Objects of a Class

In this lesson, you will create your first object of a class and learn how to use its class members.

We'll cover the following

Instantiating a Class

Once a class has been defined, you can create objects from the class blueprint using the `new` keyword followed by a class identifier which is further followed by parenthesis (`()`).

> The `new` keyword became optional in Dart 2.

```
new classIdentifier()
```

OR

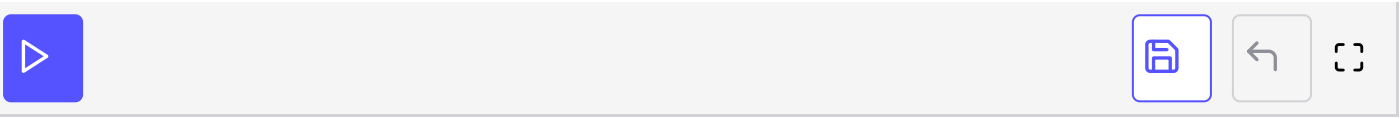
```
classIdentifier()
```

We usually don't create objects for the sake of just creating them, rather, we want to work with them in some way. For this reason, we assign the object to a variable. Let's instantiate our `Person` class.

For ease, the code for creating the class is also provided below.

```
class Person{
  String name;
  String gender;
  int age = 0;

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

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


int main() {
  // Creating an object of the Person class
  var firstPerson = Person();
}
```



Using Class Members

Now that we have our object, `firstPerson`, let's learn how to use instance variables and methods. In Dart, we use a dot (`.`) to refer to an instance variable or method.

`instanceOfObject.instanceVariable`

AND

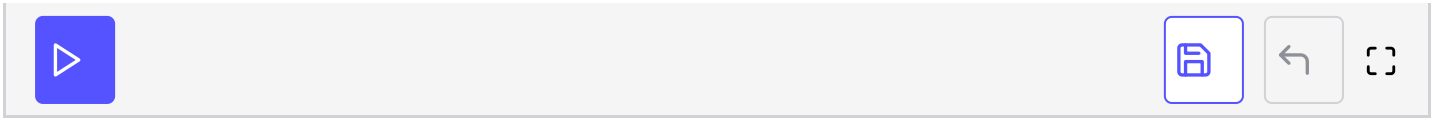
`instanceOfObject.method`

If you initialize an instance variable where it is declared, the value is set when the instance is created. So as soon as `firstPerson` was created, the value of `age` was set as **0**. Let's set the value of `name` and `gender` using the dot operator while also reassigning `age` a new value.

```
class Person{
  String name;
  String gender;
  int age = 0;

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

int main() {
  var firstPerson = Person();

  firstPerson.name = "Sarah";
  firstPerson.gender = "female";
  firstPerson.age = 25;

  print(firstPerson.name);
  print(firstPerson.gender);
  print(firstPerson.age);
}
```

When you call a method, you invoke it on an object; the method has access to that object's methods and instance variables. Let's call the `walking` and `talking` methods and see what happens.

```
class Person{
  String name;
  String gender;
  int age = 0;

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

int main() {
  var firstPerson = Person();

  firstPerson.name = "Sarah";
  firstPerson.gender = "female";
  firstPerson.age = 25;

  firstPerson.walking();
  firstPerson.talking();
}
```



`walking()` is being invoked on the object `firstPerson` and prints a variable `name`. Since the method is called on an object, it determines if the variable used is one of the instance variables of that object. As `name` is defined for the object `firstPerson`, the method will print its value.

In the code above, `firstPerson.walking()` takes the value of `Sarah` and prints `Sarah is walking`. `firstPerson.talking()` performs a similar operation, however, prints `Sarah is talking` instead.

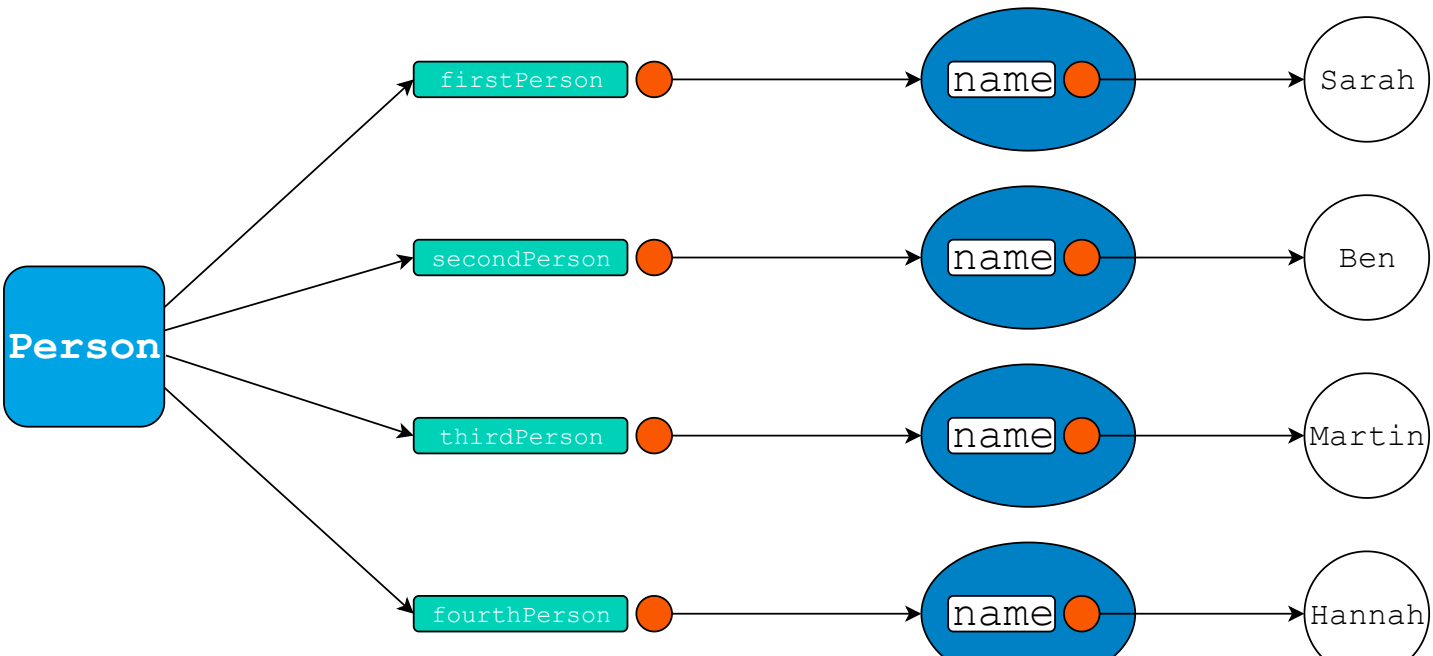
Multiple Objects of the Same Class

As classes provide reusable code, it makes sense that we can create multiple objects using the same class blueprint. Along with `firstPerson`, let's create more objects of the `Person` class.

```
class Person{
  String name;
  String gender;
  int age = 0;
```

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

}

int main() {
  var firstPerson = Person();

  firstPerson.name = "Sarah";
  firstPerson.gender = "female";
  firstPerson.age = 25;

  // Creating an object of the Person class
  var secondPerson = Person();
  secondPerson.name = "Ben";

  // Creating an object of the Person class
  var thirdPerson = Person();
  thirdPerson.name = "Martin";

  // Creating an object of the Person class
  var fourthPerson = Person();
  fourthPerson.name = "Hannah";

  // Driver Code
  print(firstPerson.name);
  print(secondPerson.name);
  print(thirdPerson.name);
  print(fourthPerson.name);
}
```

Even though there are multiple `name` variables, they all are referenced by different objects, hence, modifying one doesn't modify the others. This is why properties are known as instance variables, because each object has its own set of those variables.



Let’s move on to constructors in the next lesson.