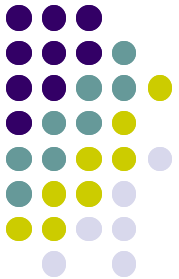
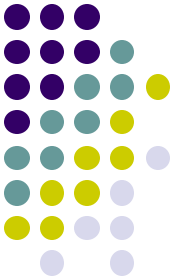
Overloading vs overriding



Ionut

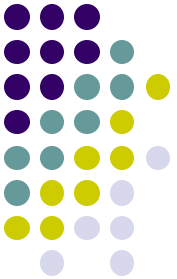
Spalatelu

Method Overloading



- Method **overloading** means providing two or more separate methods in a class with the **same name** but different parameters
- Method **return type** may or may not be different
- Both **static** and **instance** methods can be overloaded
- Usually overloading happens inside a single class, but a method can also be treated as overloaded in the subclass of that class
- This is because a subclass inherits one version of the method from the parent class and then the subclass can have another overloaded version of the method

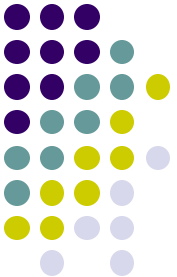
Method Overloading rules



- Methods will be considered overloaded if both of the following rules:
  - Methods must have the same method name
  - Methods must have different parameters

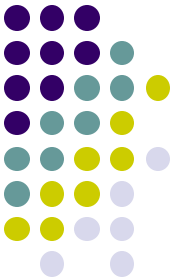
- If methods follow the rules above then may or may not:
  - Have different return types
  - Have different access modifiers
  - Throw different checked or unchecked exceptions

Method Overriding



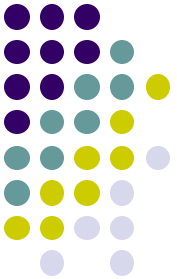
- Method **overriding** means defining a method in a **child** class that already exists in the **parent** class with **same signature**
- By extending the parent class the child class gets all the methods defined in the parent class (inherited methods)
- Method overriding is also known as **Runtime Polymorphism** and **Dynamic Method Dispatch**, because the method that is going to be called is decided at runtime by the JVM

Method Overriding rules



- Methods are overridden if we follow these rules:
  - **Signatures** must be the **same**
  - Return type:
    - If is a **primitive** or **void** in the parent class then both return types must **match exactly**
    - If is a class in the parent class then return type can be a subclass of the one in the parent class
  - It **can't** have a **lower access modifier**
  - It **can't** throw a **broader checked exception** than the method in the parent class
  - If there is an **unchecked exception** in the parent class there is **no restriction**
  - Method from the parent class cannot be **final**

Method Overriding key points



- There are some important points about method overriding to keep in mind
  - **Only inherited methods can be overridden**, in other words methods can be overridden only in child classes
  - **Constructors** and **private methods cannot be overridden**
  - A subclass can use **super.methodName()** syntax to call the superclass version of an overridden method

Method Overriding vs Overloading

OVERRIDING

same name
same parameters

```
class Dog {

    public void bark() {
        System.out.println("woof");
    }
}

class GermanShepherd extends Dog {

    @Override
    public void bark() {
        System.out.println("woof woof
woof");
    }
}
```

OVERLOADING

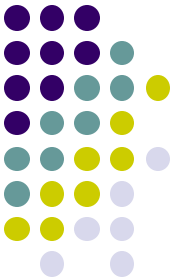
same name
different parameters

```
class Dog {

    public void bark() {
        System.out.println("woof");
    }

    public void bark(int number) {
        for(int i = 0; i < number; i++) {
            System.out.println("woof");
        }
    }
}
```

Summary



Method Overloading	**Method Overriding**
Provides functionality to reuse a method name with different parameters.	Used to override a behavior which the class has inherited from the parent class.
Usually in a single class but may also be used in a child class.	**Always in two classes** that have a child-parent or IS-A relationship.
Must have different parameters.	**Must have** the same parameters and same name.
May have different return types.	Must have the same return type or covariant return type (child class).
May have different access modifiers(private, protected, public).	**Must NOT** have a lower modifier but may have a higher modifier.
May throw different exceptions.	**Must NOT** throw a new or broader checked exception.

Questions