Hyperion Development

Task: Introduction to Object Orientated Programming

www.hyperiondev.com

Introduction

What is Object Oriented Programming (OOP)?

OOP is a fundamental style of programming for developing larger pieces of software. Up until now, the programs you have written are simple enough to be run from just one file. In the real world of software development, multiple programmers work on large projects that may have hundreds of different files of code that implement the functionality of the project.

Your first step to building more complex programs is understanding OOP. This may be the first truly abstract concept you encounter in programming, but don't worry - practice will show you that once you get past the terminology, OOP is very simple.

Why OOP?

Imagine we want to build a program for a university. This program has a database of students, their information, and their marks. We need to perform computations on this data, such as finding the average grade of a particular student. Here are some observations about the above problem:

- A university will have many students that have the same information stored in the database, for example, age, name, and gender. How can we represent this information in code?
- We need to write code to find the average of a student's grades by simply summing their grades for different subjects, and dividing by the number of subjects taken. How can we only define this code once and reuse it for many students?

OOP is the solution to the above problems, and indeed many real world implementations of the above systems will use OOP.

Learning to do it the Java way!
You've had experience in dealing with OOP in the python section, so let's take a look at how to implement the same concepts in Java.

Start by creating a new project in Eclipse and call it 'Human'. Follow the same instructions as we did in the setting up Eclipse task. Name this class 'Human' as well. You should now have a class with a main method. This class will be used to define a 'Human' object. Your class should look similar to the one below:

Human.java

```
1
2  public class Human {
3
4      public static void main(String[] args) {
5          // TODO Auto-generated method stub
6
7      }
8
9  }
10
```

You will notice the words 'public class Human' on line 2. These words define the class and call it 'Human'.

N.B. The words in purple are keywords used by Java and have a specific meaning and use. You can not use these words as variable or method names.

Whenever declaring a class, the name of the class must be preceded by the keyword 'class'. This is to let Java know that you are declaring a class.

Now within this class you can declare variables and methods which define your class. When declaring variables within a class it is good practice to make them private variables. This is due to variable access levels which will be discussed in more detail in a later task. To do this, type the Java keyword 'private' before the datatype of the variable you are declaring.

All this means is that the variable can not be directly accessed from outside the class and can only be accessed through public methods of the class. Don't let this confuse you now. It will be explained further in a later task.

Inside your class (but outside your main method) declare four variables. One for each of the following: First Name, Surname, Gender and Age. Make sure all the variables are private and have relevant data types. Your code should look similar to the following:

```
1
2  public class Human {
3      private String firstname;
4      private String surname;
5      private String gender;
6      private int age;
7
8      public static void main(String[] args) {
9          // TODO Auto-generated method stub
10
11     }
12
13 }
14
```

You have now declared a few defining fields of your human object. All humans should have a first name, surname, age and gender so any person you create from this class will have those characteristics.

As explained earlier, these fields are private which means that they can not be directly accessed. This means we have to create methods to be able to access and change these variables. These methods are known as setters and getters, or more formally, as **Mutators** and **Accessors** respectively.

Mutator methods are used to change private variables within a class. They almost always take a parameter (possibly more than one) and they change the relevant variable based on the parameter(s). These methods don't return anything so their return type is 'void', but they do take parameters so keep that in mind when creating the method.

Accessor methods generally just return the relevant variable but sometimes they manipulate the way in which it is returned. For example, you may want to return the name of a person in a particular format i.e. in capital letters or with the surname first. These methods also always return something so when creating this method keep that in mind when choosing your return type.

You may be wondering why accessor and mutator methods are useful. The most important reason is because it gives you control over how variables within your class are interacted with. Keep in mind that your class can be used by other people and, if this is the case, allowing them to change variables directly could cause other functions in your class to not work as expected.

Secondly, as with all methods, accessors and mutators allow for the reusability of code. If the names of all humans, for example, need to start with a capital letter then this formatting can be done within the setter or getter instead of multiple times for each human.

There are certain naming conventions when creating setters and getters. For setters it is in the format 'setVariableName' and for getters it is in the format 'getVariableName'. Below are examples of setters and getters for the First Name field we created earlier:

```
//Accessor method (Getter)
public String getFirstName(){
    return firstname;
}
//Mutator method (Setter)
public void setFirstName(String firstname){
    this.firstname = firstname;
}
```

Take note that these methods are public unlike our fields which were private. This gives access to the private variables, but only through these functions.

Create setters and getters for all the other fields we created earlier. Make sure you are careful with the return types particularly for the Age field. Just like the fields you created earlier, setters and getters must be created within your class but outside your main method.

We have learned about how to define fields, setters and getters but we have not yet learned how to create an actual object from the class we have made. Let's now put all this knowledge to use and start creating humans!

We will create multiple instances of this class to illustrate how you can make multiple objects of the same type. Think of classes as a data type. If you create a Human class, you can create objects of type Human instead of, for example, type String. Objects of the Human type are declared, instantiated and initialised in the following way:

```
Human John = new Human();
```

- The first 'Human' is used to state the data type of the object
- 'John' is the variable name of the object
- 'new' is a Java keyword which is used to instantiate an object. This is just the assignment of memory within RAM to hold that object.
- 'Human()' calls the constructor of the Human class. You don't need to worry about this now. Constructors will be covered in a later task.

Keep in mind, this is done in the 'main' method of your program which, in this case, is within your class. An object can also be declared and initialised on separate lines as follows:

```
Human John;
John = new Human();
```

You can create multiple objects of the Human class**. Do so now by creating a Human called Thabang.**

The methods of a class are accessed by using the dot operator on an object of that class. For example, we can set the first name of John like this:

```
John.setFirstName("John");
```

Getting the First Name can be done in a similar way. Play around with the setters and getters for each of the humans you created.

Instructions

First read example.java, open it using jGRASP (Right click the file and select 'Open with jGRASP").

- In this folder there is a file called example.java
- Open this folder using the JGRASP program. You can either do this by right clicking on the example.java file, going to the 'Open with' menu, and selecting JGrasp.exe. Alternatively, you can run the JGRASP program on your computer, go to the top left corner of the program, select File->Open and navigate to example.java on your hard drive and double click on it to open it.
- Once example.java is open in JGRASP please read all of its content very carefully. This will help you understand the basic structure of a Java program.
- There is a compulsory task at the end of the example.java document. The instructions of what to do to complete the task can be found here. You must complete this exercise to continue onto the next task of this course.

Compulsory Task 1

Follow these steps:

- Create a new file called OOP.java
- Create an Accessor method which returns the full name of the human in the format "John Smith".
- This can be accomplished in two ways. You can access the variables directly or you can make use of the accessor methods you created earlier. Try both.
- Compile, save and run your file.

Compulsory Task 2

Follow these steps:

- Modify you OOP.java program to do the following.
- Implement the Human class as demonstrated in the task document.
- Next, define two new Human objects, John and Thabang.
- For both objects, assign each a First name, Surname, Age, and Gender property.
- Print out each object with all information on the object displayed on one line as such: *John Smith Male 24*
- Compile, save and run your file.

Compulsory Task 3

Follow these steps:

- Modify you OOP.java program to do the following.
- Extend your Human class to have at least another 3 fields which are relevant to all humans. Try to make use of different data types.
- Ensure you create accessor and mutator methods for all the new fields.
- Try make your human as detailed as possible. (Can you see the usefulness of classes and OOP?).
- Using your new properties, assign further information on your two Human objects.
- Print out each object with all information on the object displayed on one line as such: *John Smith Male 24 Engineer Toyota*
- Compile, save and run your file.

Things to look out for

1. Make sure that you have installed and setup all programs correctly. You have setup **Dropbox** correctly if you are reading this, but **jGRASP or Java** may not be installed correctly.
2. If you are not using Windows, please ask your tutor for alternative instructions.

Still need help?

Just write your queries in your comments.txt file and your tutor will respond.

Task Statistics

Last update to task: 29/01/2016.
Authors: Jared Ping, Brandon Haschick and Tason Matthew.
Main trainer: Umar Randeree.
Task Feedback link: Hyperion Development Feedback.