Deel 5: Inheritance

Subclass in extension inheritance kunnen uitleggen.

Extension inheritance is a type of inheritance where the subclass inherits all the characteristics and all the methods defined at the level of its superclass, without any change to those.

Java uses the keyword **extends** to indicate that this class inherits from a superclass. Subclasses inherit *all the instance methods and all the instance variables*. The implementation of the methods and variables is left untouched in extension inheritance. Java *subclasses do not inherit static methods and static variables from their superclass*. Constructors are not inherited either, as constructors are static methods. You can invoke the constructor of the superclass by callng **super**(...) as the first line of code in the constructor of your subclass.

In extending subclasses, the subclass introduces new instance properties and class properties, which will only apply to the subclass itself.

> Good to know
>
> Method **overloading** happens in the same class, shares the same method name but each method should have different number of parameters or parameters having different types and order. But in method **overriding,** derived class have the same method with same name and exactly the same number and type of parameters and the same return type as a parent class.

Open-closed principle:
Grant subclasses access to getters and setters defined at the level of their superclass by qualifying them (at least) protected. This means that classes, models, function, etc. should be open for extension, but closed for modification.

Abstract class in extension inheritance kunnen uitleggen

> Good to know
>
> **Object class** is present in java.lang package. Every class in Java is directly or indirectly derived from the Object class. E.g. toString(), equals(), etc.

An **abstract method** is a method that has no implementation specified in the superclass. Thus, a subclass must override them, it cannot simply use the version defined in the superclass. Therefore, that class becomes an **abstract class**. So the sole purpose of an abstract class is to introduce characteristics an behaviour common to a series of subclasses that inherit from it. Every abstract class inherits from the concrete Object class. Abstract class can inherit from concrete classes and vice versa. An abstract class cannot have any objects of its own.

Polymorphism kunnen uitleggen

There is distinction between a static and a dynamic type of a variable. The **static type** of a variable is the type stated in its declaration. The **dynamic type** of a variable is the type of the contents that the variable currently store.

Example

```
DiskItem:

DiskItem test;
File file1 = new File();
Test = file1;
```

The static type of test is **DiskItem**, and the dynamic type is **File**.

You can only invoke methods to a reference variable if the *static* type (class) that has that method implemented or if you type cast that variable to such a class. See the resume on page 6 for an example.

The static type of a variable with value semantics is at all times the same as its dynamic type.

Overriding of methods kunnen uitleggen.

Subclasses can work out a more appropriate/specific implementation of instance methods they inherit from their superclass. Concrete subclasses that inherit from an abstract superclass *must* work out an implementation of each abstract method they inherit. Also, develop class hierarchies in such a way that any extension with new subclasses has no impact at all on the definition of existing classes in the hierarchy.

When *overriding* **abstract methods** at the level of a subclass, the only changes that are allowed are the *return type* that can be replaced by a subtype, the *access rights* can be widened and the *list of exceptions* can be shortened. The compiler interprets the method as a new one if these rules are not respected.

When *overriding* **concrete methods** at the level of a subclass, the versions of all methods defined at the level of its superclass are still available.

Dynamische binding kunnen uitleggen.

Given the fact that several versions of an instance method can exist throughout a hierarchy of classes, **dynamic binding** selects the version to be executed based on the dynamic type of the expression whose evaluation yields the prime objects. The selection of the actual version of the method to be executed is performed at runtime.

A final method cannot be overridden at the level of a direct or indirect subclass.