4544 users solved this topic. Latest completion was about 15 hours ago.

Theory: Protected modifier

15 minutes 0 / 5 problems solved

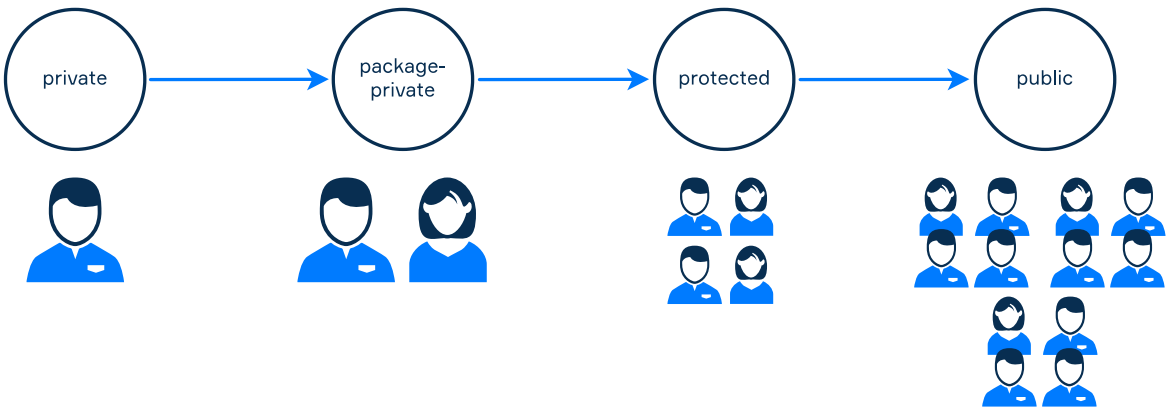
Skip this topic Start practicing

§1. Protected modifier

As you may remember, an access modifier describes who can use your piece of code. There are 4 of them in Java:

- `private`: available only for the class itself;
- `default`: available for classes from the same package (= **package-private**);
- `protected`: available for classes from the same package and the extending classes.
- `public`: available from everywhere;

We've already considered most of them, but there's the most interesting one left: the `protected` access modifier. Let's place it among the modifiers you already know:



This modifier means that only subclasses and any classes from the same package can use a class member. A top-level class cannot be protected, but an inner one can be declared this way. This is where the importance of a proper package decomposition comes in.

Now let's define the difference between `protected` and its scale neighbors, `private` and "package-private" (default).

§2. Comparing protected and other modifiers

Protected vs default. You can think of classes from the same package as the neighbors and subclasses as the children of a particular class. There are some things you can share or do with your neighbors, for example, discuss a plan of the building repairs or share the basement. These things and actions would be package-private (default).

There are also things you can do for children and close friends, like borrow some money or go for a walk in a park on Sunday. These things will be `protected`.

Protected vs private. This distinction is even easier: if a variable, a method or an inner class is used only by the class itself, then it is `private`, otherwise, it is `protected`. Following the main rule:

> Use the most restrictive access level that makes sense for a particular member.

if you're not sure if the method is useful for other classes, it's better to first make it private and expand its availability later if needed.

§3. Example

Current topic:

Protected modifier

Topic depends on:

✓ Access modifiers Stage 6

✓ Inheritance Stage 7

Topic is required for:

Referencing subclass objects

Table of contents:

↑ Protected modifier

§1. Protected modifier

§2. Comparing protected and other modifiers

§3. Example

§4. Let's recap

Feedback & Comments

In the example below, there are a `Laptop`, a `SmartPhone` and a `SmartWatch` classes in the package `org.hyperskill.bluetooth`. All the gadgets in the package can be connected via Bluetooth. `Laptop` has a method `receiveInfo()`, responsible for getting any information from connected gadgets.

```
package org.hyperskill.bluetooth;

public class Laptop {

    private String info;

    void receiveInfo(String info) {
        this.info = info;
    }


}
```

The `Laptop` class has only a single field `info` which is not directly accessible since it is declared as private. But all classes from the same package can access it invoking the `receiveInfo` method which is declared as **package-private** (no modifier).

We consider that `SmartPhone` and `SmartWatch` classes extend the same `MobileGadget` class with the `printNotification` method:

```
package org.hyperskill.bluetooth;

public class MobileGadget {

    protected void printNotification(String data) {
        System.out.println(data);
    }
}
```

The `printNotification` method is accessible for all subclasses of this class as well as for all classes in the same package (including the `Laptop` class).

There is `SmartPhone` class which can access the `receiveInfo` method of the `Laptop` class and the `printNotification` method of the `MobileGadget` class.

```
package org.hyperskill.bluetooth;

public class SmartPhone extends MobileGadget {

    private Laptop connectedLaptop;

    public SmartPhone() {
        this.connectedLaptop = new Laptop();
    }


    private void sendInfoToLaptop(String data) {

        printNotification("Sending data to laptop : "  + data);

        connectedLaptop.receiveInfo(data);

    }

}
```

The `SmartWatch` class has a private method `countHeartRate`, which is not available from other classes (even from a “brother” class `SmartPhone`). It also uses the `Laptop`'s method of receiving data and a parent's method to print the notification:

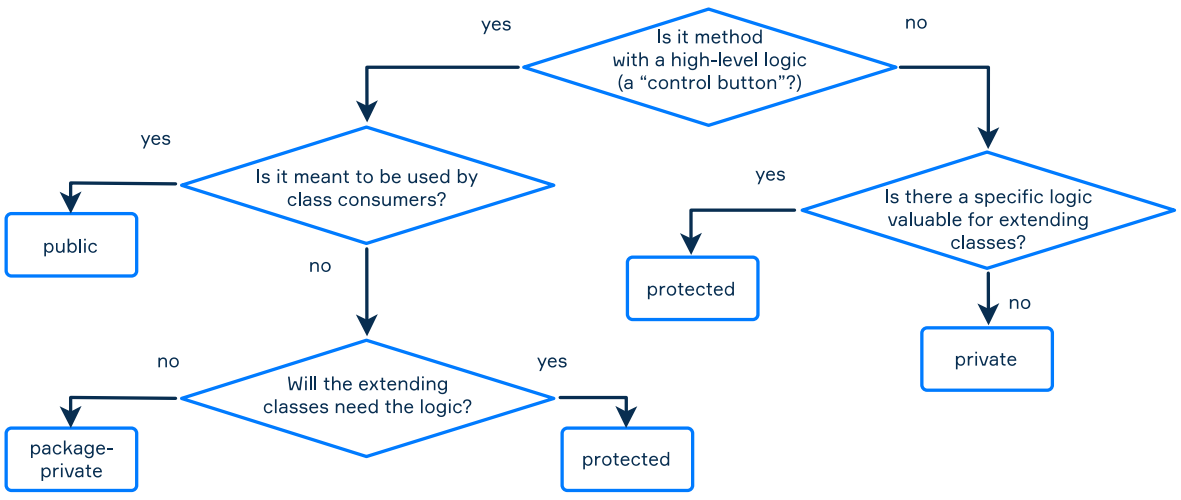
```
package org.hyperskill.bluetooth;

public class SmartWatch extends MobileGadget {

    private int avgHeartRate;
    private Laptop connectedLaptop;

    public SmartWatch() {
        this.avgHeartRate = 75;

        this.connectedLaptop = new Laptop();

    }



    private int countHeartRate() {

        System.out.println("Counting heart rate");

        return avgHeartRate;

    }



    private void sendInfoToLaptop(String data) {

        printNotification("Sending data to laptop : "  + data);

        connectedLaptop.receiveInfo(data);

    }

}
```

We hope you understand all modifiers clearly now!

The complete code example is available on GitHub. It has a slightly different package structure which is closer to a real project. You can navigate it in the GitHub web interface. You can copy this code and try to change it to better understand the example.

§4. Let's recap

Now, it's time to put all the access modifiers together:



The scheme is the same as it was earlier, but the questions now are specified with regard to inheritance.

Report a typo

357 users liked this theory. **28** didn't like it. **What about you?**

Start practicing

Comments (13) Hints (1) Useful links (0) Show discussion