OBJECT ORIENTED PROGRAMMING IN JAVA

Working With Objects

LAB 03 TABLE OF CONTENTS:
3.1 Java Naming Rules and guidelines
3.2 Using a JFrame object
3.3 Using JOptionPane to Print a Message
3.4 Reading Error Messages
3.5 Combining Statements and the Newline Character
3.6 Using JOptionPane to Input a String
3.7 String Methods
3.8 Style Guidelines
3.9 Programming Exercises

PURPOSE OF LAB 03
Working with objects is central to programming in an Object - Oriented Language such as Java. In this lab you will work with
- objects that are used to create a Graphical User Interface, or GUI
- Strings are used for both input and output in our programs.

TO PREPARE LAB 03
- Read Wu: Read Chapters 1 and 2. You may omit the following:
  - pages 51 – 54 Ch2FunTime.java and Ch2MyWebBrowser.java
  - pages 66 – 68 Section 2.4.3 Date and SimpleDateFormat (this may be covered in class)
- Read through this laboratory session
- Using your memory device, create a directory called lab03 in which you should save all of your work. There are no files to copy for this lab.

TO COMPLETE LAB 03
- This is an individual lab. You may get help from other students as well as the lab tutor. Read the honesty policy for guidelines.
- When you have completed the lab,
  - See the lab tutor, who will give you a 10 point, open note, individual quiz.
  - For 10 points, hand in to the lab tutor, a printed copy of the final versions of the two programs in this lab, HelloWorld.java and StringTest.java

When you have finished this lab, see the lab tutor, who will give you an open note, ten - point quiz. For ten points, hand in printouts of your final versions of HelloWorld.java and StringTest.java. This will be your grade for Lab 03.
3.1 JAVA NAMING RULES AND GUIDELINES

An identifier is a word that can be used to name a class, method, variable, or constant. Java has rules for choosing a legal identifier. If a rule is not obeyed, the code will not compile. Recall that Java is case sensitive.

Java Rules for Choosing Identifiers

1. An identifier may use all letters, a–z and A–Z, all digits, 0–9, the underscore ( _) and $. 
2. An identifier may not begin with a digit. 
3. An identifier may not be a Java reserved word. Reserved words, or key words are words that have a special meaning in Java. Here is the list with the words that we have already used highlighted.

```
abstract boolean break byte case catch
char class const continue default do
double else extends final finally float
for goto if implements import instanceof
int interface long native new package
private protected public return short static
strictfp super switch synchronized this throw
throws transient try void volatile while
```

Java Naming Guidelines

In addition to the naming rules, there are also guidelines, or conventions, that were established by the authors of the Java API. If a guideline is not obeyed, your code will compile, but it will be more difficult for you and others to read. Knowing these guidelines, will help you understand code that uses Java API classes.

Identifiers: Choosing names that indicate the purpose of the class, method or data value is known as **self – documentation**.

**ClassNames:** begin with a capital letter, additional words are capitalized. Examples:

First, HelloWorld, JFrame, JOptionPane

**variableNames:** begin with a lower case letter, additional words are capitalized.

Examples:

myWindow, visible, width, height, interestRate ← Note: no parentheses

**methodNames:** begin with a lower case letter, additional words are capitalized.

Examples:

setTitle(), setSize(), showMessageDialog() ← Note: always parentheses

**CONSTANTS:** are completely capitalized, additional words are separated with an underscore character _. Examples:

PI, MAX_VALUE, INTEREST_RATE

At all times, you are expected to follow the Java guidelines for choosing identifiers. Failure to do so will result in points being deducted from the code you write.
3.2 USING A JFRAME OBJECT

Java provides us with a library of classes called the Java Application Programming Interface, or Java API. In this session, we will use the predefined classes JFrame, JOptionPane, and String. You will learn to define your own classes in a later session.

Java is an object-oriented programming language. Before an object can exist, a class must define what characteristics and behaviors an object will have. The class is not the same as the object. Rather, the class is a template from which objects can be created.

A **class** is a template that describes the characteristics and behaviors that objects of this type will have. The characteristics are *data values* and the behaviors are *methods*.

An **object** is an instance of a class.

**Experiment 1: HelloWorld.java**

Traditionally, a programmer writes a Hello World program as his first program in a new programming language. For you, this will be your second program. Instead of printing to `System.out`, as done in `First.java`, the message will be displayed in a `JDialog` box that is centered in a `JFrame` window. A `JFrame` object can be displayed on the computer monitor as a window with characteristics, or data values, that include height, width and title.

To better understand the code that displays this message, the program will be written in small increments that can be compiled and executed. Space is allotted for you to record the results of each step.

**Step 1:** Begin by opening TextPad and enter the code required for all Java applications:

```java
class HelloWorld
{
    public static void main(String[] args)
    {
        //
    }
}
```

All Java applications must contain a main method. When an application is run, or executed, a message is sent to the application class to execute its main method.

Save the file with the required name `HelloWorld.java` in your `lab02` directory.

Compile the program. Even though there are no statements to execute in the main method, it is a good idea to compile programs frequently. This way, if an error has been made, it can found more easily.

The statements that are executed when the application is run are placed, in the order they are to be executed, inside the body of the main method.
Declaring and Creating an Object

To use an object, it must first be declared and created. The object declaration statement:

```java
JFrame frame;
```

allocates memory, known as `frame`, to refer to a `JFrame` object. To create, or instantiate, an object, use the `new` operator and the name of the class. The object creation – assignment statement

```java
frame = new JFrame();
```

is read right to left, "create a new JFrame object and assign it to frame". To the right is a state of memory diagram illustrating what is stored in memory after the two statements are executed.

Step 2: Add the two new statements to the main method of your program.

```java
class HelloWorld
{
    public static void main(String[] args)
    {
        JFrame frame; // declare a variable of type JFrame
        frame = new JFrame(); // create a JFrame object & assign it to frame
    }
}
```

Compile the code. Record the "essence of" the compiler error message.

________________________________________________________________________________________________
________________________________________________________________________________________________
________________________________________________________________________________________________

The Java API library of classes is organized into various directories that contain definitions of classes that have some commonality. The `JFrame` class is stored in the directory `swing`, which is a subdirectory of the directory `javax`. The compiler must be told where the `JFrame` class can be found.

Step 3: To correct the compiler error, add an `import` statement as the first line of code in the file.

```java
import javax.swing.JFrame;
```

Compile the code and run the program. Record the results.

________________________________________________________________________________________________
________________________________________________________________________________________________
________________________________________________________________________________________________

Using Instance Methods

A `JFrame` object has characteristics that include its height, width and title. Values can be assigned to these instance variables, or data values, by sending messages to the object.

To send a message to an object, a method is invoked on the object. A method that is invoked on an object is an instance method. To invoke an instance method follow the name of the object with a dot (.) followed by the name of the method, followed by a pair of parentheses. Information that is passed to the method is called an argument. Arguments are separated by commas and placed between the parentheses.
Step 4: The instance method `setTitle` must have a `String` argument, representing the title to be displayed in the `JFrame` object's titlebar.

```java
frame.setTitle("My Hello World Program");
```

The instance method `setSize` has two arguments that represent the width and height of the `JFrame` measured in a number of pixels.

```java
frame.setSize(500, 500);
```

Add these two lines of code to the end of the body of the main method, Compile the code and run the program. Record the results.

________________________________________________________________________________________________
________________________________________________________________________________________________

The `JFrame` object has an instance variable `visible` that initially stores `false`. The `JFrame` instance method `setVisible` must have a `boolean` argument, either `true` or `false`.

Step 5: Add the statement

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

at the end of the body of the method `main`. Compile the code and run the program. Record the results.

________________________________________________________________________________________________
________________________________________________________________________________________________

Note: On a PC, close this program by click on the X in the upper right – hand corner of all open windows.

Step 6: Experiment by changing the integer arguments in the statement that sets the size, in the number of pixels, of the `JFrame` object. Each time compile and run the program. Then answer this question. Which argument (first or second) represents the height of the `JFrame` object and which argument represents the width? In your answer indicate how you made this determination.

________________________________________________________________________________________________
________________________________________________________________________________________________
________________________________________________________________________________________________
________________________________________________________________________________________________

3.3 USING JOPTIONPANE TO PRINT A MESSAGE

A `JDialog` object is used to communicate with the user. On page 3.3 a message dialog box is pictured that is used to print a message. When the OK button is clicked, the dialog box closes. The `JDialog` also has characteristics such as width, and height that need to be set to dimensions that accommodate long or multiple lines. Later we will use a `JDialog` for user input.

Using Class Methods

Rather than create the `JDialog` objects directly, the `JOptionPane` class defines class methods that create `JDialog` boxes with appropriate dimensions, title and components, such as buttons. Since these methods are class methods, they are invoked on the name of the class in which they are defined rather than on an object. The `JOptionPane` class method `showMessageDialog()` is used to display a message in a dialog box. To invoke the method we write

```java
JOptionPane.showMessageDialog(frame, "Hello World");
```
There are two arguments that must be passed to the method. `frame` identifies the parent component on which the dialog box is centered and the `String` is the message that is printed.

**Step 7:** Add the statement above at the end of the `main` method. In addition, because the `JOptionPane` class is also defined in the `javax.swing` package, add an `import` statement to the beginning of the file. Your `HelloWorld.java` file should now contain the following code:

```java
import javax.swing.JOptionPane;
import javax.swing.JFrame;

class HelloWorld {
    public static void main(String[] args) {
        JFrame frame;
        frame = new JFrame();
        frame.setSize(500, 500);
        frame.setTitle("My Hello World Program");
        frame.setVisible(true);
        JOptionPane.showMessageDialog(frame, "Hello World!");
    }
}
```

Compile the code and run the program. Record the results. Does your program work as expected?

---

### 3.4 READING ERROR MESSAGES

Learning to read error messages is an important skill that you should develop as the semester continues. As your programs become larger, the number of error messages will increase. During the semester, you will learn how to control the number of error messages by implementing code in small, logical sections.

**Instructions for Recording Error Messages**

In this lab, you will work with controlled errors. That is, the file will contain only one intentional error at a time. For each error you should:

1. Introduce the single error into your code.
2. Save the file containing the error.
3. Compile the program.
4. If there is a compiler error, record "the essence" of the error message and the line number on which it was detected. If the program compiles, run it.
5. Correct the error.

To begin, be sure that your program `HelloWorld.java` compiles and runs.

**Syntax Errors**

A **syntax error** occurs when the rules of the language are violated. These errors are always found by the compiler. The error message is determined by the compiler that is used.

**Error 1:** Eliminate the first double quote (" ) in the statement

```java
    frame.setTitle("My Hello World Program");
```
When I made Error 1 and compiled the program on my home PC, I received two error messages for the single error. Notice, that both error messages found the correct line on which the error was made, but one states the exact error. In TextPad, to view line numbers in the source file click on **View – Line Numbers**.

<table>
<thead>
<tr>
<th>File name</th>
<th>line #</th>
<th>the error</th>
</tr>
</thead>
<tbody>
<tr>
<td>C:\Documentsand Settings\Marian Manyo\My Documents\cosc060\HelloWorld.java:11: ')' expected frame.setTitle(&quot;My Hello World Program&quot;);</td>
<td>^</td>
<td>the line of code where error was detected</td>
</tr>
<tr>
<td></td>
<td></td>
<td>character where error was detected</td>
</tr>
<tr>
<td>C:\Documentsand Settings\Marian Manyo\My Documents\cosc60\HelloWorld.java:11: unclosed string literal frame.setTitle(&quot;My Hello World Program&quot;);</td>
<td>^</td>
<td>the line of code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>character where error was detected</td>
</tr>
</tbody>
</table>

2 errors

Now compile your code. Record all of the error messages.

__________________________________________________________

Error 2: Correct Error 1. Then, eliminate the second " in the statement

frame.setTitle("My Hello World Program");

Compile the code. Record only the "essence" of either the first or the most informative error message.

__________________________________________________________

Error 3: Correct Error 2. Then, eliminate the semicolon at the end of the statement.

frame.setTitle("My Hello World Program")

Compile the code. Record only the "essence" of either the first or the most informative error message.

__________________________________________________________

Error 4: Correct Error 3. Then misspell the word frame in the line

JOptionPane.showMessageDialog(frame, "Hello World!");

Compile the code. Record only the "essence" of either the first or the most informative error message.

__________________________________________________________
Java Lab Manual

Error 5: Correct Error 4. Then, use incorrect capitalization in the word `JOptionPane` in the statement

```java
JOptionPane.showMessageDialog(frame, "Hello World!");
```

Compile the code. Record only the "essence" of either the first or the most informative error message.

Error 6: Correct Error 5. Then, omit one of the arguments in a method call. Replace the line

```java
JOptionPane.showMessageDialog(frame, "Hello World!");
```

with the line

```java
JOptionPane.showMessageDialog("Hello World!");
```

Compile the code. Record only the "essence" of either the first or the most informative error message.

---

**Runtime errors**

A *runtime error* occurs during the execution, or running, of a program. Basically, the computer is instructed to do something that it cannot do. Run-time errors cause the program to stop and error messages to be printed to monitor. For example, if a calculation involves division by zero, a runtime error message that declares an Arithmetic Exception may be printed in the terminal window.

Error 7: Correct Error 6. Then change the spelling of the word `main` to `Main`

```java
public static void Main(String[] args)
```

Compile the code and run the program. Record any error message.

---

Error 8: Correct Error 7. The following error may be a runtime error, or, depending on the compiler, it may be found by the compiler. Comment out the statement that creates the `JFrame` object

```java
//frame = new JFrame();
```

If the code compiles, run the program. Is this error a runtime or compiler error? Record the error message.

---

**Note:** In the future, commenting out lines of code is a very good way to determine where errors are made.
Logic errors

A logic error occurs when the semantics of the code is correct, but the meaning of the code, the semantics, is incorrect. A logic error is often called a bug. If your program has a bug, it cannot be found by the compiler or by the computer when the program is run. Instead, it must be found by the programmer or by a person specifically assigned to test the program. Examples of logic errors are calculations that give incorrect results and special situations that are not handled or considered. You are expected to carefully test your programs to be sure that they are free of bugs. Since our program is small, not many logic errors are possible.

Error 9: Correct Error 8. Then change the spelling of Hello to Hellow

    frame.setTitle("My Hellow World Program");

Compile the code. Run the program, if the code compiles. Record any error message.

3.5 COMBINING STATEMENTS AND THE NEWLINE CHARACTER

Step 8: Make three additional changes to the current program.

Using the wildcard, *

When one or more classes are imported from a specified package, it is often easier to import all classes from the specified package. To do this, use the wildcard symbol, *, to represent all class files in the package. This does not reduce the performance of a program. Replace the statements

    import javax.swing.JFrame;
    import javax.swing.JOptionPane;

with the single statement

    import javax.swing.*;

Combining declaration and initialization statements

The declaration and object creation statements can be combined into a single statement. Replace the statements

    JFrame frame;
    frame = new JFrame();

with the single statement

    JFrame frame = new JFrame();

that both declares a JFrame variable frame and initializes it to refer to a new JFrame object.

Using the newline character '\n'

Special characters, such as the newline character, can be printed using a sequence of two characters, the escape character, '\', followed by another character whose normal meaning "is escaped". The combination of the characters \n represents the newline character \n, which returns the printer carriage to the beginning of the next line.

Modify the message that is displayed in the JDialog message box

    JOptionPane.showMessageDialog(frame, "Hello\nWorld!");
The entire program should now be

```java
import javax.swing.*;
class HelloWorld
{
    public static void main(String[] args)
    {
        JFrame frame = new JFrame();
        frame.setSize(500,400);
        frame.setTitle("My Hello World Program");
        frame.setVisible(true);
        JOptionPane.showMessageDialog(frame, "Hello
World!");
    }
}
```

Compile the revised code and run the program. Record the results

________________________________________________________________________________________________
________________________________________________________________________________________________
________________________________________________________________________________________________

**Step 11:** Make changes to the statement

```
JOptionPane.showMessageDialog(frame, "Hello\nWorld!");
```

So that it prints the message in a column:

```
Hello
World!
```

Record your change to the previous program. Compile and run the program to make sure it works. What happened to the dimensions of the JDialog box?

________________________________________________________________________________________________
________________________________________________________________________________________________
________________________________________________________________________________________________
________________________________________________________________________________________________

Leave the modified statement in this program as we continue to add more to the program. See the Post-Lab Exercises for more escape characters.
3.6 USING JOPIONPANE TO INPUT A STRING

The JOptionPane class method showInputDialog is used to read a user input string. The expression

```java
JOptionPane.showMessageDialog(frame, "Enter your name")
```

does not allow for the user name to be displayed in the text field. The expression

```java
JOptionPane.showMessageDialog(frame, "Enter your name: 

String name = JOptionPane.showInputDialog(frame, "Enter your name");
```

declares a String variable name and assigns to it the String returned by the showInputDialog() method. Due to the length of the line of code, it has been broken into two lines by using the <Enter> key at an appropriate place. Long lines should always be broken into multiple lines rather than using the word – wrap feature of a text editor. Lines, including leading spaces, should never be longer than 80 characters.

The operator + can be used to concatenate, or add, two String objects to form a new String object. Therefore, if name refers to the String "Mable", the expression

```java
String sentence = "My name is " + name
```

creates the new String "My name is Mable" and assigns it to the String variable sentence.

The statement

```java
JOptionPane.showMessageDialog(frame, "My name is " + name);
```

creates the same string but without storing it to sentence.

**Step 12:** Edit your modified HelloWorld.java as follows:

- Before any message is printed, prompt the user for his/her name.
- After the Hello World message is printed, print the user’s name.

Compile and run the program. After entering your name in the input dialog box, click the OK button. Record the results.

Run the program a second time. This time click the Cancel button instead. Record the results.

Print a copy of this last version of HelloWorld.java and be sure that this final version is stored on you memory device in the lab03 directory.
3.9 STRING METHODS

We will look at only a small number of the methods in the String class.

**Experiment 2 StringTest.java**

**Step 1:** To experiment with these methods, create a new program file called StringTest.java similar to HelloWorld.java. Declare the class and create the main method. Inside the main method, create a JFrame frame object, and using an input dialog box, prompt the user for a word, saving the input in the variable String word.

Compile the code and run the program. Record the results.

All methods have a method header, which tells the user how to use the method. The *method header* consists of a return type, the name of the method name followed by a set of parentheses surrounding an optional list of parameters. The parameters, another name for variable, tell the user the number and types of arguments that must be passed to the method when it is invoked. We will look at these String instance methods with headers:

<table>
<thead>
<tr>
<th>return type</th>
<th>method Name</th>
<th>(&lt;optional parameter list&gt;)</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>toUpperCase()</td>
<td></td>
</tr>
<tr>
<td>int</td>
<td>length()</td>
<td></td>
</tr>
<tr>
<td>int</td>
<td>indexOf(String s)</td>
<td></td>
</tr>
<tr>
<td>char</td>
<td>charAt(int index)</td>
<td></td>
</tr>
<tr>
<td>String</td>
<td>substring(int beginIndex, int endIndex)</td>
<td></td>
</tr>
<tr>
<td>String</td>
<td>substring(int beginIndex)</td>
<td></td>
</tr>
</tbody>
</table>

In the explanations of the methods that follow, this refers to the object on which the method is invoked. Therefore, in the statement

```
word.toUpperCase();
```

word is this object.

**String toUpperCase():** is used to create and return a new String whose letters are equivalent to this String but are all UPPER CASE letters. No arguments are passed to the method.

**Step 2:** Add these statements to the end of the main method.

```
word.toUpperCase();
JOptionPane.showMessageDialog(frame, "You entered " + word);
```

Compile the revised code and run the program. Record the results.
**Step 3:** Invoking the method `toUpperCase()` on a `String` does not change the existing `String`, rather, it creates a new `String`. Therefore, to change `word` to its upper case equivalent, the returned `String` needs to be assigned to `word` or to some other `String` variable.

Replace the statement
```
word.toUpperCase();
```
with the statement
```
word = word.toUpperCase();
```
that assigns the returned `String` to `word`.

Compile the revised code and run the program. Record the results.

---

**Step 4:** If we only want to print `word` in upper case letters, then another option is to invoke `toUpperCase` on `word` in the `showMessageDialog` method. **Remove the statement**
```
word = word.toUpperCase();
```
and change the statement
```
JOptionPane.showMessageDialog(frame, "You entered " + word);
```
to
```
JOptionPane.showMessageDialog(frame, "You entered " + word.toUpperCase());
```
Compile the revised code and run the program. Record the results.

---

**Step 5:** Predict what is stored in `word` after the new statement is executed. Add code to test your prediction. Record the code that you added.

```java
int length ()
```
returns the number of characters in this `String`. No arguments are passed to the method.

**Step 6:** Without removing any statements, add to the end of `main` the statement
```
JOptionPane.showMessageDialog(frame,
    word + " has length " + word.length());
```
Compile and run the revised program, entering your full name. Record the two printed messages. Was your prediction in Step 5 correct?
Every character in a String has an index, or position in the String. The index of the first character is 0, and the index of the last character is always one less than the length of the String.

**int indexOf (String s):** expects a String argument and returns an integer representing the index of the first occurrence of s in the String on which the method is invoked.

<table>
<thead>
<tr>
<th>M a r q u e t t e</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8</td>
</tr>
</tbody>
</table>

If String word = "Marquette"; the method call

```
word.indexOf("q")
```

returns 3 because "q" is found beginning at index 3.

**Step 8:** Without removing any statements from main, add the statement

```
JOptionPane.showMessageDialog(frame,
    word + 
    "index of e is " + word.indexOf("e") + "\n" +
    "index of ar is " + word.indexOf("are") + "\n" +
    "index of M is " + word.indexOf("Mar") + "\n" +
    "index of m is " + word.indexOf("m"));
```

as the last statement in main. Predict the output of the last message dialog box if "Marquette" is entered. Then, compile and run the revised program, entering "Marquette". Correct your predictions if necessary.

---

**Answer these questions:**

1. If the argument is contained in the object more than once, which index is returned?

2. If the argument is not contained in the object, what index is returned?

**char charAt(int index):** An integer argument representing an index into this String, must be passed to the method. The character at index is returned.

**Step 9:** Without removing any statements, add the statement

```
JOptionPane.showMessageDialog(frame,
    word + 
    "char at index 0 is " + word.charAt(0) + "\n" +
    "char at index 3 is " + word.charAt(3) + "\n" +
```
"char at index 8 is " + word.charAt(8));
as the last statement in main. Predict the output of the last message dialog box if "Marquette" is entered. Then, compile and run the revised program, entering "Marquette". Correct your predictions if necessary.

Run the program one more time. This time enter "Warriors" when prompted for a word. Record what happens. Can you explain why this happens?

String substring (int beginIndex, int endIndex): Two integer arguments representing indices into this String must be passed to the method. A substring of this String, beginning with the character at beginIndex and up to, but not including, the character at endIndex is returned.

String substring (int beginIndex): One integer argument representing an index into this String must be passed to the method. A substring of this String, beginning with the character at beginIndex and going to the end of this String is returned.

For example, if word is "Marquette" then the expression

word.substring(3,6)

returns "que", which is the substring from the character at index 3 up to, but not including, the character at index 6. However, the expression

word.substring(3)

returns "quette", which is the substring from the character at index 3 to the end of the string.

Step 10: Without removing any statements, add the statement

JOptionPane.showMessageDialog(frame,
name + "\n" +
"substring from 1 to 5 is " + name.substring(1,5) + "\n" +
"substring from 1 is " + name.substring(1) + "\n" +
"substring from 4 to 8 is " + name.substring(4,8) + "\n" +
"substring from 4 is " + name.substring(4) + "\n" +
"substring from 0 to 1 is " + name.substring(0,1) + "\n" +
"substring from 0 to length is " +
name.substring(0,name.length()) + "\n" +
name.substring(0,name.length()));
as the last statement in main. Predict the output of the last message dialog if the entered string is "Marquette". Then, compile and run the revised program, entering "Marquette". Correct your predictions if necessary.
Answer this question:

**Overloaded methods.** are two methods in the same class that have the same name. How does the computer know which of the two substring methods to execute?

---

### 3.10 STYLE GUIDELINES

You are expected to follow these guidelines when writing your programs. This is a subset of the Java style guidelines specified by Sun. Following these guidelines makes code more readable and easier to maintain.

**Identifiers:** Following the naming guidelines (See 3.1) established by the authors of the Java API makes code easier to write and to understand. Choose names that indicate the purpose of the class, method or data value. Choosing good names is part of self – documentation.

**Indentation:** Each level of indentation should be four spaces. Tabs should never be used since they are not consistently spaced from program to program or computer to computer. Also, the default tab stop is usually eight spaces. This amount of indentation is too much, since we will soon be writing programs with five or six levels of indentation.

```java
class Sample {
    public static void main(String[] args) {
        JFrame frame = new JFrame();
        // ...
    }
}
```

**Long statements:** should be broken, by entering <Enter>, anyplace in a statement except in the middle of an identifier or in the middle of a String. Notice above, that the long string is broken into two shorter strings that are concatenated. No line should ever be longer than 80 characters.

**Whitespace:** Use whitespace, spaces and blank lines, to make code more readable. Blank lines should be used to separate sections of code that are logically cohesive. Examples of this will be seen when our classes are more involved. Spaces should be used separate operators from operands and to separate words in comma separated lists. For example:

```java
JFrame frame = new JFrame();
```
spaces

String first, middle, last;

Comments: You should include a block comment at the beginning of each of your assignment programs stating your name, and the purpose of the program. Inline comments may be used to add a comment at the end of a line or on a new line. These comments should provide documentation and explanations for code that is not obvious. Do not use comments to explain statements that are obvious.

3.11 POST LAB PROGRAMMING EXERCISES

1. Write a program MadLib.java that prompts the user for words that are either parts of speech or specific types of words, such as noun, verb, adjective, animal, number etc.. The entered words are then printed as part of one or more sentences. For example: Prompt the user, using four separate input dialogs, for an animal, plural noun, number and adjective. If the user enters, respectively, "elephant", "books", "13", and "beautiful", then the program prints, using a message dialog: "My pet ELEPHANT likes to eat BOOKS. He eats 13 times a day and is beginning to look quite BEAUTIFUL." The user-entered words should be capitalized when printed. Make up your own MadLib that uses at least four user-entered strings.

2. Write a program Pattern.java that prompts the user for a four letter word using an input dialog. Then, print the following pattern with the word using the substring method and string concatenation. For example: If the user enters the word "Java", your program should print one of the following in a message dialog,

A.  J      B.  J
   Ja     Ja
   Jav    Jav
   Java   Java
   Ja     Ja
   J      J

3. Use String methods to write a program called Name.java that prompts the user for his First Middle Last names in one input dialog. That is, a string such as "Thomas Patrick Jones" will be entered. Use a message dialog box to print the name Last, First MiddleInitial. Therefore, the string "Jones, Thomas P." would be printed in a message dialog.

4. Use String methods to write a program called Jumble.java that prompts the user for his First Last names in one input dialog. That is, a string such as "Thomas Jones" will be entered. Use a message dialog box to print the first and last names with the first letters interchanged. Therefore, the String " Jhomas Tomes" would be printed in a message dialog.

5. Use String methods to write a program called TitleCase.java that prompts the user for his First Last names in one input dialog. That is, a string such as "THOMAS jones" could be entered. Use a message dialog to print the name in title case, i.e. first letters of each name are capitalized, all other letters are in lower case. Therefore, the String " Thomas Jomes" would be printed in a message dialog. Note: The String class contains the method String toLowerCase() which returns the lower case equivalent of the String on which it is invoked.

6. Use String methods to write a program Reverse.java that prompts the user for a four letter word using an input dialog. Use a message dialog to print the word in reverse order. For example, if the user enters the word "Java", your program should print.

    "Java"
Hint: Use the charAt method and string concatenation. Begin with the empty string:
String reverse = "";
7. Write a program `Rhyme.java` that prints the following nursery rhyme in a message dialog. Spacing is important!!

```java
Jack and Jill p a hill
    went u
To fetch a pail of water.
Jack fell d o
    w
n and broke his crown.
And Jill came t u
    m b
l i
n g after
```

8. Write a Java application, `NameGame.java`, that reads in a single name from the user and then uses that name to print the following verse. Two sample outputs are given. The portions of the verse that depend on the user input are highlighted.

| User input = "Roger"                          | User input = "Caroline"
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Roger, Roger, bo-Boger</td>
<td>Caroline, Caroline, bo-Baroline</td>
</tr>
<tr>
<td>Banana-fana fo-Foger</td>
<td>Banana-fana fo-Faroline</td>
</tr>
<tr>
<td>Fee, Fie, mo-Moger</td>
<td>Fee, Fie, mo-Maroline</td>
</tr>
<tr>
<td>ROGER!</td>
<td>CAROLINE!</td>
</tr>
</tbody>
</table>

9. Write a Java application, `Testing.java`, that tests whether the following are legal Java expressions. You should determine this by trying to print each of the following. Write up your answer by showing your test code. And, include a written explanation of what is wrong with any of the illegal expressions. Explain the result of any of the legal expressions.

a. `toUpperCase("Java");`
b. "Java".substring()
c. "I love ".concat("Java")
d. "Java".Length()
e. "Java".charAt(4)
f. "Java".charAt(1, 2)

10. In 3.6, we used the escape character '\n'. The additional escape characters, used for printing, are:

- `\"` is used to print a double quote
- `\'` is used to print a single quote
- `\t` is used to print a tab
- `\` is used to print a backslash
- `\r` is used to print a return, which returns the carriage to the beginning of the same line
- `\n` is used to print a newline, which returns the carriage to the beginning of the next line

Write a Java application, `Escape.java`, that prints this sentence in a message dialog box
```
She said, "Hit the ON\OFF switch on Mike's computer."
```