Java Object Oriented Programming

Inheritance

Hiba ALQASIR

2021-2022

télécom saint-étienne

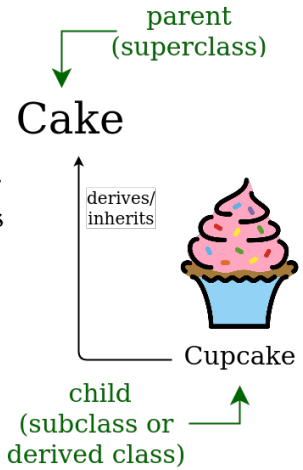
école d'ingénieurs / nouvelles technologies

Today

1. Inheritance: concept, syntax, example
2. The keywords **super** and **final**
3. Method overriding
4. Polymorphism
5. The **Object** class
6. Abstraction

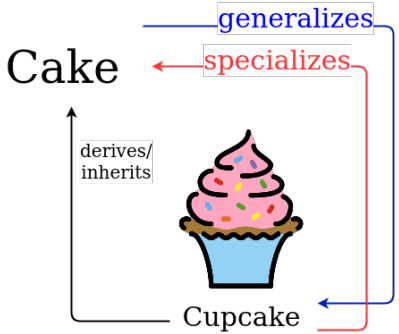
Inheritance principle

- Declare a new class based on an existing class.
- The new class inherits the members (attributes and methods) from the other class.
- The new class has its own members.



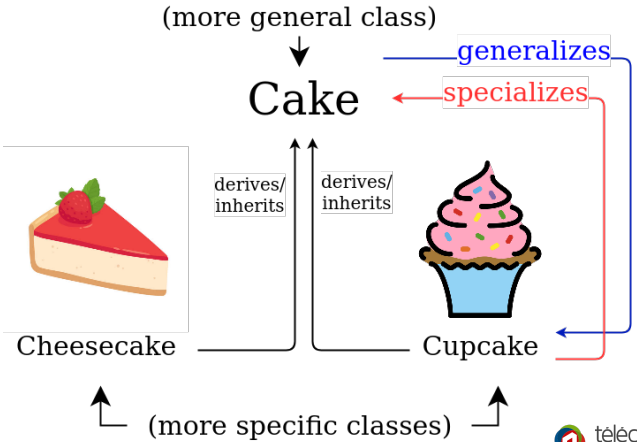
Inheritance principle

- Translation of «to be»: Cupcake is Cake
- Cake generalizes Cupcake
- Cupcake specializes Cake



Inheritance advantages

- Flexibility
- Reusability
- Overriding
- Extensibility
- Data hiding



Java Access Modifiers

One can change the access level of *fields*, *constructors*, *methods*, and *classes* by applying an access modifier on them.

- public +

  this modifier doesn't put any restriction on the access.
- default

  the package only.
- private -

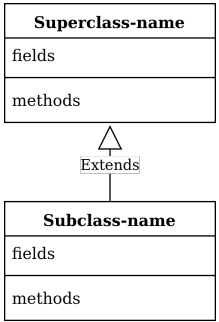
  the class only.
- protected #

  the package and the subclasses present in any package.

Inheritance syntax in Java

```
class Superclass-name
{
    // methods and fields
}
```

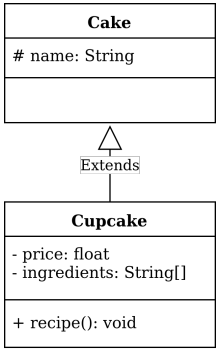
```
class Subclass-name extends Superclass-name
{
    // different methods and fields
}
```



Example

```java
public class Cake {

    protected String name;

}
```

```java
public class Cupcake extends Cake {

    private float price;
    private String[] ingredients;

    public void recipe() {
       System.out.println("Mix the ingredients in a cup, "
                          + "then put it in the oven!");
    }
}
```



super keyword

Whenever you create an instance of subclass, an instance of parent class is created implicitly.

- **super** is used to refer immediate parent class instance.
- **super** can be used to invoke immediate parent class method.
- **super()** can be used to invoke immediate parent class constructor.

super keyword

Example

```java
public class Cake {
    protected String name = "Cake";

    public Cake(String name){
        this.name = name;
    }
}
```

```java
public class CheeseCake extends Cake {
    private float calories;

    public CheeseCake() {
        name =  "Cheese " + super.name ;
    }

    public CheeseCake(String name, float calories) {
        super(name);
        this.calories = calories;
    }
}
```

super keyword

Example

```
1 public class Main{
2   public static void main(String[] args){
3     CheeseCake cheeseCake1 = new CheeseCake();
4     System.out.println(cheeseCake1.name);
5
6     CheeseCake cheeseCake2 = new CheeseCake("Strawberry Cheese Cake", 250);
7     System.out.println(cheeseCake2.name);
8   }
9 }
```

Output:

```
Cheese Cake
Strawberry Cheese Cake
```

final keyword

- **final** variable: one cannot change its value.
- **final** method: one cannot override it.
- **final** class: one cannot extend it.

Method Overriding

Method overriding means providing a specific implementation (in the subclass) of a method that is already implemented in the super class.

1. must be in inheritance context.
2. the method must have same name as in the super class.
3. the method must have same parameters as in the super class.

```
/* Don't confuse the concepts of overloading and overriding */
```

Method Overriding vs. Overloading

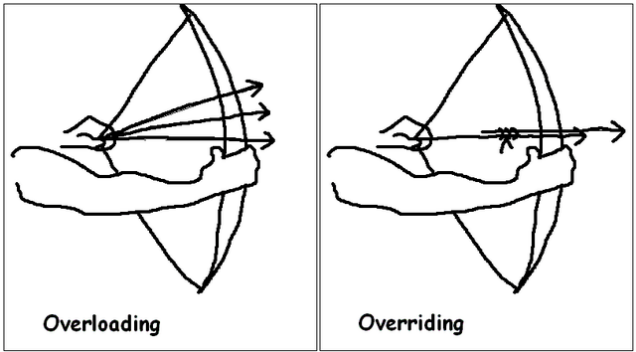
- Overloading refers to several methods with the same name in the same class, but with different signatures.
- Overriding refers to two methods, one in a super class and one in a sub class, that have the same signature.

```
1 class SuperDuper{
2   void marvelous(int i){
3     ...
4   }
5   void marvelous(char c){
6     ...
7   }
8 }
```

```
1 class SuperDuper{
2   void marvelous(int i){
3     ...
4   }
5 }
6 class JuniorSuperDuper extends SuperDuper{
7   void marvelous(int i){
8     ...
9   }
10 }
```

Method Overriding vs. Overloading

- Overloading is used to define a similar operation in different ways for different data.
- Overriding is used to define a similar operation in different ways for different types of objects.

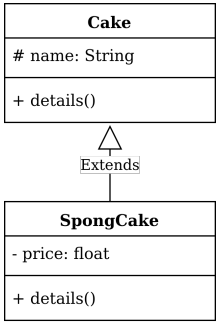


Method Overriding in Java

Example

```java
public class Cake {
    protected String name = "Cake";
    public void details() {
        System.out.println("Cake name is: "+ name);
    }
}
```

```java
public class SpongeCake extends Cake {
    private float price = 25;
    public void details() {
        System.out.println("Cake name is "+ name);
        System.out.println("Cake price is "+ price);
    }
}
```



Method Overriding in Java

Example

```java
public class Main{

    public static void main(String[] args){

        Cake cake = new Cake();
        cake.details();
    }
}
```

Output:

```
Cake name is Cake
```

```java
public class Main{

    public static void main(String[] args){

        SpongeCake cake = new SpongeCake();
        cake.details();
    }
}
```

Output:

```
Cake name is Cake
Cake price is 50
```

Method Overriding in Java

- One cannot override a static method.
- One cannot override a private method.
- One can override an overloaded method.

Polymorphism

Poly[multiple]-morphism[forms]
Polymorphism is to provide a single symbol to entities of different types.

Polymorphism

Example

```java
public class Cake {
    public void description(){
        System.out.println("All cakes are sweet.");
    }
}
```

```java
public class SpongeCake extends Cake {
    public void description() {
        System.out.println("Sponge cake does not contain any fat.");
    }
}
```

```java
public class ChocolateCake extends Cake {
    public void description() {
        System.out.println("Chocolate cake is made with chocolate.");
    }
```

Polymorphism

Example

```
1  public class Main{
2      public static void main(String[] args){
3          Cake cake1, cake2, cake3;
4
5          cake1 = new Cake();
6          cake1.description();
7
8          cake2 = new SpongeCake();
9          cake2.description();
10
11         cake3 = new ChocolateCake();
12         cake3.description();
13     }
14 }
```

Output:

```
All cakes are sweet.
Sponge cake does not contain any fat.
Chocolate cake is made with chocolate.
```

Polymorphism

Example

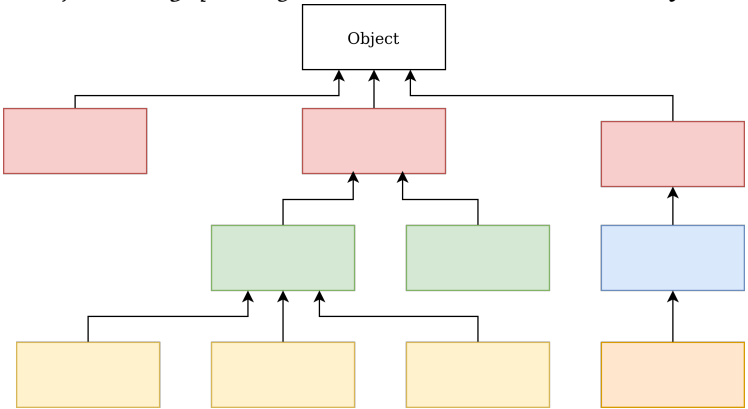
```
1  public class Main{
2      public static void main(String[] args){
3          Cake cake1, cake2;
4
5          cake1 = new Cake();
6          cake2 = new SpongeCake();
7          System.out.println(cake1 instanceof Cake);
8          System.out.println(cake1 instanceof SpongeCake);
9          System.out.println(cake2 instanceof Cake);
10         System.out.println(cake2 instanceof SpongeCake);
11     }
12 }
```

Output:

```
true
false
true
true
```

The Object Class

In *java.lang* package there is a class named **Object**



Object class is the ultimate root of all class hierarchies

The Object Class

Example

```
1  public class Main{
2      public static void main(String[] args){
3
4          Cake cake1 = new Cake();
5          SpongeCake cake2 = new SpongeCake();
6          CheeseCake cake3 = new CheeseCake();
7
8          System.out.println(cake1 instanceof Object);
9          System.out.println(cake2 instanceof Object);
10         System.out.println(cake3 instanceof Object);
11     }
12 }
```

Output:

```
true
true
true
```

Abstraction

- Abstract method
- Abstract class
- Interface (details in next TD)

Abstract method

- Abstract method is declared with **abstract** keyword.
- It has no body.
- Abstract methods must always be in abstract classes.

```
1 abstract void methodName(); //no method body
```

Example

```
1 public abstract class Cake {
2     protected String name = "Cake";
3     public abstract void description(); // abstract method without a body
4 }
```

Abstract class

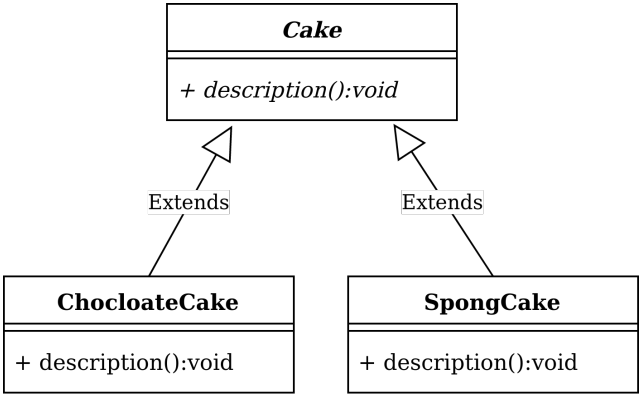
- Abstract class is declared with **abstract** keyword.
- It can have abstract and non-abstract methods, constructors, and static methods.
- It cannot be instantiated.

Example

```
1 public abstract class Cake { // abstract class
2     String name = "Cake";
3     public abstract void description();
4 }
```

Abstract class

Example



Abstract class

Example

```java
public abstract class Cake {
    public abstract void description();
}
```

```java
public class SpongeCake extends Cake {
    @Override
    public void description() {
        System.out.println("Sponge cake does not contain any fat.");
    }
}
```

```java
public class ChocolateCake extends Cake {
    @Override
    public void description() {
        System.out.println("Chocolate cake is made with chocolate.");
    }
```

Abstract class

Example

```
1  public class Main{
2
3      public static void main(String[] args){
4
5          SpongeCake spongeCake = new SpongeCake();
6          spongeCake.description();
7
8          ChocolateCake chocolateCake = new ChocolateCake();
9          chocolateCake.description();
10     }
11 }
```

Output:

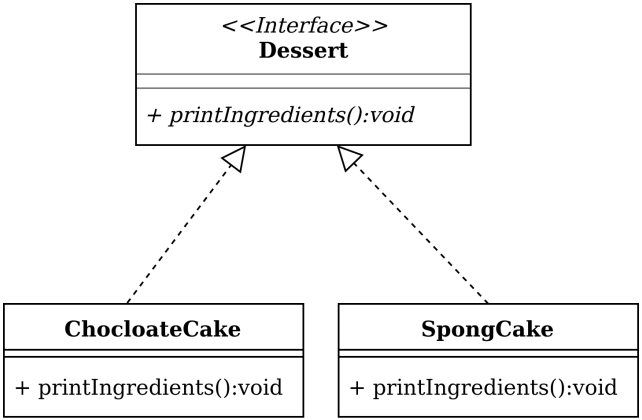
```
Sponge cake does not contain any fat.
Chocolate cake is made with chocolate.
```

Interface

- The interface is declared with **interface** keyword.
- It can have only abstract methods and static constants.
- It cannot be instantiated.

Interface

Example



Interface

Example

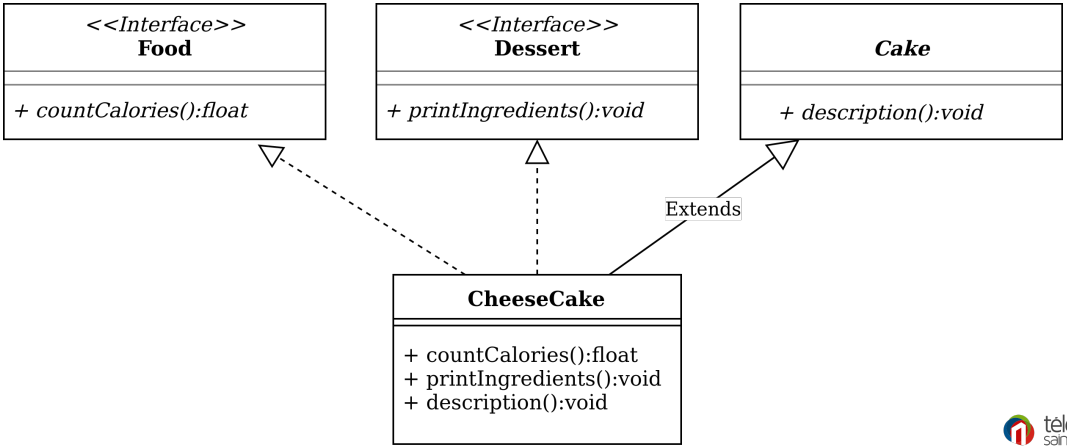
```java
interface Dessert {
    public abstract void printIngredients();
}
```

```java
public class SpongeCake implements Dessert {
    @Override
    public void printIngredients() {
        System.out.println("Eggs, sugar, and flour");
    }
}
```

```java
public class ChocolateCake implements Dessert {
    @Override
    public void printIngredients() {
        System.out.println("Eggs, sugar, flour, chocolate, and butter");
    }
}
```

Single inheritance and Multiple inheritance

Java supports single inheritance, however the **interface** is used to achieve multiple inheritance.



Single inheritance and Multiple inheritance

```java
public class CheeseCake extends Cake implements Food, Dessert {

    @Override
    public void description() {
        System.out.println("Cheese cake has cheese in it.");
    }

    @Override
    public void printIngredients() {
        System.out.println("Soft cheese, eggs, sugar and biscuits.");
    }

    @Override
    public int countCalories() {
        return 250;
    }
}
```

Exercises

Consider the following class:

```
1  public class SuperDuper{
2
3      public SuperDuper()
4      {
5          System.out.println("SuperDuper class constructor called");
6      }
7
8      void marvelous(int i)
9      {
10         System.out.println("SuperDuper class marvelous method "
11         + "called with integer i = " + i);
12     }
13     void marvelous(double d)
14     {
15         System.out.println("SuperDuper class marvelous method "
16         + "called with double d = " + d);
17     }
18 }
```

Exercises

And the following one:

```
1  public class JuniorSuperDuper extends SuperDuper {
2
3      public JuniorSuperDuper()
4      {
5          System.out.println("JuniorSuperDuper class constructor called");
6      }
7
8      void marvelous(double d)
9      {
10         System.out.println("JuniorSuperDuper class marvelous method "
11         + "called with double d = " + d);
12     }
13 }
```

Exercise #1

What is the output of this Java program?

```
1 public class Main
2 {
3     public static void main(String[] args)
4     {
5         JuniorSuperDuper jsd = new JuniorSuperDuper();
6     }
7 }
```

Try to answer before compiling and executing it.

Exercise #2

What is the output of this Java program?

```
1 public class Main
2 {
3     public static void main(String[] args)
4     {
5         JuniorSuperDuper jsd = new JuniorSuperDuper();
6         jsd.marvelous(13);
7     }
8 }
```

Try to answer before compiling and executing. 

Exercise #3

Consider the following classes:

```java
class A {
    private int a = 25;
    public void mA() { a *= a; }
}
```

```java
class B extends A {
    protected int a = 7;
    protected int b = 1;
    public void mB() { b *= a; }
}
```

```java
class C extends B {
    protected int b = 5;
    protected int c = 13;
    public void mA() { c += b; super.a++; }
    public void mC() { mA(); }
}
```

Exercise #3

What is the output of the following Java program?

```
1 public class Main
2 {
3     public static void main(String[] args)
4     {
5         C obj = new C();
6         obj.mC();
7         System.out.println(obj.a + " " + obj.b + " "+ obj.c);
8     }
9 }
```

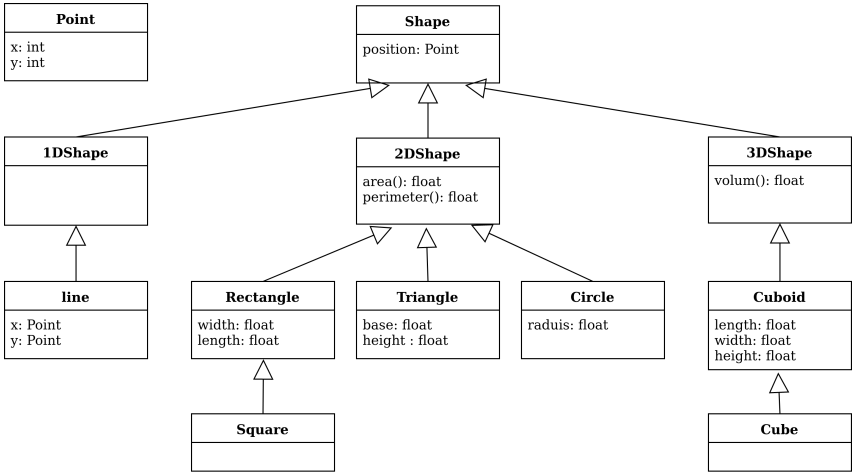
Try to answer before compiling and executing it.

Exercise #4

Implement the following class hierarchy, considering:

- You can and should add methods to the classes (getters and setters for example).
- Determine which classes or methods should be abstract.
- Reuse elements of super classes.
- Be creative and have fun!

Exercise #4


Point
x: int
y: int
Shape
position: Point
1DShape
2DShape
area(): float
perimeter(): float
3DShape
volum(): float
line
x: Point
y: Point
Rectangle
width: float
length: float
Triangle
base: float
height : float
Circle
raduis: float
Cuboid
length: float
width: float
height: float
Square
Cube