**Working with classes in Java**

The best way of solving a complex problem is by using the rule of “**Divide and conquer**”. The complex problem is divided into smaller manageable portions and each portion solved separately. The individual solutions make up the whole solution to the complex problem.

When Object Oriented programming languages were developed, what the developers of these languages had in mind is solving the problems using “Divide and conquer” approach. The problem can be divided into smaller manageable portion called “**Objects**” in Object Oriented programming and each solved by writing a **Class** code for each defined object.

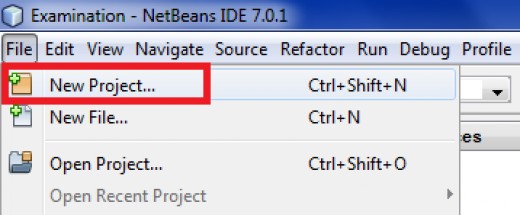
When we create a class in Object Oriented programming, we are writing code to do a particular job. That job might be to do with company employees, but not the company sales at the same time. We would write a separate class for the company sales.

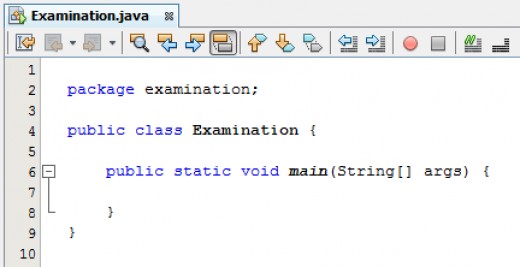
By creating different classes for specific jobs, the classes can be **re-usable**. For instance, we can re-use the employee class in the same project or in other projects.

That is one beauty aspect of Object Oriented programming in solving business problems, classes can be reusable throughout the project without necessarily creating new ones for every problem to be solved.

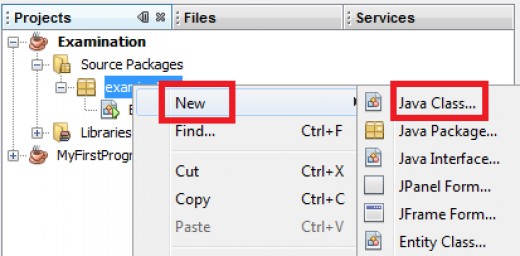
In this lesson, we shall learn how to create and use classes in Java programming language and how to call such classes into action. When we create a class, we are writing code to solve a problem. To use such a class and solve the intended problem, we have to create the object of that class. We did this in the previous lessons but we’ll do it in more details here.

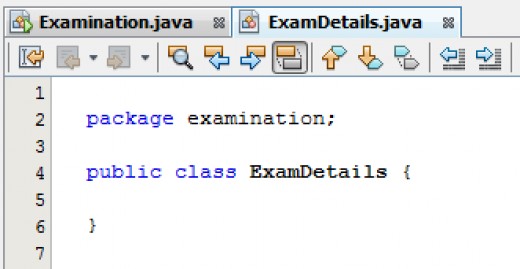
For this lesson, you’ll need to create a new project. On the Java NetBeans IDE click **File** >> **New Project**, call the project “**Examination**”, you can delete all the comments for now. See the images below;





In the project, we are going to add another class that will handle the examination data. So, right-click **Examination** package, select “**New**” and create another Java class, call it “**ExamDetails**”.



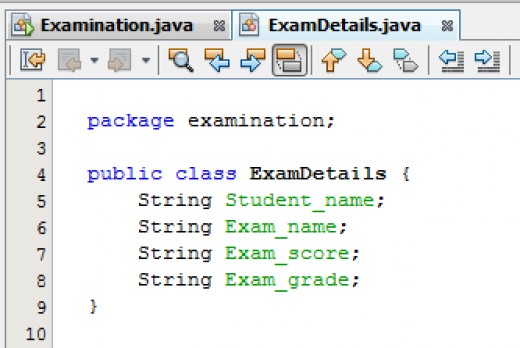


**Java global or field variables**

In the previous lessons, we have been dealing with variables that cannot be accessed outside their parent methods. Such variables are called local variables and are said to have a local scope.

For us to be able to access variables from any method, we need those variables to have a global scope. Such variables are called global, instance or field variables.

Let us create four global variables that will hold the examination data. These four variables will be available to all the methods that we’ll write in this class, and won't be local to any one method. Set the four global variables in the class **ExamDetails** as shown below;

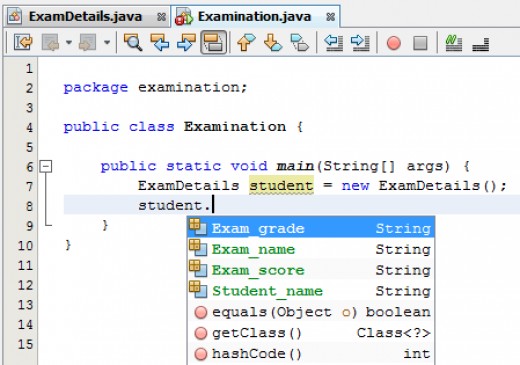


**Visibility of global variables in Java**

Just to test whether our four global variables are really global, we are going to test them in the main class. Go to the main class, **Examination**, and type the following line of code;

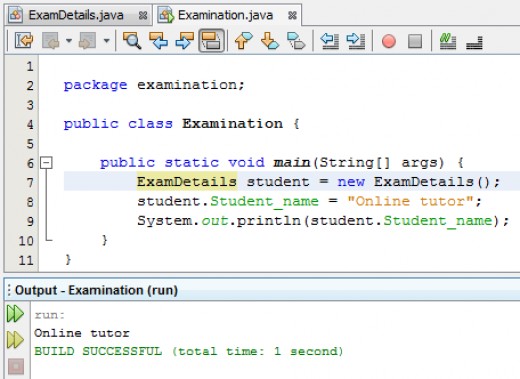
**ExamDetails student = new ExamDetails();**

In the above code, we are creating an object of the **ExamDetails** class in another method i.e. the main method. Now, attempt to access members (in this case, variables) of **ExamDetails** class in the main method and see if Java will recognize and list them.



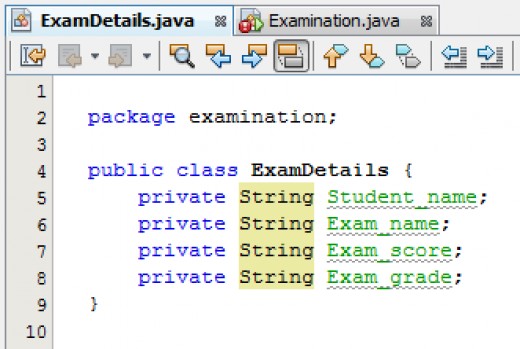
Notice that as soon as you type the dot after the object name (**student**) Java NetBeans display a popup list of methods and properties available for the object. Our four global variables are listed as properties since they are not methods. The fact that they are on the list means that they have global scope. If they had local scope they wouldn't be on the list.

Now we can even set values to the properties and display them on the console. Try the following code;

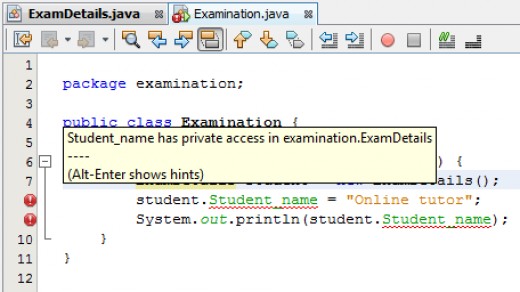


It is not a good programming practice to make global variables to have a wide global scope like we have done. By doing so, we can lose track of what values are being held in them, and it therefore makes debugging our code a lot harder and messy. It's considered good coding to narrow the scope of global variables.

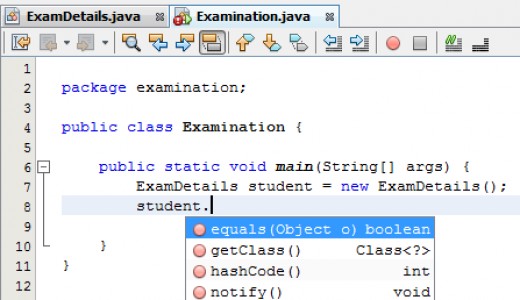
To make a global variable available only to a particular class, we add the Java keyword **private** just before the variable declaration. See the image below;



Including the keyword “**private**” will make the global variable only accessible inside **ExamDetails** class and not any other class. If you check on the main class now you’ll get a warning like shown below;



The warning states that the variable **Student\_name**is private to **ExamDetails** class and cannot be accessed from the **Examination** class. Let us delete the extra code and try our earlier test to see if Java still recognizes our global variable.



As you can see, Java failed to recognize our four variables because they no longer have a global scope and therefore can't be seen from the **Examination** class.