The `public` and `private` Keywords

In Java, the keywords `public` and `private` define the access of classes, instance variables, constructors, and methods.

`private` restricts access to only the class that declared the given structure while `public` allows for access from any class. We'll see more accessor keywords in future lessons. For now, we'll focus on these two, as they're the most used.

#Encapsulation

Encapsulation is a technique used to keep implementation details hidden from other classes. Its aim is to create small reusable bundles of logic. The `private` keyword in Java helps us achieve encapsulation.

#The `private` Keyword

In Java, instance variables are encapsulated by using the `private` keyword. This prevents other classes from directly accessing these variables:

```
public class CheckingAccount {
// Three private instance variables:
private String name;
private double balance;
private int id;
}
```

Preventing access means that these values can neither be read nor altered by outside objects. This helps protect them from accidental mutation (changing), but also aids us in keeping data from being seen when it's not needed.

#Accessor Methods

Since the instance variables in a class are usually marked as `private` to the class that made them, programmers provide **public *methods*** that allow safe access to the instance variable values in a class. **Accessor methods** also called **get methods** or **getters**, provide a safeguarded way to get the value of each instance variable from outside of the class. In the next lesson, we will see **mutator methods**, also called **set methods** or **setters**, that provide a scripted way to change the values of the instance variables. For now, let's just talk about getting that data.

Java programmers write **get methods** for each instance variable according to the following syntax. Notice that the get method returns the instance variable's value, and it has a return type that is the same as the variable that it is returning:

```
class ExampleTemplate {

//Instance variable declaration
private typeOfVar varName;

// Accessor (getter) method template
public typeOfVar getVarName() {
return varName;
}

}
```

Below, there's an example of an accessor method called `getName()` for the `Student` class, which also demonstrates how to call `getName()` using a `Student` object. Notice that the signature **does not include the keyword** `static`, as most of our code has before. A static method cannot access **instance** variables since it is called on a class, not an object. A non-static method must be called on a specified object, as shown on line 15 with `s.getName()` where `s` is an instantiated object of the `Student` class.

```
class Student {

//Instance variable name
private String name;

/** getName() example
* @return name */
public String getName() {
return name;
}

public static void main(String[] args) {
// To call a get method, use objectName.getVarName()
Student s = new Student();
System.out.println("Name: " + s.getName() );
}
```

InfoWarningTip

Debugging note! Some common errors with methods that return values are:

- Forgetting a return type like `int` before the method name.
- Forgetting to use the `return` keyword to return a value at the end of the method.
- Forgetting to do something with the value returned from a method (like saving it into a variable or printing it out once it arrives in the requesting object).

Try the following code. Note that this IDE code window has two classes! The main method is in a separate **TesterClass**. It does not have access to the private instance variables in our other `Student` class. Note that when you use multiple classes in an IDE like Eclipse, you usually put them in separate files, giving the files the same name as the public class they represent. Sometimes though, you can put two classes in 1 file, as demonstrated here, but only one of the classes in the file can `public` and have a `main` method. If you'd like to dig deeper, you can view the fixed code above in the Java visualizer.

#Coding Exercise

Here is the code. Note that it has a bug! It tries to access the private instance variable email from outside the class Student in the Tester Class. Change the `main` method in the `Tester` class so that it uses the appropriate public accessor method (`get` method) to access the `email` value instead. Notice the two files in the IDE, `Student.java`, and `TesterClass.java`. You need to run `TesterClass.java` to test the code. To choose which file to run, click `v` arrow next to the Run button and select the appropriate file.

2

Run

Student.java TesterClass.java

```
/** Class Student keeps track of name, email, and id of a Student. */
public class Student {
   private String name;
   private String email;
   private int id;

   public Student(String initName, String initEmail, int initId) {
      name = initName;
      email = initEmail;
      id = initId;
   }

   // accessor methods - getters
   /** getName()  @return name */
   public String getName() {
      return name;
   }
   /** getEmail()  @return email */
   public String getEmail() {
      return email;
   }
   /** getName()  @return id */
   public int getId() {
      return id;
   }
}
 public class TesterClass {


   // main method for testing
   public static void main(String[] args) {
      Student s1 = new Student("Skyler", "skyler@sky.com", 123456);
      System.out.println("Name:" +  s1.getName());

      // Fix the bug here!
      System.out.println("Email:" +  s1.getEmail() );
      System.out.println("ID: " + s1.getId() );
   }
}
```

# `toString()`

Another common method that returns a value is the `toString()` method, which returns a String description of the object's instance variables. This method is called automatically to try to convert an object to a String when it is needed. One example of this is the `print` statement.

Here is the Student class again, but with a toString() method. Note that when we call `System.out.println(s1);` it will automatically call the `toString()` method to cast the object into a String. The `toString()` method will return a String that is then printed out. Watch how the control moves to the `toString()` method and then comes back to main in the Java visualizer here.

In the IDE below, adjust the `toString()` method in the Student class so it returns the `id`, `name` and `email` of the student in the following format: `id + ": " + name + ", " + email;`

Run

Student.java

```
public class Student {
   private String name;
   private String email;
   private int id;

   public Student(String initName, String initEmail, int initId) {
      name = initName;
      email = initEmail;
      id = initId;
   }

   // toString() method
   public String toString() {
     return id + ": " + name + ", " + email; // Replace the empty String with
the format specified in the instructions above.
   }

   public static void main(String[] args) {
      Student s1 = new Student("Skyler", "skyler@sky.com", 123456);
      System.out.println(s1);
   }

}
```

#Practice

Consider the following Party class. The `getNumOfPeople` method is intended to allow methods in other classes to access a `Party` object's `numOfPeople` instance variable value; however, it does not work as intended. Which of the following best explains why the `getNumOfPeople` method does NOT work as intended?

```
public class Party {

private int numOfPeople;

public Party(int num) {
numOfPeople = num;
}

private int getNumOfPeople() {
return numOfPeople;
}
}
```

The getNumOfPeople method should be declared as public.

The return type of the getNumOfPeople method should be void.

The getNumOfPeople method should have at least one parameter.

The variable numOfPeople is not declared inside the getNumOfPeople method.

The instance variable num should be returned instead of numOfPeople, which is local to the constructor.

Consider the following class definition. The class does not compile.

CopyCC#C++ClojureCSSDartGoHaskellHTMLJavaJavaScriptJSONJSXKotlinMarkdown PascalPerlPHPPlain TextPythonRRubyRustSchemeShellSQLSwiftTypescriptVB.NETVBScriptXMLYAML

```
public class Student {
private int id;
public getId() {
return id;
}
// Constructor not shown
}
```

The accessor method `getId` is intended to return the `id` of a `Student` object.

Which of the following best explains why the class does not compile?

The id instance variable should be public.

The getId method should be declared as private.

The getId method requires a parameter.

The return type of the getId method needs to be defined as void.

The return type of the getId method needs to be defined as int.

#Summary

- An accessor method allows other objects to obtain the value of instance variables or static variables.
- A non-void method returns a single value. Its header includes the return type in place of the keyword `void`.
- Accessor methods that return primitive types use “return by value,” where a copy of the value is returned.
- When the return expression is a reference to an object, a copy of that reference is returned, not a copy of the object.
- The `return` keyword returns the control flow to the point immediately following where the method or constructor was called.
- The `toString` method is an overridden method included in classes to describe a specific object. It generally includes what values are stored in the object's instance data. (More on overriding soon.)
- If `System.out.print` or `System.out.println` is passed to an object, that object’s `toString` method is called, and the returned string is printed.

#Extra Resources

If you'd like to hear another angle on encapsulation, feel free to watch the YouTube video below;

Open in new tab

https://youtu.be/eboNNUADeIc