Theory: Abstract class vs interface

§1. Differences between abstract classes and interfaces

Abstract class and **interface** are both tools to achieve abstraction that allow us to declare the abstract methods. We cannot create instances of abstract classes and interfaces directly, we can only do that through classes that inherit them.

Since Java 8, an interface can have default and static methods that contain an implementation. It makes interface more similar to an abstract class. So, the important question is: what is the difference between interfaces and abstract classes?

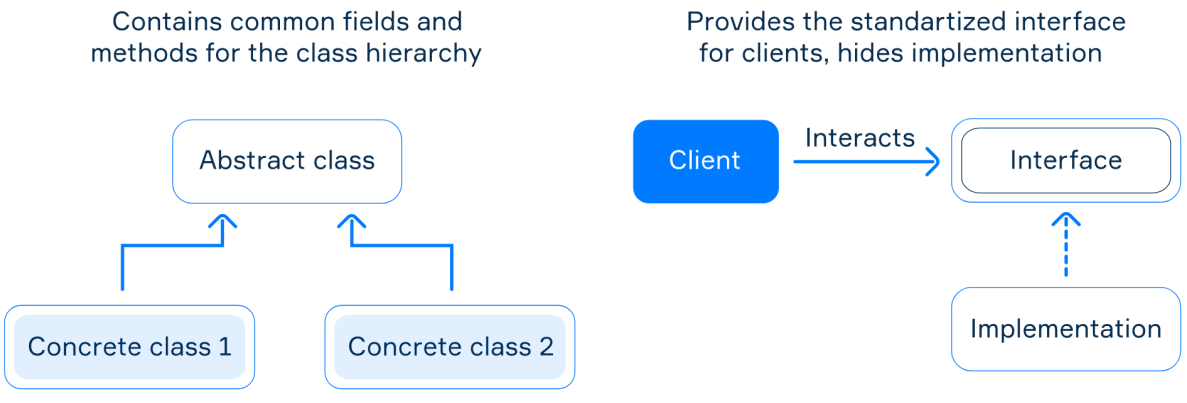
Below you can see a list of some important differences between these two concepts.

- an **abstract class** can have *abstract* and *non-abstract* instance methods while an **interface** can have *abstract* or *default* instance methods;
- an **abstract class** can extend another abstract or regular class and an **interface** can only extend another interface;
- an **abstract class** can extend only one class while an **interface** can extend any number of interfaces;
- an **abstract class** can have *final*, *non-final*, *static*, *non-static variables* (regular fields) while an interface can only have *static final variables*;
- an **abstract class** can provide an implementation of an interface but an **interface** cannot provide an implementation of an abstract class;
- an **abstract class** can have a constructor and an **interface** cannot;
- in an **abstract class**, the keyword `abstract` is mandatory to declare a method as an *abstract* one while in an **interface** this keyword is optional.

Remember, a class **extends** another class, a class **implements** an interface, but an interface **extends** another interface.

The provided list of differences is by no means complete. **Abstract classes** and **interfaces** have a lot of other differences but the main one is their purpose.

Typically, interfaces are used to decouple the interface of a component (class) from the implementation while abstract classes are often used as base classes with common fields to be extended by subclasses.



The typical use of abstract classes and interfaces

The picture above demonstrates the last statement.

§2. Using abstract classes and interfaces together

Current topic:

Abstract class vs interface

Topic depends on:

- Abstract class
- Interface

Topic is required for:

- Factory method

Table of contents:

Sometimes interfaces and abstract classes are used together to make a class hierarchy more flexible. In this case, an abstract class contains common members and implements one or multiple interfaces, and concrete classes extend the abstract class and possibly other interfaces.

See the following simple example.

```
interface ManagedDevice {

    void on();

    void off();
}

abstract class AbstractDevice implements ManagedDevice {


    protected String serialNumber;

    protected boolean on;



    public AbstractDevice(String serialNumber) {

        this.serialNumber = serialNumber;

    }



    protected void setOn(boolean on) {

        this.on = on;

    }

}


class Kettle extends AbstractDevice {



    protected double volume;



    public Kettle(String serialNumber, double volume) {

        super(serialNumber);

        this.volume = volume;

    }



    @Override

    public void on() {

        // do complex logic to activate all electronic components

        setOn(true);

    }



    @Override

    public void off() {

        // do complex logic to stop all electronic components

        setOn(false);

    }

}
```

Using both concepts (interfaces and abstract classes) makes your code more flexible. Use suitable abstractions or their combination when designing your class hierarchies.

As an example, you may see class hierarchies in the standard Java class library. An example of that is the collections hierarchy. It combines abstract classes and interfaces to make the hierarchy more maintainable and flexible to use in your code.

Report a typo

248 users liked this theory. **10** didn't like it. **What about you?**

Start practicing

Comments (9) Hints (0) Useful links (0) Show discussion