Exercise 3

Parameters and return values

By the end of this exercise you will be able to

- Define and call a method which has *parameters*.
- Define and call a method which has a *return value*.
- Ignore the return value of a method.
- Use *method overloading*.

Introduction

If you have successfully completed the introduction to methods, properties and constructors of exercise 2, then you are now ready for a more detailed look at how methods work.

Methods are like functions for those of you who have a background in non objected-oriented programming languages like C or Pascal. This exercise will go over *arguments* and *parameters*, or how to send information into a method, and *return values*, or how to get information out of a method. Also we will mention the more advanced ideas of ignoring a method's return value and what it means to *overload* a method.

The questions that follow refer to the `Citizen` and `CitizenTest` classes that are shown in the program listing.

Questions

1. Fetch the file `CitizenTest.java`, which contains the code from the program listing.
2. Write a `salaryRise` method that has one `int` parameter called `amount` which increases the person's salary by that amount. Add some code to the `main` method to raise Ernie's salary by $10,000.

   HINT: Look at the `setSalary` method to see how to write a method with one parameter.
3. Write a `netWorth` method that returns the difference of the person's savings and their loan. Then put some code in the `main` method to print out Ernie's net worth.

   HINT: Look at the `getSalary` method to see how a method can return a value to the method which called it.
4. If we take away the `System.out.println( )` method call from the line that says: `e.getSalary()`, then nothing is printed to the screen when we run it. If we don't use the value returned by the call to `getSalary`, then why does Java even allow this code to compile?
5. Write a `toString` method that has no parameters and returns a string containing all the information about the citizen, including their name, salary, savings and their loan, if they have one. Put a some code into the `main` method to test it out.

Program listing

```
class Citizen {

   // Properties of the class...
   private String name;
   private int    salary;
   private int    savings;
   private int    loan;

   // Constructor of the class...
   public Citizen(String aName, int aSalary, int aSavings, int aLoan) {
      name    = aName;
      salary  = aSalary;
      savings = aSavings;
      loan    = aLoan;
   }

   // Methods of the class...
   public int getSalary() {
      return salary;
   }
   public void setSalary(int aSalary) {
      salary = aSalary;
   }

}

class CitizenTest {

   // The main method is the point of entry into the program...
   public static void main(String[] args) {

      Citizen e = new Citizen("Ernie", 50000, 2000,   0);
      Citizen b = new Citizen("Bert", 100000,10000,5000);

      System.out.println("Ernie's salary is: " + e.getSalary());
   }
}
```

Classes often have a `toString` method and you will learn more about this later.

HINT: Use the `+` operator to build up a `String` object and return this object at the end of the method.

6. Next to the constructor, write a second constructor that has one `String` parameter called `name`. Make it so that the constructor sets the person's name and sets every other property to zero dollars. Add the following line of code to the `main` method to test out this second constructor that you have just written:

```
Citizen f = new Citizen("Fred");
```

NOTE: Having more than one constructor in a class is an example of method overloading. You can overload any method, not just the constructor, by having several methods all with the same name but different parameters and in the same class.