



Northeastern
University

Lecture 12: Inheritance and Polymorphism - 2

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.

Outline

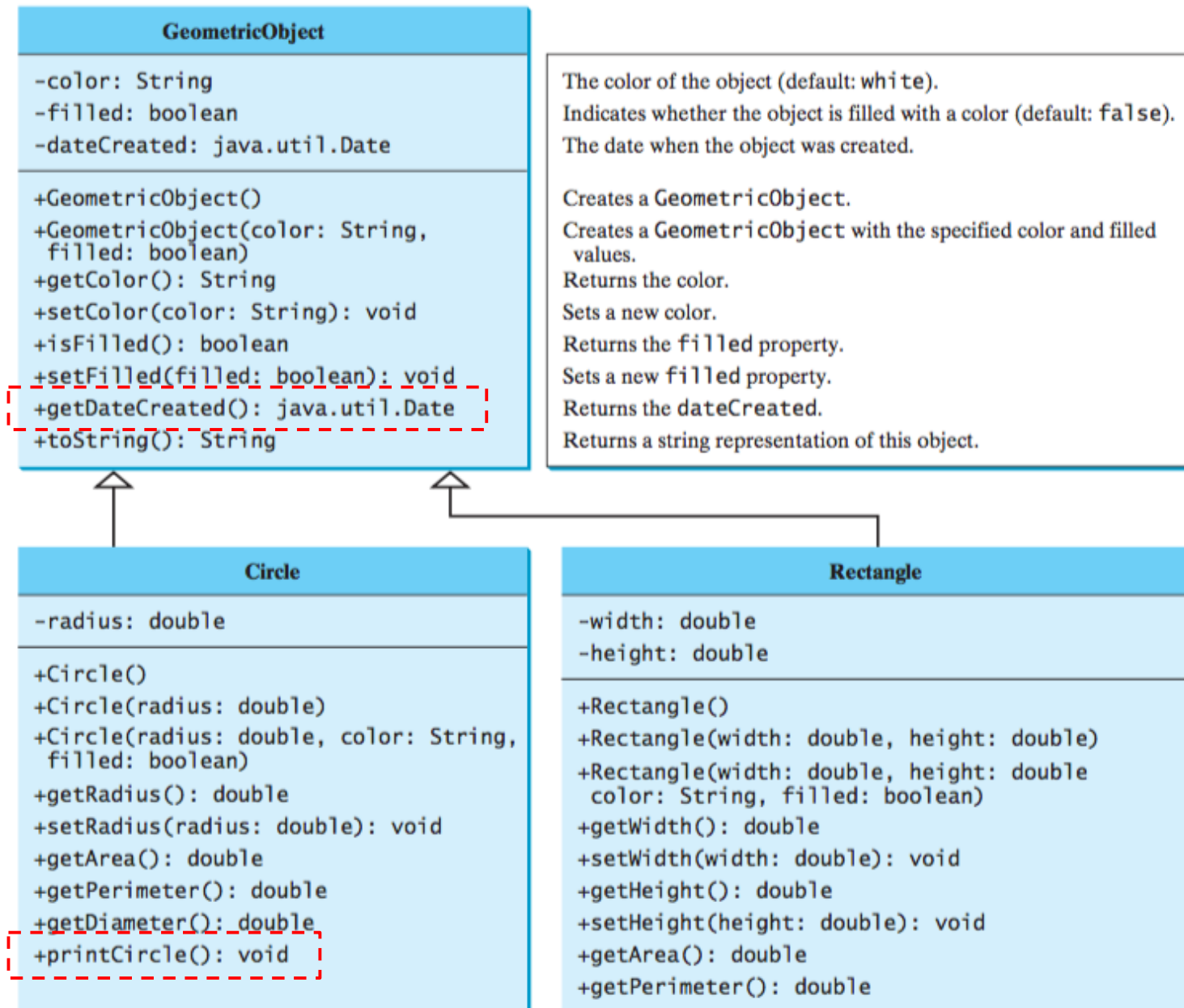
- Inheritance
- Polymorphism

Inheritance (cont.)

Defining a Subclass

- A subclass inherits from a superclass. (Use “**extends**” keyword)
- You can also:
 - » Add new properties (data fields)
 - » Add new methods
 - » **Override the methods** of the superclass

Superclasses and Subclasses



Calling Superclass Methods

- You could call superclass methods.
- For example, you can implement 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);  
  
}
```

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**.

```
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;  
    }  
}
```

Notes

- An **instance method** can be overridden **only if it is accessible**.
- Therefore, **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**.

Notes (cont.)

- 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.

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);  
    }  
}
```

Overriding

```
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);  
    }  
}
```

Overloading

The Object 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 {  
    ...  
}
```

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**.

```
Loan loan = new Loan();  
System.out.println(loan.toString());
```

The toString() method in Object

- 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.

Exercise

- In terms of **inheritance**, what is the effect of keeping a constructor private?

Answer

- Declaring a constructor private allows **no one outside of the class** to instantiate the class.
- It means if the constructor is private, the class can not be inherited.

Private Constructors in Java:

<https://www.baeldung.com/java-private-constructors#:~:text=Private%20constructors%20allow%20us%20to,is%20known%20as%20constructor%20delegation>

Polymorphism

Polymorphism

- A **class** defines a **type**.
- A type defined by a subclass is called a **subtype**, and a type defined by its superclass is called a **supertype**.
- Polymorphism means that a variable of a supertype can refer to a subtype object.
- Therefore, you can say that **Circle** is a subtype of **GeometricObject** and **GeometricObject** is a supertype for **Circle**.

Exercise

- Please type following code as “`PloymorphismDemo.java`”, run the program and see the results.

```
public class PolymorphismDemo {
    public static void main(String[] args) {
        m(new GraduateStudent());
        m(new Student());
        m(new Person());
        m(new Object());
    }

    public static void m(Object x) {
        System.out.println(x.toString());
    }
}

class GraduateStudent extends Student {
}

class Student extends Person {
    public String toString() {
        return "Student";
    }
}

class Person extends Object {
    public String toString() {
        return "Person";
    }
}
```

PolymorphismDemo.java

Exercise

```
public class PolymorphismDemo {
    public static void main(String[] args) {
        m(new GraduateStudent());
        m(new Student());
        m(new Person());
        m(new Object());
    }


    public static void m(Object x) {
        System.out.println(x.toString());
    }
}

class GraduateStudent extends Student {
}

class Student extends Person {
    public String toString() {
        return "Student";
    }
}

class Person extends Object {
    public String toString() {
        return "Person";
    }
}
```

Method *m* takes a parameter of the Object type. You can invoke it with any object.



Output:

Student
Student
Person
java.lang.Object@6d06d69c

Polymorphism and Dynamic Binding

- When the method `m(Object x)` is executed, the argument `x`'s `toString` method is invoked.
- `x` may be an instance of `GraduateStudent`, `Student`, `Person`, or `Object` .
- Classes `GraduateStudent`, `Student`, `Person`, and `Object` have their own implementation of the `toString` method.
- Which implementation is used will be determined dynamically by the Java Virtual Machine at runtime. This capability is known as *dynamic binding*.