# Classes

Matthew Meisel

Some portions of this material are modified from *Surverying the Field of Computing*, 4th ed., by Carl Burch, 2002.

## Classes

A class is a variable. Unlike a primitive type, however, it is a new type of object that a programmer can create to store information in a customized manner. Before we can use a class in our program, we have to design it.

For our purposes, we can say that a class has five main parts:

- instance variables
- static variables
- a constructor method
- instance methods
- static methods

First, we must program a class, and then our program can create new **instances** of this class for use in our program. The class itself is merely a template for variables of that type.

## Instance variables

Every class holds certain useful data. For example, a class representing an employee might have instance variables that hold that employee's name (a string), phone number (an integer), and salary (an integer). We call these variables **instance variables** because they are unique to each instance of a class. The values of the instance variables will probably be different in each instance of the class.

We declare instance values to be private so that only that instance of the class can access them. We use the constructor method and instance methods to assign values to instance variables.

## Static variables

On occasion, it is useful to store variables in the class that are constant in every instance of the class. These variables are **static variables**. For example, if we are programming a class that is working with trigonometric data, it might be useful to store pi ($\pi$) as a static variable. The advantage of declaring some variables to be static is that it is easy to change them *for every instance of the class* if some fact of our simulation changes. In the example class later, we declare the variable `payInterval` to be static, so we can go back into our program and change its value if, say, our company switched from monthly to bi-weekly pay intervals.

We use the `static` keyword with static variables. We usually declare static variables to be private so that only methods within that class or within instances of that class can access them.

## Constructor method

A class' **constructor method** is called by the program to create a new instance of the class. Therefore, the constructor method of a class should assign initial values to some, if not all, of the instance variables.

Constructor methods always take the name of the class they are in. (Example: in the class `Employee`, the name of the constructor class will always be `public Employee`.) Constructor methods do not return a value, but by convention we do not call them "void." We always declare constructor methods to be public so that the program (or other classes) can call them. Like all methods, constructor methods may be overloaded, that is, there may be more than one constructor method with different parameters (inputs).

## Instance methods

Once a class is created, a program communicates with it through the class' **instance methods**. Instance methods can perform calculations on the class' variables and return the result to the program. Commonly, methods beginning with the words `get` and `set` are used to ask for and change, respectively, a class' instance variables.

We usually declare instance methods to be public so that the program (or other classes) can call them. Instance methods may also be overloaded.

We call them from our program using this code:

```
variableName.methodName(parameters)
```

## Static methods

In some classes, we will want to create methods that perform useful operations on objects of that class. These are **static methods**. Whereas instance methods interact with instance variables of that class, static methods often interact with the instances themselves. For example, in a class representing a complex number, we would use an instance method to find the norm of that specific complex number, but we would use a static method to add two complex numbers together.

We use the `static` keyword with static methods. We usually declare static methods to be public so that the program (or other classes) can call them. Static methods may also be overloaded.

We call them from our program using this code:

```
ClassName.methodName(parameters)
```

## An example class

The following class represents an employee at a company. See if you can spot each of the following: the constructor method, instance variables, static variables, instance methods, and static methods.

```
 1  public class Employee {
 2
 3      private String name;
 4      private long homePhone;
 5      private int salary;
 6
 7      private static int payInterval = 2;
 8
 9      public Employee (String n, long hp, int s) {
10          name = n;
11          homePhone = hp;
12          salary = s;
13      }
14
15      public String getName () {
16          return name;
17      }
18
19      public long getHomePhone () {
20          return homePhone;
21      }
22
23      public int getSalary () {
24          return salary;
25      }
26
27      public void setName (String n) {
28          name = n;
29      }
30
31      public void setHomePhone (long hp) {
32          homePhone = hp;
33      }
34
35      public void setSalary (int s) {
36          salary = s;
37      }
38
39      public int payDayAmt () {
40          return payInterval*salary/52;
41      }
42
43      public static int avgSalary (Employee a, Employee b) {
44          int totalSalary = a.getSalary() + b.getSalary();
45          return totalSalary/2;
46      }
47
48  }
```

Let's look at the code in order:

Line 1: Indicates that we're defining a class named `Employee`. We could choose any name we wanted here, but Employee is a good choice, since it denotes what an instance of this class represents. By convention in Java, all classes are named with a capital letter.

Lines 3-5: Declares the instance variables. Notice that each looks like the local variable declarations we've been using, except that we have added the word `private`, and the definitions are outside of all methods.

Line 7: Declares a static variable. Do you see why it is advantageous to the programmer to declare this variable static?

Lines 9-13: Defines a constructor method.

Lines 15-41: Defines many instance methods.

Lines 43-46: Defines a static method. Again, do you see why it is advantageous to the programmer to declare this method static?

## Creating a class in a program

We create a class in a program by first declaring a variable of the given type:

```
Employee e;
```

Then, we use the keyword `new` and call the class' constructor method (or one of its constructor methods):

```
e = new Employee("Joe Schmoe", 5553141, 45000);
```

We have now created an instance of the Employee class called `e`, and we can call any of `e`'s public methods from our program. If Joe Schmoe has just received a raise, and we want our program to reflect that, we would write:

```
e.setSalary(55,000);
```

## *Exercise 1*

Based on the above code for the `Employee` class, determine what the output would be of the following program. (Note: do not actually program this into your computer, just write what its output would be.)

```
public class TrivialApplication {

    public static void main(String args[]) {

       Employee e = new Employee("Joe Schmoe", 5553141, 45000);

       System.out.println(e.getSalary());
       e.setSalary(55000);
       System.out.println(e.getSalary());

       System.out.println(e.getHomePhone());
       e.setHomePhone(1234567);
       System.out.println(e.getHomePhone());

       System.out.println(e.getName());
       e.setName("Joe H. Schmoe");
       System.out.println(e.getName());

       System.out.println(e.payDayAmt());

       Employee e2 = new Employee("Jane Schmoe", 5554321, 65000);

       System.out.println(Employee.avgSalary(e,e2));

    }
}
```

Notice that void methods (like those beginning with `set`) are written alone as their own lines of code because they are commands; whereas, methods that return a value (like those beginning with `get`) must be part of another piece of code (that will usually store the return value in a variable, or, in this case, print the return value).

## *Exercise 2*

Write a complex number class.

Your class should contain:

- two instance variables, one each to represent the number's real and imaginary parts
- no static variables
- a constructor method whose parameters are two real numbers that will become the real and imaginary parts
- instance methods which: get the real part, get the imaginary part, set the real part, set the imaginary part, and get the norm of the complex number
- four static methods, whose parameters are two complex numbers, which: add, subtract, multiply, and divide the two numbers and return the result.