TASK

Object Orienting Programming — Inheritance

Visit our website

Introduction

Welcome to the Inheritance Task!

Inheritance is quite a complicated concept to grasp. However, it is an important feature of OOP that you will need to understand to be a proficient programmer. It is a useful feature used to make your code neater and more logical.

Get in touch

Connect for support

Remember that with our courses, you're not alone! You can contact your mentor to get support on any aspect of your course.

The best way to get help is to login to **www.hyperiondev.com/portal** to start a chat with your mentor. You can also schedule a call or get support via email.

Your mentor is happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

What is inheritance?

Inheritance is one of the features of Object-Oriented Programming (OOP). Inheritance allows a class to use (or inherit) the properties and methods of another class. In other words, the **derived class** inherits the states and behaviors from the **base class**. The derived class is also called the subclass, and the base class is also known as the **superclass**. The derived class can add its own additional variables and methods. These additional variables and methods differentiate the derived class from the base class.

Inheritance is a compile-time mechanism. A superclass can have any number of subclasses. But a subclass can have only one superclass. This is because Java does not support **multiple inheritance**. The superclass and subclass have an "is-a" relationship between them.

To help us understand inheritance, we are going to consider an example using animals. Consider two animals – a lion and a cheetah. Although these are not the same animal, they are both animals, so they have certain things in common. Let's say that all animals have four legs and a head and teeth. Because all animals have these features, we can define a lion and a cheetah as animals.

Now let's put this in a programming context. If we had to create a class for a lion and one for a cheetah, we would find that we would be duplicating a lot of the same methods and fields in the two classes because the two animals are similar and therefore have features in common. Programmers are lazy! So we always want to try to do things as efficiently as possible. Therefore, we minimise the number of duplicates we create. To do this in this example, we could make use of inheritance. Start by creating a class for animals in which we defined all the common methods and fields for the two animals. Create another two classes – one for the lion and one for the cheetah. These would both inherit all the methods and fields from the superclass – the animal class. This means the lion and cheetah classes would both have all the properties that the animal class has as well as others, which would be specific to that class. Think of the lion and cheetah class as subclasses of the animal class. They are derived from the animal class.

Inheritance — hands on!

Let's create the scenario above. Create an animal class with variables common to both the lion and the cheetah, e.g. number of teeth, spots(Boolean), weight. We will now create the lion class. An example animal class has been included in the task folder in case you get confused.

It is, however, advised that you create one on your own to practise your skills. The *animal* class is very similar to previous classes you have created, so there should be nothing new there. The *lion* class will seem quite different from any other class you have created. Do not worry, you will understand everything shortly.

Below is the initial setup of the lion class:

```
public class Lion extends Animal{

    public Lion(int numTeeth, boolean spots, int weight) {
        super(numTeeth, spots, weight);
    }

}
```

You will notice many unusual things here. The first will be the word *'extends'*. This is a Java keyword which tells Java that you are using inheritance. It also tells Java that the class name which follows this keyword is the class you will be inheriting from – in this case it is the *'Animal'* class.

The next thing you will notice is the constructor. If the class you are inheriting from has a constructor which requires parameters, these parameters have to be passed into the subclass. The superclass object must be constructed using these parameters.

You will see that the keyword *'super'* is used to denote the superclass. The constructor of the superclass is used here without defining a specific object for it. This is a special case because it is being used in inheritance and therefore it will not cause an error. When accessing methods and variables of the superclass, the *'super'* keyword will be used along with the dot operator.

You can now add anything to this class that is specific to a *'lion'*. It will have those features in addition to those which are specific to all animals.

Please note that when you use a field or a method of the superclass, it will not change for all instances of the superclass. It will only change for that particular instance. Inheritance is not the same as using static variables or methods.

Compulsory Task 1

Follow these steps:

- Modify the existing **Animal.java** file for this task.
- Using the Lion class template, as shown in this pdf file, expand the class to have features specific to a lion:
    - Add a field for lion *type* (cub, male, female).
    - Add a method in this class which sets the lion type based on its weight (note that the weight is a derived field from the superclass).
    - If the weight is less than 80kg, it's type should be a cub. If less than 120kg, it should be female. If greater than 120kg, it is a male.
    - Include a method that will print out a description of a lion object.
- Compile, save and run your file.

Compulsory Task 2

Follow these steps:

- Modify the existing **Animal.java** file for this task.
- Create a class called *'Cheetah'* that:
    - Inherits from the *Animal* class.
    - Makes use of at least one static field which needs to have a static setter and getter.
    - Contains a constructor.
    - Contains a *toString()* method.
    - Has an array as one of its fields.
- Create an application class. Within the main method, create a *Cheetah* object and print it out the details that describe this object.
- Compile, save and run your file.

Rate us

Share your thoughts

Hyperion strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think the content of this task, or this course as a whole, can be improved or think we've done a good job?

Click here to share your thoughts anonymously.