CITP 190 - Intro to Programming in JAVA
Writing a Class

Classes are useful because they enable code reuse. Once you write a class, you can use it in many programs.
Look at the Section Class Handout. It contains the code for a class called Section formatted to class standards. Let's examine this code.

This line of code:

```
	public class Section
```

declares the class Section. It was created for you when you created the new class.

Most classes have data associated with them. The data is contained in instance variables. In the Section class, we have a variable to hold the section number and a variable to hold the number of students in the section. These instance variables are declared with these lines of code:

```
	// instance variables
	private String sectNum;		// the section number of the section
	private int numStudents;	// the number of students in the section
```

Usually we declare the instance variables as `private`. This makes sure that the only way the data can be changed is by using one of the methods of the class. When only the methods of the class can change the data it is called **data hiding**.

Our class will not be of much use to us unless we can create objects of the class and use them. The method that creates an object is called a constructor. The name of the constructor is always the same as the name of the class. The following code defines a constructor for the Section class:

```
	public Section(String section, int number)
	{
		sectNum = section;
		numStudents = number;
	}
```

This constructor takes two arguments, section and number. It assigns the value of section to the instance variable `sectNum`. It assigns the value of number to the instance variable `numStudents`.

The constructor is always a `public` method. This allows us to use it in other classes, such as the class that contains the `main()` method. The constructor is the only method that does not have a return type. JAVA knows the method is a constructor because it has the same name as the class and it does not have a return type. If you include a return type, the method is no longer a constructor!

You may see classes that contain data but do not contain a constructor. If you do not include a constructor in your class, JAVA will create a constructor for you. It will set all String values to null, all numeric values to 0, and all Boolean values to false. This constructor is called the **default constructor**.

A class can have more than one constructor. Each constructor must have a different signature. The signature of a method is its return type plus the types of its arguments. The signature for the constructor in our class is: `String, int`.

We could have explicitly included the default constructor by including the following code:

```
	public Section()
	{
		sectNum = null;
		numStudents = 0;
	}
```

The signature for this constructor is nothing, i.e. there are no arguments.

We could also have defined a constructor with one argument such as:

```
	public Section(String section)
	{
		sectNum = section;
		numStudents = 0;
	}
```

The signature for this constructor is: `String`. When you have more than one method with the same name, but different signatures, it is called **overloading**.

The next method in the Section class allows us to change the number of students by adding to it. The code is:

```
    public void addStudents(int number)
    {
        numStudents += number;
    }
```

Notice that this method does not return anything. If we need to use the number in our program, we will have to use a getter method.

The next method formats the data nicely for printing. The code is:

```
    public String formatString()
    {
        NumberFormat withCommas = NumberFormat.getNumberInstance();
        return "Section:            " + sectNum + "\nNumber of Students: " +
            withCommas.format(numStudents);
    }
```

Because we declared the variables **`private`**, we cannot access them outside the class. If we need to know the value of one of the instance variables in some other code (such as the `main()` method) we need to have a method that will return the value. These methods are called getters. There is normally one getter for each instance variable. The getter for the `numStudents` instance variable is:

```
    public int getNumStudents()
    {
        return numStudents;
    }
```

It is a typical getter in that all it does is return the value for the instance variable. It also has a typical getter name - the word `get` followed by the name of the instance variable. Although this is typical, it is not required that getters be named in this way.

The getter for the section instance variable is:

```
    public String getSectNum()
    {
        return sectNum;
    }
```

If we cannot access instance variables outside the class to see their values, we also cannot access them to change their values. If we need to change the value of one of the instance variables in some other code (such as the `main()` method) we need to have a method that will change the value. These methods are called setters. There is normally one setter for each instance variable. The setter for the `numStudents` instance variable is:

```
    public void setNumStudents(int number)
    {
        numStudents = number;
    }
```

If the specifications for the class require that an instance variable have only certain values, the setter method must validate the value it is given <u>before</u> making any change. If the value does not meet the criteria, the method <u>must not</u> change the value of the instance variable.

The setter for the `sectNum` instance variable is:

```
    public void setSectNum(String section)
    {
        sectNum = section;
    }
```

Now that we have a class, we can use it in a program. Let's look at the following program:

```
	public static void main(String[] args)
	{
		// instantiate two variables of the Section class
		Section sect1 = new Section("10225", 10);
		Section sect2 = new Section("52895", 1004);

		// user the formatString() method
		// to display the contents of the variables
		System.out.println(sect1.formatString());
		System.out.println(sect2.formatString());
		System.out.println();

		// add 35 students to sect1
		sect1.addStudents(35);

		// display the new contents of the variable
		System.out.println(sect1.formatString());
		System.out.println();
	}
```

You create an instance (object) of the Section class using the `new` operator. The following code creates a new object:

```
		Section sect1 = new Section("10225", 10);
```

This code calls the constructor for the Section class. If you look at the code for the constructor, you will see it has two arguments, `String section` and `int number`. When we call the constructor, the `section` argument will be set to the value `"10225"` and the `number` argument will be set to the value 10. The constructor code the sets the value of the `sectNum` instance variable to the string `"10225"` and the value of the `numStudents` instance variable to 10 for the object named `sect1`.

The following code creates another new object:

```
		Section sect2 = new Section("52895", 1004);
```

This code sets the value of the `sectNum` instance variable to the string `"52895"` and the value of the `numStudents` instance variable to 1004 for the object named `sect21`.

We use the `formatString()` method to format the contents of each object and then use the `println()` method to display the contents with these lines of code:

```
		System.out.println(sect1.formatString());
		System.out.println(sect2.formatString());
		System.out.println();
```

We now add 35 to the `numStudents` instance variable for the `sect1` object with the following code:

```
		sect1.addStudents(35);
```

Finally, we display the new value of the `sect1` object with the code:

```
		System.out.println(sect1.formatString());
		System.out.println();
```

The output of this program is:

```
Section:            10225
Number of Students: 10
Section:            52895
Number of Students: 1,004

Section:            10225
Number of Students: 45
```