Learning Objectives - Getters and Setters

- **Define the terms getter, setter, and data validation**
- **Demonstrate how to access private attributes with getters**
- **Demonstrate how to change private attributes with setters**
- **Demonstrate validating data with setters**

Getters

Getters

While all attributes are private, that does not mean that the user will not need to access the values of these attributes. A public method can be used to access an attribute. This type of method has a special name, a getter (also called an accessor). The whole purpose of a getter is to return an attribute. These methods get their name because the method always start with `get` followed by the attribute name.

```
//add class definitions below this line

class Phone {
  private String model;
  private int storage;
  private int megapixels;

  public Phone(String mo, int s, int me) {
    model = mo;
    storage = s;
    megapixels = me;
  }

  public String getModel() {
    return model;
  }
}

//add class definitions above this line
```

The method `getModel` is an example of a getter. They are very simple, straightforward methods that do only one thing — return a private attribute. Getters can be treated just as you would treat the attribute (except for changing its value).

```
    //add code below this line

    Phone myPhone = new Phone("iPhone", 256, 12);
    System.out.println(myPhone.getModel());
    System.out.println(myPhone.getModel().toUpperCase());
    System.out.println(myPhone.getModel().startsWith("b"));
    System.out.println(myPhone.getModel().indexOf("n"));

    //add code above this line
```

challenge

Try this variation:

Create getters for the `storage` and `megapixels` attributes.

▼ **Possible Solution**

```
public int getStorage() {
  return storage;
}

public int getMegapixels() {
  return megapixels;
}
```

Benefits of Getters

Using a getter is the same thing as accessing a public attribute. Why not make the attribute public? That would mean writing less code. Is that not a good thing? A public attribute makes no distinction between accessing its value and changing its value. If you can access it, you can change it (or vice versa). Using a getter with a private attribute makes this distinction clear; you can access the value, but you cannot change it.

```
    //add code below this line

    Phone myPhone = new Phone("iPhone", 256, 12);
    System.out.println(myPhone.getModel());
    myPhone.model = "Pixel 5";

    //add code above this line
```

The code above generates an error because an instance cannot alter a private attribute. Using a getter allows limited access to an attribute, which is preferable to the full access a public access modifier allows.

Setters

Setters

Setters are the compliment to getters in that they allow you to set the value of a private attribute. Setter methods are also called mutators. Use the `Phone` class from the previous page.

```
//add class definitions below this line

class Phone {
  private String model;
  private int storage;
  private int megapixels;

  public Phone(String mo, int s, int me) {
    model = mo;
    storage = s;
    megapixels = me;
  }

  public String getModel() {
    return model;
  }

  public int getStorage() {
    return storage;
  }

  public int getMegapixels() {
    return megapixels;
  }
}

//add class definitions above this line
```

Add the `setModel` method to the `Phone` class. As this is a setter method, start the name with `set` followed by the name of the attribute. Setters do not return anything, so their type is `void`. Finally, setters have a parameter — the new value for the attribute.

```
public void setModel(String newModel) {
  model = newModel;
}
```

Now that you are implementing both getters and setters, you should now be able to access and modify private attributes.

```
  //add code below this line

  Phone myPhone = new Phone("iPhone", 256, 12);
  System.out.println(myPhone.getModel());
  myPhone.setModel("Pixel 5");
  System.out.println(myPhone.getModel());

  //add code above this line
```

challenge

Try this variation:

Create setters for the `storage` and `megapixels` attributes.

▼ **Possible Solution**

```
public void setStorage(int newStorage) {
  storage = newStorage;
}

public void setMegapixels(int newMegapixels) {
  megapixels = newMegapixels;
}
```

Comparing Getters and Setters

Getters and setters have a lot in common. Their names are similar, they have the same number of lines of code, etc. However, getters and setters also differ in a few important ways. The table below highlights these similarities and differences.

Category	Getters	Setters
Has `public` keyword	X	X

Has `private` keyword	-	-
Has `return` statement	X	-
Has `void` type	-	X
Performs only 1 task	X	X
Has parameter	-	X

Data Validation

Java already has a type system which will flag errors when a boolean value is passed to a method that takes an integer. However, just because a method takes an integer does not mean that all integers are valid for that method. Data validation is the process of asking if this data is appropriate for its intended use. Take a look at the `Person` class.

```
//add class definitions below this line

class Person {
  private String name;
  private int age;

  public Person(String n, int a) {
    name = n;
    age = a;
  }

  public String getName() {
    return name;
  }

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

  public int getAge() {
    return age;
  }

  public void setAge(int newAge) {
    age = newAge;
  }
}

//add class definitions above this line
```

The `setAge` method will assign any value to the attribute `age` as long as the value is an integer. There are some integers which makes no sense when thought of as an age. The code sample below sets the age of `myPerson` to `-100`. `-100` is a valid integer, but it is not a valid age. This is why data validation is important. Java's compiler is not sufficient to catch all errors.

```
    //add code below this line


    Person myPerson = new Person("Calvin", 6);
    System.out.println(myPerson.getName() + " is " +
        myPerson.getAge() + " years old.");
    myPerson.setAge(-100);
    System.out.println(myPerson.getName() + " is " +
        myPerson.getAge() + " years old.");


    //add code above this line
```

Another benefit to using setters is that data validation can take place before the new value is assigned to the attribute. Modify `setAge` so that it will only update the `age` attribute if the new value is greater than or equal to 0. This way you can ensure that the `Person` object always has a valid age.

```
public void setAge(int newAge) {
  if (newAge >= 0) {
    age = newAge;
  }
}
```

challenge

Try these variations:

- Change the data validation for `setAge` method so that values over 200 are not valid.

▼ **Possible Solution**

Here is one possible solution.

```
    public void setAge(int newAge) {
      if (newAge >= 0 && newAge <= 200) {
        age = newAge;
      }
    }
```

- Add data validation to the `setName` method so that all strings with one or more characters are valid.

  ▼

  Possible Solution

  Here is one possible solution.

  ```
    public void setName(String newName) {
      if (newName != "") {
        name = newName;
      }
    }
  ```