Mutator Methods

Corresponding to each `get` method, programmers also provide a public **set method** to change the value of a private instance variable in a class. These are called **mutator methods** (or **setters/set** or **modifier** methods). They are **void methods** meaning that they do not return a value, but they do take a **parameter** which will become the new value for the instance variable. Here are some examples of how to write a `set` method for an instance variable:

```
class ExampleTemplate {
//Instance variable declaration
private typeOfVar varName;

// Mutator (setter) method template
public void setVarName(typeOfVar newValue) {
varName = newValue;
}
}
```

Here's an example of the `Student` class with a mutator method called `setName()`:

```
class Student {

//Instance variable name
private String name;

/** setName sets name to newName
* @param newName */
public void setName(String newName) {
name = newName;
}

public static void main(String[] args) {
// To call a set method, use objectName.setVar(newValue)
Student s = new Student();
s.setName("Ayanna");
}
}
```

Notice the difference between `set` (mutator) and `get` (accessor) methods in the following figure. Getters return an **instance variable's** value and have the same return type as the variable and no parameters. **Setters have a void return type and take a new value as a parameter to change the value of the instance variable.**

```
//instance variable declaration
private typeOfVar varName;
```

```
/** setVar sets varName to a newValue
  *  @param newValue
  */
public void setVarName(typeOfVar newValue)
{
      varName = newValue;
}
```

```
/** getVar() returns varNa
  *  @return varName
  */
public typeOfVar getVar
{
     return varName;
}
```

Comparison of set and get methods

#Why use Accessor and Mutator Methods?

While it is possible to write code without getters and setters by directly accessing and modifying the instance variables, it is generally considered good practice to use getters and setters for the reasons described below.

Here are some scenarios in which using getters (accessor methods) and setters (mutator methods) is recommended:

1. **Encapsulation and Data Hiding:** Getters and setters help achieve encapsulation by keeping the internal state of an object private and providing tightly-controlled access to it. By declaring the instance variables as `private` and using getters and setters, you can ensure that the object's state is accessed and modified only through defined methods. This helps maintain data integrity and prevents unwanted modifications from external sources.
2. **Access Control and Validation:** Getters and setters allow you to control access to fields and apply validation logic if necessary. For example, you can enforce certain conditions or constraints when setting a value with a setter method. This ensures that the values assigned to the fields are valid and consistent with the expected behaviour of the class.
3. **Flexibility and Maintainability:** By using getters and setters, you can modify the internal implementation of a class without affecting the external code that uses

the class. If you later decide to change the way a field is stored or retrieved, you can do so within the getter and setter methods without impacting the rest of the class/codebase.

4. **Object-Oriented Design and Best Practices:** Using getters and setters aligns with the principles of object-oriented design, such as encapsulation and promotes clean and modular code by separating the internal representation of an object from its external interface.

In short, **accessor and mutator methods provide a level of abstraction and encapsulation, making your code more robust, maintainable, and easier to understand.**

#Coding Exercise

Try the `Student` class below, which has `set` methods added this time. You'll need to fix one error to get the code to run. The main method is in a separate `Tester` class and does not have access to the private instance variables in the `Student` class. Change the `main` method to use a public mutator (`set` method) to access the value instead.

Fix the main method to include a call to the appropriate `set` method:

Run

TesterClass.java

```
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(s1);
      s1.setName("Skyler 2");

      // Main doesn't have access to email, use set method!
      s1.email = "skyler2@gmail.com";
      System.out.println(s1);
   }
 }

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;
    }
    // mutator methods - setters
    public void setName(String newName) {
      name = newName;
    }
    public void setEmail(String newEmail) {
      email = newEmail;
    }
    public void setId(int newId) {
      id = newId;
    }
    // accessor methods - getters
    public String getName() {
       return name;
    }
    public String getEmail() {
       return email;
    }
    public int getId() {
       return id;
    }
    public String toString() {
       return id + ": " + name + ", " + email;
    }
}
```

#Check your understanding

Consider the class `Party`, which keeps track of the number of people at the party.

```
public class Party {
//number of people at the party
private int numOfPeople;

/* Missing header of set method */
{
numOfPeople = people;
}
}
```

Which of the following method signatures could replace the missing header for the set method in the code above so that the method will work as intended?

public int getNum(int people)

public int setNum()

public int setNum(int people)

public void setNumOfPeople(int people)

public int setNumOfPeople(int p)

Options

accessor method

gets and returns the value of an instance variable

mutator method

sets the instance variable to a value in its parameter

constructor

initializes the instance variables to values

public

accessible from outside the class

private

accessible only inside the class

Mutator methods do not *have* to have a name with "set" in it, although most do. They can be any method that changes the value of an instance variable for a given object or a static variable in the class itself.

#Knowledge Check

Consider the following class definition.

```
public class Liquid {
private int currentTemp;

public Liquid(int temp) {
currentTemp = temp;
}

public void resetTemp() {
currentTemp = newTemp;
}
```

```
}
```

Which of the following best identifies the reason the class does not compile?

The constructor header does not have a return type.

The resetTemp method is missing a return type.

The constructor should not have a parameter.

The resetTemp method should have a parameter.

The instance variable currentTemp should be public instead of private.

In the `Party` class below, the `addPeople` method is intended to increase the value of the instance variable `numOfPeople` by the value of the parameter `additionalPeople`. The method does not work as intended.

```
public class Party {
private int numOfPeople;

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

public int addPeople(int additionalPeople) { // Line 8
numOfPeople += additionalPeople; // Line 10
}
}
```

Which of the following changes should be made so that the class definition compiles without error and the method `addPeople` works as intended?

Replace line 10 with numOfPeople = additionalPeople;

Replace line 10 with return additionalPeople;

Replace line 10 with additionalPeople += 3;

Replace line 8 with public addPeople(int additionalPeople)

Replace line 8 with public void addPeople(int additionalPeople)

#Summary

- A **void** method does not return a value. Its header contains the keyword `void` before the method name to denote this fact.

- A **mutator method** is often `void` and changes the values of instance variables or static variables.

#Extra Resources

Let's get another voice in the conversation. If you want to see a bit more about mutator methods (and accessors), check out Marta's explanation:

Open in new tab
https://youtu.be/kFsNB4vu8z4