyword **super**.



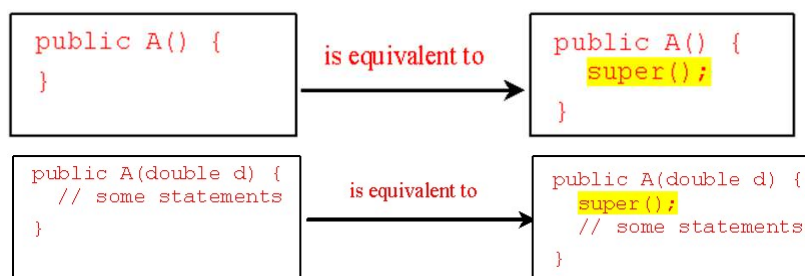
*If the keyword **super** is not **explicitly** used, the superclass's **no-arg constructor** is **automatically** invoked.*



5

## Superclass's Constructor is Always Invoked

- ❖ A constructor may invoke an **overloaded** constructor **or** its superclass's constructor.
- ❖ **If** none of them is invoked explicitly, the compiler puts **super()** as the first statement in the constructor.
- ❖ For example:



## Using the Keyword **super**

- ❖ The keyword **super** refers to the superclass of the class in which super appears.
- ❖ **super** keyword can be used in two ways:
  - To call a superclass constructor.
  - To call a superclass method.



7

## 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 that the statement that uses the keyword **super** appear first in the constructor.



8

## Constructor Chaining

Constructing an instance of a class invokes all the superclasses' constructors along the inheritance chain. This is called **constructor chaining**.

```

public class Faculty extends Employee {
    public static void main(String[] args) {
        Faculty f = 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");
    }
}

```

Super(); →

Super(); →

Super(); →



## Example on the Impact of a Superclass without no-arg Constructor

❖ Find out the **errors** in the following program:


```

public class Apple extends Fruit {

}

public class Fruit {
    public Fruit(String name) {
        System.out.println("Name: " + name);
    }
}

```



10

## Defining a Subclass

❖ A **subclass** inherits from a superclass.  
You can also:

👉 **Add new properties.**

👉 **Add new methods.**

👉 **Override** the methods of the superclass.



11

## Calling Superclass Methods

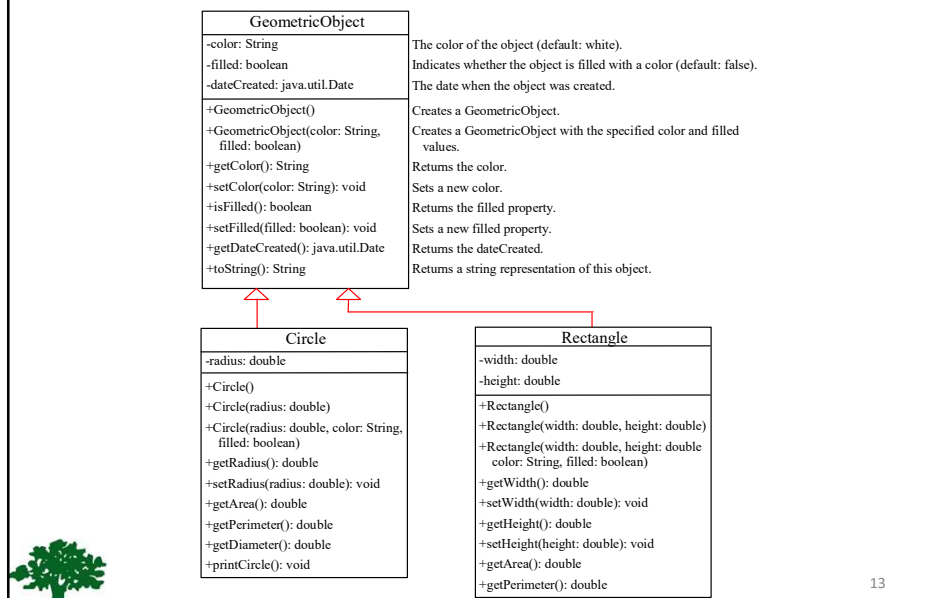
❖ You could rewrite the **printCircle()** method in the **Circle** class as follows:

```
public void printCircle() {  
    System.out.println("The circle is created " +  
        super.getDateCreated() +  
        " and the radius is " + radius);  
}
```



12

# Superclasses and Subclasses



## Overriding Methods in the Superclass

❖ Sometimes it is necessary for the subclass to **modify** the implementation of a method defined in the superclass.

❖ This is referred to as **method overriding**.

```

public class Circle extends GeometricObject {
    // Other methods are omitted

    /** Override the toString method defined in GeometricObject */
    public String toString() {
        return super.toString() + "\n radius is " + radius;
    }
}
  
```

## Note

- ❖ An **instance method** can be overridden **only if** it is accessible.
  - Thus a **private method** cannot be overridden, because it is not accessible outside its own class.
  - If a method defined in a subclass is **private** in its superclass, the two methods are completely unrelated.



15

## Overriding vs. Overloading

```
public class Test {
    public static void main(String[] args) {
        A a = new A();
        a.p(10);
        a.p(10.0);
    }
}

class B {
    public void p(double i) {
        System.out.println(i * 2);
    }
}

class A extends B {
    // This method overrides the method in B
    public void p(double i) {
        System.out.println(i);
    }
}
```



16



## Overriding vs. Overloading

```
public class Test {
    public static void main(String[] args) {
        A a = new A();
        a.p(10);
        a.p(10.0);
    }
}

class B {
    public void p(double i) {
        System.out.println(i * 2);
    }
}

class A extends B {
    // This method overloads the method in B
    public void p(int i) {
        System.out.println(i);
    }
}
```

17

## The Object Class

- ❖ Every class in **Java** is descended from the **java.lang.Object** class.
- ❖ If no inheritance is specified when a class is defined, the superclass of the class is **Object**.

<pre>public class Circle {     ... }</pre>	Equivalent =====	<pre>public class Circle extends Object{     ... }</pre>
--	---------------------	--

18

## The **toString()** method in **Object**

- ❖ The **toString()** method returns a string representation of the **object**.
- ❖ The default implementation returns a string consisting of:
  - A **class name** of which the object is an instance.
  - The at sign (@).
  - A **number** representing this object.



19

## The **toString()** method in **Object**

```
Circle c = new Circle();
System.out.println(c.toString());
```

- ❖ The code displays something like:
 

**Circle@15037e5**
- ❖ This message is not very helpful or informative.
- ❖ Usually you should **override** the **toString** method so that it returns an informative string representing the object.



20

