Set 18: Adding Functionality with Methods

Skill 18.01: Interpret how objects are referenced in memory

Skill 18.01 Concepts

In our last lesson we learned about *instance variables*, that is variables that are declared inside a class, but outside of a code block.

```
public class Student{

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

    public Student(){

    }
}
```

We also learned that whenever an object of is created, so too are the instance variables associated with the object.

Student

```
public class Student{

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

    public Student(){

    }
}
```

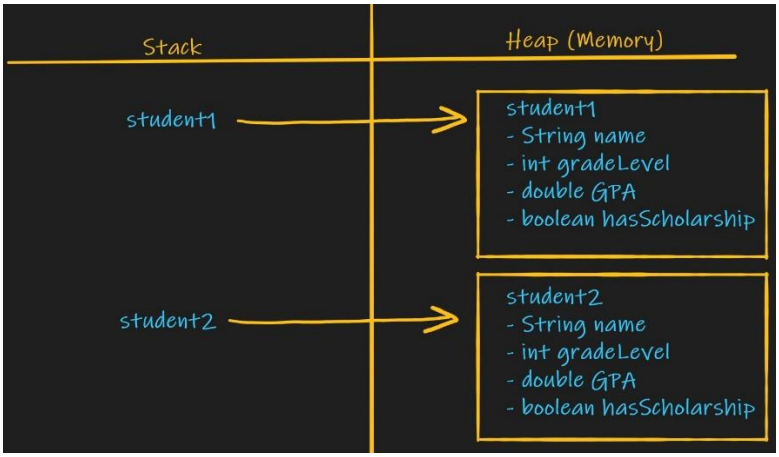
StudentMaker

```
public class StudentMaker{

    public static void main(String args[]){

        Student student1 = new Student();
        Student student2 = new Student();
    }
}
```

When the two students above are created in the *StudentMaker* class for example, memory is set aside for the instance variables.



To assign values to the *instance* variables, *name*, *gradeLevel*, *GPA*, and *hasScholarship*, requires the use of *setter* methods.

Student

```
public class Student{

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

    public Student(){

    }

    public void setName(String n){

        name = n;

    }

    public void setGradeLevel(int g){

        gradeLevel = g;

    }
    public void setGPA(double gpa){

        GPA = gpa;

    }
    public void setHasScholarship(boolean s){

        hasScholarship = s;

    }
}
```

StudentMaker

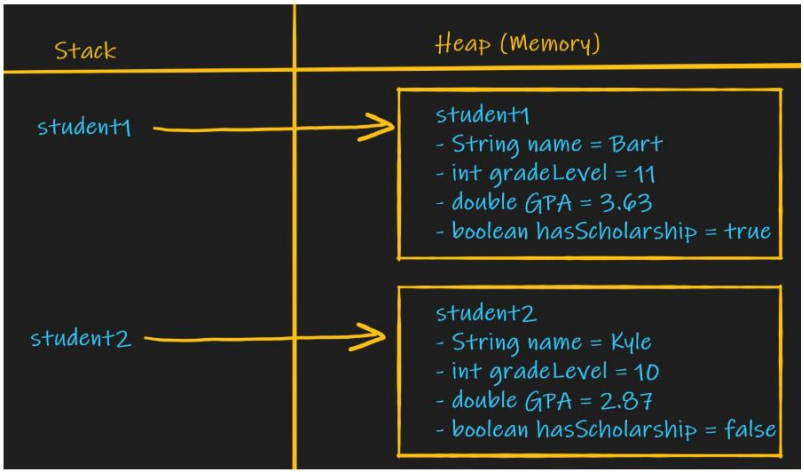
```
public class StudentMaker{

    public static void main(String args[]){

        Student student1 = new Student();
        student1.setName("Bart");
        student1.setGradeLevel(11);
        student1.setGPA(3.63);
        student1.setHasScholarship(true);


        Student student2 = new Student();
        student2.setName("Kyle");
        student2.setGradeLevel(10);
        student2.setGPA(2.87);
        student2.setHasScholarship(false);
}
```

When the code above in the *StudentMaker* class is executed, what is stored in memory now appears as follows.



Now that we have assigned values to our *instance* variables with *setter* methods, we can retrieve their values with *getter* methods.

Student

```
public class Student{

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

    public Student(){

    }

    public void setName(String n){

        name = n;

    }

    public void setGradeLevel(int g){

        gradeLevel = g;

    }
    public void setGPA(double gpa){

        GPA = gpa;

    }
    public void
setHasScholarship(boolean s){

        hasScholarship = s;

    }
}
```

StudentMaker

```
public class StudentMaker{

    public static void main(String args[]){

        Student student1 = new Student();
        student1.setName("Bart");
        student1.setGradeLevel(11);
        student1.setGPA(3.63);
        student1.setHasScholarship(true);


        Student student2 = new Student();
        student2.setName("Kyle");
        student2.setGradeLevel(10);
        student2.setGPA(2.87);
        student2.setHasScholarship(false);


        System.out.println(student1.getName());
        System.out.println(student1.getGradeLevel());
        System.out.println(student1.getGPA();
        System.out.println(student1.getHasScholarship());
}
```

Output

```
Bart
11
3.63
true
```

[Skill 18.01: Exercise 1]

Skill 18.02: Add functionality to a class with methods

Skill 18.02 Concepts

A method is a task that an object of a class performs. The *setters* and *getters* we wrote above are all examples of methods because they are tasked with setting and getting the values of different instance variables.

More functionalities can be added to our class, however, with additional methods. For example, what if we wanted a method that calculated the year a student should graduate, or another method, that calculated a student's weighted GPA? All this can be done by writing additional methods.

Just as we saw with our *setters* and *getters*, all methods are composed of two basic parts: a signature and a body.

```
public void setName(String n){

     name = n;

}
```

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 that there is no specific output from the method. That is, nothing is returned.
- setName is the name of the method.
- String n is a parameter that is passed to the body of the method

The body of the method is contained inside the curly brackets that follow the signature. What is contained within these brackets defines the functionality of the method.

Now, let's consider a method that computes the year a student should graduate.

```
public int getGradYear(){

   int gradYear = 0;
   int year = YearMonth.now().getYear();
   int month = YearMonth.now().getMonthValue();
       if(month>=6){
           gradYear = 12 - gradeLevel + year + 1;
       }else{
           gradYear = 12 - gradeLevel + year;
       }
   return gradYear;
}
```

In the above example the term *public* in the signature indicates that the method is visible by other classes in the same project. The term *int* indicates that the method *returns* an integer data type. If no data type was returned, int would be replaced with *void.*

Implementing this method is done in the same way we implemented our *getters* and *setters*.

Student

```
public class Student{
 private String name;
 private int gradeLevel;
 private double GPA;
 private boolean hasScholarship;

 public Student(){     }

 public void setName(String n){
   name = n;
 }

 public void setGradeLevel(int g){
   gradeLevel = g;
 }

 public void setGPA(double gpa){
   GPA = gpa;
 }

 public void setHasScholarship(boolean s){
   hasScholarship = s;
 }

 public String getName(){
   return name;
 }

 public int getGradeLevel(){
   return gradeLevel;
 }

 public double getGPA(){
   return GPA;
 }

 public boolean getHasScholarship(){
   return hasScholarship;
 }

 public int getGradYear(){
   int gradYear = 0;
   int year = YearMonth.now().getYear();
   int month =
YearMonth.now().getMonthValue();
     if(month>=6){
       gradYear = 12 - gradeLevel + year - 1;
     }else{
       gradYear = 12 - gradeLevel + year;
     }
   return gradYear;
 }
}
```

StudentMaker

```
public class StudentMaker{

    public static void main(String args[]){

    Student student1 = new Student();
    student1.setName("Bart");
    student1.setGradeLevel(11);
    student1.setGPA(3.63);
    student1.setHasScholarship(true);


    System.out.println(student1.getName());
    System.out.println(student1.getGradeLevel());
    System.out.println(student1.getGPA();
    System.out.println(student1.getHasScholarship());
    System.out.println(student1.getGradYear());
    }
}
```

Output

```
Bart
11
3.63
```

```
true
2022
```

[Skill 18.02: Exercise 1]

Skill 18.03: Interpret the *return* keyword

Skill 18.03 Concepts

As aforementioned, methods declared as *void* type do not *return* a value. As a review, consider the *getSum()* method below.

```
public class SumNums{

    private int num1, num2, sum;

    public SumNums(int a, int b){

        num1 = a;
        num2 = b;
    }
    public void getSum(){

        sum = num1 + num2;
    }
}
```

- public means that other classes can access this method.
- The void keyword means that there is no specific output from the method. That is, nothing is returned.
- getSum is the name of the method.

Because, the *getSum()* method is a *void* type method, nothing is returned. In other words, the following code would produce an error,

SumNums	**SumNumsDriver**

`public class SumNums {`
 	
`    private int num1, num2, sum;`
 	
`    public SumNums(int a, int b) {`
`        num1 = a;`
`        num2 = b;`
`    }`
 	
`    public void getSum() {`
`        sum = num1 + num2;`
`    }`
`}`
	`public class SumNumsDriver {`
	`   public static void main(String args[]){`
	
	`        SumNums s = new SumNums(1, 2);`
	`        System.out.println(s.getSum());// ERROR`
	`   }`
	`}`

Output	
Syntax error	

Fixing the error requires that we define the return type of the *getSum()* method. This is illustrated below,

```
public class SumNums{

    private int num1, num2, sum;

    public SumNums(int a, int b){

        num1 = a;
        num2 = b;
    }
    public int getSum(){

        sum = num1 + num2;
        return sum;
    }
}
```

- public means that other classes can access this method. We will learn more about that later.
- The int keyword means that the method returns an *int* data type..
- getSum is the name of the method.
- return sum returns an *int* data type.

Now that a return type has been defined, we can print the value associated with the method. The code shown below will now run without an error.

SumNums

```
public class SumNums {

    private int num1, num2, sum;

    public SumNums(int a, int b) {
        num1 = a;
        num2 = b;
    }

    public int getSum() {
        sum = num1 + num2;
        return sum;
    }
}
```

SumNumsDriver

```
public class SumNumsDriver {
   public static void main(String args[]){

        SumNums s = new SumNums(1, 2);
        System.out.println(s.getSum());// ERROR
   }
}
```

Output

3

A *return* statement can be used at various places in a method but we need to ensure that it is the last statement to get executed in a method. For example,

```
public class SumNums{

        private int num1, num2, sum;

        public SumNums(int a, int b){

                num1 = a;
                num2 = b;
        }
        public String getSum(){

                sum = num1 + num2;

                if(sum >= 18)
                        return "I'm an adult!";

                return "Not old enough!";
        }
}
```

If the condition (sum >= 18) is true, then *return "I'm an adult!";* executes and the flow of the program *returns* the String, *"I'm an adult!"*, to the method that called it,

SumNums

```
public class SumNums{

    private int num1, num2, sum;

    public SumNums(int a, int b) {

        num1 = a;
        num2 = b;
    }

    public String getSum() {

        sum = num1 + num2;

        if (sum >= 18)
            return "I'm an adult!";

        return "Not old enough!";
    }
}
```

SumNumsDriver

```
public class SumNumsDriver {
    public static void main(String args[]) {

        SumNums s = new SumNums(10, 9);
        System.out.println(s.getSum());

    }
}
```

Output

```
I'm an adult
```

- For a method with a **void** return type, a return statement is optional.
- For a method with any other return type, it **MUST** return a piece of data of the correct type.
- Methods end as soon as they reach a return statement, even if there is more code.

Skill 18.03: Exercise 1

Skill 18.04 Identify the scope of a variable

Skill 18.04 Concepts

Scope refers to the extent a variable within a program can be accessed. The *scope* of a variable is determined by the curly brackets (also called domain) within which it was declared.

Recall that *instance* variables are variables declared at the top of the class, because they are part of the curly brackets that define the class, they are said to be part of the class's scope. That is, they are accessible by any member of the class.

Variables, on the other hand declared within a constructor or a method, are confined to the curly brackets that define them. Such variables are only within the scope of the constructor or method within which they were declared and cannot be accessed by other members outside of the curly brackets

Consider the *SavingsAccount* class below,

```
public class SavingsAccount{

        public String name;
        public double balance;

        public SavingsAccount(String n, double b){

                name = n
                balance = b
        }
        public void checkBalance(){

                String message = "Your balance is $" + balance;
                System.out.println(message);
        }
}
```

The variable *message*, which is declared and initialized inside of *checkBalance()*, cannot be used inside any other method. It only exists within the scope of the *checkBalance()* method. However, *name* and *balance*, which are declared at the top of the class, can be used inside all methods in the class, since they are in the scope of the whole class.

Skill 18.04: Exercise 1

Skill 18.05 Apply *public* and *private* access modifiers to manage data members

Skill 18.05 Concepts

The *public* access modifier has the widest scope among all other access modifiers. Classes, methods, or data members which are declared as public are accessible from everywhere in the program, with no restrictions.

While declaring some parts of our program as *public* is necessary, it creates enormous security issues, and the potential of unintentionally modifying a portion of our program. These issues can be solved with the *private* access modifier, which is specified using the keyword *private*.

```java
public class SumNums {

    private int num1, num2, sum;

    public SumNums(int a, int b) {
        num1 = a;
        num2 = b;
    }

    public int getSum() {
        return num1 + num2;
    }

    public void setNums(int a, int b) {
        num1 = a;
        num2 = b;
    }
}
```

Notice, in the code above, the instance variables *num1*, *num2*, and *sum* are declared as *private*. Declaring instance variables as *private* is always a good practice. If accessing or changing the instance variables is necessary, *getter* and *setter* methods should be used. While the above code illustrates this practice, additional *getter* and *setter* methods could be implemented to provide more control over the program.

```java
public class SumNums {

    private int num1, num2, sum;

    public SumNums(int a, int b) {
        num1 = a;
        num2 = b;
    }

    public int getSum() {
        return num1 + num2;
    }

    public int getNum1() {
        return num1;
    }

    public int getNum2() {
        return num2;
    }

    public void setNum1(int a) {
        num1 = a;
    }

    public void setNum2(int b) {
        num2 = b;
    }
}
```

Instance variables should always be declared as **private**. Public **getter** and **setter** methods should be used to retrieve and modify instance variables.

Skill 18.05: Exercise 1