**Java class constructors and setter methods**

In the previous lesson, we have made an attempt to limit the scope of the global variables by use of **private** keyword. By doing so, the variables are no longer accessible from the **Examination** class.

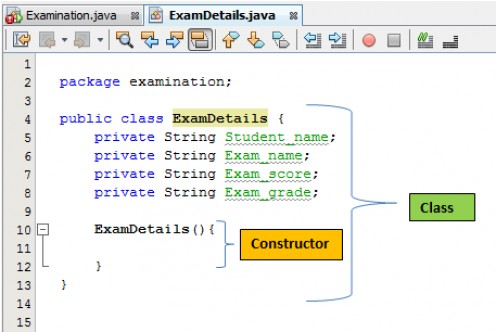
To solve this problem, we have to use **Java class constructors**. Revisit the classes lesson and have a look of what we learnt about and how we created the two Java classes.

Java constructor is a method that is used to set initial values for field variables. When the object is created, Java calls the constructor first.

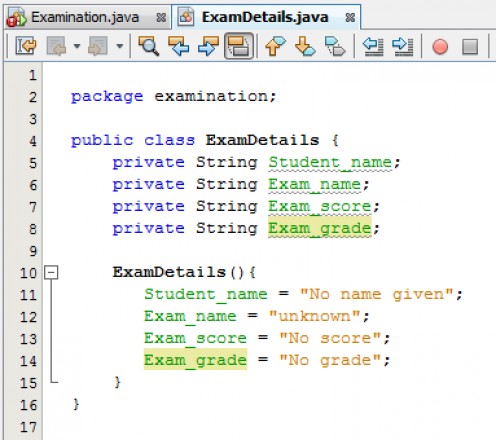
Any code contained within the constructor method is executed. The call to Java constructor happens automatically when a new object is created.

Constructor methods take the same name as the class name, they do not have return types nor return value but we can pass values to them.

Let us add a Java constructor to the **ExamDetails** class. The constructor will be called **ExamDetails**just like the class itself. See the image below:

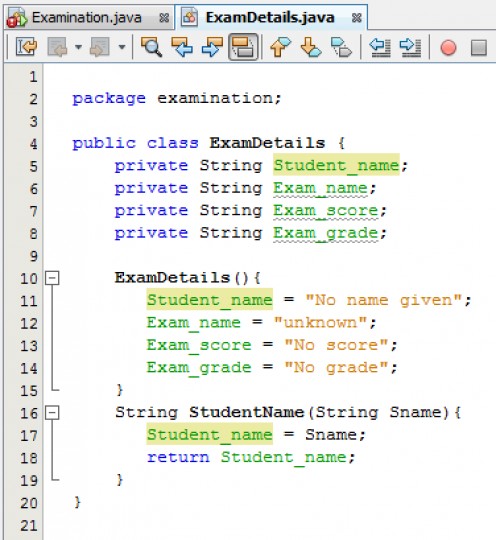


When we create a constructor, it is a good practice to set **default values** for the global variables. When the object of **ExamDetails** class is created, these values will be assigned to the variables. The default values will be replaced by the real values later in the program. See in the image below how we have set default values for the constructor method.



**How to access class variables**

To change the default values and set new ones, we use a method that sets some different values for them. Because this method basically sets new values other than the default ones it is usually called the **setter method**. We are going to add a setter method that will set the **Student\_name**. Add the following code into the class as shown in the image below:



From the above image, we have called the setter method **StudentName()**. All what this method does is to assign a new name value to the **Student\_name** variable and overwrite the default value.

The method sets a value for the **Student\_name** field and returns this field as a value. We have made the setter method simple enough as we are just beginning to understand Java classes otherwise we could have added more functionality to validate input errors such as formatting or case sensitivity.

**How to assign values using Java setter methods**

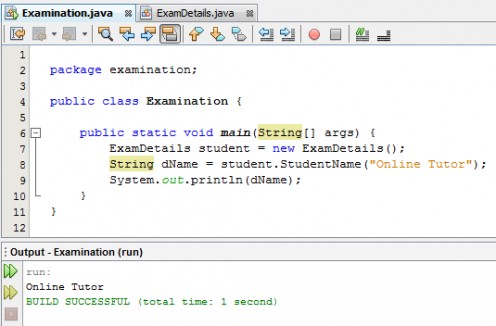
Now, let us see if we can assign a new name using the setter method and output on the console. Add the following line of code in the main class i.e. the **Examination** class.

**String dName = student.StudentName("Online Tutor");**

**System.out.println(dName);**

Notice that we are calling the **StudentName** method using the **student** object. We are handing over a value of "**Online Tutor**" to this method which will be the new value for the **Student\_name** field. The **Student\_name** value is then returned and stored in the variable **dName**.

The complete code for the**Examination** class should now look like shown below. Run the **Examination** class and you should be able to see the output.



Now that we have learnt how to create **constructor methods**, **setter methods** and how to assign values and access those values, we are going to write more useful methods that actually have more functionality such as decision making and error checking.