Chapter 11 Inheritance

Creating classes from other classes!

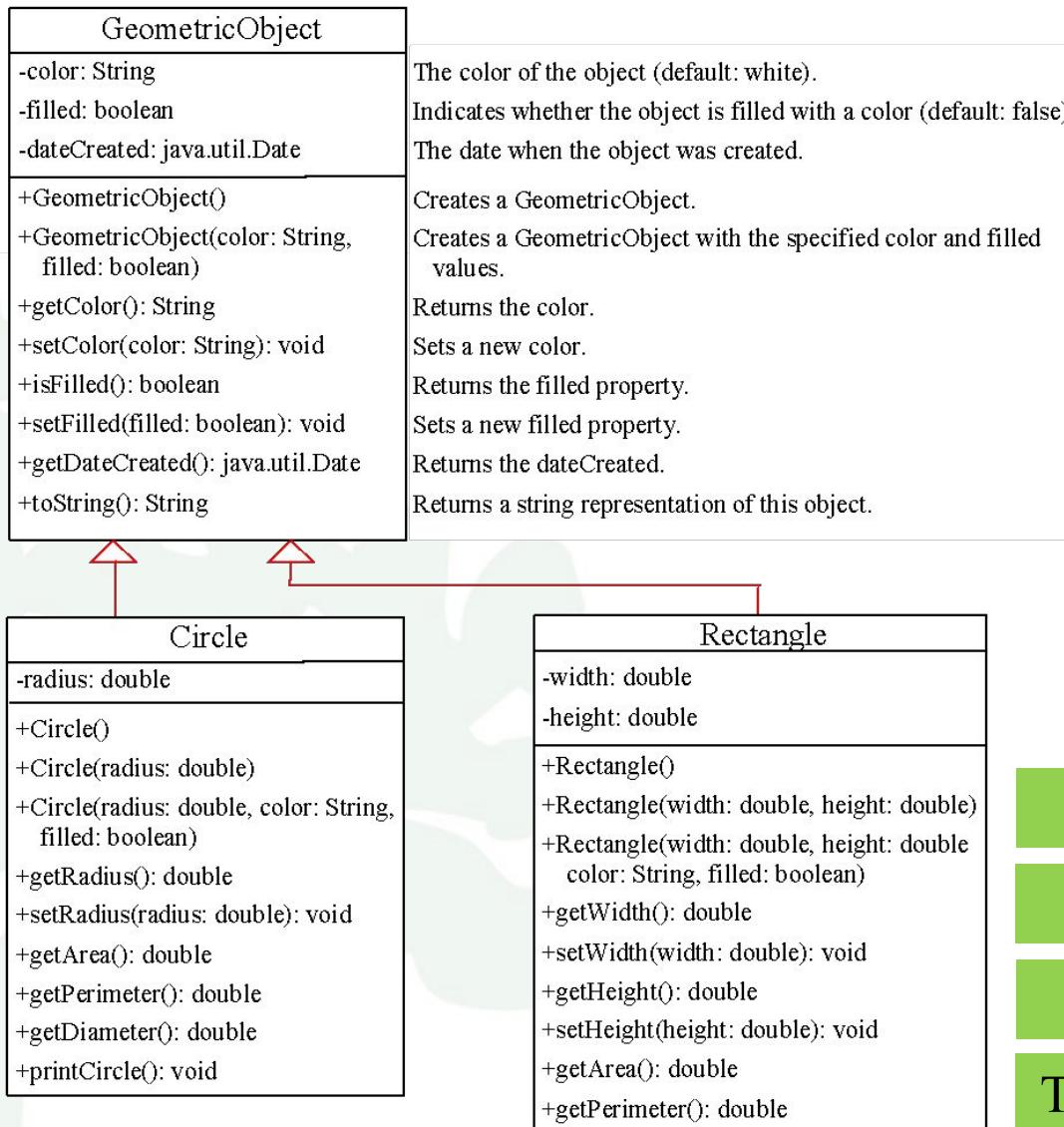Liang, Introduction to Java Programming, Eleventh Edition, (c) 2017 Pearson Education, Inc. All rights reserved.

1

Uploaded By: Jibreel Bornat

Inheritance cont.

- **Subclasses**: a subclass may inherit the structure and behaviour of it's **superclass**.

Uploaded By: Jibreel Bornat
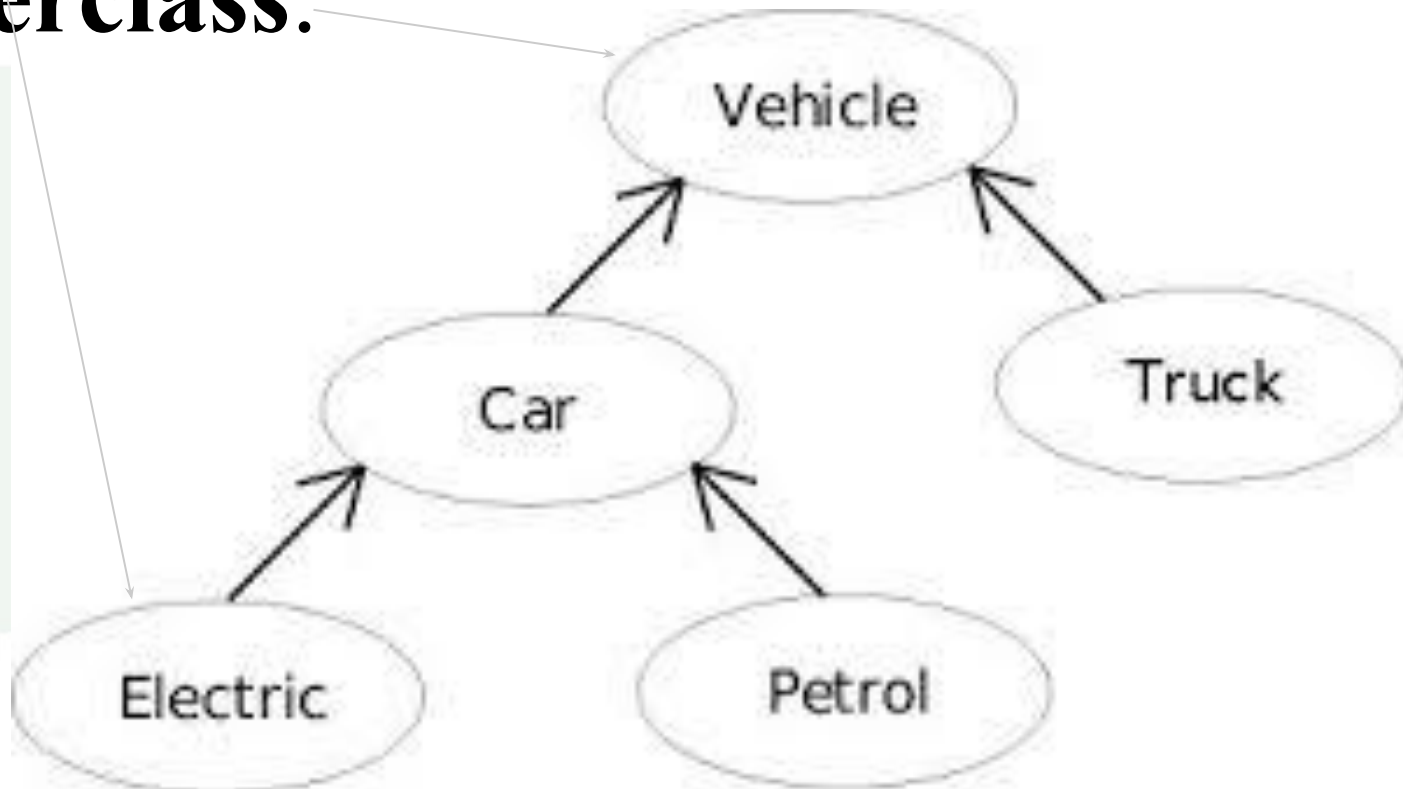
Inheritance

- You use a class to model objects of the same type.

- Different classes may have some common properties and behaviors, which can be generalized in a class that can be shared by other classes.

- You can define a specialized class that extends the generalized class.

- The specialized classes inherit the properties and methods from the general class.

Superclasses and Subclasses

GeometricObject	
-color: String	The color of the object (default: white).
-filled: boolean	Indicates whether the object is filled with a color (default: false).
-dateCreated: java.util.Date	The date when the object was created.
+GeometricObject()	Creates a GeometricObject.
+GeometricObject(color: String, filled: boolean)	Creates a GeometricObject with the specified color and filled values.
+getColor(): String	Returns the color.
+setColor(color: String): void	Sets a new color.
+isFilled(): boolean	Returns the filled property.
+setFilled(filled: boolean): void	Sets a new filled property.
+getDateCreated(): java.util.Date	Returns the dateCreated.
+toString(): String	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

GeometricObject

Circle

Rectangle

TestCircleRectangle   Run

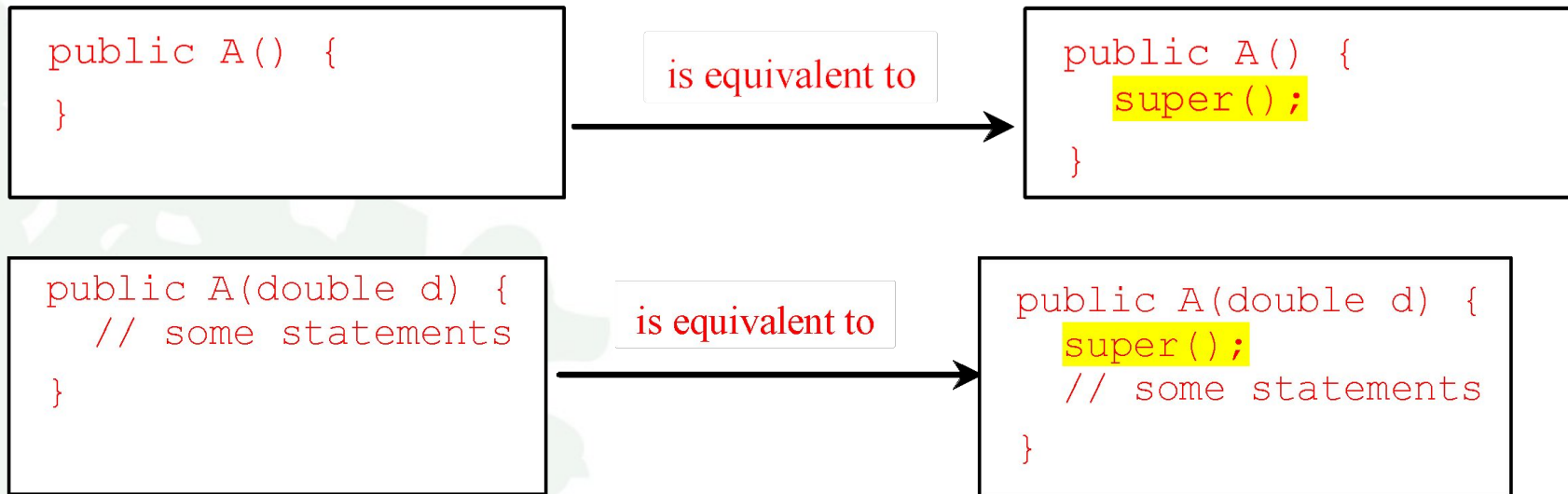# A subclass inherits accessible data fields and methods from its superclass

Does it inherit class constructors?!!

A constructor is used to construct an instance of a class. 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 <u>super</u>. *If the keyword <u>super</u> is not explicitly used, the superclass's no-arg constructor is automatically invoked.*

> *What if there is no-arg constructor defined in the super class?!!!*

Uploaded By: Jibreel Bornat

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,

```
public A() {

}
```

is equivalent to

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

}
```

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

is equivalent to

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

}
```

⚠ Caution

You must use the keyword **super** to call the superclass constructor, and the call must be the first statement in the constructor. Invoking a superclass constructor's name in a subclass causes a syntax error.

Using the Keyword `super`

The keyword `super` refers to the superclass of the class in which `super` appears. This keyword can be used in two ways:

❑ To call a superclass constructor

❑ To call a superclass method

Uploaded By: Jibreel Bornat

7

CAUTION

You must use the keyword <u>super</u> 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 <u>super</u> appear first in the constructor.

STUDENTS-HUB.com

Uploaded By: Jibreel Bornat

Constructor Chaining

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

```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");
  }
}
```

Uploaded By: Jibreel Bornat

9

Output

```
Faculty() {                Employee() {               Employee(String s) {       Person() {
                              this("(2) ...");

  Performs Faculty's          Performs Employee's        Performs Employee's        Performs Person's
  tasks;                      tasks;                     tasks;                     tasks;

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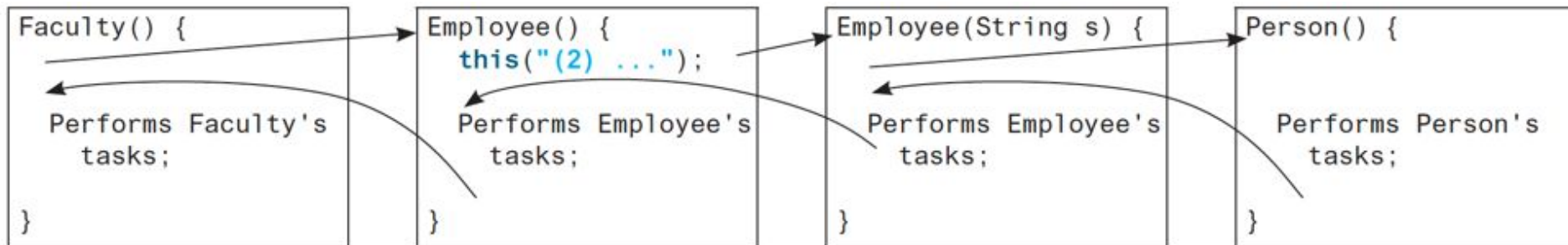
```
(1) Performs Person's tasks
(2) Invoke Employee's overloaded constructor
(3) Performs Employee's tasks
(4) Performs Faculty's tasks
```

10

What is the output?

```java
class A {
  public A() {
    System.out.println(
      "A's no-arg constructor is invoked");
  }
}

class B extends A {
}

public class C {
  public static void main(String[] args) {
    B b = new B();
  }
}
```

Uploaded By: Jibreel Bornat

What is the Output?

```java
class A {
  public A(int x) {
  }
}

class B extends A {
  public B() {
  }
}

public class C {
  public static void main(String[] args) {
    B b = new B();
  }
}
```

Uploaded By: Jibreel Bornat

12

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

Find out the errors in the program:

```java
public class Apple extends Fruit {
}

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

Design Guide

If possible, you should provide a no-arg constructor for every class to make the class easy to extend and to avoid errors.

13

Uploaded By: Jibreel Bornat

Defining a Subclass

A subclass inherits from a superclass. You can also:

- ❑   Add new properties

- ❑   Add new methods

- ❑   Override the methods of the superclass

STUDENTS-HUB.com   Uploaded By: Jibreel Bornat

Calling Superclass Methods

You could rewrite the <u>printCircle()</u> method in the <u>Circle</u> class as follows:

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

Uploaded By: Jibreel Bornat

15

Overriding Methods in the Superclass

A subclass inherits methods from a 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*.

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


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

}
```

To override a method, the method must be defined in the subclass using the same signature as in its superclass.

Uploaded By: Jibreel Bornat 16

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.

STUDENTS-HUB.com

Uploaded By: Jibreel Bornat

NOTE

Like an instance method, a static method can be inherited. However, a static method cannot be overridden. If a static method defined in the superclass is redefined in a subclass, the method defined in the superclass is hidden.

The hidden static methods can be invoked using the syntax SuperClassName.staticMethodName.

Uploaded By: Jibreel Bornat

18

Overriding vs. Overloading

```java
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);
  }
}
```

```java
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);
  }
}
```

- Overridden methods are in different classes related by inheritance; overloaded methods can be either in the same class, or in different classes related by inheritance.

- Overridden methods have the same signature; overloaded methods have the same name but different parameter lists.

STUDENTS-HUB.com

Uploaded By: Jibreel Bornat

19

Problems?!

```java
public class Circle {
  private double radius;

  public Circle(double radius) {
    radius = radius;
  }

  public double getRadius() {
    return radius;
  }

  public double getArea() {
    return radius * radius * Math.PI;
  }
}

class B extends Circle {
  private double length;

  B(double radius, double length) {
    Circle(radius);
    length = length;
  }

  @Override
  public double getArea() {
    return getArea() * length;
  }
}
```

This annotation denotes that the annotated method is required to override a method in its superclass. If a method with this annotation does not override its superclass's method, the compiler will report an error

venth Edition, (c) 2017 Pearson Education, Inc. All reserved.

20

The <u>Object</u> Class and Its Methods

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.

```
public class Circle {
  ...
}
```

Equivalent

```
public class Circle extends Object {
  ...
}
```

Uploaded By: Jibreel Bornat

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 (@), and a number representing this object (memory add.).

```
Loan loan = new Loan();

System.out.println(loan.toString());
```

The code displays something like Loan@15037e5 . This message is not very helpful or informative. Usually you should override the toString method so that it returns a digestible string representation of the object.

Liang, Introduction to Java Programming, Eleventh Edition, (c) 2017 Pearson Education, Inc. All rights reserved.

STUDENTS-HUB.com

Uploaded By: Jibreel Bornat

The `final` Modifier

- The `final` class cannot be extended:

```
final class Math {
   ...
}
```

- The `final` variable is a constant:

```
final static double PI = 3.14159;
```

- The `final` method cannot be overridden by its subclasses.