Getter and Setters

In this lesson, you will learn how to read and write properties using getters and setters.

We'll cover the following

Introduction

Getters and **setters** are special methods that provide read and write access to an object's properties. Recall how we were able to retrieve and set the values of instance variables using the dot operator (`.`). Each instance variable has an implicit getter and setter which we have been using up until now. In this lesson, we will explore the getter and setter methods.

Getters

Getters are functions that are used to retrieve the values of an object's properties and are defined using the `get` keyword.

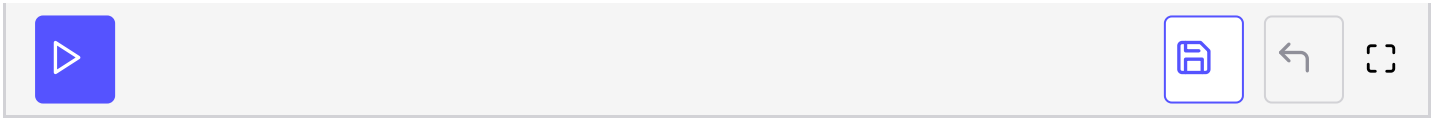
The syntax is as follows:

```
type get identifier
```

Let's look at a very simple example of `get`.

```
class Person{
  String name;
  String gender;
  int age;

  Person(this.name, this.gender, this.age);

  Person.newBorn(){
```

```
    this.age = 0;
  }


  // Getter function getting the value of name
  String get personName => name;

  walking() => print('$name is walking');
  talking() => print('$name is talking');
}

int main() {
  var firstPerson = Person("Sarah","Female",25);
  print(firstPerson.personName);
}
```



On **line 13**, we are creating a getter function which returns the value of `name` of the current instance. On **line 21**, we are calling the getter function and the output should display **Sarah**.

Setters

Setters are functions that are used to write the values of an object's properties and are defined using the `set` keyword.

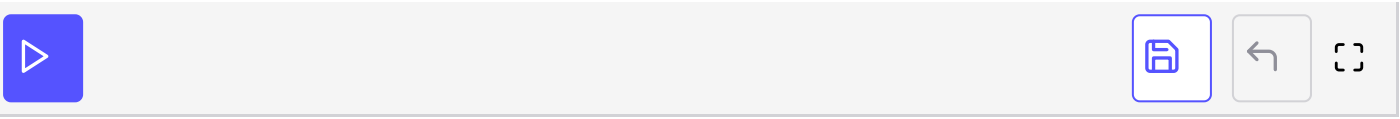
The syntax is as follows:

```
type set identifier(parameterList)
```

Let's expand on the example above.

```
class Person{
  String name;
  String gender;
  int age;

  String get personName => name;

  // Setter function for setting the value of age
  void set personAge(num val){
    if(val < 0){
      print("Age cannot be negative");
    } else {
      this.age = val;
    }
  }

  walking() => print('$name is walking');
  talking() => print('$name is talking');
```

```
}

int main() {

  var firstPerson = Person();
  firstPerson.personAge = -5;
  print(firstPerson.age);
}
```



From **line 9** to **line 15**, we are creating a setter function which sets the value for `age`. The setter has a condition which makes sure that the user does not input a negative age.

On **line 23**, we are setting the value of age of `firstPerson` using the `personAge` setter function.

> Getter functions and setter functions are not called the way a normal function is called, i.e., using parentheses (`()`) after the function name. They are called similar to the way instance variables are called, i.e., a dot operator (`.`) simply followed by the function name.

Learning by Example

Let's look at a slightly more complex problem.

The class `Figure` has **4** properties, `left`, `top`, `width`, and `height`. We are going to create two getter functions which will calculate the value of two new properties, `right` and `bottom`.

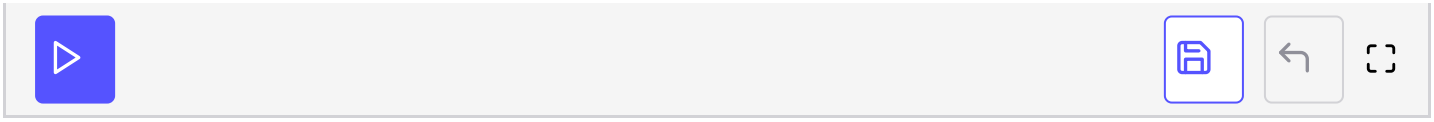
We will also create a setter function for `right`. The properties `right` and `left` are interdependent. This means that based on the value set for `right`, the value of `left` needs to be modified. This is being handled by the setter function.

Likewise, `bottom` and `height` are interdependent. Hence, the setter for `bottom` modifies the value of `height` according to the value of `bottom`.

Let's take a look at the code below.

```
class Figure {
  num left, top, width, height;

  Figure(this.left, this.top, this.width, this.height);
```

```
  Figure(this.left, this.top, this.width, this.height);

  // Define two calculated properties: right and bottom.
  num get right => left + width;
  set right(num value) => left = value - width;
  num get bottom => top + height;
  set bottom(num value) => top = value - height;
}

main() {
  var fig = Figure(3, 4, 20, 15);
  print(fig.left);
  print(fig.right);
  fig.right = 12;
  print(fig.left);
}
```



On **line 15**, the value of `left` is **3**. However, when we call the setter function `right` on **line 17**, the value of `left` is modified to **-8**. Which is being displayed on **line 18**.

Let’s move onto inheritance in the next lesson.