Methods

A method is something you can do with an object. A method might (or might not):

- Take arguments,
- Return a result,
- Change the object's state,
- Have other side-effects (e.g. printing to the console).

We often want methods which simply get or set the value of a field; these methods are called "getters" and "setters", and they're easy to write. For example, consider the Person class below:

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

    Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    String getName() {
        return this.name;
    }

    int getAge() {
        return this.age;
    }

    void setName(String name) {
        this.name = name;
    }

    void setAge(int age) {
        this.age = age;
    }
}
```

This class has fields for the person's name (of type String) and their age (of type int). There is a constructor which sets initial values for these fields, and there are four methods:

- getName takes no arguments, and returns the person's name (a String).
- getAge takes no arguments, and returns the person's age (an int).
- setName takes a new name (as a String), assigns it to the name field, and returns nothing.

- setAge takes a new age (as an int), assigns it to the age field, and returns nothing.

In Java, you must declare a method's parameter types and return type. The return type is written before the method name, and the parameter types are written before each parameter's name. The special type void means the method does not return a result.

By convention, the "getter" method for a field named foo is named getFoo, and the "setter" method is named setFoo.

Calling methods

To call a method you need first to create an object (and a variable referencing it) using the new keyword and choosing a name and an age. (Since the name is a string, you must write it in double-quotes.)

```
Person alice = new Person("Alice", 19);
```

Once you have created an object (and a variable referencing it), the syntax for calling methods in Java is the same as in Python: for example, if alice refers to an object then alice.getAge() calls its getAge method. Try the following in the Code Pad:

```
Person alice = new Person("Alice", 19);
System.out.println(alice.getName());

returns  "Alice" (String)

System.out.println(alice.getAge());

returns 19 (int)

alice.setAge(20);
System.out.println(alice.getAge());

returns 20 (int)
```

Note that you need a semicolon, ; after each line, unless it's an expression which you want to see the result of. (The "setter" methods return nothing, so they aren't expressions.)

After calling a "setter" method, you can also see that the field's value has changed:

```
System.out.println(alice.age)
returns 20 (int)
```

"Getter" and "setter" methods are simple, but they're often needed, so it's worth practicing writing them.

It's usually better to use "getter" and "setter" methods rather than access an object's fields directly; we'll get into the reasons for this later.