

## Objectives

After completing the lab you should be able to

1. Use more features of Eclipse.
2. Use several methods of `String` class.
3. Test Java programs.
4. Construct an object using the `new` operator.
5. Extract useful information from Java API.
6. Use accessor and mutator methods of a class.
7. Use simple 2-dimensional graphics methods to draw a frame and display your name in a color.
8. Become more comfortable asking questions.

## References

Chapter 2 in *Big Java* by Cay Horstmann. **Please bring your textbook to lab. You will need it!**

You will use the following web pages in this lab.

- *Course website*: `<http://www.eg.bucknell.edu/~csci203/>`
- *Java API website*: `<http://java.sun.com/javase/6/docs/api/>`

## On Being Productive — Multiple Workspaces

Today we start with a tip that will make you more productive on the LINUX system.

Do you wish you had a larger computer screen? You can! Look for four small rectangles in the lower right corner of your screen. These correspond to your screen's four workspaces. We will place a web browser in one workspace and Eclipse in another.

Start a web browser and open the web page for today's lab. Click on one of the gray rectangles to switch to a different workspace. Now start Eclipse in the new workspace by selecting **Programming** → **Eclipse** from the Red Hat menu.

Watch how you can quickly switch between the two workspaces by clicking on the proper small rectangle.

Sometimes you want to move a window to another workspace. This is easy. Just right click on the top bar of the window and select **Move to Another Workspace**. You can also have a window, such as a calculator, remain on all four workspaces by selecting **Always on Visible Workspace**.

## Some More Features in Eclipse

If you have not started Eclipse, do that now. Create a new Java Project by selecting **File** → **New** → **Java Project** and give it the project name of `lab04-xyz01` where `xyz01` is your login name. Create a new class called `GreatStyle` by selecting **File** → **New** → **Class**. As usual, check the boxes for generating main and comments.

Eclipse will automatically indent and format your Java programs. Enter inside the `main` method the following as one line as shown.

```
int a; a=2; a=a*2+4; System.out.println("a =" +a);
```

Now save the file and you will see that Eclipse will reformat the Java code to adhere to *Sun's Java Programming Style Standard*: <http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html>. However, Eclipse does not do it all! For example, it will not select good variable names for you. It will not add blank lines between logical sections of the code. It will not place the statements in the proper order for you! You still have to think about your code and present it in a readable manner.

Run the program and observe that Eclipse does *not* reformat the text inside of double quotes, i.e., strings in your program. Add a space after the equal sign in `"a ="` and run the program again.

## Create a new `readme.txt` File

Select **File** → **New** → **Untitled Text File**. Select **File** → **Save As...**, now select your `lab04-xyz01` folder as the parent folder and save the file as `readme.txt`. Open your `banner` file (**File** → **Open File...**) and copy and paste your banner at the top of the file. Modify the date and lab portions and save again.

## Using Strings

One of the first kinds of objects introduced in CSCI 203 is the Java `String` class. See pages 21, 34–42 in *Big Java*. Strings are sequences of characters such as “Where is Hogwarts?”.

To create a `String` object that is assigned to the *reference variable* `s1` and initially holds the characters `"Bison"`, we would write:

```
String s1 = "Bison";
```

Note that the characters are enclosed in double quote marks.

The `String` class comes with a large number of *methods* that can operate on `String` objects. A method essentially describes the actions that can be applied to an object. You will learn a few basic methods for `String` in this lab. You will learn more as the semester progresses. One `String` method asks for its length, i.e., the number of characters it holds. For example,

```
s1.length()
```

will return 5. In Java, *messages* (or method calls) to an object have the following format:

```
objectName.methodName(parameter1, parameter2, ...)
```

The parentheses are *always* required even if a method has no parameters as in the `length` method above. A *message* is a request for the object (*receiver*) to do something and may possibly return a value.

Another `String` method asks for the  $i^{\text{th}}$  character. The method `charAt` returns the character at position  $i$ . Java starts counting positions at zero. Therefore, the message

```
s1.charAt(0)
```

returns the first character in the string `"Bison"` which is the capital letter B. You can also ask if one string is part of another string. The `String` method `indexOf` takes a `String` parameter and returns the index of the first character in the string that begins with string you are looking for in the given string. If the parameter string does not exist in the object that invokes the method, `indexOf` returns  $-1$ . For example, the message

```
s1.indexOf("son")
```

returns 2 because `son` is found starting at the third character of the string `"Bison"`, and the message

```
s1.indexOf("soon")
```

returns  $-1$  because `soon` is not found in `"Bison"`.

Strings may be used in assignment statements and use the plus symbol (+) to concatenate or combine string values.

```
String s2 = ""; // empty string with zero characters
s2 = "Go " + s1 + "!";
System.out.println(s2);
```

The above will print

Go Bison!

## Exercise 1: Using Strings

In Eclipse, select the `src` folder, create a new `Exercise1` class and write a Java program that assigns two strings to two `String` reference variables. Your program should concatenate the two values and store the result in a new `String` object `s3`. Print the length of `s3` and the second, fourth and sixth characters of `s3`. Then, print the results of checking if the second input string is part of the first input string.

For testing purposes, run your program with the following two strings: `"them"` and `"the"` (double-quotes are not part of the string). Also, do another run of two strings where the second string is not part of the first string.

Copy the output of the correct runs to your `readme.txt` file.

## Testing Programs

Software testing is the process of looking for errors in programs. This is a very important process that is discussed on pages 47–48 of *Big Java*. The approach we will use in this course is to print out a variable's *expected value* after a section of code. For example, you may want to check that a method returns a particular value.

## Exercise 2: More Using Strings and Testing Programs

In Eclipse, select the `src` folder, create a new `Exercise2` class and write a Java program that assigns your full name, for example, “Britney Jean Spears”, to the `String` variable `name`. The program displays the value of the `String` variable. Use the `charAt` method and the `+` (concatenation) operator to display your initials. For Britney, the program would display BJS. For testing purposes, on the next line display the expected value as shown on page 48.

Initials: ABC

Expected: ABC

To turn the characters into a string, append them to an empty string. For example, suppose your initials are in positions 0, 10 and 20 of the string. You can form a string with your initials by using

```
" " + name.charAt(0) + name.charAt(10) + name.charAt(20)
```

Copy the output of a correct run to your `readme.txt` file.

## Constructing Objects with `new`

Usually we *construct* (create) objects using the `new` operator in Java. (Note that the `String` class is a special case where we are not required to use `new`.) For example, to construct a `Rectangle` object we would write something like this:


```
Rectangle rect1 = new Rectangle(3, 4, 7, 8);
```

To the left of the assignment operator (`=`), this statement declares `rect1` to be a reference to a `Rectangle` object. To the right of the assignment operator, the `new` operator asks the `Rectangle` *constructor* to create a new rectangle. Recall that a *class* is an object factory that constructs new objects. `new` is an operator that instructs a class to construct a new object by invoking its constructor. The parameters to the constructor specify how the object should be initialized.

For some classes, e.g., `String`, their description is known to the Java compiler. But for most classes, you have to tell the Java compiler where a class's description is. That is the purpose of the `import` statement. The class `Rectangle` is in `java.awt.Rectangle`.

## Having Eclipse Enter the `import` Statement

A common problem is remembering the location in the Java library of a particular class that you need to import. Why spend precious minutes searching through textbooks when you can have Eclipse do it for you?

In Eclipse, create a new class called `Exercise3`. In the `main` method enter the above line with `Rectangle` using `new`. After you have typed the line, you will notice that Eclipse underlines both occurrences of “`Rectangle`” in red and places a red rectangle with an X in the margin . Hover your mouse pointer over this symbol or the underlined words, and Eclipse will tell you what's wrong. In this case, it will tell you that `Rectangle` is not a recognized type. Click on the red symbol in the margin, and it will offer suggestions on how to correct the problem.

Use the up and down arrow keys on your keyboard to move through the suggestion list. As each item is highlighted, Eclipse will show you what will happen if

you select that item. In this case, you want to import `Rectangle` from `java.awt`. Select that item and press Enter. Eclipse will add the necessary `import` statement to your code. Find the line `import java.awt.Rectangle;` near the top of your program.

**Note:** When Eclipse offers you suggestions, make sure you select the proper one and one you understand! Otherwise Eclipse may modify your program in ways that can lead to deep trouble. If you accidentally select the wrong suggestion, don't panic! Just undo it.

### Read API about `Rectangle` Class

For information of using the Java API, see pages 50–52 in *Big Java*.

Visit the *Java API website*: <http://java.sun.com/javase/6/docs/api/> and find the `Rectangle` class. Scroll down to the “Constructor Summary” and find the four parameter constructor. Read its description. Scan down through the methods that are available. Read about `getWidth`, `getHeight`, `getX`, `getY`, `setSize` and `translate` methods. The methods with the word “Deprecated” means they were in older versions of Java and should not be used.

It is important that you be comfortable accessing the API library and learn how to extract useful information. Don't memorize! Look it up in the API!

### Exercise 3

Add code to the main method of a `Exercise3` class to do the following:

1. Construct a `Rectangle` object `rect1` with location `(2,3)` and width of 4 and height of 6.
2. Construct a `Rectangle` object `rect2` with location `(5,1)` and width of 2 and height of 4.
3. For both rectangles, display the x and y locations and width and height in a form that is readable. Use the `getX`, `getY`, `getWidth` and `getHeight` methods. What type of values do these methods return? Look in the left hand column in their API entry for the answer.
4. For rectangle `rect1`, use the `translate` method to move the rectangle to location `(5,4)`. Display the four values in a readable form. Note that this method is listed as having a `void` return type which means it does not return a value. Also, for testing purposes, you should display the expected values.

- For rectangle `rect2`, use the `setSize` method to resize the rectangle to width of 1 and height of 10. Display the four values in a readable form. Also, for testing purposes, you should display the expected values.

Copy the output of a correct run to your `readme.txt` file.

Methods that only access the state of an object are called *accessor* methods. Methods that change the state are called *mutator* methods. See pages 46–47 in *Big Java*.

Here is a question to answer and place in your `readme.txt` file. In the methods you used in this exercise which are accessor methods and which are mutator methods?

## Practicing Simple 2-dimensional Graphics Methods

The reference for this section is pages 58–70 in *Big Java*.

To display a simple frame (window) in Java, one has to use a bit of “plumbing,” i.e., statements that you don’t necessarily understand but you copy close to verbatim. See pages 58 and 59 on how to write a Java program with the minimal “skeleton” to display an empty frame. Whenever, you want to write a graphics program, you should start with this skeleton.

You might want to type in the Java program on page 59 and run it.

Assuming one starts with the “skeleton,” one constructs components and adds them to the `JFrame` object (Inserted above the `frame.setVisible(true);` line). In Exercise 4, you will construct a `JTextField` component and set its background and text.

## Exercise 4

Type in the following Java program in Eclipse and add your banner.

```
import java.awt.Color;
import javax.swing.JFrame;
import javax.swing.JTextField;

public class FrameTester {

    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.setSize(200, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Add a JTextField component that displays
```

```
// some text on the frame

JTextField myText = new JTextField("Hello, World!");
myText.setBackground(Color.PINK);
frame.add(myText);

frame.setVisible(true);
}
}
```

Run the program to see the results.

Modify the program as follows:

1. Double the frame size in each direction.
2. Change the greeting to “Hello, *your name!*” where you insert your name.
3. Change the background color to pale green (See section 2.13.1 on page 67 of *Big Java*).

For this exercise nothing is to be copied to your `readme.txt` file. However, since the TA will grade this exercise by running your program, make sure your modified `FrameTester.java` program is in the proper place under your `lab04-xyz01` project.

## What To Submit

Make sure that your `readme.txt` file contains the following.

1. The results of the two runs from Exercise 1.
2. The results of a run from Exercise 2.
3. The results of a run from Exercise 3 and the answer to the question.

Make sure your `FrameTester.java` file from Exercise 4 and `readme.txt` are inside your `lab04-xyz01` lab directory (folder) in proper place.

Open a File Browser by double clicking on the home folder icon on your desktop. Double click on your `csci203` folder. Drag your `lab04-xyz01` lab folder onto the CSCI203 *lab* drop box on the left:

[CSCI203 Lab drop\\_box](#)