FAKULTAS
ILMU
KOMPUTER

Inheritance and Polymorphism

Dasar – Dasar Pemrograman 2

Dinial Utami Nurul Qomariah

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

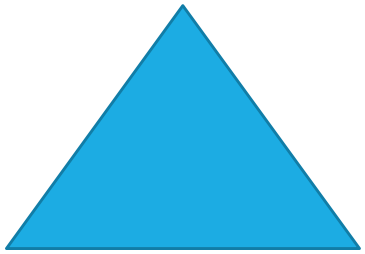
FAKULTAS ILMU KOMPUTER

Credits

- Liang, Introduction to Java Programming, 11th Edition, Ch. 1
- Downey & Mayfield, Think Java: How to Think Like a Computer Scientist, Ch. 1
- Slide Kuliah Dasar-Dasar Pemrograman 2 Semester Genap 2019/2020

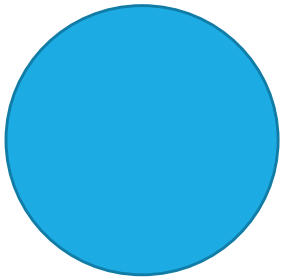
Motivation

- We have some objects, Define the Classes of these objects!



- Color
- Filled
- Height
- Base

- Color
- Filled
- Length
- Height



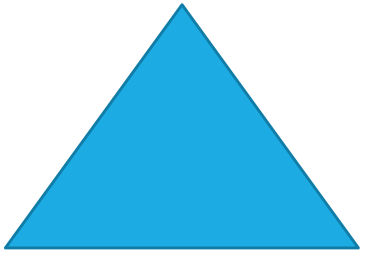
- Color
- Filled
- Diameter

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

FAKULTAS ILMU KOMPUTER

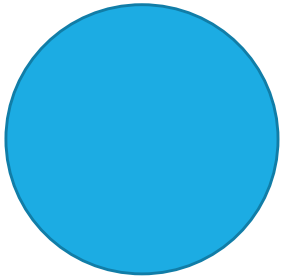
Motivation

- We have some objects, Define the Classes of these objects!



- Color
- Filled
- Height
- Base

- Color
- Filled
- Length
- Height



- Color
- Filled
- Diameter

These classes have common features. What is the best way to design these classes so to avoid redundancy?

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

FAKULTAS ILMU KOMPUTER

Inheritance

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

FAKULTAS ILMU KOMPUTER

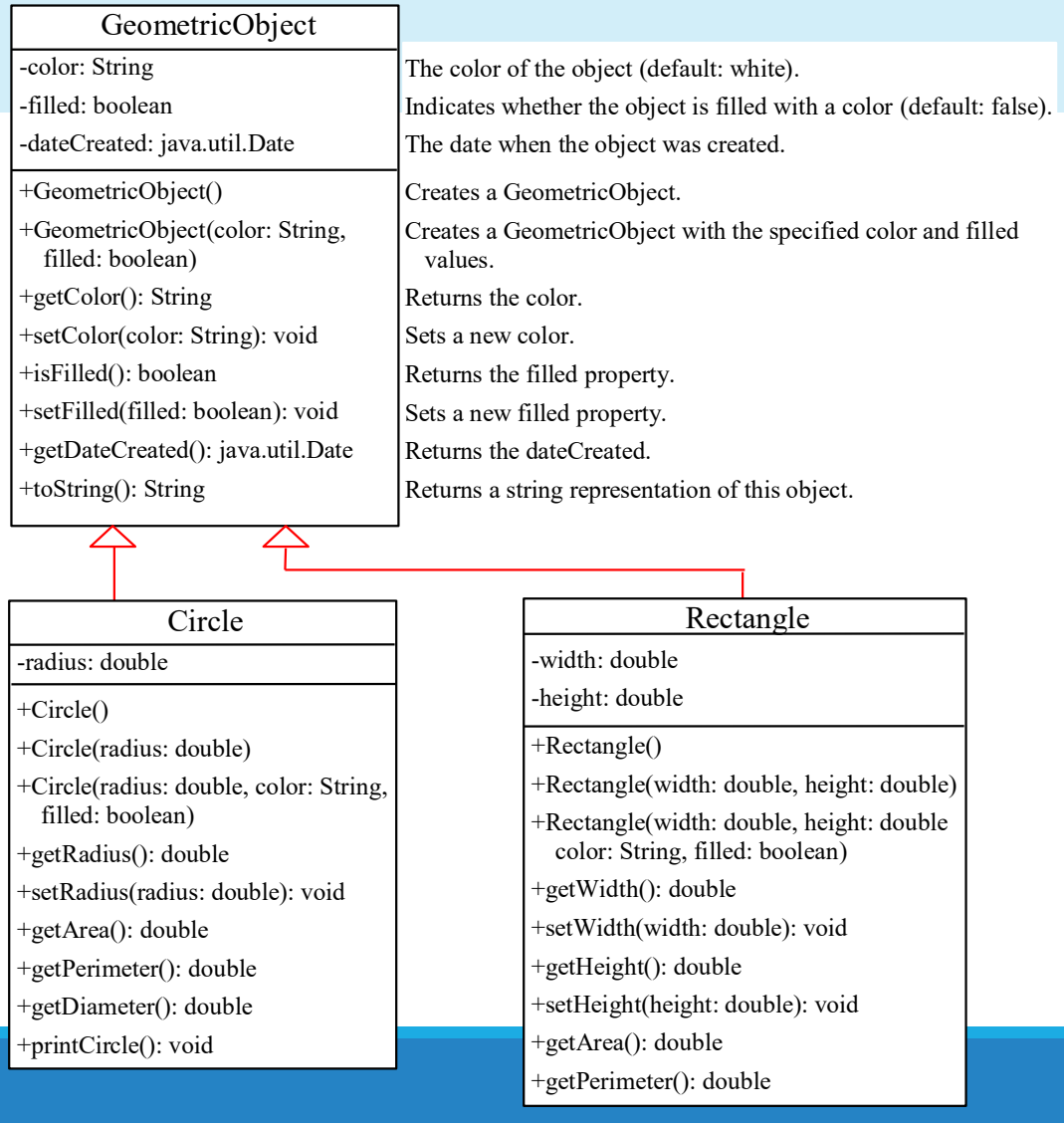
Superclasses and Subclasses

Superclass = parent class / base class
Subclass = derived class / extended class / child class

Subclass memiliki hubungan IS-A dengan superclass.

Setiap subclass mewarisi field (atribut), method, dan nested class dari superclass.

Yang tidak dapat diwariskan:
-. Constructor



Geometric Object

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.

```
public class GeometricObject{
   private String color = "white";
   private boolean filled;
   private java.util.Date dateCreated;
```

https://liveexample.pearsoncmg.com/html/SimpleGeometricObject.html

Circle

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

```
public class Circle extends GeometricObject{
      private double radius;
```

Keyword extends digunakan untuk menjadikan suatu class sebagai turunan dari class lain.

https://liveexample.pearsoncmg.com/html/CircleFromSimpleGeometricObject.html

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

FAKULTAS ILMU KOMPUTER

Rectangle

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

```
public class Rectangle extends GeometricObject{
      private double width;
      private double height;
```

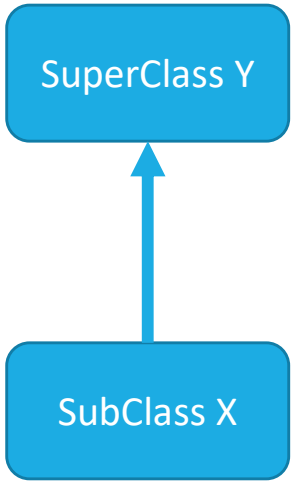
Keyword extends digunakan untuk menjadikan suatu class sebagai turunan dari class lain.

https://liveexample.pearsoncmg.com/html/RectangleFromSimpleGeometricObject.html

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

FAKULTAS ILMU KOMPUTER

Are superclass's Constructor Inherited?



- **Subclass Inherits all Properties and methods**
- 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**.
- *If the keyword **super** is not explicitly used, the superclass's no-arg constructor is automatically invoked.*

Using the Keyword `super`

The keyword super refers to the superclass of the class in which super appears.

The keyword can be used in two ways:

- To call a superclass constructor
- To call a superclass method

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.

```
public A() {
}
```

is equivalent to

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

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

is equivalent to

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

Using the Keyword `super`

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.

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

FAKULTAS ILMU KOMPUTER

Constructor Chaining

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

Konstruktor Berjenjang :

- subclass yang paling bawah akan memanggil konstruktor dari superclass di atasnya.
- konstruktor superclass atas tersebut juga memanggil konstruktor superclass lain yang lebih atas dan seterusnya.

Constructor Chaining

First Super Class is ‘Person’

```
class Person {
  public Person() {
     System.out.println("(1) Person's no-arg constructor
     is invoked");
  }
}
```

Constructor Chaining

Second Class that extends from 'Person' class is Employee, the Employee is Subclass from Person class.

```
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);
  }
}
```

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

FAKULTAS ILMU KOMPUTER

Constructor Chaining

And the Employee have a sub class, the Faculty class.

```
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");
  }
}
```

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

FAKULTAS ILMU KOMPUTER

Trace Execution of Constructor Chaining

```
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");
  }
}
```

1. Start from the main method

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

FAKULTAS ILMU KOMPUTER

Trace Execution of Constructor Chaining

```
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");
  }
}
```

2. Invoke Faculty constructor

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

FAKULTAS ILMU KOMPUTER

Trace Execution of Constructor Chaining

```
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");
  }
}
```

3. Invoke Employee's no-arg constructor

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

FAKULTAS ILMU KOMPUTER

Trace Execution of Constructor Chaining

```
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");
  }
}
```

4. Invoke Employee(String) constructor

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

FAKULTAS ILMU KOMPUTER

Trace Execution of Constructor Chaining

```
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");
  }
}
```

5. Invoke Person() constructor

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

FAKULTAS ILMU KOMPUTER

Trace Execution of Constructor Chaining

```
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");
  }
}
```

6. Execute println

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

FAKULTAS ILMU KOMPUTER

Trace Execution of Constructor Chaining

```
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");
  }
}
```

7. Execute println

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

FAKULTAS ILMU KOMPUTER

Trace Execution of Constructor Chaining

```
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");
  }
}
```

8. Execute println

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

FAKULTAS ILMU KOMPUTER

Trace Execution of Constructor Chaining

```
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");
  }
}
```

9. Execute println

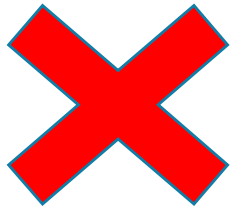
UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

FAKULTAS ILMU KOMPUTER

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

```
public class Apple extends Fruit {
}

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



- Error karena Class Constructor dengan parameter di Superclass harus di akses di class Child/ Sub class

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

```
public class Apple extends Fruit {

}

class Fruit {

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

Add a constructor with a call to `super("name");` here

Or Add a no-argument constructor here

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

```
public class Apple extends Fruit {
    public Apple(String name){
        super(name);
    }
}

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

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

FAKULTAS ILMU KOMPUTER

Defining a Subclass

A subclass inherits from a superclass. You can also:

- Add new properties
- Add new methods
- **Override the methods** of the superclass

Calling Superclass Methods

You could rewrite the printCircle() method in the Circle class

```
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 {


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

}
```

Overriding Methods in the Superclass

```
public class GeometricObject {

   private String toString() {
      return "created on " + dateCreated + "\ncolor: " + color +
      " and filled: " + filled;
   }

}
```

How about if Like This? It's Still Override?

```
public class Circle extends GeometricObject {

   public String toString() {
      return super.toString() + "\nradius is " + radius;
   }

}
```

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

FAKULTAS ILMU KOMPUTER

Overriding vs. Overloading

- Kunci dari Overriding “Jika kita ingin megubah fungsi dari method yang ada di Super Class”.

```
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);
  }
}
```

```
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);
  }
}
```

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.

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.

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

FAKULTAS ILMU KOMPUTER

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 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.