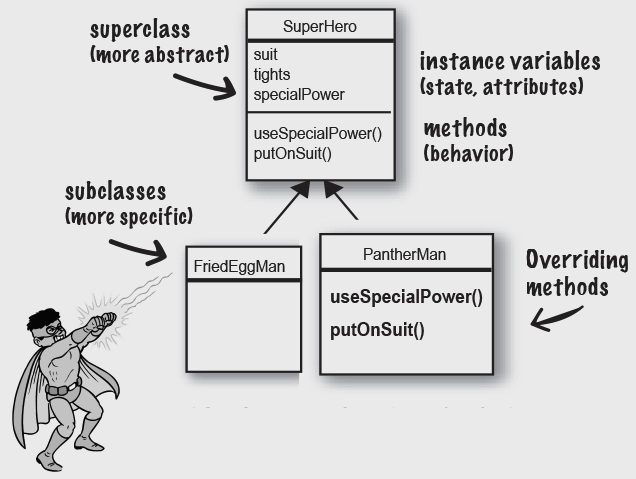
**SubClasses & Inheritance**

Louis Botha, louis.botha@tuni.fi

Head First Java

Source: Head First Java, 3rd Edition

Source: Java All in One

Source: Learn Java 17

Read more:  [Introduction to object-oriented programming](https://java-programming.mooc.fi/part-4/1-introduction-to-object-oriented-programming)

Read more:  [W3Schools](https://www.w3schools.com/java/java_oop.asp)

**Inheritance**

Inheritance refers to a feature of object-oriented programming that lets you create classes that are derived from other classes.

* A class that’s based on another class is said to **inherit** the other class.
* The class that is inherited is called the **parent class**, **the base class**, or the **superclass**.
* The class that does the inheriting is called the **child class**, the **derived class**, or the **subclass**.

The parent-child relationship in Java is expressed using the **extends** keyword:

class A { }

class B extends A { }

The **B** class inherits from the **A** class.

Inheritance is best used to implement is-a-type-of relationships. For example:

* Class **B** is a type of Class **A**
* Solitaire is a type of game
* a truck is a type of vehicle
* an invoice is a type of transaction

In each case, a particular kind of object is a specific type of a more general category of objects.

You need to know a few important things about inheritance:

* A derived class automatically takes on all the behavior and attributes of its base class.
* A derived class can add features to the base class it inherits by defining its own methods and fields.
* A derived class can also change the behavior provided by the base class.
* Public members of a class **are** inherited
* Private members of a class **are** not inherited
* *attention A class can extend only one other class in Java*

**Types of inheritance in Java**

* Single inheritance
* Subclass inherit characteristics from a single superclass

class A { }

class B extends A { }

* Multilevel inheritance
* A subclass can have its own subclasses.

class A { }

class B extends A { }

class C extends B { }

* Hierarchal inheritance
* Superclass can be the parent to multiple levels of subclasses

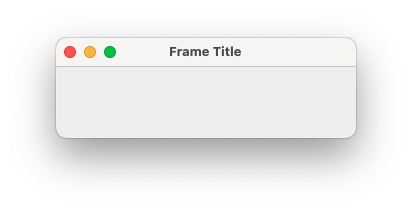
class A { }

class B extends A { }

class C extends A { }

class D extends A { }

Inheritance is used a lot with Graphical User Interface programming.



When inheriting from the a graphics library class, the class will have all the same properties as the default class.

import javax.swing.\*;

class SimpleFrame extends JFrame {

   public SimpleFrame() {

       super("Frame Title");

       setSize(300, 100);

       setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

       setVisible(true);

   }

   public static void main(String[] arguments) {

       SimpleFrame sf = new SimpleFrame();

   }

}

By inheriting JFrame, the SimpleFrame has for example all the properties of a UI window.

**Inheritance and Constructors**

When you create an instance of a subclass, Java automatically calls the default constructor of the base class before it executes the subclass constructor.

class Animal {

    public Animal() {

        System.out.println(

            "Hello from the ANIMAL constructor");

    }

}

class Dog extends Animal {

    public Dog() {

        System.out.println(

            "Hello from the DOG constructor");

    }

}

class Main {

    public static void main(String[] args) {

        Dog myDog = new Dog();

    }

}

// Output

❯ java Main

Hello from the ANIMAL constructor

Hello from the DOG constructor

class Animal {

    private String name;

    public Animal(String name) {

        this.name = name;

    }

    public String getName() {

        return this.name;

    }

}

class Dog extends Animal {

    public Dog(String name) {

        super(name);

    }

}

class Main {

    public static void main(String[] args) {

        Dog myDog = new Dog("Lassie");

        System.out.println("Hello " + myDog.getName());

    }

}

//Output

❯ java Main

Hello Lassie

**Method overriding**

Method Overriding is the process when the subclass or a child class has the same method as declared in the parent class.

Example of Method Overriding in Java.

class Animal {

    public void run() {

        System.out.println("The ANIMAL run");

    }

}

class Dog extends Animal {

    public void run() {

        System.out.println("The DOG run");

    }

}

class Main {

    public static void main(String[] args) {

        Dog myDog = new Dog();

        myDog.run();

    }

}

// Output

❯ java Main

The DOG run

Only the *The DOG run* printed as the class overwrites the run method of the parent class.

**Using Super**

The super keyword works similarly to this but refers to the instance of the base class rather than the instance of the current class.

class Animal {

    public void run() {

        System.out.println("The ANIMAL run");

    }

}

class Dog extends Animal {

    public void run() {

        System.out.println("The DOG run");

        super.run();

    }

}

class Main {

    public static void main(String[] args) {

        Dog myDog = new Dog();

        myDog.run();

    }

}

// Output

❯ java Main

The DOG run

The ANIMAL run

**Using Final**

**final variable**

Java has a final keyword that serves three purposes.

When you use final with a variable, also called a constant, is a variable whose value you can’t change after it’s been initialised.

To declare a final variable, you add the final keyword to the variable declaration, like this:

final int WEEKDAYS = 5;

**final method**

A *final method* is a method that can’t be overridden by a subclass.

class Animal {

    private String name;

    final public String getName() {

        return this.name;

    }

}

As the method getName is declared as final, any class that uses the Animal class as a base class can’t override the getName method. If it tries, the compiler issues the error message ("Overridden method final").

**final class**

A *final class* is a class that can’t be used as a base class.

final class Animal {

}

No one can use the Animal class as the base class for another class.