IT 179

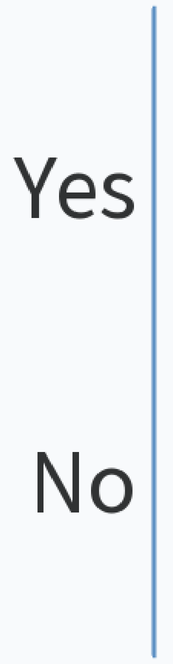
Inheritance & Polymorphism

When poll is active, respond at **pollev.com/abdelmounaam190**

Text **ABDELMOUNAAM190** to **37607** once to join

Have you watched the videos?

Do you have troubles reading from and writing to files in Java?



Any Questions about the Videos?

Inheritance:
Write Code ONCE
(where possible)

A Basic Principle of Good Program Design

Consider

- A university directory with Faculty, Student, and Staff classes
- A payroll system for a company with both salaried and hourly employees.
- An investment system managing stocks, bonds, and savings accounts
- A drawing program with different shapes

Inheritance

- A subclass inherits public instance **variables** (ivars) and **methods** from its super class
- To declare a class a subclass of another class use the `extends` keyword

```
class A {

}

class B extends A {


}
```

Inheritance

- Create parent class
- Child class (sub class) "**inherits**" attributes and behaviors from parent (super class)
- There is an **is-a** relationship between subclasses and their super class.
- Examples
  - A Dog **is-a** Mammal
  - A Rectangle **is-a** Shape
  - A Circle **is-a** Shape

Inheritance - Example

```
class Bike {
      float tireSize;
      void setBrakeType();
}
class RoadBike extends Bike {
      //inherits tireSize
      //inherits setBrakeType();
      private int gears;
      public void setGear();
}
```

Is this valid?

```
class A {

}

class B extends A {


}

class C extends B {


}
```

Is this valid?

```
class A {

}

class B extends A {


}

class C extends B {


}
```

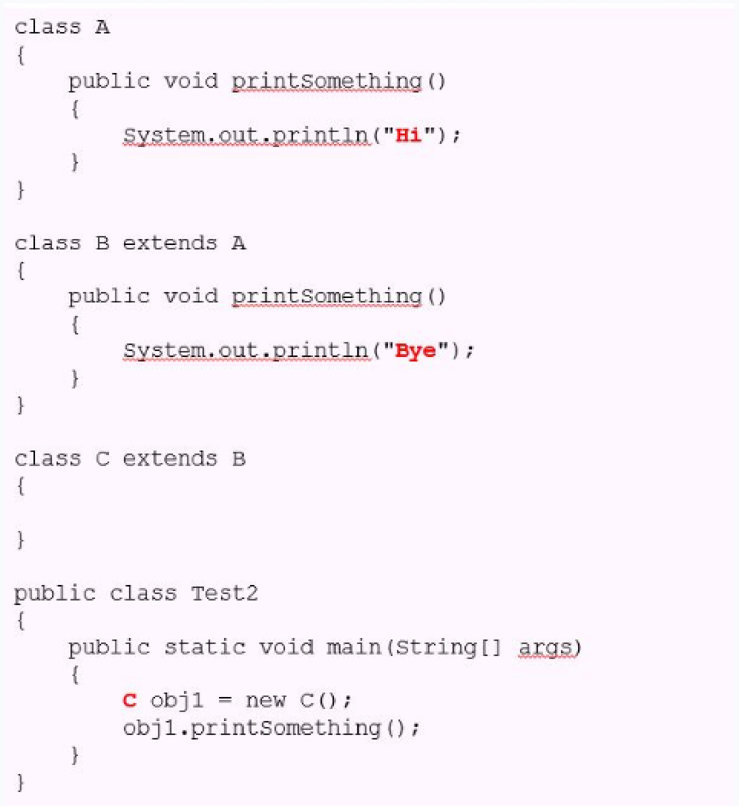
Yes

No

What will this print out?

```
class A
{
    public void printSomething()
    {
        System.out.println("Hi");
    }
}

class B extends A
{
    public void printSomething()
    {
        System.out.println("Bye");
    }
}

class C extends B
{

}

public class Test2
{
    public static void main(String[] args)
    {
        C obj1 = new C();
        obj1.printSomething();
    }
}
```

what will this print out?



```
class A
{
    public void printSomething()
    {
        System.out.println("Hi");
    }
}

class B extends A
{
    public void printSomething()
    {
        System.out.println("Bye");
    }
}

class C extends B
{

}

public class Test2
{
    public static void main(String[] args)
    {
        C obj1 = new C();
        obj1.printSomething();
    }
}
```

Hi

Bye

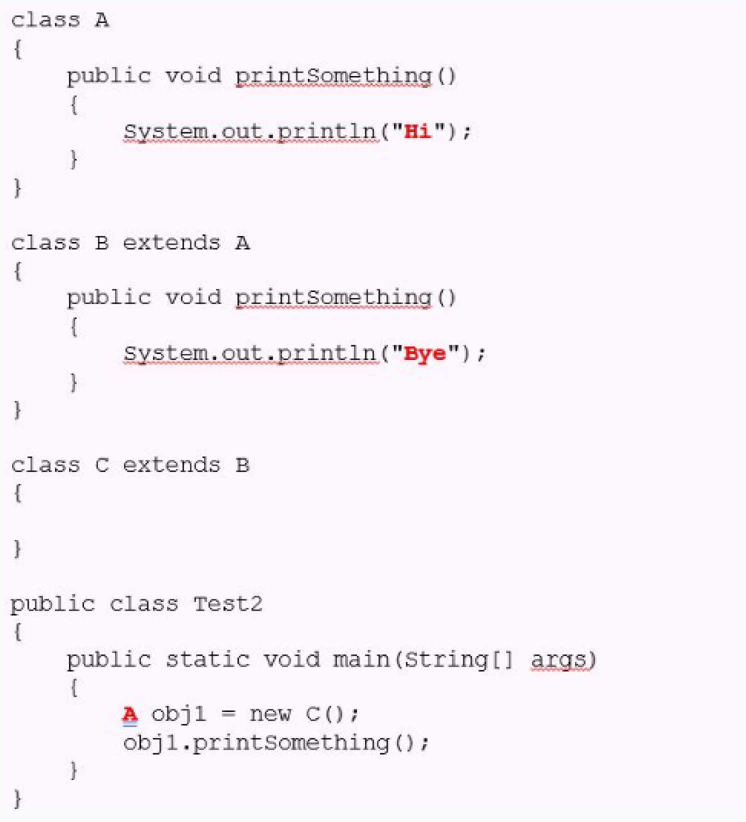
Nothing

A runtime error will occur. No method printSomething() in the class C.

What will this print out?

```
class A
{
    public void printSomething()
    {
        System.out.println("Hi");
    }
}

class B extends A
{
    public void printSomething()
    {
        System.out.println("Bye");
    }
}

class C extends B
{

}

public class Test2
{
    public static void main(String[] args)
    {
        A obj1 = new C();
        obj1.printSomething();
    }
}
```

What will this print out?



```
class A
{
    public void printSomething()
    {
        System.out.println("Hi");
    }
}

class B extends A
{
    public void printSomething()
    {
        System.out.println("Bye");
    }
}

class C extends B
{

}

public class Test2
{
    public static void main(String[] args)
    {
        A obj1 = new C();
        obj1.printSomething();
    }
}
```

Hi

Bye

The code has a compilation error.

There will be a run-time error.

Is this good programming?

```
class Bike {
	public float tireSize;
	void setBrakeType();
}
class RoadBike extends Bike {
	//inherits tireSize
	//inherits setBrakeType();
	private int gears;
	public void setGear();
}
```

Declaring ivars as public breaks encapsulation.

Which problem do we now have?

```
class Bike {
      private float tireSize;
      void setBrakeType();
}
class RoadBike extends Bike {
      ~~//inherits tireSize~~
      //inherits setBrakeType();
      private int gears;
      public void setGear();
}
```

What will happen?

```
class Bike {
      private float tireSize;

}

class RoadBike extends Bike {

    void m() {
        tireSize = 50;
    }
}
```

Field tireSize not visible

Public ivars or instance fields

- While you inherit ivars, in practice they are declared private and are not directly accessible in subclasses

- Use the parent classes getter/setter methods to access ivars.

Public ivars or instance fields

```
class Bike {
      private float tireSize;

      void setTireSize(float ts) {
            tireSize = ts;
      }
}

class RoadBike extends Bike {
      void m() {
            setTireSize(50);
      }
}
```

Overloading methods

- Overloading = create methods with the same name but different number or type of arguments
- Signature of method = name and parameter types

```
class A
{
    public int m(int j)
    {
        return 1;
    }

    public int m(float j)
    {
        return 1;
    }

}
```

Is this valid?

- Overloading = create methods with the same name but different number or type of arguments
- Signature of method = name and parameter types

```
class A
{
    public int m(int j)
    {
        return 1;
    }

    public float m(int j)
    {
        return 1.5;
    }

}
```

Overloading methods

- Overloading = create methods with the same name but different number or type of arguments
- Signature of method = name and parameter types
- The return type is not part of the signature.

Overloading methods

- Subclasses can overload inherited methods

```
class A
{
    int m(int j)
    {
        return 1;
    }

}

class B extends A
{
    int m(double d) {
        return 3;
    }
}
```

Overloading methods

- Constructors are commonly overloaded
- Overloading occurs at compile time

```
Circle c1 = new Circle(); //create a unit circle, centered at (0,0)
Circle c2 = new Circle(3); //create a circle with a radius of 3
                           //and centered at (0,0)

Circle c3 = new Circle(5, 5, 5); //createa circle with radisu 5 and
```

Overriding methods

- Subclasses can define methods that have the exact same signature as a method in a superclass

- The subclass method is said to override the method of the superclass

- Overriding replaces the implementation of the method
  This is called subtype polymorphism

Overriding methods

- Can use `@Override` marker to tell the compiler you are overriding a method.

- Compiler will give a warning if the methods don't match.

- Good practice to use `@Override`

Overriding - Example

```
class Parent {
    void show()
    {
        System.out.println("Parent's show()");
    }
}

class Child extends Parent {
    // This method overrides show() of Parent
    @Override
    void show()
    {
        System.out.println("Child's show()");
    }
}
```

```
class Main {
  public static void main(String[]
args) {
  Parent obj1 = new Parent();
  obj1.show();

  //This is run time polymorphism
  Parent obj2 = new Child();
  obj2.show();
     }
}
```

Back to the Bike Example

```
class Bike {
      public float tireSize;
      public void setBrakeType();
}
class RoadBike extends Bike {
      //inherits tireSize
      //inherits setBrakeType();
      private int gears;
      public void setGear();
}
```

References

- What is the effect of:

```
RoadBike rb = new RoadBike();
```

- 1) creates an object new RoadBike()
- 2) creates a reference named rb of type RoadBike
- 3) makes the reference point to the object.

- reference type AND object type are the same

References and Super Types

- What is the effect of this:

```
Bike bike = new RoadBike();
```

- reference type and the object type are NOT the same.
- The reference is the super class (Bike); the object created is of type RoadBike

This is polymorphism

References and Super Types

- Anything that extends the reference type can be assigned to the reference variable.

- There is an IS-A relationship between the object and the reference.

- Polymorphism allows the reference type and object type to be different.

Example 1 – Animal Farm

Example 2 – Shapes

- Class Shape
  - float area() {}
- Triangle as a subclass of Shape
- Circle as a subclass of Shape
- Rectangle as a subclass of Shape
- Square as a subclass of Rectangle