

## Set 17: Getters and Setters

**Skill 17.01: Describe the purpose of encapsulation**

**Skill 17.02: Interact with instance variables with getters and setters**

**Skill 17.01: Describe the purpose of encapsulation**


### Skill 17.01 Concepts


In our last lesson we saw how to give objects state through instance fields. For example we described our student data types below with the following instance variables: *name*, *gradeLevel*, *GPA*, and *hasScholarship*.

Student student1			Student student2		
data type	name	value	data type	name	value
String	name	Bart	String	name	Kyle
int	gradeLevel	10	int	gradeLevel	11
double	GPA	2.6	double	GPA	3.5
boolean	hasScholarship	false	boolean	hasScholarship	true

Recall that *instance* variables are variables that can be accessed by any member of the class. And for this reason, they were declared at the top of our Student class.

```
public class Student{  
    public String name;  
    public int gradeLevel;  
    public double GPA;  
    public boolean hasScholarship;  
  
    public Student(){  
    }  
}
```

 **Instance variables**

 **Constructor**

Declaring our instance variables with the public access modifier allowed us to modify their values from another class.

Student	StudentMaker
<pre> public class Student{      public String name;     public int gradeLevel;     public double GPA;     public boolean hasScholarship;      public Student(){      }  } </pre>	<pre> public class StudentMaker{      public static void main(String args[]){          Student student1 = new Student();         student1.name = "Bart";         student1.gradeLevel = 10;         student1.GPA = 2.6;         student1.hasScholarship = false;      }  } </pre>

In general, declaring instance variables as *public* is bad practice. Instead, it is standard practice to declare all instance variables as *private*. Declaring instance variables with the *private* access modifier ensures that they will be hidden from other classes. This process of *data hiding* is also referred to as *encapsulation*.

```

public class Student{

    private String name;
    private int gradeLevel;
    private double GPA;
    private boolean hasScholarship;

    public Student(){

    }

}

```

Instance variables

Constructor

**Instance variables** are used to define the object's state and should always be declared as private.

**Encapsulation** is the process of hiding data from other classes.

But, while declaring *instance variables* as private is good practice, it prevents us from modifying their values from other classes. The following code for example would fail because the variables *name*, *gradeLevel*, *GPA*, and *hasScholarship* are now *private*.

Student	StudentMaker
<pre> public class Student{      private String name;     private int gradeLevel;     private double GPA;     private boolean hasScholarship;      public Student(){      }  } </pre>	<pre> public class StudentMaker{      public static void main(String args[]){          Student student1 = new Student();         student1.name = "Bart";         student1.gradeLevel = 10;         student1.GPA = 2.6;         student1.hasScholarship = false;      }  } </pre>

To interact with our *private* instance variables requires the use of additional methods, otherwise referred to as *getters* and *setters*. Which just happens to be the topic of this lesson.

#### [Skill 17.01: Exercises 1 & 2](#)

#### Skill 17.02: Interact with instance variables with getters and setters

##### Skill 17.02 Concepts

Interacting with private instance variables requires the use of *getter* and *setter* methods. As their names imply, *getter* methods allow us to retrieve the value of an instance variable and *setter* methods allow us to modify the value of an instance variable.

Below is an example of *getter* method that retrieves the instance variable *name*

```

public String getName(){

    return name;

}

```

The first line, `public String getName()`, is the method signature. It gives the program some information about the method. each part of this line is described below,

- `public` means that other classes can access this method.
- The `String` keyword means the method returns a String data type
- `getName` is the name of the method.

The statement `return name;` is in the body of the method, which is defined by the curly braces.

Now that we have added the `getName()` method to our Student class, it also allows us access to the *name* instance variable associated with any Student object. We can do this by calling the method on the Student object we create, for example.

Student	StudentMaker
<pre> public class Student{      private String name;     private int gradeLevel;     private double GPA;     private boolean hasScholarship;      public Student(){      }      public String getName(){         return name;     }  } </pre>	<pre> public class StudentMaker{      public static void main(String args[]){          Student student1 = new Student();         System.out.println(student1.getName());     }  } </pre>
Output	
<pre> null </pre>	

But, in the above example, *null* is printed. Why, because we never set the name of the student. To do this requires a setter method.

Below is an example of *setter* method that sets the instance variable *name*

```

public void setName(String n){

    name = n;

}

```

Once again, the first line, `public void setName(String n)`, is the method signature. It gives the program some information about the method. each part of this line is described below,

- **public** means that other classes can access this method.
- The **void** keyword means the method does not return a data type.
- **setName** is the name of the method.
- **String n** is referred to as a parameter. A parameter is specified when the method is called.

The statement `name = n;` is in the body of the method, which is defined by the curly braces. When the method is called, whatever value the user specifies for the parameter will be assigned to the *name* instance variable.

The below example illustrates how *setters* and *getters* can be used to interact with the instance variable *name* in the Student class.

Student	StudentMaker
<pre> public class Student{      private String name;     private int gradeLevel;     private double GPA;     private boolean hasScholarship;      public Student(){      }      public String getName(){         return name;     }      public void setName(String n){         name = n;     }  } </pre>	<pre> public class StudentMaker{      public static void main(String args[]){          Student student1 = new Student();         student1.setName("Bart");         System.out.println(student1.getName());      }  } </pre>
Output	
Bart	

We learned previously that we can also set the initial values of state variables in the constructor.

This can be done using parameter values specified by the user or by using values specified by the program itself. Below is an example.

```

public class Student{

    private String name;
    private int gradeLevel;
    private double GPA;
    private boolean hasScholarship;

    public Student(String n, int g, double gpa){
        name = n;
        gradeLevel = g;
        GPA = gpa;
        hasScholarship = false;
    }

    /* Other methods not shown */
}

```

In the above example, the values of the instance variables *name*, *gradeLevel*, and *GPA* are specified by the user as parameters. However the value of *hasScholarship* is false for all new Student objects.

[Skill 17.02: Exercises 1 thru 3](#)