



POLYTECHNIC UNIVERSITY OF THE PHILIPPINES
COLLEGE OF COMPUTER AND INFORMATION SCIENCES
DEPARTMENT OF COMPUTER SCIENCE | DEPARTMENT OF INFORMATION TECHNOLOGY

INSTRUCTIONAL MATERIAL FOR STUDENTS

Compiled by:

ALFRED M. PAGALILAWAN
ELIAS A. AUSTRIA
RACHEL A. NAYRE

TABLE OF CONTENTS

TOPIC		PAGE
Lesson 1	Introduction to OOP and Basic Components of Java	
	Object-Oriented Programming	5
	Features of OOP	5
	History of Java	5
	Java Platform	8
	Parts of Java Language	9
	Variables and Data Type	10
	Operators	12
	Exercises	15
Lesson 2	Input and Control Statements	
	Input Statements	18
	Displaying Text in a Dialog Box	20
	Control Flow Statements	
	if-else statement	24
	switch statement	25
	Loop statements	27
	Exercises	30
Lesson 3	Arrays and Strings	
	Array	32
	String	
	String Methods	33
	Exercises	36
Lesson 4	Methods	
	Definition of Method	39
	Passing Array to Method	44
	Method Overloading	45
	Exercises	46
Lesson 5	Classes and Objects	
	Class and Object Definition	49
	Instance Method	50
	Declaring Objects	51
	Instance Object	55
	Laboratory Exercises	56
Lesson 6	Overloading Constructors	
	Constructor Method	59
	Overloading Constructors	63
	Laboratory Exercises	66
Lesson 7	Prewritten Classes and Methods	
	Prewritten Constants and Methods	68
	Prewritten Imported Methods	70
	Laboratory Exercises	72
Lesson 8	Frames	
	GUI Components	74
	Swing Overview	74
Lesson 9	Database	

	Database Definition	108
	JDBC	108
	Steps in Creating DB	109
	SQL Syntax	109
	Laboratory Exercises	122
Bibliography		123

Lesson 1

Introduction to OOP and Basic Components of Java

Overview:

This lesson covers object oriented programming and its features, history of Java language and characteristics, Java platform and basic components of Java language

Objectives:

- To enumerate the features of OOP
- To describe the characteristics of Java language
- To understand how the Java program works
- To form variable name based on the rules
- To list the data types and its sizes
- To understand how the Java operators work

Object-Oriented Programming

When you program without using OOP (Object-Oriented Programming), you are programming procedurally. A *procedural* program is a step-by-step program that guides the application through sequence of instructions. A procedural program executes each statement in the literal order of its commands, even if those commands cause the program to branch into all directions. C, Pascal, Qbasic and COBOL are all examples of procedural programming languages

Object-Oriented Programming is an extension of procedural programming in which you take slightly different approach to writing computer programs. Thinking in an object-oriented manner involves envisioning program components as objects that are similar to concrete objects in the real world. Writing object-oriented programs involves both creating objects and creating applications that use those objects. This approach is better suited to most tasks because most problems are complex and multifaceted and do not conform easily to linear approach.

Some Features of OOP

1. **Encapsulation**
Capturing data and keeping it safely and securely from outside interface.
2. **Inheritance**
This is the process by which a class can be derived from a base class with all features of base class and some of its own. it increases code reusability.
3. **Polymorphism**
Ability to exist in various forms.
4. **Abstraction**
The ability to represent data at a very conceptual level without any details. It hides information to other class.
5. **Message Passing**
Objects communicates through invoking methods and sending data to them. This feature of sending and receiving information among objects through function parameters.

Brief History of Java

Java, having been developed in 1991, is a relatively new programming language. At that time, **James Gosling** from Sun Microsystems and his team began designing the first version of Java aimed at programming home appliances which are controlled by a wide variety of computer processors.

Gosling's new language needed to be accessible by a variety of computer processors. In 1994, he realized that such a language would be ideal for use with web

browsers and Java's connection to the internet began. In 1995, Netscape Incorporated released its latest version of the Netscape browser which was capable of running Java programs.

Why is it called Java? It is customary for the creator of a programming language to name the language anything he/she chooses. The original name of this language was Oak, until it was discovered that a programming language already existed that was named Oak. As the story goes, after many hours of trying to come up with a new name, the development team went out for coffee and the name Java was born.

While Java is viewed as a programming language to design applications for the Internet, it is in reality a general all- purpose language which can be used independent of the Internet.

What is Java?

Just Another Vague Acronym!!!

Definitely not! Java is two things: a programming language and a platform.

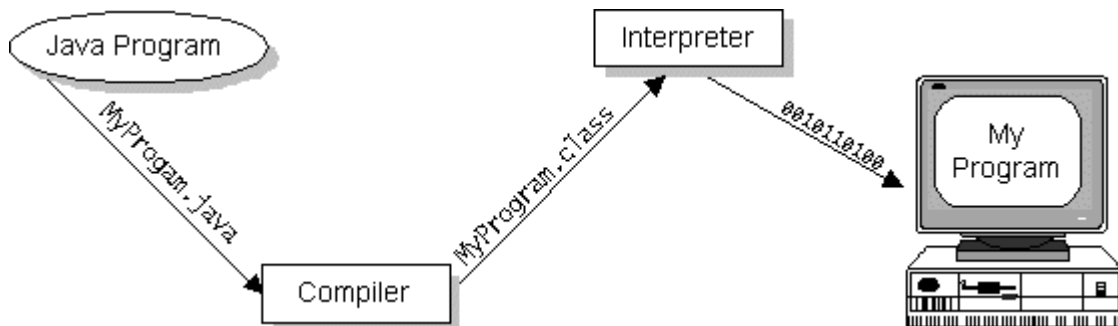
The Java programming language was developed by SUN (Stanford University Network) Microsystems as an object-oriented language that is used both for general-purpose business programs and interactive World Wide Web-based internet programs. Some of the advantages that made Java programming language so popular in recent years are its security features, and the fact that it is architecturally neutral, which means that you can use Java to write program that will run on any platform.

Java is a high-level programming language that is all of the following:

- ☐ Simple
- ☐ Object-oriented
- ☐ Distributed
- ☐ Interpreted
- ☐ Robust
- ☐ Secure
- ☐ Architecture-neutral
- ☐ Portable
- ☐ High-performance
- ☐ Multithreaded
- ☐ Dynamic

Java is also unusual in that each Java program is both compiled and interpreted. With a compiler, you translate a Java program into an intermediate language called **Java bytecodes** or simply **bytecode**—the platform-independent codes interpreted by the Java interpreter. With an interpreter, each Java bytecode instruction is parsed and run on

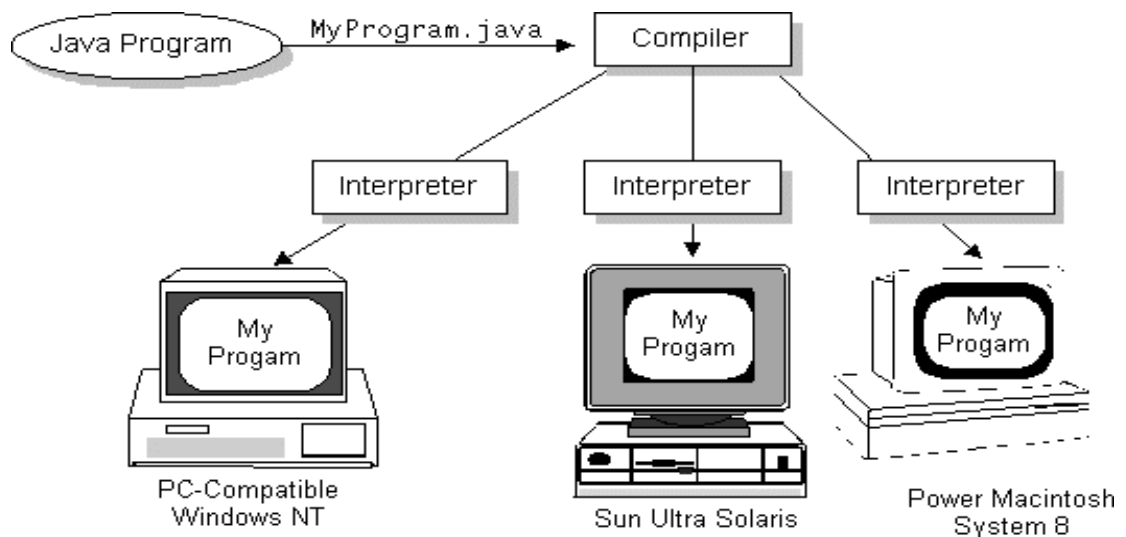
the computer. Compilation happens just once; interpretation occurs each time the program is executed. This figure illustrates how this works.



You can think of Java bytecodes as the machine code instructions for the *Java Virtual Machine* (Java VM). Every Java interpreter, whether it's a Java development tool or a Web browser that can run Java applets, is an implementation of the Java VM. The Java VM can also be implemented in hardware.

Java bytecodes help make "write once, run anywhere" possible. You can compile your Java program into bytecodes on any platform that has a Java compiler. The bytecodes can then be run on any implementation of the Java VM. For example, the same Java program can run on Windows NT, Solaris, and Macintosh

You can think of Java bytecodes as the machine code instructions for the *Java Virtual Machine* (Java VM). Every Java interpreter, whether it's a Java development tool or a Web browser that can run Java applets, is an implementation of the Java VM. The Java VM can also be implemented in hardware.



The Java Platform

A platform is the hardware or software environment in which a program runs. The Java platform differs from most other platforms in that it's a software-only platform that runs on top of other, hardware-based platforms. Most other platforms are described as a combination of hardware and operating system.

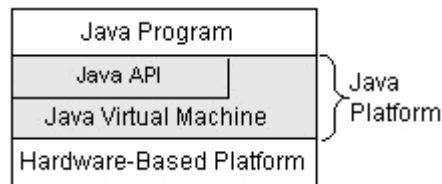
The Java platform has two components:

- The *Java Virtual Machine* (Java VM)
- The *Java Application Programming Interface* (Java API)

The Java VM the base for the Java platform and is ported onto various hardware-based platforms to name a few common in the market are Microsoft Internet Explorer and Netscape Navigator.

The Java API is a large collection of ready-made software components that provide many useful capabilities, such as graphical user interface (GUI) widgets. The Java API is grouped into libraries (*packages*) of related components.

The following figure depicts a Java program, such as an application or applet, that's running on the Java platform. As the figure shows, the Java API and Virtual Machine insulates the Java program from hardware dependencies.



As a platform-independent environment, Java can be a bit slower than native code. However, smart compilers, well-tuned interpreters, and just-in-time bytecode compilers can bring Java's performance close to that of native code without threatening portability.

What Can Java Do?

Probably the most well-known Java programs are *Java applets*. An applet is a Java program that adheres to certain conventions that allow it to run within a Java-enabled browser. This you will be learning as we go through each lessons.

The most common types of programs in java are probably applets and applications, where a Java application is a standalone program that runs directly on the Java platform.

Java Program: In Birds Eye View

```
/**
 * The HelloWorld class implements an application that
 * simply displays "Hello World!" to the standard output.
 */
```

```
public class HelloWorld //class header
```



```

{
    public static void main(String args[])
    {
        System.out.println("Hello World!");
    }
}

```

Parts of a Java Program

1. Class name
2. Method main

```

1  /**
2   * The HelloWorld class implements an application that
3   * simply displays "Hello Java!" to the standard output.
4   */
5   public class HelloWorld
6   {
7       public static void main(String args[])
8       {
9           System.out.println("Hello World!");
10      }
11  }

```

Programming Analysis:

Line 1 to 4: Are what we call **doc comments**. The program simply ignores any statements that are inside a comments.

Java Comments: there are 3 kinds of comments in Java

/** */	documentation comment	this is used at the beginning of the program
/* */	block comment	used to comment a block or lines of program
//	line comment	used to comment a line of program

Line 5: Is the class heading. Everything that you use must be a part of a class. When you write public class HelloWorld, you are defining a class named HelloWorld. The reserved word public is an access modifier. An access modifier defines the circumstances under which a class can be accessed. Public access is the most liberal type of access.

Line 6: Begins the body of the program with the use of open brace {

Line 7: The first method that java program executes is the **main()**. In the method header, static means showing little change or stationary. Static also means unchanging and indicates that every member created for the HelloWorld class will have an identical,

unchanging `main()`. Within the method `main(String[] args)` is an argument. An argument consist of information that a method requires to perform its task. String represents a Java class that can be used to represent character strings. The identifier **args** is used to hold any Strings that might be sent to `main()`. The `main()` method could do something with those arguments such as printing them but in line 7, the `main()` does not actually use the `args` identifier. Nevertheless, you must place an identifier within the `main()` method's parenthesis. The identifier need not be named `args`. It could be any legal Java identifier but by convention, we use the `args` identifier.

Line 8: Begins the body of the `main()` with the use of the symbol `{`

Line 9: This statement displays **Hello World!** on the screen. Within the statement `System.out.println("Hello World");` `System` is a class. `Out` is an object. The `out` object represents the screen. One of the `System`'s object is the `out` object. `Println` is a method that prints a line of output on the screen then positions the cursor on the next line. The dots in the statement `System.out.println` are used to separate Class, object and method. *Classes, objects and methods are further discussed in the next module*

Line 10: Ends method `main()`

Line 11: Ends class `HelloWorld`

Where To code your Java Programs?

You can use Notepad, Textpad or Eclipse to create java programs, be sure that the file name is the same as the class name and the extension name should be **java**.

example:

HelloWorld.java

Rules Of Java Programming.

Java just like its predecessor C is case-sensitive language. Always check for the proper casing when writing and encoding java program. Statements must terminate with ;

Variables and Data Types

Variables are the nouns of a programming language-that is, they are the entities (values and data) that act or are acted upon. A variable declaration always contains two components: the type of the variable and its name. The location of the variable declaration, that is, where the declaration appears in relation to other code elements, determines its scope.

Data Types

All variables in the Java language must have a data type. A variable's data type determines the values that the variable can contain and the operations that can be performed on it. For example, the declaration `int count` declares that `count` is an integer (`int`). Integers can contain only integral values (both positive and negative), and you can use the standard arithmetic operators (+, -, *, and /) on integers to perform the standard arithmetic operations (addition, subtraction, multiplication, and division, respectively).

Primitive Data Type

Type	Size/Format	Description	Range
<i>(integers)</i>			
byte	8-bit two's complement	Byte-length integer	-128 to 127
short	16-bit two's complement	Short integer	-32,768 to 32,767
int	32-bit two's complement	Integer	-2,147,483,648 to 2,147,483,647
long	64-bit two's complement	Long integer	
<i>(real numbers)</i>			
float	32-bit IEEE 754	Single-precision floating point	
double	64-bit IEEE 754	Double-precision floating point	
<i>(other types)</i>			
char	16-bit Unicode character	A single character	
boolean	true or false	A boolean value (true or false)	

Variable Names

A program refers to a variable's value by its name.

In Java, the following must hold true for a variable name

1. It must be a legal Java identifier comprised of a series of Unicode characters. Unicode is a character-coding system designed to support text written in diverse human languages. It allows for the codification of up to 65,536 characters, currently 34,168 have been assigned. This allows you to use characters in your Java programs from various alphabets, such as Japanese, Greek, Cyrillic, and Hebrew. This is important so that programmers can write code that is meaningful in their native languages.
2. It must not be a keyword or a boolean literal (true or false).
3. It must not have the same name as another variable whose declaration appears in the same scope.

By Convention: Variable names begin with a lowercase letter and class names begin with an uppercase letter. If a variable name consists of more than one word, such as **is visible**, the words are joined together and each word after the first

begins with an uppercase letter, therefore **isVisible** is proper variable name.

Variable Initialization

Local variables and member variables can be initialized with an assignment statement when they're declared. The data type of both sides of the assignment statement must match.

```
int count = 0;
```

The value assigned to the variable must match the variable's type.

Final Variables

You can declare a variable in any scope to be final, including parameters to methods and constructors. The value of a final variable cannot change after it has been initialized.

To declare a final variable, use the final keyword in the variable declaration before the type:

```
final int aFinalVar = 0;
```

The previous statement declares a final variable and initializes it, all at once. Subsequent attempts to assign a value to aFinalVar result in a compiler error. You may, if necessary, defer initialization of a final variable. Simply declare the variable and initialize it later, like this:

```
final int blankfinal;  
...  
blankfinal = 0;
```

A final variable that has been declared but not yet initialized is called a blank final. Again, once a final variable has been initialized it cannot be set and any later attempts to assign a value to blankfinal result in a compile-time error.

Operators

Arithmetic Operators

The Java language supports various arithmetic operators for all floating-point and integer numbers. These include + (addition), - (subtraction), * (multiplication), / (division), and % (modulo). For example, you can use this Java code to add two numbers:

```
addThis + toThis
```

Or you can use the following Java code to compute the remainder that results from dividing divideThis by byThis:

```
divideThis % byThis
```

This table summarizes Java's binary arithmetic operations:

Operator	Use	Description
+	op1 + op2	Adds op1 and op2
-	op1 - op2	Subtracts op2 from op1
*	op1 * op2	Multiplies op1 by op2
/	op1 / op2	Divides op1 by op2
%	op1 % op2	Computes the remainder of dividing op1 by op2

The + and - operators have unary versions that perform the following operations:

Operator	Use	Description
+	+op	Promotes + to int if it's a byte, short, or char
-	-op	Arithmetically negates op

There also are two short cut arithmetic operators, ++ which increments its operand by 1, and -- which decrements its operand by 1.

Operator	Use	Description
++	op++	Increments op by 1; evaluates to value before incrementing
++	++op	Increments op by 1; evaluates to value after incrementing
--	op--	Decrements op by 1; evaluates to value before decrementing
--	--op	Decrements op by 1; evaluates to value after decrementing

Relational and Conditional Operators

A relational operator compares two values and determines the relationship between them. For example, != returns true if the two operands are unequal.

Operator	Use	Return true if
>	op1 > op2	op1 is greater than op2
>=	op1 >= op2	op1 is greater than or equal to op2
<	op1 < op2	op1 is less than op2
<=	op1 <= op2	op1 is less than or equal to op2
==	op1 == op2	op1 and op2 are equal

<code>!=</code>	<code>op1 != op2</code>	op1 and op2 are not equal
-----------------	-------------------------	---------------------------

Relational operators often are used with the conditional operators to construct more complex decision-making expressions. One such operator is `&&`, which performs the *boolean and* operation. For example, you can use two different relational operators along with `&&` to determine if both relationships are true.

Java supports five binary conditional operators, shown in the following table:

Operator	Use	Returns true if
<code>&&</code>	<code>op1 && op2</code>	op1 and op2 are both true, conditionally evaluates op2
<code> </code>	<code>op1 op2</code>	either op1 or op2 is true, conditionally evaluates op2
<code>!</code>	<code>! op</code>	op is false
<code>&</code>	<code>op1 & op2</code>	op1 and op2 are both true, always evaluates op1 and op2
<code> </code>	<code>op1 op2</code>	either op1 or op2 is true, always evaluates op1 and op2

In addition, Java supports one other conditional operator--the `?:` operator. This operator is a ternary operator and is basically short-hand for an if-else statement:

expression ? op1 : op2

The `?:` operator evaluates expression and returns op1 if it's true and op2 if it's false.

example: `int x = 5, y = 6, z = 0;`

`x > y ? z=x : z=y;`

In the previous example since the expression `x > y` evaluates to false `x++` operator is performed.

Assignment Operators

You use the basic assignment operator, `=`, to assign one value to another. The `countChars` method uses `=` to initialize `count` with this statement:

`int count = 0;`

Java also provides several short cut assignment operators that allow you to perform an arithmetic, logical, or bitwise operation and an assignment operation all with one operator. Suppose you wanted to add a number to a variable and assign the result back into the variable, like this:

`i = i + 2;`

You can shorten this statement using the short cut operator `+=`.

`i += 2;`

The two previous lines of code are equivalent.

This table lists the shortcut assignment operators and their lengthy equivalents:

Operator	Use	Equivalent to
<code>+=</code>	<code>op1 += op2</code>	<code>op1 = op1 + op2</code>
<code>-=</code>	<code>op1 -= op2</code>	<code>op1 = op1 - op2</code>

<code>*=</code>	<code>op1 *= op2</code>	<code>op1 = op1 * op2</code>
<code>/=</code>	<code>op1 /= op2</code>	<code>op1 = op1 / op2</code>
<code>%=</code>	<code>op1 %= op2</code>	<code>op1 = op1 % op2</code>

Expressions

Expressions perform the work of a Java program. Among other things, expressions are used to compute and assign values to variables and to help control the execution flow of a program. The job of an expression is two-fold: perform the computation indicated by the elements of the expression and return some value that is the result of the computation.

Definition: An *expression* is a series of variables, operators, and method calls (constructed according to the syntax of the language) that evaluates to a single value.

Exercises

Write the Java statement for the following: (Statements must be properly terminated)

1. Declare x,y and z to be variables of type int with values 5,10 and 15 respectively
2. Store the result of the comparison between x and y to z. Use the comparison operator >
3. Store the sum of x and y to variable sum
4. Display the value of x,y and z using only one print or println statement in this method
: The sum of 5 and 10 is 15
5. Declare isBoolean to have an initial value of "true"

Laboratory Exercises

1. Write, compile and test a program that displays the following patterns on the screen

a)	<pre> * *** ***** ***** *** * </pre>	b)	<pre> 1 12 123 1234 12345 </pre>
----	--------------------------------------	----	----------------------------------

2. Write a program that prints your complete name on the first line and course, year and section on the second line using only one System.out.println statement.

3. Given the following variables and their values:

Sagot = true
A = 2
B = 4
Letter = 'c'
Pi = 3.14

Requirements:

a) Write a Java program that prints the values of the variables above in this manner

The value of A is 2 while B is 4
Letter c
Initial value of Sagot is true
Pi contains the value 3.14
Sagot is now false

** for the last line of output, determine if A is greater than B

b) Display the product of A and B without declaring another variable. Display the result in this manner : $2 * 4 = 8$

4. Write a program that inputs the hourly rate and number of hours worked.

Compute and display the gross pay (hourly rate * hours worked), your withholding tax, which is 15% of your gross pay and your net pay (gross pay – withholding tax).

Sample output : Hourly rate : 104.65
 Hours worked : 22
 Gross pay : 2302.3
 Withholding tax : 345.345
 Net pay : 1956.955

5. Write a program that declares a variable that represents the minutes worked on a job and assign a value. Display the value in hours and minutes. For example 125 minutes becomes 2 hours and 5 minutes

Sample output : Given : 125 minutes
 Converted hours : 2 hours and 5 minute/s

6. Write a program that displays the conversion of 1887 into 1000's, 500's, 100's, 50's 20's ,10's, 5's and 1's

Sample output Cash on hand : 1887
 Denominations :
 1000 – 1
 500 - 1
 100 - 3
 50 – 1
 20 – 1
 10 – 1
 5 - 1
 1 - 2

Lesson 2: Input statements and Control Flow Statements

Overview:

This lesson covers the different input statements and control flow statements.

Objectives:

- To utilize the different input statements
- To apply the output statement in the program
- To identify the control flow statements

Java Input Statements

To put data into variables from the standard input device (keyboard) we can use the predefined Java class such as `BufferedReader` and `Scanner`.

- **BufferedReader**

A java class to read the text from an input stream by buffering characteristics that seamlessly reads characters, arrays or lines.

Syntax:

```
BufferedReader buffread = new BufferedReader(new
    InputStreamReader(System.in));
```

This statement creates the input stream object buffread (user-defined word) and associates it with the standard input device. (Note that the BufferedReader is a predefined Java class and the above statement creates buffread to be an object of this class). The object buffread reads the input as follows:

- a. for integer
`Integer.parseInt(buffread.readLine())`
- b. for floating-point number
`Double.parseDouble(buffread.readLine())`
- c. for strings
`buffread.readLine()`

Example:

```
import java.io.*;
class InputData1
{ public static void main(String[] args) throws
    IOException
    { String firstname, lastname;
      int age;
      double weight;
      BufferedReader buffread = new BufferedReader(new
          InputStreamReader(System.in));
      System.out.print("Enter firstname: ");
      firstname = (buffread.readLine());
      System.out.print("Enter lastname: ");
      lastname = (buffread.readLine());
      System.out.print("Enter age: ");
      age = Integer.parseInt(buffread.readLine());
      System.out.print("Enter weight: ");
      weight = Double.parseDouble(buffread.readLine());

      System.out.println("Your Name is: " + firstname + "
          " + lastname);
      System.out.println("Your Age is: " + age);
      System.out.println("Your Weight is: " + weight);
    }
}
```

Note: When using `BufferedReader`, import `java.io.*`; (java input/output package) must be declared before the class declaration and throws `IOException` keywords must be included in the main declaration.

- **Scanner**

Used to get user input of the primitive types.

Syntax:

```
static Scanner console = new Scanner (System.in);
```

This statement creates the input stream object `console` and associates it with the standard input device. (Note that the `Scanner` is a predefined Java class and the above statement creates `console` to be an object of this class). The object `console` reads the next input as follows:

- a. for integer
`console.nextInt()`
- b. for floating-point number
`console.nextDouble()`
- c. for string
`console.next()`
- d. for string with a line
`console.nextLine()`

Example:

```
import java.util.*;
class InputData2
{ static Scanner console = new Scanner(System.in);
  public static void main(String[] args)
  { String firstname, lastname;
    int age;
    double weight;
    System.out.print("Enter firstname and lastname
                     separated by space: ");
    firstname = console.next();
    lastname = console.nextLine();
    System.out.print("Enter age and weight separated
                     by space: ");
    age = console.nextInt();
    weight = console.nextDouble();
    System.out.println("Your Name is: " + firstname
                      + " " + lastname);
    System.out.println("Your Age is: " + age);
    System.out.println("Your Weight is: " + weight);
  }
}
```

Note: When using `Scanner`, import `java.util.*`; (java utilities package) must be declared before the class declaration.

- **Parsing (Parse method)**

A static method and can have one argument or two that reads the value of one object to convert it to another type.

Example:

```
class InputData3
{ public static void main(String[] args)
{ String firstname, lastname;
  int age;
  double weight;
  firstname = args[0];
  lastname = args[1];
  age = Integer.parseInt(args[3]);
  weight = Double.parseDouble(args[4]);
  System.out.println("Your Name is: " + firstname
    + " " + lastname);
  System.out.println("Your Age is: " + age);
  System.out.println("Your Weight is: " + weight);
}
}
```

Note: When using parsing/parse, you must input the value/s when executing the program. *Example: InputData3 juan cruz 20 110.00.*

- **InputDialog Box**

Used to get input from the user.

Example:

```
import javax.swing.JOptionPane;
public class InputData4
{ public static void main(String args[])
{ String name;
  name=JOptionPane.showInputDialog("Enter your name ");
  System.exit(0);
}
}
```

Displaying Text in a Dialog Box

Java's numerous predefined classes are grouped into categories of related classes called packages. The packages are referred collectively as the Java class library, or Java applications programming interface (API). The packages of the Java API are split into *core packages* and *extension packages*. The names of the packages begin with either "java" (core packages) or "javax" (extension packages). Java's class JOptionPane provides prepackaged dialog boxes that enable programs to display messages to users.

Example:

```
1 // Java extension package
2 import javax.swing.JOptionPane;
3 public class Dbox
4 {
```

```

5    public static void main (String args[ ])
6        {JOptionPane.showMessageDialog(null,"Welcome to \nJava 2 programming");
7        System.exit(0); //terminate application
8        } //end of method main
9    } // end of class Dbox

```

output



Line 2 tells the compiler to load the JOptionPane class from javax.swing package. This package contains many classes that help in defining graphical user interfaces (GUIs). GUI components facilitate data entry by the user of your program and formatting or presentation of data outputs to the user of your own program. Line 6 indicate a call to method showMessageDialog of class JOptionPane. The method in line 6 contains 2 arguments. The first argument helps the Java application determine where to position the dialog box. When the first argument is “null”, the dialog box appears at the center of the screen. The title bar of the dialog box contains the string **Message**, to indicate that the dialog box is presenting a message to the user. The dialog box automatically contains the OK button that allows the user to dismiss (hide) the dialog box by clicking the button. Line 7 uses a static method *exit* of class System to terminate the application. The argument 0 to method exit indicates that the application has terminated successfully. A non-zero value normally indicates that an error has occurred.

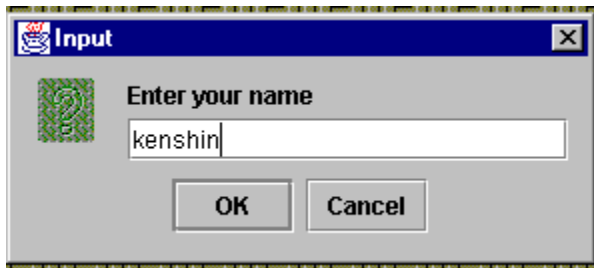
Entering data in a Dialog box

```

1    import javax.swing.JOptionPane;
2    public class Hello
3    { public static void main(String args[])
4        { String name;
5        name=JOptionPane.showInputDialog("Enter your name ");
6        JOptionPane.showMessageDialog(null,"Hello " +
7        name,"Welcome!",JOptionPane.PLAIN_MESSAGE);
8        System.exit(0);
9    }

```

output



user enters "kenshin" then clicks OK

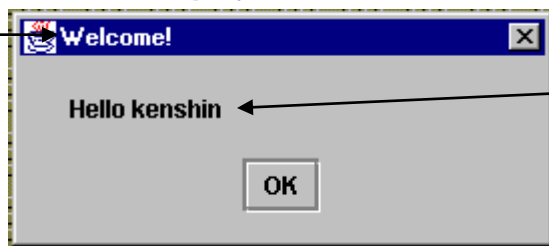


The program in the previous page uses another predefined dialog box from JOptionPane called an input dialog that allows the user to input a value for use in the program. The program also uses a message dialog to display the value entered in the input dialog box. Line 4 declares a string variable named "name" that will hold whatever value is entered in the input dialog. Line 5 reads from the user a string value. Method JOptionPane.showInputDialog displays the input dialog in the previous page. The argument to showInputDialog indicates what the user should type in the textfield. The result of JOptionPane method showInputDialog is assigned to variable name. After the input has been made, line 6 displays the message dialog box. In this example, the method showMessageDialog contains four arguments. The first argument indicates that the message dialog will appear at the center, the second argument is the message to display. In this case the second argument is :

"Hello " + name





The third and fourth arguments represent the string that should appear in the dialog box's title bar and dialog box type respectively. The fourth argument – JOptionPane.PLAIN_MESSAGE is a value indicating the type of message dialog to display. This type of message **does not display an icon** to the left of the message.

argument 3



argument 2

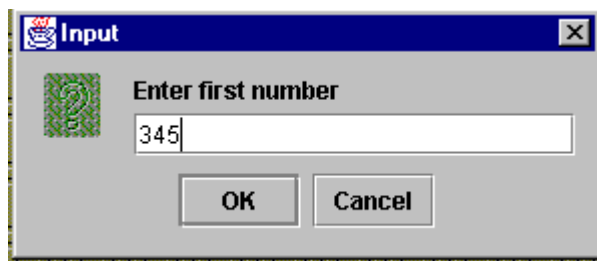
All message dialog types except PLAIN_MESSAGE display an icon indicating to the user the type of message.

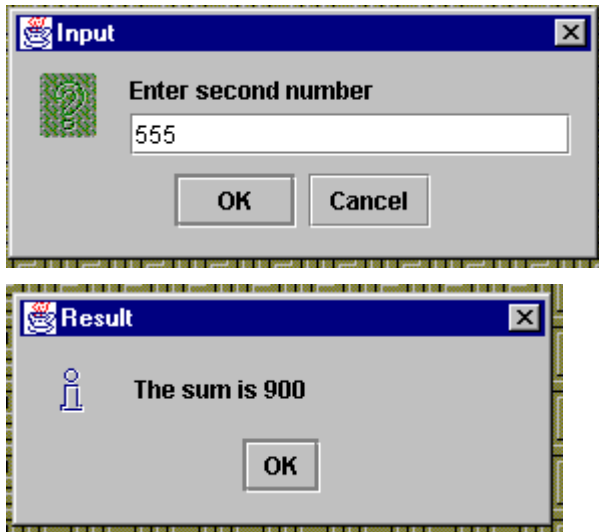
Message dialog type	Icon	Description
.ERROR_MESSAGE		Displays a dialog that indicates an error to the user
.INFORMATION_MESSAGE		Displays a dialog with an informational message to the user. The user can simply dismiss the dialog
.WARNING_MESSAGE		Displays a dialog that warns the user of a potential problem
.QUESTION_MESSAGE		Displays a dialog that poses a question to the user
.PLAIN_MESSAGE		Displays a dialog that simply contains a message with no icon.

Adding 2 integers in a dialog box

```
import javax.swing.JOptionPane;
```

```
public class Sum
{ public static void main(String args[])
  { String firstnum,secondnum;
    int sagot;
    firstnum=JOptionPane.showInputDialog("Enter first number ");
    secondnum=JOptionPane.showInputDialog("Enter second number ");
    sagot = Integer.parseInt(firstnum) + Integer.parseInt(secondnum);
    JOptionPane.showMessageDialog(null,"The sum is " +
sagot,"Result",JOptionPane.INFORMATION_MESSAGE);
    System.exit(0);
  }
}
```





Control Flow Statements

Control flow statements determine the order in which other statements are executed. Java language supports several control flow statements, including:

Statement	Keyword
decision making	If-else, switch-case
loop	for, while, do-while
exception	try-catch-finally, throw
miscellaneous	break, continue, label: , return

The if-else Statement

Java's if-else statement provides your programs with the ability to selectively execute other statements based on some criteria.

This is the simplest version of the if statement: the statement governed by the if is executed if some condition is true. Generally, the simple form of if can be written like this:

```
if (expression)  
statement to do if expression is true;
```

What if you wanted to perform a different set of statements if the *expression* is false? Well, you can use the else statement for that.

```
if (expression)  
statement to do if expression is true;
```

```
else  
statement to do if expression is false;
```

What if you intend to do more than one statements if the expression is true? What about if you intend to do more than one statements if the expression is false? Well just place them in a block – using the open brace as the start and ending it with a close brace.


```

if (expression)
{
    statement1 to do if expression is true;
    statement2 to do if expression is true;
    statementN to do if expression is true;
}
if (expression)
{
    statement1 to do if expression is true;
    statement2 to do if expression is true;
    statementN to do if expression is true;
}
else
{
    statement1 to do if expression is false;
    statement2 to do if expression is false;
    statementN to do if expression is false;
}

```

Other format of if-else statement is the nested if-else. Consider the sample below. Where else is followed by another if.

```

int testscore;
char grade;

if (testscore >= 90)
{
    grade = 'A';
}
else if (testscore >= 80)
{
    grade = 'B';
}
else if (testscore >= 70)
{
    grade = 'C';
}
else if (testscore >= 60)
{
    grade = 'D';
}
else {
    grade = 'F';
}

```

The switch Statement

Use the switch statement to conditionally perform statements based on some expression. For example, suppose that your program contained an integer named month whose value indicated the month in some date. Suppose also that you wanted to display the name of the month based on its integer equivalent. You could use Java's switch statement to perform this feat:

```

int month;
...
switch (month) {
case 1: System.out.println("January"); break;
case 2: System.out.println("February"); break;
case 3: System.out.println("March"); break;
case 4: System.out.println("April"); break;
case 5: System.out.println("May"); break;
case 6: System.out.println("June"); break;
case 7: System.out.println("July"); break;
case 8: System.out.println("August"); break;
case 9: System.out.println("September"); break;
case 10: System.out.println("October"); break;
case 11: System.out.println("November"); break;
case 12: System.out.println("December"); break;
}

```

The switch statement evaluates its expression, in this case, the value of month, and executes the appropriate case statement. Of course, you could implement this as an if statement:

```

int month;
...
if (month == 1) {
    System.out.println("January");
} else if (month == 2) {
    System.out.println("February");
}
...
// you get the idea
...

```

Deciding whether to use an if statement or a switch statement is a judgment call. You can decide which to use based on readability and other factors. Each case statement must be unique and the value provided to each case statement must be of the same data type as the data type returned by the expression provided to the switch statement.

Another point of interest in the switch statement are the break statements after each case. The break statements cause control to break out of the switch and continue with the first statement following the switch. The break statements are necessary because case statements fall through. That is, without an explicit break control will flow sequentially through subsequent case statements. In the previous example, you don't want control to flow from one case to the next, so you have to put in break statements. However, there are certain scenarios when you do want control to proceed sequentially through case statements. Like in the following Java code that computes the number of days in a month according to the old rhyme that starts "Thirty days hath September, April, June and November all the rest is Thirtry-one except February..":

```

int month;
int numDays;
...
switch (month) {
case 1:
case 3:
case 5:
case 7:
case 8:

```

```

case 10:
case 12:
    numDays = 31;
    break;
case 4:
case 6:
case 9:
case 11:
    numDays = 30;
    break;
case 2:
    if (((year%4==0) && !(year % 100 == 0))||(year % 400 == 0) )
        numDays = 29;
    else
        numDays = 28;
    break;
}

```

Finally, you can use the default statement at the end of the switch to handle all values that aren't explicitly handled by one of the case statements.

```

int month;
...
switch (month) {
case 1: System.out.println("January"); break;
case 2: System.out.println("February"); break;
case 3: System.out.println("March"); break;
case 4: System.out.println("April"); break;
case 5: System.out.println("May"); break;
case 6: System.out.println("June"); break;
case 7: System.out.println("July"); break;
case 8: System.out.println("August"); break;
case 9: System.out.println("September"); break;
case 10: System.out.println("October"); break;
case 11: System.out.println("November"); break;
case 12: System.out.println("December"); break;
default: System.out.println("Hey, that's not a valid month!");
    break;
}

```

Loop Statements

```

do {
    statement to do until expression becomes false;
} while (expression) ;

```

```

while (expression) {
    statement to do while expression becomes is true;
}

```

```

for (initialization; expression or terminating condition; increment or decrement)
{
    statement to do while expression is true;
}

```

break statement

The **break** statement has two uses. The first is to terminate a case in the switch statement; the second use is to force immediate termination of a loop, bypassing the normal loop conditional test. When the break statement is encountered inside a loop, the loop is immediately terminated and program control resumes at the next statement following the loop.

```
int x;
for(x=0; x<100; x++)
{
    System.out.println(x);
    if ( x == 10)
        break;
}
```

This prints the numbers **0** through **10** on the screen and then terminates because the break causes immediate exit from the loop, overriding the conditional test **x<100** built into the loop.

continue statement

The continue statement works somewhat like the break statement. But, instead of forcing termination, continue forces the next iteration of the loop to take place, skipping any code in between. For example the following routine displays only odd numbers from 1 to 100:

```
int x = 1;
do {
    if ( x % 2 == 0)
        continue;
    } while ( x != 100);
```

Nested Loops

Just as if statements can be nested, so can loops. You can place a while loop within a while loop or a for loop within a for loop. In short, when we say nested loop, we have a loop within a loop

Example : Try creating a program that displays numbers from 1-5 in this manner

```
1
12
123
```

If the output is that short, you need not bother yourself constructing a nested loop, you can accomplish it with a series of `System.out.println("1")`, followed by another this time

printing "12" and a third one printing "123". However, if the last number increases as it moves to the next row as in:

```
1
12
123
1234
12345
```

and so on then printing it manually is not recommended since you can accomplish it using a single print method to print the number no matter how deep the number goes. Study the program segment below:

```
int x,y;
for (x=1; x<=3;x++) //outer loop
    for(y=1;y<=x;y++) // inner loop
        System.out.print(y);
        System.out.println(); /*forces the cursor to move to the next row when the
inner loop is
                                completed */
```

Analysis:

The outer loop is executed first with x having an initial value of 1. It then performs the next loop where another variable is initialized to 1. This loop will terminate if the value of y is greater than x. At this point, the statement inside the inner loop will only execute once since the value of x is only 1 therefore printing the value of y which is also 1. The cursor does not go down the next line. There is no {} that covers the two print methods therefore the second println statement will be executed once the inner loops condition is satisfied. The outer loops value will only increment once the inner loops condition is satisfied or completed.

Value of x	Value of y
1	1
2	1
	2
3	1
	2
	3

Therefore if you want to increase the number being produced by the loop, change the value of x in the outer loop (changing it to 4 would display the output below).

```
1
12
123
1234
```

Puzzle: How would you generate this output?

```
1
22
333
```

*** you do not have to reconstruct your loop. You only have to change 1 value from the loop.

Laboratory Exercises

Using a nested loop, create a program that produces the following output:

a)	123	b) 321	c) 321	d) 3
	12	32	21	32
	1	3	1	321
e)	123	f) 3	g) 333	h) 111
	23	22	22	22
	3	111	1	3

Lesson 3: Arrays and Strings

Overview:

This lesson covers arrays, strings and string methods.

Objectives:

- To develop programs with array
- To distinguish what string function to be used

Array - a collection of elements having the same data type; array is an object.

Array declaration

```
int[] anintarray = new int[5]; //allocating 5 elements
```

or

```
int anintarray[] = new int[5]; //allocating 5 elements
```

Assigning values

```
anintarray[0] = 100;
```

.

.

```
anintarray[4] = 500;
```

```
int k=110;
```

```
for (int i = 0;i<5;i++)
```

```
{
```

```
    anintarray[i] = k-=10;
```

```
}
```

Initialization

```
int anintarray[] = {100,200,300,400,500};
```

//automatic allocation based on the initial values

or

```
int[] anintarray = {100,200,300,400,500};
```

```
int anintarray[5] = {100,200,300,400,500};
```

// Cannot explicitly declare an array size while declaring initial values

//Class name Array.java

// this program prints the contents of an array

```
public class Array
```

```
{ public static void main(String[] args)
```

```
{ int[] x = {2,4,70,33,3};
```

```
int y;
```

```
System.out.println("Contents of array x ");
```

```
for (y=0;y<=4;y++)
```

```
    System.out.println("x["+y+"]=" + x[y]);
```

```
}
```

```
}
```

output:

x[0]=2

x[1]=4

x[2]=70

x[3]=33

x[4]=3

STRING

String (with a capital S) is one of the new datatype, but actually String is a class built in to Java. The class String is defined in java.lang.String which is automatically imported into every program you write.

Lets look at the following Java code:

Declaration

```
1 String strSample;  
2 StrSample = new String("Hello");
```

Line 1: is the object declaration, String followed by an object name

Line 2: is the object initialization, the keyword String followed by an open-close parenthesis, a string value as its parameter.

Those 2 lines of can be shorten using this line

```
3 String strSample = new String("Hello");
```

Initialization

```
String str1 = new String("Hello!"); //convention
```

OR

```
String str1 = "Hello!"; // String shortcut
```

Assignment

```
str1 = "Hi";
```

Concatenation

```
str1 = str1 + str2;
```

```
str1 = str1 + " Again!";
```

```
str1 = "PG" + 13;
```

```
str1 = 13.1516 + "PI";
```

```
str1 = 143 + 44; // exception : can't convert int to string
```

SAMPLE PROGRAM

```
/** StringDemo.java  
*/  
public class StringDemo  
{  
    public static void main(String args[])  
    {  
        String str1="Hi!"; //declaration with initial value  
        System.out.println(str1);  
        str1 = "Hello"; //assigning new value  
        str1 = str1 + ",World!"; //assigning using concatenation  
        System.out.println(str1);  
    }  
}  
} //end StringDemo
```

Different String Methods

Comparing Strings

1. **equals() method** – returns true only if two Strings are identical in content. Thus a String holding “CSIT ” with a space after “T” is not equivalent to String holding “CSIT” with no space after “T”

Ex.

```
String name1 = "kenshin";
String name2 = "himura";
if (name1.equals(name2))
    System.out.println("same name");
else
    System.out.println("not the same name"); // result
                                         is false

//another example
if (name1.equals("kenshin ") //there's a space after n
    System.out.println("same name"); //result is true
else
    System.out.println("not the same name");

// last example
if (name1.equals("KENSIN"))
    System.out.println("same name");
else
    System.out.println("not the same"); //result is
                                         false
```

7. **equalsIgnoreCase() method** – similar to the equals() but it ignores case when determining if two Strings are equivalent. This method allows you to test entered data without regard to capitalization

```
Ex. String name1 = "kenshin";
    String name2 = "KENSIN";
    String name3 = "Kenshin";
    if (name1.equalsIgnoreCase(name2))
        System.out.println("same name different case"); //result is
                                                         true
    else
        System.out.println("not the same");
// another example
if (name1.equalsIgnoreCase(name3))
    System.out.println("same name different case"); //result is
                                                         true
else
    System.out.println("not the same");
```

8. **toUpperCase() method** – convert any String to its uppercase equivalent

```
Ex. String word1 = "big";
    word1 = word1.toUpperCase();
    System.out.print(word1);           // result : BIG
```

9. **toLowerCase() method** – convert any String to its lowercase equivalent

```
Ex. String word1 = "BIG";
    word1 = word1.toLowerCase();
```

```
System.out.print(word1);           //result :big
```

10. **indexOf() method** – determines whether a specific character occurs within a String. If it does, the method returns the position of the character. The value –1 is returned if the character does not exist. Like array, positions begin at zero

```
Ex.
String name1="KENSHIN";
int pos =0;
pos = name1.indexOf('K');
System.out.println("position of K in KENSHIN is " + pos);
// output : position of K in KENSHIN is 0

//below is another example
pos = name1.indexOf('k');
System.out.println("position of k in KENSHIN is " + pos);
// output : position of k in KENSHIN is – 1
//value returned is negative since no match is found (indexOf is case-sensitive)

// an example wherein there is more than 1 occurrence of the char being searched
pos = name1.indexOf('N')
System.out.println(pos)
// result is 2. It only returns the position of the first occurrence
```

6. **replace() method** – allows you to replace all occurrences of some character within a String.

```
Ex.
String name1 = "kexshix";
String name2;
name2 = name1.replace('x','n');
System.out.println(name2);
//result is kenshin
//this method is also case-sensitive
```

Converting Strings to Numbers

If a string contains all numbers as in "2470", you can convert it from String to integer for computation purposes. To convert a String to an integer, use the Integer class. A method of the Integer class is the `parseInt()` which takes a String argument and returns its integer value. In Java, to parse a String means to break its separate characters into a numeric format.

```
Ex.
String numstr="2470";
int number ;
number = Integer.parseInt(numstr);
number+=1;
System.out.println(number); //result is 2471
```

Array of Strings

Declaration String[] names = new String[4]; Assigning Values names[0] = "Beth"; . . names[3] = "Arnie";	Initialization String[] names = {"Beth","Mely","Pearl","Arnie"}; or String names[] = {"Beth","Mely","Pearl","Arnie"};
---	--

```
//A complete example of a program containing an array of strings
//class name :Strings.java
public class Strings
{ public static void main(String[] args)
  { String[] jleague = {"Superman","Batman","Hawkgirl","Flash","Green Lantern"};
    int x;
    System.out.println("Members of the Justice League");
    for (x=0;x<=4;x++)
      System.out.println(jleague[x]);
  }
}
```

```
output:
Members of the Justice League
Superman
Batman
Hawkgirl
Flash
Green Lantern
```

Lesson 4 Exercises

- I. Write the Java statement for the following.
 1. Declare an array of integer named *numbers* allocating 10 elements

 2. Declare an array of integer named *num2* with initial values 2,4,6,8,and 10

 3. Store the value 10 to the first index of the array used in number 1

 4. Declare an array of String named *dbz* allocating 4 elements

 5. Assign “gohan” to the first index of the array used in number 4

 6. Declare an array of String named *jleague* with values “Clark, Bruce and John”

 7. Display the value of the first index in array *jleague*

8 Display all the values of array *numbers* (use a loop)

9-10. Increase all the values of array *numbers* by 1 then display the new values (use a loop)

Laboratory exercise

Given an array named *n* with values 33,2,70,4,52,42,8,35,9,211

Write a program that will :

- a) Separate all odd from even numbers
- b) Display the highest number (without sorting)
- c) Display the lowest number (without sorting)
- d) Sort the numbers in ascending order

Note: You are to write a separate program for every requirement displaying first the value of the array. Assign your own class name and provide your own screen display.

Lesson 4

Using Methods

Overview:

This lesson covers methods and how to use them.

Objectives:

- To know how declare methods
- To utilize the different methods in the program

What is a method?

- Method is a blocked statement that performs a specific task/s.
- We use the term procedure, function or sub-routines in a procedural-language while we use the term method, in an object-oriented language.

Different Samples of method

Simple Method

- returns no value & accepts no argument

```
/** File name: MySecond
    Version : 1
 */

public class MySecond
{
    public static void main(String args[])
    {
        greet();
    } // end main

    public static void greet()
    {
        System.out.println("Greetings from method greet()!");
    } //end greet()

} //end class
```

Output:
Greetings from method greet()!

Analyzing our program

```
public class MySecond
{
    public static void main(String args[])
    {
        greet(); // method call
    }
} // end main

public static void greet() // method header
{
    System.out.println("Greetings from method greet()!");
} //end greet() // method definition
} //end class
```

Parts Of A Method Header

1. **Access modifier** - The access modifier for a method can be any of the modifiers public, private, friendly, protected, private protected or static. Most often methods are given public access

2. Return type
3. Method name
4. Parameter List

Methods with argument

- Method that require a single argument but does not return any value
- Method with more than one argument but does not return any value
- Method overloading
 - A method with the same name but differs on the parameter list.

Methods that require a single argument with no return value

Some methods require additional information. If a method could not receive communications from you, called arguments, then you would have to write an infinite number of methods to cover every possible situation. For instance, if you design a method that squares numeric values, it makes sense to design a square method that you can supply with an argument that represents the value to be squared, rather than having to create method square1(), square2() and so on. When you write the method declaration for a method that can receive an argument, you need to include the following items within the method declaration parenthesis:

- The type of argument
- A local name for the argument

The example below is a class that contains a method that squares a number

```
public class Square
{ public static void main(String[] args)
  { int num=30;
    sqr(20); //method call with value 20 being sent
    sqr(num); //value of num is being sent
  }
  public static void sqr(int number) // number takes the value passed from main()
  { System.out.println("The square of " + number + " is " + (number * number));}
} //end of class square
```

Output:

```
The square of 20 is 400
The square of 30 is 900
```

*** The arguments you send to the method must match in both number and type the parameter listed in the method declaration

Methods that require multiple arguments with no return value

A method can require more than one argument. You can pass multiple arguments to a method by listing the arguments within the call to the method and separating them with commas. The example below contains a method that accepts two integer values. The first argument determines what operation to perform on the second argument. If the value of the first argument is 1, the method will compute and display the square of the second argument. If the value of the first argument is 2, the method will compute and print the cube of the second argument. Any other value for the first argument will simply display the value of the second argument

```
public class Sqr_cube
{ public static void main(String[] args)
  { compute(1,5);
    compute(2,5);
    compute(3,5);
  }
  public static void compute (int x,int y)
  { if (x==1)
    { System.out.println("The square of " + y + " is " + (y*y));
    }
    else if (x==2)
    { System.out.println("The cube of " + y + " is " + y*y*y);
    }
    else
    { System.out.println(y);
    }
  } // end of method compute
} // end of class Sqr_cube
```

Output

```
The square of 5 is 25
The cube of 5 is 125
5
```

Methods that return values

A method with or without argument can return a value. The return type for a method can be any type used in Java which includes the int, double, char and so on. A method's return type can be identified through the method's type.

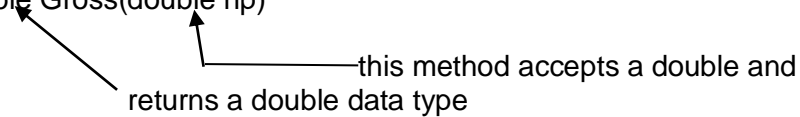
Example:

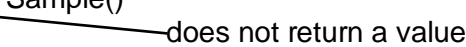
```
public static int Status()
```

← the return type for this method is int

```
public static char Sex_code()
```

← return type is char

`public static double Gross(double np)`

 this method accepts a double and
 returns a double data type

`public static void Sample()`

 does not return a value

The sample program below takes an integer value and returns an integer value. The method returns 1 if the accepted value is an odd number and will return 0 if the accepted value is an even number

```

public class Odd_even
{
    public static void main(String[] args)
    {
        int x;
        x= is_odd(5); //method call sending 5. x takes the returned value
        if (x==1)
            System.out.println("The number is odd");
        else
            System.out.println("The number is even");
    }
    public static int is_odd(int y)
    {
        if (y/2 * 2 == y)
            return 0;
        else
            return 1;
    }
    // end of method is_odd
} // end of class Odd_even

```

//Another example of methods that returns values

/** File name: MyThird

Version : 1

*/

```

public class MyThird
{
    public static void main(String args[])
    {
        int z;
        z = square(35) + square(25);
        System.out.println(z);
        z = square(3,5);
        System.out.println(z);
    }
    // end main

    public static int square(int x)
    {
        int y=1;
        y = x * x;
    }
}

```

```

        return y;

    } //end square(int x)

    public static int square(int x, int y)
    {
        int r;
        r = x * y;
        return r;
    } //end square(int x, int y)

} //end class

```

What we have learned and explored is a method called - **class method**. We need the keyword `static` in order for Java compiler to know that the method we have created is a class method. Class method/s are directly being invoked within a class it has been defined by calling the method name or it can also be called by another class by calling the classname preceded by a dot "." then preceded by the method name. An example is given on the next page

Calling a method from another class

```

/** FirstClass.java
 * This program contains a class method.
 * This demonstrates that the class method could be called
 * from another class.
 */

public class FirstClass
{
    public static void main(String args[])
    {
        greet();
    }
    public static void greet()
    {
        System.out.println("Hello!");
    }
} //end FirstClass

```

```

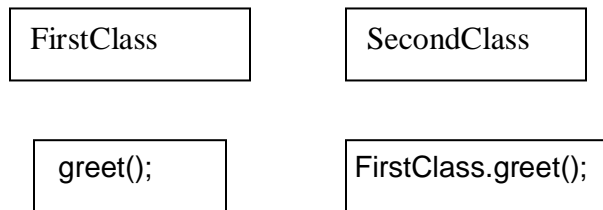
/** SecondClass
 *
 */
public class SecondClass
{
    public static void main(String args[])
    {
        System.out.println("I summon greet of FirstClass");
        FirstClass.greet();
    }
}
//end SecondClass

```

Note:

FirstClass should be in **bytecode**, before **SecondClass** can use it. Therefore compile **FirstClass.java** before compiling and running **SecondClass.java**

Illustration:



//method call for greet at FirstClass method call for greet() being called at SecondClass

Passing Arrays to Methods

An array can be used as an argument to a method, thus permitting the entire array to be passed to the method. To pass an array to a method, the array name must appear by itself without the square brackets or subscripts, as an actual argument within the method call. When declaring an array as an argument, the array name is written with a pair of square brackets before (after the data type i.e int [] x) or after the array name (i.e int x[]). When an entire array is passed to a method, any changes made in the array within the method makes its changes directly to the original array that was declared in the main(). As shown in the example below, array x in main contains the value 1,2,3 and 4. Before the method call, the contents of array x is printed. After printing the values, the entire array is passed to method pass_array (line 8). Within the method pass_array, all values are increased by 1 making the values 2,3,4 and 5 then displaying them. After the method call, the values of the array is again printed at the main().

```

public class Pass
{
    public static void main(String[] args)
    {
        int[] x = { 1,2,3,4};
    }
}

```

```

int y;
System.out.println("In main");
for (y=0;y<=3;y++)
    System.out.print(x[y] + " ");
pass_array(x);
System.out.println("\nAfter method call");
for (y=0;y<=3;y++)
    System.out.print(x[y] + " ");
}
public static void pass_array(int[] y)
{ int z;
  System.out.println("\nIn method pass_array");
  for (z=0;z<=3;z++)
  { y[z]+=1;
    System.out.print(y[z]+" ");}
}
}

```

Output

```

In main
1 2 3 4
In method pass_array
2 3 4 5
After method call
2 3 4 5

```

Method Overloading

Overloading involves using one term to indicate diverse meanings. When you overload a Java method, you write multiple methods with a shared name. The compiler understands your meaning based on the arguments you use with the method. An example of method overloading is provided below.

```

/** File name: MySecond
    Version : 2
    */
public class MySecond
{
    public static void main(String args[])
    {
        greet(); //invoke method greet()
        greet(5); //invoke method greet(with argument int)
        double z=greet(6.0); //invoke method greet(with argument int);
        System.out.println(z);
    } // end main

    public static void greet() //no parameter function need no "void"
    {
        System.out.println("Greetings from method greet()!");
    }
}

```

```

    }//end greet()#1

public static void greet(int x) //no parameter function need no "void"
{
    int i;
    for(i=1;i<=x;i++)
        System.out.println("Greetings #" + i + " from method
                                greet(with argument)!");
}

} //end greet()#2

public static double greet(double x) //no parameter function need no "void"
{ double y;
    y = x + x;
    return y;
} //end greet()#3
} //end class

```

```

Greetings from method greet()!
Greetings #1 from method greet(with argument)!
Greetings #2 from method greet(with argument)!
Greetings #3 from method greet(with argument)!
Greetings #4 from method greet(with argument)!
Greetings #5 from method greet(with argument)!
12.0

```

Activities:

I. Write the proper method declaration/ method call for the following

1. A method named CS that does not accept and does not return a value

2. Call the method in number 1

3. A method named IT that does not accept but returns an integer value

4. Call method IT

5. A method named CSIT that accepts an integer and returns a character

6. A method named CCMIT that accepts two integers and returns a double

7. A method named Pass_array that accepts an array and returns no value

8. Call method Pass_array sending array x to it

9. A method named max that requires two integers and returns an integer

10. A method named Test that accepts an integer and returns a boolean value

Laboratory exercises

1. Create a class whose main() holds two integer variables. Assign values to the variables. Create two methods named sum() and difference, that compute the sum and difference between the two variables respectively. Each method should perform the computation and display the results. In turn, call the two methods passing the values of the 2 variables. Create another method named product. The method should compute the product of the 2 numbers but will not display the answer. Instead, the method should return the answer to the calling main() which displays the answer. Provide your own screen display and class name.
2. Create a class whose main() holds an array containing 10 integers. Create two methods that accept the array. The first method named lowest returns the lowest from the 10 numbers while the second method named highest returns the highest number from the list. Determine the highest and lowest number without sorting the numbers. Your output should contain the original list of numbers, the highest and the lowest number from the list. Supply the necessary values for the array
3. Create a class named Commission that includes 3 variables: a double sales figure, a double commission rate and an integer commission rate. Create 2 overloaded methods named computeCommission(). The first method takes 2 double arguments representing sales and rate, multiplies them and then displays the results. The second method takes 2 arguments: a double sales figure and an integer commission rate. This method must divide the commission rate by 100.0 before multiplying by the sales figure and displaying the commission. Supply appropriate values for the variables. Test each overloaded method

Lesson 5

Creating Classes and Objects

Overview:

This lesson covers the classes and objects, how to create and use them

Objectives:

- To understand the use and purpose of classes and objects in the program
- To create programs with classes and objects

What is a class?

Class is a concept or anything abstract. It is a collection of variables, objects and sub-classes.

Example: cell phone -

Q: What are its attribute or traits or characteristics?

A: Keypad, LCD, antenna, body, model, brand

Q: What are its other characteristics?

A: Turn-on, silent, dirty, broken, ringing.

What is an object?

- An instance of a class (one tangible example of a class)
 - An object is a software bundle of variables and related methods.
 - Objects inherit attributes from classes
 - All objects have predictable attributes because they are members of certain classes
-
- Nokia 3210 is an object of class cell phone or therefore Nokia 3210 is an **instance** of class cellphone!

Instance is an image, therefore all of a attributes of the class will be inherit by the instantiated object.

- **Ford Lynx** is an object of class **Automobile**, wherein we know that an automobile consist of - wheels, chassis, body, engine, steering wheel.

It is said earlier that a class can contain a sub-class, example: an automobile has a fuel gauge, a temperature gauge, a speedometer all of these are objects of **class - measuring device** which is a part of a super class automobile.

A Complete Class

In order for us to say it is a complete class it should have a state and behavior.

example:

A Class dog has a state - breed, age

The same Class dog should have a behavior like- barking, eating, sleeping, playing.

- These **real-world objects** share two characteristics: they all have **state** and they all have **behavior**. For example, dogs have **state** (*breed and color*) and dogs have **behavior** (*barking, sleeping and slobbering on your newly cleaned slacks*). Bicycles have **state** (*current gear, current pedal cadence, two wheels, number of gears*) and **behavior** (*braking, accelerating, slowing down, changing gears*).
- **Software objects** are modeled after real-world objects in that they, too, have state and behavior. A software object maintains its state in **variables** and implements its behavior with **methods – methods are sub-routines, procedures or functions** in other programming languages.

Creating a class

When you create a class, first you must assign a name for that class and then you must determine what data and methods will be part of the class. Suppose you want to create a

class named Mystudent. One instance variable of Mystudent might be age and two necessary methods might be a method to set (or provide a value) the student's age and another method to retrieve the age(see Instance methods). To begin, you create a class header with three parts:

- An optional access modifier
- The keyword class
- Any legal identifier you choose for the name of the class

Ex.

```
public class Mystudent
```

The keyword public is a class access modifier. Other access modifiers that can be used when defining a class are final or abstract. Public classes are accessible by all objects, which means that public classes can be **extended** or used as a basis for any other class. After writing the class header public class Mystudent, you write the body of the Mystudent class, containing its data and methods between a set of curly brackets

Ex.

```
public class Mystudent
{
    // instance variables and methods are placed here
}
```

You place the instance variables or fields for the Mystudent class as statements within the curly brackets

```
public class Mystudent
{
    int age = 18; // variable with no access modifier is public by default
    // other methods go here
} // end of class Mystudent
```

The allowable modifiers are private, public, friendly, protected, static and final. Private access means no other classes can access a fields values and only methods of the same class are allowed to set, get or otherwise use private variables. Private access is sometimes called **information hiding**, and is an important component of object-oriented programs. A class' private data can be changed or manipulated only by a class' own methods and not by methods that belong to other class. In contrast, most class methods are not usually private.

Instance Methods

Besides data, classes can contain methods. In the Mystudent class for example, you could write a method that contains two methods : One to set the student's age (named below as AlterAge) and another to retrieve the student's age (named GetAge).

Ex.

```
public class Mystudent
{   int age=18;
    void AlterAge(int n) /* method with no access modifier is    public by default */
    {
        age = n;
    } //end of method AlterAge()
    public void GetAge()
    {
        return age;
    }
} //end of class Mystudent
```

The methods in the above example do not contain the keyword **static**. The keyword **static** is used for classwide methods but not for methods that belong to objects. If you are creating a program with `main()` that you will execute to perform some task then most of your methods will be **static** so you call them within the `main()`. However if you are creating a class from which objects will be instantiated, most methods will be **nonstatic** as you will be associating the methods with individual objects. Methods used with object instantiations are called **instance methods**.

Declaring Objects

Declaring a class does not create an actual object. A class is just an abstract of what an object will be like if any objects are actually instantiated. You can create a class with fields and methods long before you instantiate any objects that are members of that class. A two-step process creates an object that is an instance of a class. First, you supply a type and an identifier just as when you declare any variable and then you allocate memory for that object. To allocate the needed memory, you must use the **new** operator.

```
Ex:   Mystudent Loyda;    //declaring an object Loyda
      Loyda = new Mystudent(); // initializing an object to have a
                                memory location
```

You can also define and reserve memory in one statement as in:

```
Mystudent Loyda = new Mystudent();
```

In the previous example, `Mystudent` is the object's type (as well as its class) and `Loyda` is the name of the object. The equal sign is the assignment operator, so a value is being assigned to object `Loyda`. The **new** operator is allocating a new, unused portion of computer memory for object `Loyda`. The value that the statement is assigning to `Loyda` is a memory address at which it is to be located. The last portion of the statement `Mystudent()` is the name of a method that constructs `Mystudent` object. `Mystudent()` is a constructor method. A

constructor method is a method that establishes an object. When you don't write a constructor method for a class object, Java writes one for you and the name of the constructor method is always the name of the class whose objects it constructs. The complete program is written below. (more of constructors on the next lesson)

```
/** Program : MyStudent.java
    Description : This program contains instance methods
                  that can be instantiated to other class
    */

public class MyStudent
{
    int age=18; // variable with no access modifier is public by default

    void AlterAge(int n) // method with no access modifier is public by default
    {
        age = n;
    }

    public int GetAge()
    {
        return age;
    }

} //end class MyStudent

/** StudentKo.java
    This program will instantiate the attributes of Mystudent.class
    */

public class StudentKo
{
    public static void main(String args[])
    {
        int studentAge;
        MyStudent Loyda; // declaring an object
        Loyda = new MyStudent(); // initializing an object to have a memory location
        /* or MyStudent Loyda = new MyStudent(); */
        studentAge= Loyda.age;
        System.out.println("Before: age is "+studentAge);
        Loyda.AlterAge(19);
        studentAge=Loyda.GetAge();
        System.out.println("After: age is "+studentAge);
    }
} //end class StudentKo
```

Before: age is 18 After: age is 19

```

//Example #2
//Class name Sample2.java
public class Sample2
{ String lname,fname,job;
  int age;

  public void setLastname(String last)
  { lname=last; }

  public String getLastname()
  { return lname; }

  public void setFirstname(String first)
  { fname = first; }

  public String getFirstname()
  { return fname;}

  public void setJob(String trabaho)
  { job = trabaho; }

  public String getJob()
  { return job; }

  public void setAge(int edad)
  { age = edad; }
  public int getAge()
  { return age;}
}

// Instance of Sample2
// Class name l_sample2.java
public class l_sample2
{ public static void main(String args[])
  { String lastname,firstname,work;
    int gulang;
    Sample2 student = new Sample2();
    student.setLastname("Himura"); //line 6
    student.setFirstname("Kenshin"); //line 7
    student.setAge(27); //line 8
    student.setJob("Slasher"); //line 9
    lastname = student.getLastname();
    firstname = student.getFirstname();
    work=student.getJob();
    gulang = student.getAge();
  }
}

```

```

        System.out.println("Name : " + lastname + ", " + firstname);
        System.out.println("Age : " + gulang);
        System.out.println("Job : " + work);
    }
}

```

Programming Activity

1.
 - Key-in the program Sample2.java and I_sample2.java
 - Compile the 2 programs
 - Run I_sample2
 - Take note of the output
 - Try to put a comment on lines 6-9 on the program I_sample2 then compile and run the program
 - Take note of the new output. *The output will be discussed in the next lesson (overloading constructors)*
2. Try to run compile **MyStudent.java**, also compile **StudentKo.java**, then run **StudentKo** bytecode.
 - a. Try to put public static void main in **MyStudent.java** program, and try to invoke **alterAge()** method directly. (a)Have you found any error? (b)How do we invoke **alterAge()** properly inside **MyStudent.java** eventhough it is a member of class MyStudent?
 - b. Create a class method greet inside StudentKo.java like these

```

public static void greet()
{
    System.out.println("OK!");
}

```

Can we invoke **greet()** directly inside main() in class Roy?

How do we invoke greet() inside main() in class RoyInstance?

ANSWERS:

1.
 - (a) During compiling Roy.java an error is - can't create a reference to a non-static methods or variables. Because as we have learned that instance methods could only be used only after creating on object of a class where throwage() method is a member of.
 - (b) Inside main method in Roy.java instantiate an object. like these

```

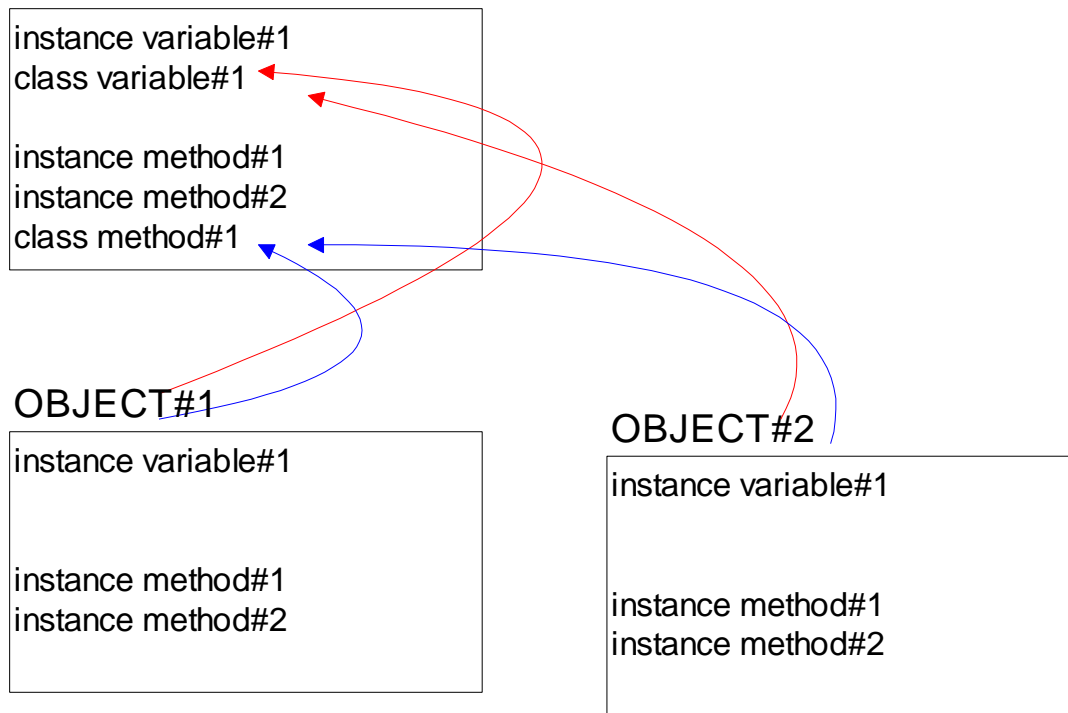
public static void main(String args[])
{
    Roy bago = new Roy();
    bago.throwage(45);
    int x = bago.fetchage();
    System.out.println(bago);
}

```

MORE ON OBJECT CREATION (INSTANCE)

- When a class is instantiated by a particular object only those instance methods and instance variables are replicated on that particular object, all of the class methods and class variables will be linked in general still to the class. Therefore any object that changes the value of the class variable all the other object that will invoke the class variable will have the same and one value.

CLASS



analyze the program below.

```
/** Program : Student.java
Description : This program contains instance methods, class
              methods and class variables
*/
```

```
public class Student
{
    static int schoolcode = 357; //class variable

    int studnum; // instance variable

    public void throwstudnum(int x) // instance method
    {
        studnum = x;
    }

    public int fetchstudnum() // instance method
    {
        return studnum;
    }
}
```

```

    }

    public int fetchschoolcode() //instance method
    {
        return schoolcode;
    }

    public void changeschoolcode(int y) //class method
    {
        schoolcode = y;
    }

} //end class Student

```

```

.....

/** StudRec.java
 */

public class StudRec
{
    public static void main(String args[])
    {
        Student iskulmateko = new Student();
        Student iskulmateniya = new Student();

        int i=iskulmateniya.fetchschoolcode();
        System.out.println("SC ni iskulmateniya: "+i);

        iskulmateniya.changeschoolcode(14344);

        int j=iskulmateko.fetchschoolcode();
        System.out.println("SC ni iskulmateko: "+j);
    }
} //end class StudRec

```

Laboratory Exercises

1. a . Create a class named Pizza. Data fields include String for toppings (such as "pepperoni"), an integer for diameter in inches (such as 12) and double for price (as in 13.99). Include methods to get and set values for each of these fields
 - b. Create a class named TestPizza that instantiates one Pizza object and demonstrates the use of the Pizza set and get methods.
2. Write a program that displays the employees Ids together with their first and last names. Use two classes. The first class contains the employee data and separate methods to set and get the ID's and names. The other class creates objects for the

employees and uses the objects to call the methods. Create several employees and display their data

3. a. Create a class named Circle with fields named radius, area and diameter. Include methods names setRadius(), getRadius(), computeDiameter() which computes a circle's diameter and computeArea which computes a circle's area. The diameter of a circle is twice its radius and the area is 3.14 multiplied by the square of the radius.
- b. Create a class named TestCircle whose main() declares three Circle objects. Using the setRadius() method, assign one circle a small radius value. Assign another circle a larger radius value and assign the third circle a radius of 1. Call computeDiameter() and computeArea() for each circle and display the results.

Lesson 6

Overloading Constructors

Overview:

This lesson covers constructors and how they work

Objectives:

- To understand the purpose of constructor
- To create a program with constructors
- To gain knowledge about overloading constructors

Constructor Methods

A constructor method is a method that is called on an object when it is created – in other words, when it is constructed. Unlike other methods, a constructor cannot be called directly. Instead, Java calls constructor methods automatically. Java does 3 things when *new* is used to create an instance of a class:

- Allocates memory for objects
- Initializes the objects instance variables either to initial values or to a default (0 for numbers, null for objects, false for Booleans or '\0' for characters)
- Calls the constructor method of the class which might be one of several methods

If a class does not have constructor methods defined, an object is still created when the *new* statement is used in conjunction with the class. However, you might have to set its instance variables or call other methods that the object needs to initialize itself. By defining constructor methods in your own classes, you can set initial values of instance variables, call methods based on those variables, call methods on other objects and set the initial properties of an object.

In the previous lesson we created a class named *Sample2* and instantiated an object with the statement:

```
Sample2 student = new Sample2();
```

you are actually calling a method named *Sample2()* that is provided by the Java compiler. A constructor method establishes an object. The constructor method named *Sample2* establishes one *Sample2* with the identifier *student* and provides the following initial values to *Sample2*'s data fields: (recall programming experiment #1 in the previous lesson)

- numeric fields are set to 0
- The object type fields are set to null

If you do not want *Sample2*'s fields to hold these default values, or if you want to perform additional task when you create *Sample2* then you write your own Constructor method. Any constructor method you write must have the same name as the class it constructs and constructor methods cannot have return type. An edited version of the class *Sample2* is provided on the next page.

```
public class Sample2
{ String lname,fname,job;
  int age;
```

```

Sample2() //constructor method containing initial values
{
    lname="Xavier";
    fname="Charles";
    job="Professor";
    age=60;
}

public void setLastname(String last)
{ lname=last; }

public String getLastname()
{ return lname; }

public void setFirstname(String first)
{ fname = first; }

public String getFirstname()
{ return fname;}

public void setJob(String trabaho)
{ job = trabaho; }

public String getJob()
{ return job; }

public void setAge(int edad)
{ age = edad; }

public int getAge()
{ return age;}
}

```

Now examine the program below. It no longer calls the setAge(),setLastname(), setFirstname() and setJob() methods.

```

public class I_sample2
{
    public static void main(String args[])
    {
        String lastname,firstname,work;
        int gulang;
        Sample2 student = new Sample2();
        lastname = student.getLastname();
        firstname = student.getFirstname();
        work=student.getJob();
        gulang = student.getAge();
    }
}

```

```

        System.out.println("Name : " + lastname + ", " + firstname);
        System.out.println("Age : " + gulang);
        System.out.println("Job : " + work);
    }
}

```

but if you run the program, it will display the output below

Name : Xavier, Charles Age : 60 Job : Professor

because of the constructor method created within the Sample2 class

Sending Arguments to Constructors

Examine the program below

```

//class name Sample3.java
public class Sample3
{
    int id;

    Sample3(int emp_Id) //this constructor accepts an int value
    { id = emp_Id; }

    public int getId()
    { return id;}
}

```

Notice the constructor method contains an int argument named emp_id. Once you instantiate Sample3, you have to pass an integer value to the constructor as shown in the program below

```

//class name: I_sample3.java
//instance of sample3
public class I_sample3
{
    public static void main(String[] args)
    {
        Sample3 test1 = new Sample3(2470); //value passed to constructor
        Sample3 test2 = new Sample3(2113);
        Sample3 test3 = new Sample3(1234);
        System.out.println("Employee id # " + test1.getId());
        System.out.println("Employee id # " + test2.getId());
        System.out.println("Employee id # " + test3.getId());
    }
}

```

When the constructor executes, the integer within the method call is passed to Sample3() and assigned to id. (id = emp_Id)

```

Output:
Employee id # 2470
Employee id # 2113
Employee id # 1234

```

In Sample3 class, instance method getId() returns the employee ID. You can display the returned value in two ways. First is to directly print the result as shown in the 3 System.out.println statement above. The second is to assign a variable that will accept the returned value of the method getId as shown below

```

int id1;
Sample3 test1 = new Sample3();
id1 = test1.getId(); System.out.println("Employee id # " + id1);

```

Sending multiple arguments to a Constructor

Multiple arguments can also be sent to a constructor method. Assume that a class contains an employee's ID number, age and department. It also contains an instance method that returns the employee ID , age and department. The constructor method of this class initializes an employee's ID number to 999, his/her age to 21 and department name as "Floating". Once you instantiate this class you have to send 3 arguments in order to replace the initial values assigned to it. The program for this problem is shown below

```

//Class name Sample4.java
public class Sample4
{
    int emp_id,emp_age;
    String emp_dept;
    Sample4(int id_no,int age,String dept)//accepts 3 arguments
    { emp_id = id_no;
      emp_age = age;
      emp_dept = dept;
    }

    public int getId()
    { return emp_id;}

    public int getAge()
    { return emp_age;}

    public String getDept()
    { return emp_dept;}
}
//end of class Sample4

```

.....

```
//class name I_sample4.java
//Instance of Sample4 class
public class I_sample4
{ public static void main(String[] args)
{
    Sample4 employee1 = new Sample4(1234,25,"Payroll");
    System.out.println("Employee ID : " + employee1.getId());
    System.out.println("Age : "+ employee1.getAge());
    System.out.println("Department : " + employee1.getDept()+"\n");

    Sample4 employee2 = new Sample4(5678,30,"Human Resources");
    System.out.println("Employee ID : " + employee2.getId());
    System.out.println("Age : "+ employee2.getAge());
    System.out.println("Department : " + employee2.getDept());
}
} //end of class I_sample4
```

```
output
Employee ID : 1234
Age : 25
Department : Payroll

Employee ID : 5678
Age : 30
Department : Human Resources
```

? In I_sample3 and I_sample4 programs, arguments were sent to the constructors when the objects were created. What happens if you do not include an argument to both constructors?

Ex. Sample3 test1 = new Sample3(); and Sample4 employee1 = new Sample4();

The result would be an error :

```
A_sample3.java : 3 :Cannot resolve symbol
Symbol : Constructor Sample3 ()
Location : class Sample3
    { Sample3 test1 = new Sample3();
      ^
```

Overloading Constructors

If you create a class from which you instantiate objects, Java automatically provides you with a constructor. Unfortunately, if you create your own constructor, the automatically created constructor no longer exists. Therefore, once you create a constructor that takes an argument, you no longer have the option of using the constructor that requires no arguments.

Fortunately, as with other methods, you can overload constructors. Overloading constructors provides you with a way to create objects with or without initial arguments, as

needed. For example, you can create a class that contains a constructor method with no arguments but contains initial values for the variables and another constructor method that accepts an argument/arguments so you can set new values. When both constructor reside within the class, you have the option of creating an object with or without an initial value. Analyze the program below.

*/*Class that contains multiple constructor methods each receiving different number of arguments*/*

```
public class Sample5
{   int stud_no,age;
    String course;

    Sample5() //constructor with no argument
    { stud_no=999;
      age=17;
      course="Fine arts"; }

    Sample5(int stud_num) //constructor with 1 argument
    { stud_no=stud_num;}

    Sample5(int stud_num,int stud_age)// with 2 arguments
    { stud_no = stud_num;
      age=stud_age;}

    Sample5(int stud_num,int stud_age,String stud_course)//3 args.
    { stud_no = stud_num;
      age=stud_age;
      course = stud_course;}

    public int getNum()
    { return stud_no;}

    public int getAge()
    { return age;}

    public String getCourse()
    { return course;}
} //end of class Sample5
```

The class contains 4 constructor methods each method accepts different number of arguments. When you create an object with no argument as in :

```
Sample5 student1 = new Sample5();
```

The constructor with no argument is called. The complete program is shown below

```
//This program illustrates how the constructor overloading works
// Instance of Sample5 class
```



```

public class I_sample5
{ public static void main(String[] args)
{
    Sample5 student1 = new Sample5();
    System.out.println("Constructor with no argument");
    System.out.println("Student number "+ student1.getNum());
    System.out.println("Age : " + student1.getAge());
    System.out.println("Course : " + student1.getCourse()+"\n");

    System.out.println("Constructor with 1 argument");
    Sample5 student2 = new Sample5(2470);
    System.out.println("Student number "+ student2.getNum());
    System.out.println("Age : " + student2.getAge());
    System.out.println("Course : " + student2.getCourse()+"\n");

    System.out.println("Constructor with 2 arguments");
    Sample5 student3 = new Sample5(2470,33);
    System.out.println("Student number "+ student3.getNum());
    System.out.println("Age : " + student3.getAge());
    System.out.println("Course : " + student3.getCourse()+"\n");

    System.out.println("Constructor with 3 arguments");
    Sample5 student4 = new Sample5(2113,25,"BSCS");
    System.out.println("Student number "+ student4.getNum());
    System.out.println("Age : " + student4.getAge());
    System.out.println("Course : " + student4.getCourse());
}
}

```

Output:

```

Constructor with no argument
Student number : 999
Age : 17
Course : Fine Arts

Constructor with 1 argument
Student number : 2470
Age : 0
Course : null

Constructor with 2 arguments
Student number : 2470
Age : 33
Course : null

Constructor with 3 arguments
Student number : 2113
Age : 25
Course : BSCS

```

Laboratory Exercises

1. a .Create a class named Circle with fields named radius, area and diameter. Include a constructor method that sets the radius to 1. Also, include methods named setRadius(), getRadius(), computeDiameter() which computes a circle's diameter and computeArea which computes a circles area (The diameter of a circle is twice its radius and the area is 3.14 multiplied by the square of the radius).

b. Create a class named TestCircle whose main() method declares 3 Circle objects. Using the setRadius() method, assign one circle a small radius value and assign another circle a larger radius value. Do not assign a value to the radius of the 3rd circle; instead, retain the value assigned at the constructor. Call computeDiameter() and ComputeArea() for each circle and display the results. Provide your own screen display.

2.Create a class named House that includes data fields for the number of occupants and the annual income as well as methods named setOccupants(), setIncome(), getOccupants() and getIncome() that set and return those values respectively. Additionally, create a constructor that requires no arguments and automatically sets the occupants field to 1 and income field to 0. Create an additional overloaded constructor . This constructor receives an integer argument and assigns the value to the occupants field. Create a third overloaded constructor this time, the constructor receives 2 arguments, the values of which are assigned to the occupants and income fields respectively. Create another class named I_house that instantiates the House class and see if the constructors work correctly.

Lesson 7

Using Prewritten Classes and Methods

Overview:

This lesson covers some of the prewritten classes and methods of Java language

Objectives:

- To know the different prewritten classes and methods
- To construct programs with that prewritten classes and methods

- **Using Automatically Imported, Prewritten Constants and Methods**

There are nearly 500 classes available for you in Java. You already used several of the prewritten classes without being aware of it. System, Character, Boolean, Byte, Short, Long, Float and double are actually classes from which you create objects. These classes are stored in a *package*, which is simply a folder that provides a convenient grouping of classes, which is sometimes called **library of classes**. There are many Java packages containing classes that are available only if you explicitly name them within your program but the group of classes that contains the previously listed classes is used so frequently that it is available automatically to every program you write. The package that is implicitly imported into every Java program is named java.lang. The classes it contains are the **fundamental classes**, or basic classes as opposed to the **optional classes** that must be explicitly named.

The class java.lang.Math contains constants and methods that you can use to perform common mathematical functions. Commonly used constant is PI. Within the Math class, the declaration for PI is public final static double PI = 3.141592653589793. PI is :

- *public* so any program can access it
- *final* so it cannot be changed
- *static* so only one copy exists
- *double* so it holds a large floating-pt value

all of the constants and methods in the Math class are static, which means they are class variables and methods. Some of the common Math class methods are listed below.

Method	Meaning
abs(x)	Absolute value of x
acos(x)	Arcosine of x
asin(x)	Arcsine of x
atan(x)	Arctangent of x
ceil(x)	Smallest integral value not less than the ceiling
cos(x)	Cosine of x
exp(x)	Exponent

Methods	Meaning
floor(x)	Largest integral value not greater than x
log(x)	Natural logarithm of x
max(x,y)	Larger of x and y
min(x,y)	Smaller of x and y
pow(x,y)	X raised to the y power
random()	Random double no between 0.0 and 1.0
round(x)	Closest integer to x(where x is a float and the return value is an integer or long
sin(x)	Sine of x
sqrt(x)	Square root of x
tan(x)	Tangent of x

//program that demonstrates the use of math class methods

```
public class MathClass
{
    public static void main(String[] args)
    {
        double num = 26.9;
        int x=5,y=3;
        System.out.println("Absolute value of num is "+Math.abs(num));
        System.out.println("The square root of " + num + " is "+
            Math.sqrt(num));
        System.out.println(num + " rounded is " + Math.round(num));
        System.out.println("PI is " + Math.PI);
        System.out.println(x + " raised to " + 2 + " is " +
            Math.pow(x,2));
        System.out.println("The larger no. between " + x + " and " + y
            + " is " + Math.max(x,y));
        System.out.println("The smaller no. between " + x + " and " + y
            + " is " + Math.min(x,y));
    }
}
```

Output:
 Absolute value of num is 26.9
 The square root of 26.9 is 5.186520991955976
 26.9 rounded is 27
 PI is 3.141592653589793
 5 raised to 2 is 25.0
 The larger no. between 5 and 3 is 5
 The smaller no. between 5 and 3 is 3

Because all constants and methods in the Math class are classwide, there is no need to create an instance. You cannot instantiate objects of type Math because the constructor for the Math class is private and your programs cannot access the constructor.

Using Prewritten Imported Methods

Only few of the classes such as java.lang – are included automatically in the programs you write. To use any of the other predefined classes, you can use any of the following:

- use the entire path with the class name
- import the class
- import the package of which the class you are using is a part

For example, the java.util class package contains useful methods that deal with dates and time. Within this package, a class named Date is defined. You can instantiate an object of type Date from this class by using the full class path as in:

```
java.util.Date date2day = new java.util.Date(); //first bullet above
```

alternatively, you can shorten the declaration of date2day to :

```
Date date2day = new Date();
```

by including the statement :

```
import java.util.Date; //second bullet above
```

as the first line in your program. An import statement allows you to abbreviate the lengthy class named by notifying the Java program that when you use Date, you mean the java.util.Date class. You must place any import statement you use before any executing statement in your program.

An alternative to importing a class is to import an entire package of classes. You can use the asterisk (*) as a wildcard symbol to represent all the classes in a package. Therefore, the import statement :

```
import java.util.*; // third bullet
```

imports the Date class and any other java.util classes as well.

The Date class has several constructors. For example, if you construct a Date object with 5 integer arguments, they become the year, month, day, hour and minutes. A Date object constructed with 3 arguments assumes the arguments to be the year, month and day and the time is set to midnight. The Date class contains a variety of other useful methods such as setMonth(), getMonth(), setDay, getDay(), setYear() and getYear();

```
import java.util.*;
public class DateClass
{ public static void main(String[] args)
```

```

{ Date date2day = new Date();
  Date datebukas = new Date();
  Date bdayko = new Date(103,10,1);
  System.out.println("My birthday is " + bdayko);
  System.out.println("Todays date is " + date2day);
  System.out.println("The month is " + date2day.getMonth());
  System.out.println("The date is " + date2day.getDate());
  System.out.println("The day is " + date2day.getDay());
  System.out.println("The year is " + date2day.getYear());
  datebukas.setDate(date2day.getDate() +1);
  System.out.println("Tomorrow is " + datebukas);
}
}

```

output :

```

My birthday is Sat Nov 01 00:00:00 CST 2003
Todays date is Wed Apr 30 15:03:39 CST 2003
The month is 3
The date is 30
The day is 3
The year is 103
Tomorrow is Thur May 01 15:11:05 CST 2003

```

The `getMonth()` method returns the number of months not the actual month. January is 0, February is 1 and so on. `getDate()` returns the current date while `getDay()` returns the current day in numbers. 1 is Mon, 2 is Tue and so on. The `getYear()` method returns the year but is 1900 less than the current year. Therefore 82 means 1982 and 103 means 2003. To correct the year displayed, edit the program and add 1900 to the `getYear()` as in:

```
System.out.println("The year is " + date2day.getYear()+1900);
```

This will now display the year as :

```
The year is 2003
```

You can perform arithmetic using dates. In the last line in the output, which displays the date on the following day, you just add the number of days to calculate the next date.

```
datebukas.setDate(date2day.getDate() +1);
```

The compiler will interpret an incorrect date such as April 31 as being May 1.

Laboratory Exercises

1. Write a Java program to determine the answers to the following:
 - a. The square of 30
 - b. The sine and cosine of 100
 - c. The value of the floor, ceiling and round of 44.7
 - d. The larger and smaller of the character K and integer 70
2. Write a program to calculate how many days it is from today until the current year
3. Write a program to determine how many days it is from today until your next birthday
4. Write a program that displays the current date in this format
Today is August 1, 2003
It is the 1st day of August
Today is Friday

Lesson 8: Using Frames

Overview:

This lesson covers basic graphical user interface (GUI) components

Objectives:

- To know the different GUI components
- To utilize GUI components in the program

Graphical User Interface Components

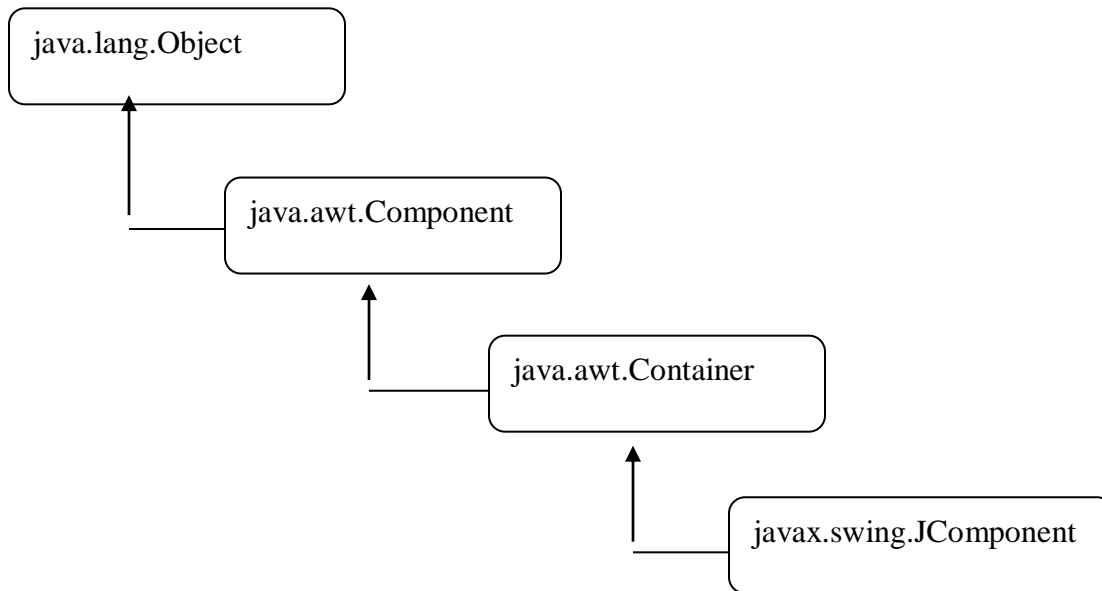
GUIs are built from GUI components (sometimes called *controls* or *widgets* (*window gadgets*)). A GUI component is an object which the user interacts via mouse, the keyboard or another form of input

Component	Description
JLabel	An area where uneditable text or icons can be displayed
TextField	An area in which the user inputs data from the keyboard. The area can also display an information
JButton	An area that triggers an event when clicked
JCheckBox	A GUI component that is either selected or not selected
JComboBox	A drop-down list of items from which the user can make a selection by clicking an item in the list or possibly by typing into the box
JList	An area where a list of items is displayed from which the user can make a selection by clicking once on any element in the list. Double-clicking an element in the list generates an action event. Multiple elements can be selected
JPanel	A container in which components can be placed

Swing Overview

The classes that create the GUI components are part of the Swing GUI components from the package `javax.swing`. Most swing components are written, manipulated and displayed completely in Java. The original GUI components from the Abstract Windowing Toolkit package are tied directly to the local platform's graphical user interface capabilities. When a Java program with an AWT GUI executes on different Java platforms, the program's GUI components display differently on each platform. The appearance and how the user interacts with the program are known as the program's *look and feel*. Swing enables programs to provide a custom look and feel for each platform or even to change the look and feel while the program is running. Swing components are often referred to as

lightweight components – they are written completely in Java so they are not “weighed down” by the complex GUI capabilities of the platform on which they are used. AWT components (many of which parallel the swing components) that are tied to the local platform are correspondingly called *heavyweight components* – they rely on the local platform’s windowing system to determine their functionality and their look and feel. Each heavyweight component has a peer (from package `java.awt.peer`) that is responsible for the interactions between the component and the local platform that display and manipulate the component. Several Swing components are still heavyweight components. In particular, subclasses of `java.awt.Window` (such as `JFrame`) that display windows on the screen and subclasses of `java.applet.Applet` (such as `JApplet`) still require direct interaction with the local windowing system. As such, heavyweight Swing GUI component are less flexible than many of the lightweight components.



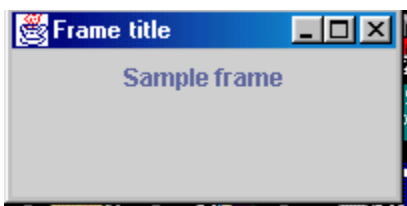
The figure above shows an inheritance hierarchy of the classes that define attributes and behaviors that are common to Swing components. Each class is displayed with its fully qualified package name and class name. Much of each GUI component’s functionality is derived from these classes. A class that inherits from the **Component** class a **component**. For example, class **Container** inherits from class Component, and class Component inherits from Object. Thus, a Container is a Component and is an Object, and a Component is an Object. A class that inherits from class Container is a Container. Thus a Jcomponent is a Container.

Class **Component** defines the common attributes and behaviors of all subclasses of **Component**. With few exceptions, most GUI components extend class **Component** directly or indirectly.

A **Container** is a collection of related components. In application with **JFrames** and in applets, we attach components to the content pane, which is an object of class **Container**. Class **Container** defines the common attributes and behaviors for all subclasses of **Container**. One method that originates in class **Container** is **add** for adding components to a **Container**. Another method that originates in class **Container** is **setLayout**, which enables a program to specify the layout manager that helps a **Container** position and size its components.

```
//This program demonstrates the use of JFrame
//Jframedemo.java
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class Jframedemo extends JFrame
{
    private JLabel lbl = new JLabel("Sample frame");
    public Jframedemo(String s)
    {
        super(s);
        Container a = getContentPane();
        a.setLayout(new FlowLayout());
        a.add(lbl);
        setVisible(true);
        setSize(200,200);
    }
    public static void main(String args[])
    {
        Jframedemo test = new Jframedemo("Frame title ");
        test.show();
        test.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```



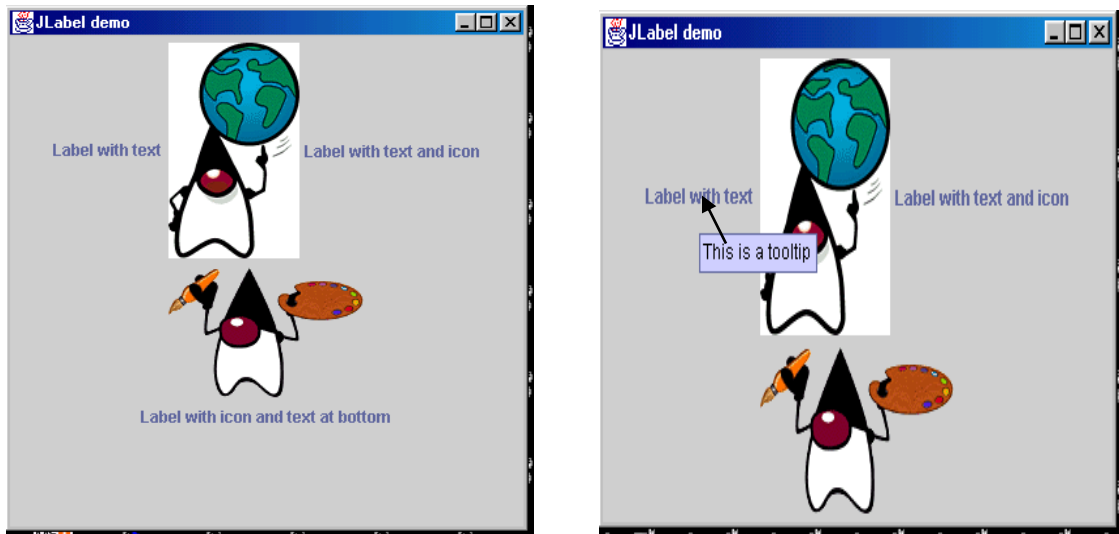
JLabel

Labels provide text instructions or information on a GUI. Labels are defined with class JLabel – a subclass of JComponent. A label displays a single line of read-only text, an image or both text and an image. A program demonstrating the use of JLabel is provided below

// This program demonstrates the use of JLabel

```
1    import javax.swing.*;
2    import java.awt.*;
3    import java.awt.event.*;

4    public class Labeldemo extends JFrame
5    { private JLabel label1,label2,label3;
6      public Labeldemo(String x)
7      { super (x);
8        //get content pane and set its layout
9        Container c = getContentPane();
10       c.setLayout(new FlowLayout());
11       //JLabel with a string argument
12       label1 = new JLabel("Label with text");
13       label1.setToolTipText("This is a tooltip");
14       c.add(label1);
15       //JLabel constructor with string, icon and alignment arguments
16       Icon iconglobe = new ImageIcon("globe.gif");
17       label2 = new JLabel("Label with text and
18         icon",iconglobe,SwingConstants.LEFT);
19       c.add(label2);
20       //JLabel with no arguments
21       label3 = new JLabel();
22       Icon iconpaint = new ImageIcon("painting.gif");
23       label3.setText("Label with icon and text at bottom");
24       label3.setIcon(iconpaint);
25       label3.setHorizontalTextPosition(SwingConstants.CENTER);
26       label3.setVerticalTextPosition(SwingConstants.BOTTOM);
27       c.add(label3);
28       setSize(400,300);
29       setVisible(true);
30     } //end of constructor
31     public static void main(String args[])
32     { Labeldemo testlabel = new Labeldemo("JLabel demo");
33       testlabel.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
34     } // end of main
35   } //end of class
```



The program declares three JLabels (line 5). The JLabel objects are instantiated in the Labeldemo constructor. Line 12 creates a JLabel object with a text “Label with text”. The label displays this text when the label.

Line 13 uses method **setToolTipText** (inherited into class JLabel from class JComponent) to specify the tooltip (see output on the right side) that is displayed automatically when the user positions the mouse over the label in the GUI. Line 14 adds label1 to the content pane.

Several Swing components can display images by specifying an **Icon** as an argument to either constructor by using a method that is normally called **setIcon**. An **Icon** is an object of any class that implements interface Icon (package javax.swing). One such class is **ImageIcon**(package javax.swing), which supports several image formats, including Graphics Interchange Format(GIF), Portable Network Graphics (PNG) and Joint Photographic Experts Group(JPEG). Line 16 and Line 21 defines an ImageIcon object. The file “globe.gif” and “painting.gif” contains the image to load and store in the ImageIcon object. This file is assumed to be in the same directory as the program. The **ImageIcon** object is assigned to **Icon** to reference iconglobe and iconpaint. Remember, class ImageIcon implements interface Icon, therefore, ImageIcon is an Icon.

Class JLabel supports the display of icons. Line 17 uses another JLabel constructor to create a label that displays the text “Label with text and icon” and the Icon to which the iconglobe refers and is left justified. Interface **SwingConstants** (package javax.swing) defines a set of common integer constants (such as SwingConstants.LEFT) that are used with many Swing components. By default, the text appears to the right of the image when a

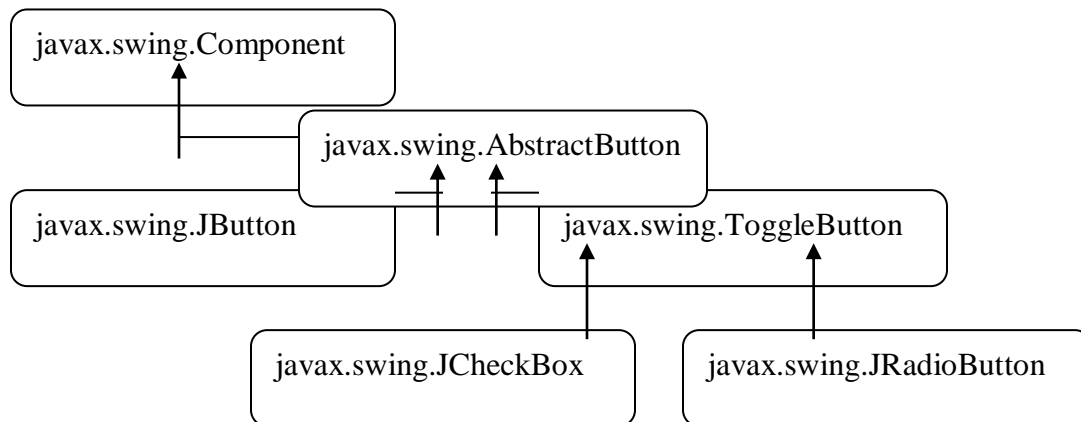
label contains both text and an image. The horizontal and vertical alignments of a label can be set with methods **setHorizontalTextPosition** and **setVerticalTextPosition** respectively.

Class JLabel provides many methods to configure a label after it has been instantiated. Line 20 creates a JLabel and invokes no argument. Such a label has no text or Icon. Line 22 uses method **setText** to set the text displayed on the label. Line 23 uses method **setIcon** to set the icon displayed on the label. Line 24 and 25 uses methods **setHorizontalTextPosition** and **setVerticalTextPosition** to specify the text position in the label. Thus, the icon will appear above the text.

JButton

A button is a component that the user clicks to trigger a specific action. A Java program can use several types of buttons including command buttons, checkboxes, toggle buttons and radio buttons. A command button generates an `ActionEvent` when the user clicks the button. Command buttons are created with class JButton, which inherits from class `AbstractButton`.

Button Hierarchy



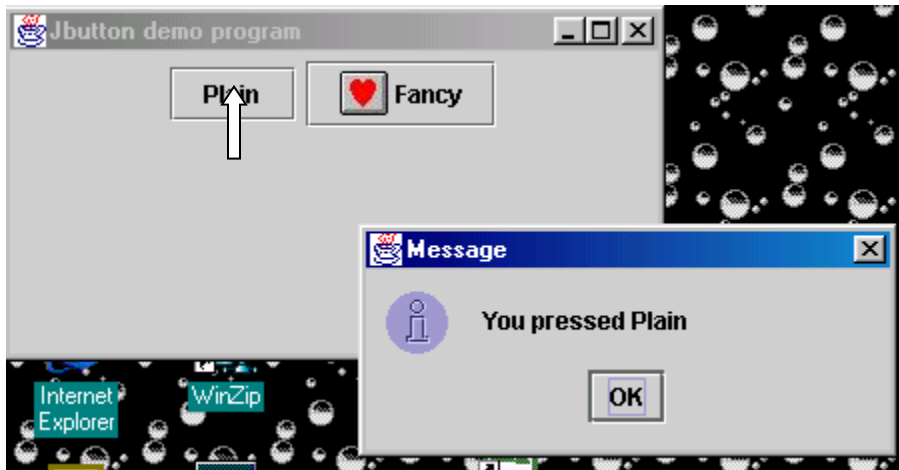
A program demonstrating the use of JButton is presented below. Like JLabels, JButton supports the display of icons. Two buttons are created in the program. The buttons are labeled “plain” and “fancy”. The fancy button is defined with an icon on it. (line 9). To provide the user with an extra level of visual interactivity with GUI, JButton can also have a *rollover Icon* – an icon that is displayed when the mouse is positioned over the Button. The icon on the button changes as the mouse moves in and out of the button’s area on the screen. Line 14 sets the rollover icon to “globe” using **setRollOverIcon()** (inherited from `AbstractButton` into class into JButton). Event handling for the button is performed by a single instance of inner class Bhandler (defined at line 23-28).

```

// Program that demonstrates the use of JButton
//Jbuttondemo.java
1   import javax.swing.*;
2   import java.awt.*;
3   import java.awt.event.*;
4   public class Jbuttondemo extends JFrame
5   {
6       JButton btn1 = new JButton("Plain");
7       Icon iconheart = new ImageIcon("heart.jpg");
8       Icon iconglobe = new ImageIcon("globe.gif");
9       JButton btn2 = new JButton("Fancy",iconheart);
10      Jbuttondemo(String x)
11      { super(x);
12        Container c = getContentPane();
13        c.setLayout(new FlowLayout());
14        btn2.setRolloverIcon(iconglobe);
15        c.add(btn1);
16        c.add(btn2);
17        //create an instance of inner class Bhandler to use for button event handling
18        Bhandler clicked = new Bhandler();
19        btn1.addActionListener(clicked);
20        btn2.addActionListener(clicked);
21        setSize(400,300);
22        setVisible(true);
23    } //end of constructor
24        //inner class for button event handling
25    private class Bhandler implements ActionListener
26    {
27        public void actionPerformed(ActionEvent y)
28        { JOptionPane.showMessageDialog(null,"You pressed "+
29            y.getActionCommand());
30        } //end of actionPerformed()
31    } //end of class bhandler
32    public static void main(String args[])
33    { Jbuttondemo test = new Jbuttondemo("Jbutton demo program");
34      test.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
35    } //end of main
36 } //end of class jbuttondemo

```

When Plain button is clicked. Dialog box appears when Plain button is clicked



New icon appears when mouse is positioned at fancy button. Dialog box appears when fancy button is clicked



JCheckBox and JRadioButton

The Swing components contain three types of *state buttons* – JToggleButton, JCheckBox and JRadioButton – that have on and off or true/false values. JToggleButton are frequently used with toolbars. Classes JCheckBox and JRadioButton are subclasses of JToggleButton. A JRadioButton is different from a JCheckBox in that there are normally several JRadioButtons that are grouped together and only one of the JRadioButtons in the group can be selected at any point.

```
//This program demonstrates the use of JCheckbox
//Jcheckboxdemo.java
1    import javax.swing.*;
2    import java.awt.event.*;
3    import java.awt.*;
```

```

4     public class Jcheckboxdemo extends JFrame
5     {
6     private JTextField jtxt1 = new JTextField("See for yourself!",15);
7     private JCheckBox bold = new JCheckBox("Bold");
8     private JCheckBox italic = new JCheckBox("Italic");
9     private JButton btnClose = new JButton("Close");
10    public Jcheckboxdemo(String x)
11    { super(x);
12      Container c = getContentPane();
13      c.setLayout(new FlowLayout());
14      c.add(jtxt1);
15      jtxt1.setEditable(false);
16      c.add(bold);
17      c.add(italic);
18      c.add(btnClose);
19          //register listeners for JCheckboxes
20      Checkhandler click = new Checkhandler();
21      Buttonhandler btnclick = new Buttonhandler();
22      bold.addItemListener(click);
23      italic.addItemListener(click);
24      btnClose.addActionListener(btnclick);
25      setSize(300,300);
26      setVisible(true);
27    } //end of constructor
28          //inner class for button event handling
29    private class Buttonhandler implements ActionListener
30    { public void actionPerformed(ActionEvent a)
31      { dispose();System.exit(0);}
32    } //end of class buttonhandler
33          //inner class for item listener handling
34    private class Checkhandler implements ItemListener
35    { private int valBold = Font.PLAIN;
36      private int valItalic = Font.PLAIN;
37      public void itemStateChanged(ItemEvent y)
38      {
39          // for bold checkbox events
40          if (y.getSource()==bold)
41              if(y.getStateChange()==ItemEvent.SELECTED)
42                  valBold = Font.BOLD;
43              else
44                  valBold = Font.PLAIN;
45          // for italic checkbox events
46          if (y.getSource()==italic)
47              if(y.getStateChange()==ItemEvent.SELECTED)
48                  valItalic = Font.ITALIC;
49              else
50                  valItalic = Font.PLAIN;
51          // set text to whatever is clicked
52          jtxt1.setFont(newFont("TimesRoman",valBold+valItalic,14));
53      } //end of itemstatechanged

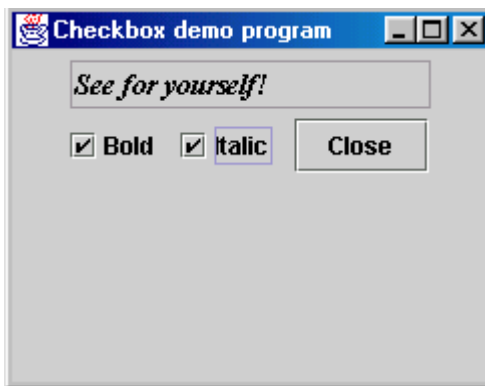
```

```

53     } //end of checkhandler
52     public static void main(String args[])
53     { Jcheckboxdemo demo = new Jcheckboxdemo("Checkbox demo program");
54       demo.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
55     } //end of main
56       } //end of class jchekcboxdemo

```

The program uses two checkbox objects to change the font style of the text displayed in the textfield and an exit button. One checkbox applies a bold style when selected and the other applies an italic style when selected. If both are selected, the text in the textfield becomes bold and italicized. Line 6 creates an object for JTextField. Line 7-8 creates a checkbox labeled Bold and Italic respectively. When the user checks an item in the checkbox, an ItemEvent occurs that can be handled by an ItemListener. In the program above, the event handling is performed by an instance inner class Checkhandler (click, line 19) and register it with method addItemListener (line 33) as the item for both bold and italic checkboxes. The output of the program is shown on the next page.



Radio Buttons (defined with class JRadioButton) are similar to checkboxes in that they have two states – selected or not selected (also called deselected). The logical relationship between radio buttons is maintained by a **ButtonGroup** object (package javax.swing). The **ButtonGroup** itself is not a GUI component. Therefore, a **ButtonGroup** object is not displayed in a user interface. Rather, the individual JRadioButton objects from the group are displayed in the GUI. (Adding a ButtonGroup object to a container is a syntax error).

```

//This program demonstrates the use of JRadioButton
//Jradiodemo.java
1     import javax.swing.*;
2     import java.awt.*;
3     import java.awt.event.*;

4     public class Jradiodemo extends JFrame
5     { private JLabel jlbgender = new JLabel("Gender");
6     private JTextField jtxt1 = new JTextField("Male",10);

```

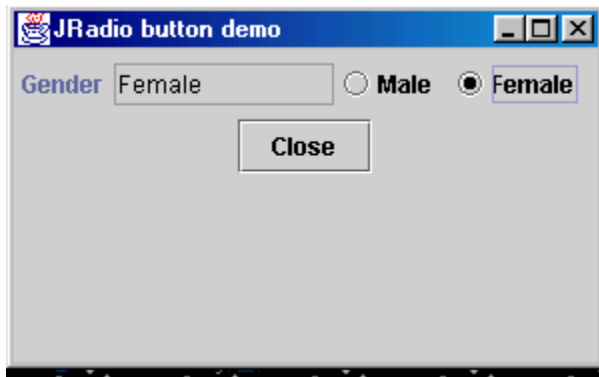
```

7  private JButton jbbtnclose = new JButton("Close");
8      //create a button group object
9  private ButtonGroup gendergroup = new ButtonGroup();
10     //create radio buttons
11  private JRadioButton male=new JRadioButton("Male",true);
12 private JRadioButton female = new JRadioButton("Female",false);
13  public Jradiodemo(String x)
14  { super(x);
15      Container c = getContentPane();
16      c.setLayout(new FlowLayout());
17      c.add(jlblgender);
18      c.add(jtxt1);
19      c.add(male);
20      c.add(female);
21      //create logical relationship between radio buttons
22      jtxt1.setEditable(false);
23      gendergroup.add(male);
24      gendergroup.add(female);
25      c.add(jbbtnclose);
26      //register events for close button
27      Buttonhandler btnexit = new Buttonhandler();
28      //register events for radio buttons
29      Radiohandler optionclick = new Radiohandler();
30      jbbtnclose.addActionListener(btnexit);
31      male.addItemListener(optionclick);
32      female.addItemListener(optionclick);
33      setSize(300,300);
34      setVisible(true);
35  } //end of constructor
36  //process close button
37  private class Buttonhandler implements ActionListener
38  { public void actionPerformed(ActionEvent a)
39      { dispose();System.exit(0);}
40  } //end of buttonhandler

41  private class Radiohandler implements ItemListener
42  { public void itemStateChanged(ItemEvent b)
43      { if (b.getSource()==male)
44          jtxt1.setText("Male");
45          else
46          jtxt1.setText("Female");
47      }
48  } //end of radiohandler

49  public static void main(String args[])
50  { Jradiodemo test = new Jradiodemo("JRadio button demo");
51      test.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
52  } //end of main
53  } //end of class jradiodemo

```



Line 9 instantiates a `ButtonGroup` object and assigns it to reference `gendergroup`. This object is the “glue” that binds the 2 radio button objects together. Class `Radiohandler` (lines 41-48) implements interface `ItemListener` so it can handle item events generated by the `JRadioButtons`. Each radio button in the program has an instance of this class (`optionclick`) registered as its `ItemListener`.

JComboBox

A combo box (sometimes called drop-down list) provides a list of items to be displayed from which the user can make a selection. Combo boxes are implemented with class `JComboBox`, which inherits from class `JComponent`. `JComboBoxes` generate `ItemEvents` like `JCheckBox` and `JRadioButton`.

//This program demonstrates the use of `JComboBox`

//Jcombodemo1.java

```

1      import javax.swing.*;
2      import java.awt.*;
3      import java.awt.event.*;
4      public class Jcombodemo1 extends JFrame
5      {   private String list[] = {"Colossus","Nightcrawler","Storm","Gambit","Prof. X"};
6          //list above will be placed in the combo box
7      private JComboBox xmen = new JComboBox(list);
8      private JTextField txt1 = new JTextField(15);
9      private JLabel lbl1 = new JLabel("Given Name");
10     public Jcombodemo1(String x)
11     {   super (x);
12         Container d = getContentPane();
13         d.setLayout(new FlowLayout());
14         d.add(lbl1);
15         d.add(txt1);
16         d.add(xmen);
17         txt1.setEditable(false);
18         xmen.setMaximumRowCount(3);//set the rows to 3 when the
           //list appears
19         // setup jcombo box and register its event
20         Listhandler showlist = new Listhandler();

```

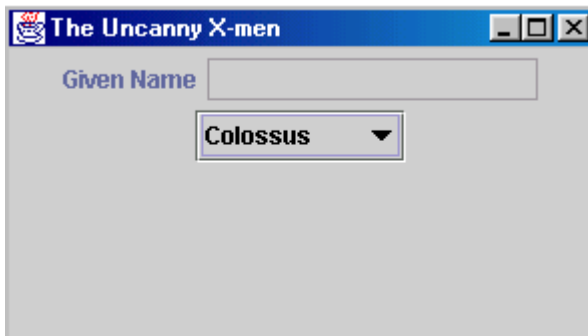
```

21  xmen.addItemListener(showlist);
22  setVisible(true);
23  setSize(300,300);
24  } //end of container
25  //inner class to handle combo box events
26  private class Listhandler implements ItemListener
27  { public void itemStateChanged(ItemEvent a)
28  { int b = xmen.getSelectedIndex();
29    if (b==0) //if first option in the list is clicked
30      txt1.setText("Vladimir Rasputin");
31    else if (b==1)
32      txt1.setText("Kurt Wagner");
33    else if (b==2)
34      txt1.setText("Ororo Monroe");
35    else if (b==3)
36      txt1.setText("Remy Lebeau");
37    else
38      txt1.setText("Charles Xavier");
39  } //end if
40  } //end listhandler

41  public static void main(String args[])
42  { Jcombodemo1 test = new Jcombodemo1("The Uncanny X-men");
43    test.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
44  } //end of main
45 } //end of class jcombodemo1

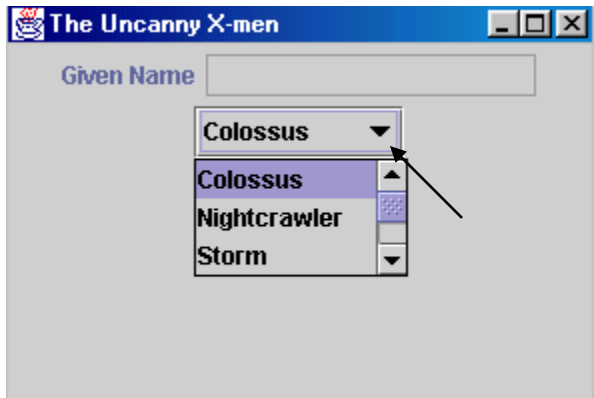
```

In the program, the options in the combo box is placed in an array of String called list(line 5). Line 7 creates a JComboBox object, using the Strings in array "list" as the elements in the list. A numeric index keeps track of the ordering of items in the combo box. The first item is at index 0, the next item is at index 1 and so on. The first item added to the combo box appears as the currently selected item when the combo box is displayed. (see frame on next page)

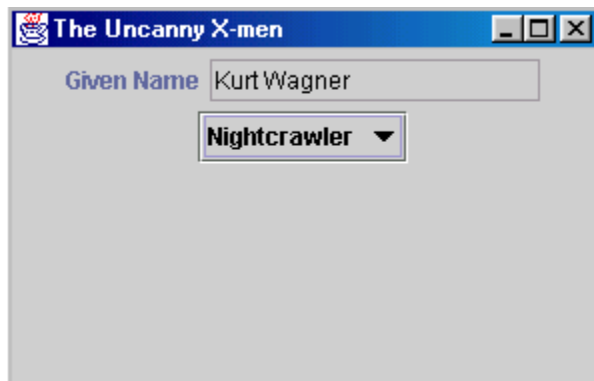


Other items are selected by clicking the combo box. When clicked, the combo box expands in a list from which the user can make a selection. Line 18 uses JComboBox method setMaximumRowCount to set the maximum number of elements that are displayed

when the user clicks the combo box. If there are more items in the combo box than the maximum number of elements that are displayed, the combo box automatically provides a scrollbar. (see frame below)



When the user makes a selection from the list, method `itemStateChanged` (lines 27-40) sets the text in the textfield based on the option selected.



```
//This program demonstrates the use of jcombo box using array
// for the options in the combo box
//Jcombodemo.java
1    import javax.swing.*;
2    import java.awt.*;
3    import java.awt.event.*;

4    public class Jcombodemo extends JFrame
5    {
6    private String iconlist[]={"Java_logo.gif", "HotJava-16.gif", "Star7.gif","Box.gif"};
7    private Icon icons[] = {new ImageIcon( iconlist[0]), new ImageIcon( iconlist[1]),new
ImageIcon( iconlist[2]),new ImageIcon( iconlist[3])};
8    private JComboBox iconcombo = new JComboBox(iconlist);
9
10   private JLabel lbl = new JLabel(icons[0]);
11   public Jcombodemo(String str)
12   { super(str);
13     Container c = getContentPane();
```

```

14  c.setLayout(new FlowLayout());
15  iconcombo.setMaximumRowCount(3);
16  Combohandler display = new Combohandler();
17  c.add(iconcombo);
18  iconcombo.addItemListener(display);
19  c.add(lbl);
20  setSize(400,400);
21  show();
22  }//end of constructor
23  private class Combohandler implements ItemListener
24  { public void itemStateChanged(ItemEvent a)
25    { if (a.getStateChange()==ItemEvent.SELECTED)
26        lbl.setIcon(Icons[iconcombo.getSelectedIndex()]);
27    }
28  } //end of combohandler
29  public static void main(String args[])
30  { Jcombodemo test = new Jcombodemo("Jcombo box demo with icons");
31    test.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
32  } //end of main
33 } //end of class jcombodemo

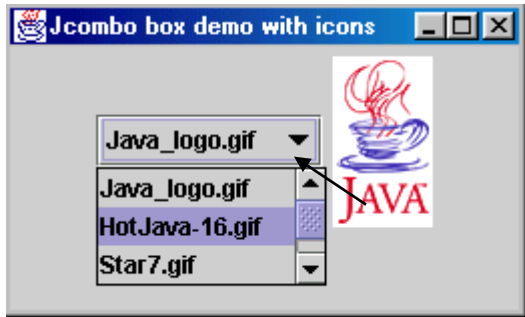
```

The program is another example of a combo box that provides a list of four image file names. When an image file is selected, the corresponding image is displayed as an icon on a JLabel. Line 7 declare and initialize an array icons with four new ImageIcon objects. String array "iconlist" (line 6) contains the names of the four image files. When the user makes a selection from the images, method itemStateChanged sets the Icon for Label. The icon is selected from array "icons" by determining the index number of the selected item in the combo box with the method getSelectedIndex.

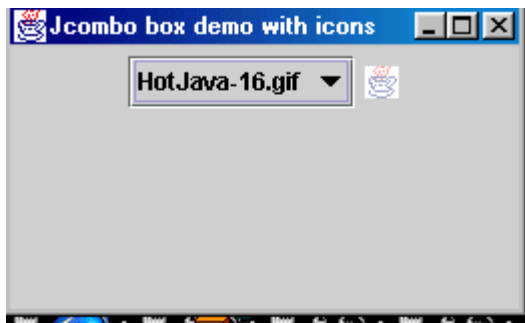
When the program is executed...



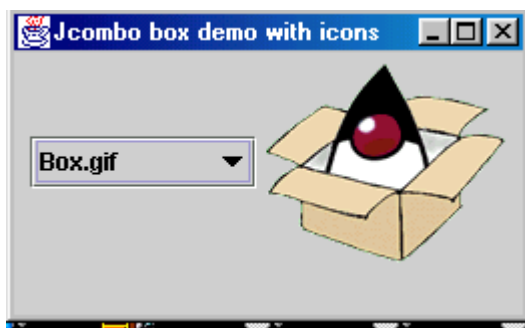
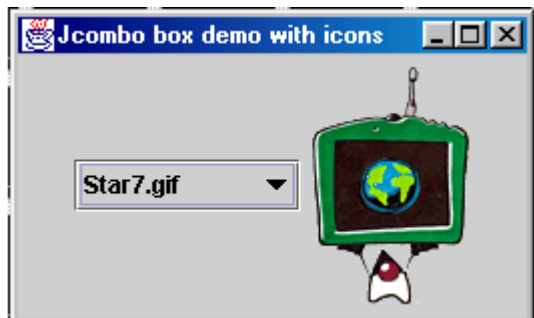
when the list is displayed...



after selecting HotJava-16.gif from the list



The last 2 frame below shows the 2 items selected from the list



Another version of the program above is presented below. This program does not use array for the icons. The output is similar to the output above.

```
//This program demonstrates the use of Jcombo box without using //array for the icons
// Jcombodemo2.java
```

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
```

```
public class Jcombodemo2 extends JFrame
{ //icons are not placed in an array
  Icon t1icon = new ImageIcon("java_logo.gif");
  Icon t2icon = new ImageIcon("hotjava-16.gif");
  Icon t3icon = new ImageIcon("box.gif");
  Icon t4icon = new ImageIcon("star7.gif");
  // items that will appear in the combo box
  private String list[]={"Java Logo 1","Java Logo 2","Box","Star"};
  private JComboBox iconcombo = new JComboBox(list);
  private JLabel lbl = new JLabel(t1icon);
  public Jcombodemo2(String str)
  { super(str);
    Container c = getContentPane();
    c.setLayout(new FlowLayout());
    iconcombo.setMaximumRowCount(3);
    Combohandler display = new Combohandler();
    c.add(iconcombo);
    iconcombo.addItemListener(display);
    c.add(lbl);
    setSize(400,400);
    show();
  } //end of constructor
  private class Combohandler implements ItemListener
  { public void itemStateChanged(ItemEvent a)
    { int b=iconcombo.getSelectedIndex();
      if (b==0)//if first item is selected
        lbl.setIcon(t1icon);
      else if (b==1) //2nd item is selected
        lbl.setIcon(t2icon);
      else if (b==2)//3rd item is selected
        lbl.setIcon(t3icon);
      else //last item is selected
        lbl.setIcon(t4icon);
    }
  } //end of combohandler
  public static void main(String args[])
  { Jcombodemo2 test = new Jcombodemo2("Jcombo box demo with icons");
    test.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  } //end of main
} //end of class jcombodemo2
```

JList

A list displays a series of items from which the user may select one or more items. Lists are created with class JList, which inherits from class JComponent. Class JList supports single-selection lists (lists that allow only one item to be selected at a time) and multiple-selection lists (lists that allow any number of items to be selected)

```
1    //This program demonstrates the use of JList
2    //Jlistdemo.java
3    import javax.swing.*;
4    import javax.swing.event.*;
5    import java.awt.*;
6    import java.awt.event.*;

7    public class Jlistdemo1 extends JFrame
8    {    private String xmenlist[] =
          {"Cyclops", "Jubilee", "Beast", "Wolverine", "Nightcrawler"};
9    private JList xmen = new JList(xmenlist);
10   private JTextField txt1 = new JTextField(15);
11   public Jlistdemo1(String s)
12   { super(s);
13     Container a = getContentPane();
14     a.setLayout(new FlowLayout());
15     xmen.setVisibleRowCount(3);
16     a.add(txt1);
17     txt1.setEditable(false);

18     //do not allow multiple selection
19     xmen.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);

20     //add a scrollpane containing jlist to content pane
21     a.add(new JScrollPane(xmen));
22     xmen.addListSelectionListener(new ListSelectionListener()
23     { public void valueChanged(ListSelectionEvent x)
24       { int b = xmen.getSelectedIndex();
25         if (b==0)
26             txt1.setText("Scott Summers");
27         else if (b==1)
28             txt1.setText("Jubilation Lee");
29         else if (b==2)
30             txt1.setText("Henry Mckoy");
31         else if (b==3)
32             txt1.setText("Logan");
33         else
34             txt1.setText("Kurt Wagner");
35       }
36     }); //end of anonymous inner class
37     setVisible(true);
38     setSize(300,300);
39 } //end of constructor
40 public static void main(String args[])
```

```

41    { Jlistdemo1 testlist = new Jlistdemo1("Jlist demo");
42      testlist.show();
43      testlist.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
44    } // end of main
45    } //end of class Jlistdemo

```

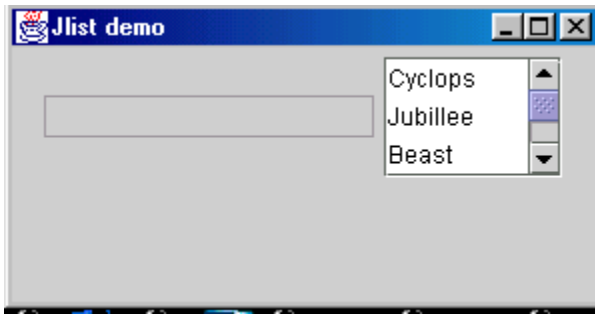
A Jlist object is instantiated at line 9 and assign to reference “xmen” in the constructor. The argument to the JList constructor is the array of Strings (xmenlist) declared at line 8 to display in the list. Line 15 uses method **setVisibleRowCount** to determine the number of items that are displayed in the list.

Line 19 uses method **setSelectionMode** to specify the list selection mode. Class **ListSelectionModel** defines constants **SINGLE_SELECTION**, **SINGLE_INTERVAL_SELECTION** and **MULTIPLE_INTERVAL_SELECTION** to specify a Jlists selection mode. A **SINGLE_SELECTION** list allows only one item to be selected at a time. A **SINGLE_INTERVAL_SELECTION** list is a multiple-selection list that allows several items in a contiguous range in the list to be selected. A **MULTIPLE_INTERVAL_SELECTION** list is a multiple list that does not restrict the items that can be selected.

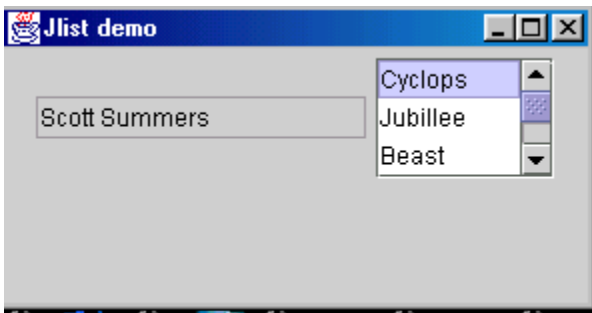
Unlike JComboBox, JLists do not provide a scrollbar if there are more items in the list than the number of visible rows. In this case, JscrollPane object is used to provide the automatic scrolling capability for the JList. Line 21 adds a new instance of class JscrollPane to the content pane. The JscrollPane constructor receives as its argument the JComponent for which it will provide automatic scrolling functionality (in this case, “xmen”). By default, the scrollbar appears only when the number of items in the list exceeds the number of visible items.

Line 22 uses JList method **addListSelectionListener** to register an instance of an anonymous inner class that implements **ListSelectionListener** (defined in the package javax.swing.event, line 4) as the listener for JList xmen. When the user makes the selection from the list, method **valueChanged** (line 23) executes and sets the value of the textfield based on the value returned by the method **getSelectedIndex**.

When the program is executed...



after selecting an option in the list....



Another version of the JList program is presented below. This program does not use an anonymous inner class. The output is similar to the output presented above.

//Jlist program that does not use an anonymous inner class

//Jlistdemo1rev.java

import javax.swing.*;

import javax.swing.event.*;

import java.awt.*;

import java.awt.event.*;

public class Jlistdemo1rev extends JFrame

{ private String xmenlist[] = {"Cyclops","Jubilee","Beast","Wolverine","Nightcrawler"};

private JList xmen = new JList(xmenlist);

private JTextField txt1 = new JTextField(15);

public Jlistdemo1rev(String s)

{ super(s);

Container a = getContentPane();

a.setLayout(new FlowLayout());

xmen.setVisibleRowCount(3);

a.add(txt1);

txt1.setEditable(false);

//do not allow multiple selection

xmen.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);

//add a scrollpane containing jlist to content pane

a.add(new JScrollPane(xmen));

Listhandler display = new Listhandler();

```

        xmen.addListSelectionListener(display);
        setVisible(true);
        setSize(300,300);
    } //end of constructor
    private class Listhandler implements ListSelectionListener
    { public void valueChanged(ListSelectionEvent x)
      { int b = xmen.getSelectedIndex();
        if (b==0)
            txt1.setText("Scott Summers");
        else if (b==1)
            txt1.setText("Jubilation Lee");
        else if (b==2)
            txt1.setText("Henry Mckoy");
        else if (b==3)
            txt1.setText("Logan");
        else
            txt1.setText("Kurt Wagner");
      }
    } //end of listhandler

    public static void main(String args[])
    { Jlistdemo1 testlist = new Jlistdemo1("Jlist demo");
      testlist.show();
      testlist.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    } //end of main
} //end of class jlistdemo1rev

```

Multiple-Selection List

A multiple-selection list enables the user to select many items from a JList. A **SINGLE_INTERVAL_SELECTION** list allows selection of a contiguous range of items in the list by clicking the first item, then holding the Shift key while clicking the last item to select in the range. A **MULTIPLE_INTERVAL_SELECTION** list allows continuous range selection as described for a SINGLE_INTERVAL_SELECTION list and allows miscellaneous items to be selected by holding the CTRL key while clicking each item to select. To deselect an item, hold the CTRL key while clicking the item a second time.

// This program demonstrates the use of jlist with multiple //selection
 //Jlistdemo2.java

```

1    import javax.swing.*;
2    import javax.swing.event.*;
3    import java.awt.*;
4    import java.awt.event.*;

5    public class Jlistdemo2 extends JFrame

```

```

6      {   private String xmenlist[] =
{"Cyclops","Jubilee","Beast","Wolverine","Nightcrawler"};
7      private JList xmen = new JList(xmenlist); //list box w/ //items to copy
8      private JList copylist = new JList(); //list box where //items will be copied
9      private JButton copybutton = new JButton("Copy>>");
10     public Jlistdemo2 (String s)
11     { super(s);
12       Container a = getContentPane();
13       a.setLayout(new FlowLayout());
14       //setup xmen list
15       xmen.setVisibleRowCount(5);
16       xmen.setFixedCellHeight(15);
17       xmen.setSelectionMode(ListSelectionModel.MULTIPLE_INTERVAL_SELECTION);
18       a.add(new JScrollPane(xmen));

19       a.add(copybutton);
20       //anonymous inner class for button event
21       copybutton.addActionListener(new ActionListener()
22       { public void actionPerformed(ActionEvent c)
23         { //place selected list in copy list
24           copylist.setListData(xmen.getSelectedValues());
25         }
26       }); //end of anonymous inner class

27       //setup copy list
28       copylist.setVisibleRowCount(5);
29       copylist.setFixedCellHeight(15);
30       copylist.setFixedCellWidth(100);
31       copylist.setSelectionMode(ListSelectionModel.SINGLE_INTERVAL_SELECTION);
32       a.add(new JScrollPane(copylist));

33       setVisible(true);
34       setSize(300,300);
35     } //end constructor
36     public static void main(String args[])
37     { Jlistdemo2 testlist = new Jlistdemo2("Jlist demo with multiple selection");
38       testlist.show();
39       testlist.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
40     }
41 }

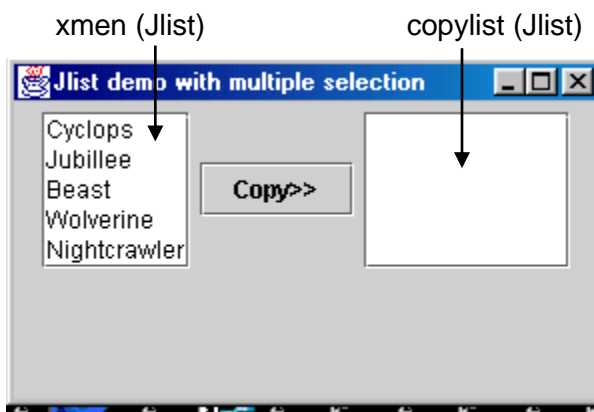
```

The program uses multiple-selection list to copy items from one JList to another. One list is `MULTIPLE_INTERVAL_SELECTION` and the other is a `SINGLE_INTERVAL_SELECTION`. Line 7 creates JList “xmen” and initializes it with the Strings in the array “xmenlist”. Line 15 sets the number of visible rows to 5. Line 16 uses JList method **setFixedCellHeight** to specify the height in pixels of each item in the list. This is to ensure that rows in both JList (xmen and copylist) have the same height. Line 17 specifies that xmen is a `MULTIPLE_INTERVAL_SELECTION` list. Line 18 adds a new

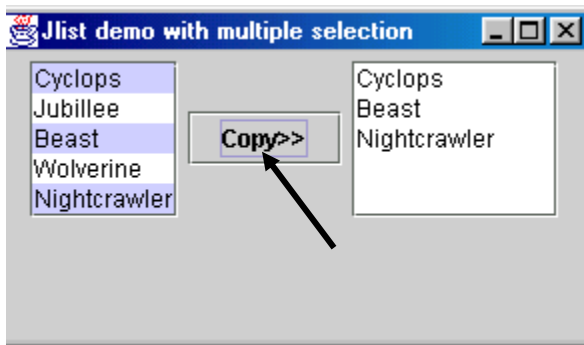
JScrollPane containing “xmenlist” to the content pane. Line 28-32 perform similar tasks for JList “copylist”, which is defined as SINGLE_INTERVAL_SELECTION list. Line 30 uses method setFixedCellWidth to copylists’ width to 100 pixels.

A multiple-selection list does not have a specific event associated with making multiple selections. Normally, an event generated by another GUI component (known as an external event) specifies when the multiple selection in the JList should be processed. In the program, the user clicks the copy button to trigger the event that copies selected items in xmen to copylist.

When the button is clicked, method actionPerformed is called. Line 24 uses method setListData to set the items displayed in copylist. In the same line, xmen list’s method **getSelectedValues** , which returns an array of Objects representing the selected items in the xmen list. In the statement, the returned array is passed as the argument to copylists’ setListData method.



Assuming Cyclops, Beast and Nightcrawler is selected from the list then copy button is clicked, the selected items will be copied to the copylist



Calling a Frame from another frame


```

//This program demonstrates the calling of a frame from another
//frame
//F2f.java
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class F2f extends JFrame
{
    Frame2 subframe = new Frame2("This is the subframe");
    Icon iconheart = new ImageIcon("Heart.jpg");
    Icon iconpaint = new ImageIcon("painting.gif");
    private JButton jbtnshow = new JButton("Show frame",iconheart);
    private JButton jbtnclose = new JButton("Close");
    public F2f(String s)
    {
        super (s);
        Container c = getContentPane();
        c.setLayout(new FlowLayout());
        Buttonhandler btnclick = new Buttonhandler();
        c.add(jbtnshow);
        c.add(jbtnclose);
        jbtnshow.setRolloverIcon(iconpaint);
        jbtnshow.addActionListener(btnclick);
        jbtnclose.addActionListener(btnclick);
        setVisible(true);
        setSize(300,300);
    } //end constructor
    private class Buttonhandler implements ActionListener
    {
        public void actionPerformed(ActionEvent y)
        {
            if (y.getSource()==jbtnshow)
                subframe.show(); //display the other frame
            else
            {
                dispose();
                System.exit(0);
            }
        } //enf if
    } //end buttonhandler
    public static void main(String args[])
    {
        System.out.println("Preparing to display frame... pls wait...");
        F2f mainframe = new F2f("This is the main frame");
        mainframe.show();
        mainframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
} //this is the other frame that will be called
private class Frame2 extends JFrame
{
    JButton jbtnexit = new JButton("Return to main");
    Frame2(String str)
    {
        super(str);
        Container x = getContentPane();
        x.setLayout(new FlowLayout());
        x.setSize(200,200);
        Bhandler btnreturn = new Bhandler();
    }
}

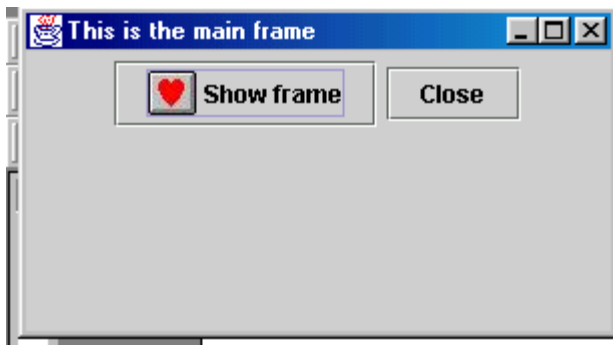
```

```

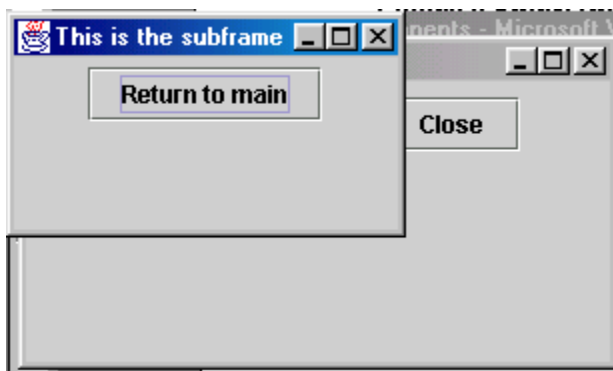
        x.add(jbtnexit);
        jbtnexit.addActionListener(btnreturn);
        setSize(200,200);
    } //constructor
    private class Bhandler implements ActionListener
    { public void actionPerformed (ActionEvent z)
      { dispose();} //return to the calling frame
    } //end bhandler
} //end frame2
} //end class f2f

```

initial output



subframe appears when the “show frame” button is clicked.



Using Menus with Frames

Menus are an integral part of GUIs. Menus allow the user to perform actions without unnecessarily “cluttering” a graphical user interface with extra GUI components. In Swing GUIs, menus can be attached only to objects of the classes that provide the method **setJMenuBar**. Two such classes are `JFrame` and `JApplet`. The classes used to define menus are `JMenuBar`, `JMenu`, `JMenuItem`, `JCheckBoxMenuItem` and `JRadioButtonMenuItem`.

Class **JMenuBar** (a subclass of `JComponent`) contains the methods necessary to manage a menu bar, which is a container for menus.

Class **JMenuItem** (a subclass of javax.swing.AbstractButton) contains the methods necessary to manage menu items. A menu item is a GUI component inside a menu that, when selected, causes an action to be performed. A menu item can be used to initiate an action or it can be a submenu that provides more menu items from which the user can select. Submenus are useful for grouping related menu items in a menu.

Class **JMenu** (a subclass of javax.swing.JMenuItem) contains the methods necessary for managing menus. Menus contains menu items and are added to menu bars or to other menus as submenus. When a menu is clicked, the menu expands to show its list of menu items. Clicking a menu item generates an action event.

Class **JCheckBoxMenuItem** (a subclass of javax.swing.JMenuItem) contains the methods necessary to manage menu items that can be toggled on or off. When a JCheckBoxMenuItem is selected, a check appears to the left of the menu item.

Class **JRadioButtonMenuItem** (a subclass of javax.swing.JMenuItem) contains the methods necessary to manage menu items that can be toggled on or off like JCheckBoxMenuItems. When multiple JRadioButtonMenuItems are maintained as part of a ButtonGroup, only one item in the group can be selected at a given time.

```
//this program demonstrates the use of menu in a frame
//the edit menu however is deactivated or no menu item was
//created to simplify the example
//Jmenudemo1.java
```

```
1    import java.awt.*;
2    import java.awt.event.*;
3    import javax.swing.*;

4    public class Jmenudemo1 extends JFrame
5    {    //create menu bar
6        JMenuBar menubar = new JMenuBar();
7        //set up options in menu bar
8        JMenu fileMenu = new JMenu("File");
9        JMenu editMenu = new JMenu("Edit");
10       //set up options in file menu
11       JMenuItem newItem = new JMenuItem("New");
12       JMenuItem openItem = new JMenuItem("Open");
13       JMenuItem exitItem = new JMenuItem("Exit");

14       public Jmenudemo1(String x)
15       { super(x);
16         //attach menu bar to the frame
17         setJMenuBar(menubar);
18         menubar.add(fileMenu);
```

```

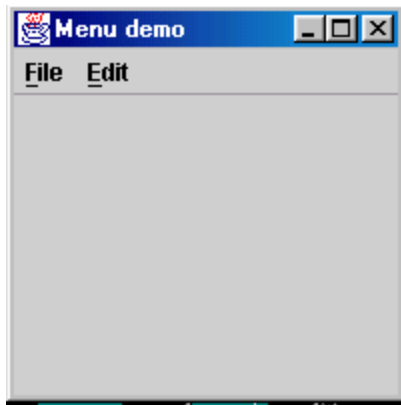
19     menubar.add(editMenu);

20     //set up menu items as submenu of the file menu
21     fileMenu.add(newItem);
22     fileMenu.add(openItem);
23     fileMenu.add(exitItem);

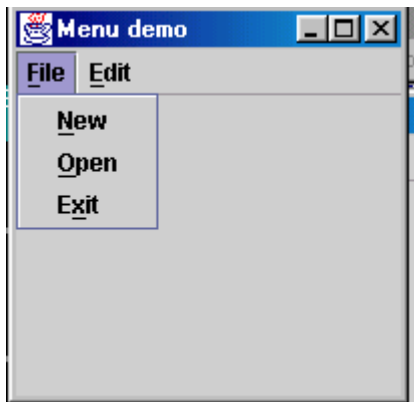
24     //create mnemonic keys
25     fileMenu.setMnemonic('F');
26     editMenu.setMnemonic('E');
27     newItem.setMnemonic('N');
28     openItem.setMnemonic('O');
29     exitItem.setMnemonic('X');

30     //set up events for menu items when selected
31     ButtonHandler click = new ButtonHandler();
32     newItem.addActionListener(click);
33     openItem.addActionListener(click);
34     exitItem.addActionListener(click);
35     setVisible(true);
36     setSize(200,200);
37 } //end of constructor
38 private class ButtonHandler implements ActionListener
39 { public void actionPerformed(ActionEvent a)
40     { if (a.getSource()==newItem) //if option new is selected
41         JOptionPane.showMessageDialog(null,"Option New
selected","New",JOptionPane.PLAIN_MESSAGE);
42     else if (a.getSource()==openItem) //if option Open is selected
43         JOptionPane.showMessageDialog(null,"Option Open
selected","Open",JOptionPane.PLAIN_MESSAGE);
44     else
45         System.exit(0);
46     } //end action performed
47 } // end of buttonhandler
48 public static void main(String args[])
49 { Jmenudemo1 test = new Jmenudemo1("Menu demo");
50     test.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
51     test.show();
52 } //end of main
53 } //end of class jmenudemo1

```



when the file menu is selected...



dialog box appears when the New option is selected in the menu item



// Another sample of using menus other containing gui //components.

//Jmenudemo2.java

```

1      import java.awt.*;
2      import java.awt.event.*;
3      import javax.swing.*;

4      public class Jmenudemo2 extends JFrame
5      {      //create menu bar
6      JMenuBar menubar = new JMenuBar();
7      //set up options in menu bar
8      JMenu fileMenu = new JMenu("File");
9      JMenu styleMenu = new JMenu("Style");
10     //set up options in file menu
11     JMenuItem newItem = new JMenuItem("New");

```

```

12     JMenuItem openItem = new JMenuItem("Open");
13     JMenuItem exitItem = new JMenuItem("Exit");
14     //set up options in style menu
15     JMenuItem bold = new JMenuItem("Bold");
16     JMenuItem italic = new JMenuItem("Italic");

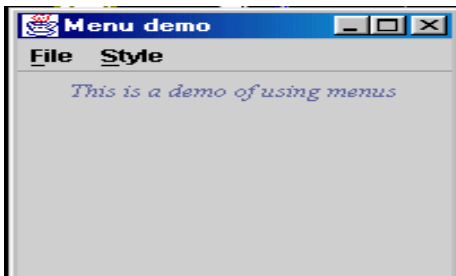
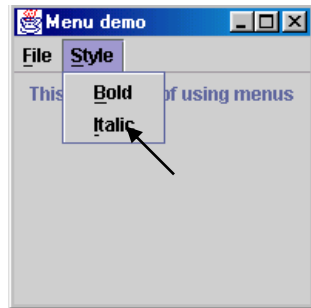
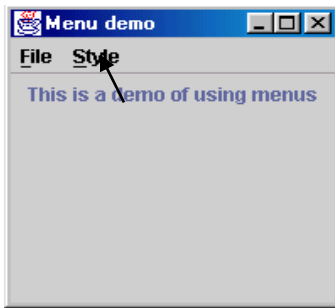
17     JLabel lbl1 = new JLabel("This is a demo of using menus");
18     public Jmenudemo2(String x)
19     { super(x);
20         Container c = getContentPane();
21         c.setLayout(new FlowLayout());
22         c.add(lbl1);
23         //set up menu items as submenu of the file menu
24         fileMenu.add(newItem);
25         fileMenu.add(openItem);
26         fileMenu.add(exitItem);
27         // set up menu items as submenu of the style menu
28         styleMenu.add(bold);
29         styleMenu.add(italic);
30         //create Mnemonic key
31         fileMenu.setMnemonic('F');
32         styleMenu.setMnemonic('S');
33         newItem.setMnemonic('N');
34         openItem.setMnemonic('O');
35         exitItem.setMnemonic('X');
36         bold.setMnemonic('B');
37         italic.setMnemonic('I');
38         //set up events for menu items when selected
39         ButtonHandler click = new ButtonHandler();
40         newItem.addActionListener(click);
41         openItem.addActionListener(click);
42         exitItem.addActionListener(click);
43         bold.addActionListener(click);
44         italic.addActionListener(click);
45         //attach menu bar to the frame
46         setJMenuBar(menuBar);
47         menuBar.add(fileMenu);
48         menuBar.add(styleMenu);
49         setVisible(true);
50         setSize(200,200);
51     } //end of constructor
52     private class ButtonHandler implements ActionListener
53     { public void actionPerformed(ActionEvent a)
54         { if (a.getSource()==newItem) //if option new is selected
55             JOptionPane.showMessageDialog(null,"Option New
selected","New",JOptionPane.PLAIN_MESSAGE);
56             else if (a.getSource()==openItem) //if option Open is selected
57                 JOptionPane.showMessageDialog(null,"Option Open
selected","Open",JOptionPane.PLAIN_MESSAGE);
58             else if (a.getSource()==exitItem) //exit option
59                 System.exit(0);

```

```

60         else if (a.getSource()==bold)//bold option in the style menu
61             lbl1.setFont(new Font("TimesRoman",Font.BOLD,12));
62         else
63             lbl1.setFont(new Font("TimesRoman",Font.ITALIC,12));
64     }
65 } //end of class buttonhandler
66 public static void main(String args[])
67 { Jmenudemo2 test = new Jmenudemo2("Menu demo");
68   test.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
69   test.show();
70 } //end of main
71 } //end of class jmenudemo2

```



//This program demonstrates the use of JCheckboxMenuItem
 // and JRadioButtonMenuItem
 //Jmenudemo3.java

```

1   import javax.swing.*;
2   import java.awt.*;
3   import java.awt.event.*;

4   public class Jmenudemo3 extends JFrame
5   { JMenuBar menubar = new JMenuBar();
6     //set up 2 options in the menu bar
7     JMenu filemenu = new JMenu("File");
8     JMenu stylemenu = new JMenu("Style");
9     //set up menu item for File menu
10    JMenuItem file_exit = new JMenuItem("Exit");
11    //setup options for Style menu
12    JMenu colormenu = new JMenu("Color");
13    JMenu bimenu = new JMenu("Typeface");

```

```

14    //sub menu for Color menu using checkbox
15    JCheckBoxMenuItem style_bold = new JCheckBoxMenuItem("Bold");
16    JCheckBoxMenuItem style_italic = new JCheckBoxMenuItem("Italic");
17    //sub menu for Typeface menu using radio buttons
18    ButtonGroup colorgroup = new ButtonGroup();
19    JRadioButtonMenuItem red = new JRadioButtonMenuItem("Red",false);
20    JRadioButtonMenuItem blue = new JRadioButtonMenuItem("Blue",false);
21    JRadioButtonMenuItem yellow = new JRadioButtonMenuItem("Yellow",false);

22    Font plain = new Font("TimesRoman",Font.PLAIN,15);
23    JLabel lbl = new JLabel("Sample Text");
24    public Jmenudemo3(String s)
25    { super(s);
26        //container for label
27        Container c = getContentPane();
28        c.setLayout(new FlowLayout());
29        //attach menu bar and its menu items
30        setJMenuBar(menubar);
31        menubar.add(filemenu);
32        menubar.add(stylemenu);
33        //create mnemonic keys
34        filemenu.setMnemonic('F');
35        stylemenu.setMnemonic('S');
36        file_exit.setMnemonic('X');
37        colormenu.setMnemonic('C');
38        bimenu.setMnemonic('T');
39        //class to handle event for exit option
40        ButtonHandler pick = new ButtonHandler();
41        filemenu.add(file_exit);
42        file_exit.addActionListener(pick);
43        setSize(300,300);

44        c.add(lbl);
45        lbl.setFont(plain);
46        //add menu items for style menu
47        stylemenu.add(colormenu);
48        //add a line separating the 2 options
49        stylemenu.addSeparator();
50        stylemenu.add(bimenu);
51        //class to handle radio button & checkbox events
52        Itemselected option = new Itemselected();
53        Checkselected checked = new Checkselected();
54        //attach radio buttons for Color option
55        colormenu.add(red);
56        colormenu.add(blue);
57        colormenu.add(yellow);
58        //create logical relationship between radio buttons
59        colorgroup.add(red);
60        colorgroup.add(blue);
61        colorgroup.add(yellow);
62        //attach checkbox for Typeface option

```



```

63     bimenu.add(style_bold);
64     bimenu.add(style_italic);

65     red.addItemListener(option);
66     blue.addItemListener(option);
67     yellow.addItemListener(option);
68     style_bold.addItemListener(option);
69     style_italic.addItemListener(option);
70 } //end of constructor
71 // inner class to handle Exit option in the File menu
72     private class Buttonhandler implements ActionListener
73     { public void actionPerformed(ActionEvent e)
74         {System.exit(0);}
75     } //end of class Buttonhandler
76     //inner class to handle checkbox events
77     private class Checkselected implements ItemListener
78     {
79         private int valBold = Font.PLAIN;
80         private int valItalic = Font.PLAIN;

81         public void itemStateChanged(ItemEvent z)
82         {
83             //bold selection
84             if (z.getSource()==style_bold)
85                 if(z.getStateChange()==ItemEvent.SELECTED)
86                     valBold=Font.BOLD;
87                 else //if bold is unchecked
88                     valBold=Font.PLAIN;
89             // for italic checkbox events

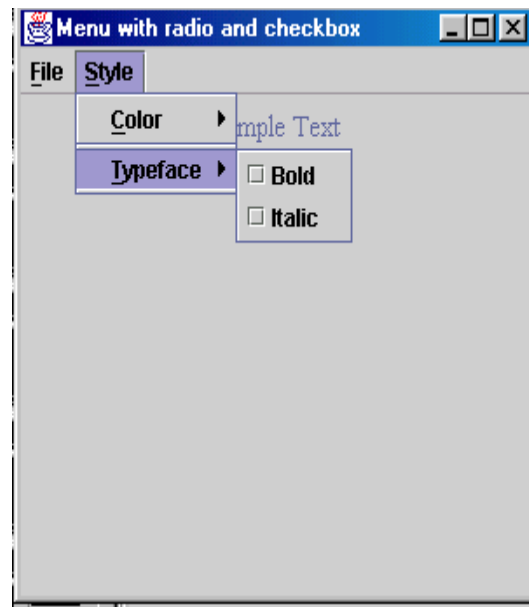
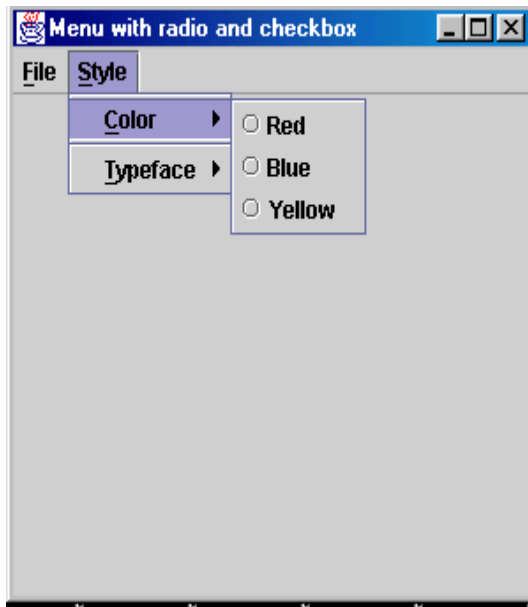
90             if (z.getSource()==style_italic)
91                 if(z.getStateChange()==ItemEvent.SELECTED)
92                     valItalic=Font.ITALIC;
93                 else // if unchecked
94                     valItalic=Font.PLAIN;
95             //set up the typeface of the text
96             lbl.setFont(new Font("TimesRoman",valBold+valItalic,15));
97         } //end of itemstatechanged
98     } //end of class Checkselected
99     //inner class to handle radio buttons
100    private class Itemselected implements ItemListener
101    { public void itemStateChanged(ItemEvent z)
102        { if (z.getSource()==red)
103            lbl.setForeground(Color.red);
104            else if(z.getSource()==blue)
105                lbl.setForeground(Color.blue);
106            else
107                lbl.setForeground(Color.yellow);
108        }
109    } //end of class Itemselected

```

```

110     public static void main(String args[])
111     { Jmenudemo3 test = new Jmenudemo3("Menu with radio and checkbox");
112       test.show();
113       test.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
114     } //end of main
115 } //end of class Jmenudemo3

```



Lesson 9: Database

Overview:

This lesson covers simple database connectivity.

Objectives:

- To learn how to create table in MS Access
- To add, edit and delete records on the DB

What is Database?

A **database** is an organized collection of structured information, or **data**, typically stored electronically in a computer system. A **database** is usually controlled by a **database** management system (DBMS). Most **databases** use structured query language (SQL) for writing and querying **data**.

Java DataBase Connectivity (JDBC)

JDBC API is a Java API that can access any kind of tabular data, especially data stored in a Relational Database. JDBC works with Java on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX.

JDBC is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.

The JDBC library includes APIs for each of the tasks mentioned below that are commonly associated with database usage.

- Making a connection to a database.
- Creating SQL or MySQL statements.
- Executing SQL or MySQL queries in the database.
- Viewing & Modifying the resulting records.

Applications of JDBC

Fundamentally, JDBC is a specification that provides a complete set of interfaces that allows for portable access to an underlying database. Java can be used to write different types of executables, such as –

- Java Applications
- Java Applets
- Java Servlets
- Java ServerPages (JSPs)
- Enterprise JavaBeans (EJBs).

All of these different executables are able to use a JDBC driver to access a database, and take advantage of the stored data.

JDBC provides the same capabilities as Object Database Connectivity (ODBC), allowing Java programs to contain database-independent code.

Create Database

To create a Database using JDBC application. Before executing the following example, make sure you have the following in place –

- You should have admin privilege to create a database in the given schema. To execute the following example, you need to replace the *username* and *password* with your actual user name and password.
- Your MySQL or whatever database you are using, is up and running.

Steps in Creating DB

The following steps are required to create a new Database using JDBC application

–

- **Import the packages:** Requires that you include the packages containing the JDBC classes needed for database programming. Most often, using *import java.sql.** will suffice.
- **Register the JDBC driver:** Requires that you initialize a driver so you can open a communications channel with the database.
- **Open a connection:** Requires using the *DriverManager.getConnection()* method to create a Connection object, which represents a physical connection with the database server.
To create a new database, you need not give any database name while preparing database URL as mentioned in the below example.
- **Execute a query:** Requires using an object of type Statement for building and submitting an SQL statement to the database.
- **Clean up the environment:** Requires explicitly closing all database resources versus relying on the JVM's garbage collection.

SQL Syntax

Structured Query Language (SQL) is a standardized language that allows you to perform operations on a database, such as creating entries, reading content, updating content, and deleting entries.

SQL is supported by almost any database you will likely use, and it allows you to write database code independently of the underlying database.

This chapter gives an overview of SQL, which is a prerequisite to understand JDBC concepts. After going through this chapter, you will be able to Create, Create, Read, Update, and Delete (often referred to as **CRUD** operations) data from a database.

1. Create Database

The CREATE DATABASE statement is used for creating a new database.

Syntax: `CREATE DATABASE DATABASE_NAME;`

2. Drop Database

The DROP DATABASE statement is used for deleting an existing database.

Syntax: `DROP DATABASE DATABASE_NAME;`

3. Create Table

The CREATE TABLE statement is used for creating a new table.

Syntax: `CREATE TABLE table_name
(
 column_name column_data_type,
 column_name column_data_type,
 column_name column_data_type ...);`

4. Drop Table

The DROP TABLE statement is used for deleting an existing table.

Syntax: `DROP TABLE table_name;`

5. Insert Data

The syntax for INSERT, looks similar to the following, where column1, column2, and so on represents the new data to appear in the respective columns

Syntax: INSERT INTO table_name VALUES (column1, column2, ...);

6. Select Data

The SELECT statement is used to retrieve data from a database.

Syntax: SELECT column_name, column_name, ... FROM
table_name WHERE conditions;

7. Update Data

The UPDATE statement is used to update data.

Syntax: UPDATE table_name SET column_name = value, column_name =
value, ... WHERE condition/s;

8. Delete Data

The DELETE statement is used to delete data from tables.

Syntax: DELETE FROM table_name WHERE condition/s;

Sample Programs:

```
// Create Database
// STEP 1. Import required packages
import java.sql.*;

public class JDBCExample {
    // JDBC driver name and database URL
    static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
    static final String DB_URL = "jdbc:mysql://localhost/";

    // Database credentials
    static final String USER = "username";
    static final String PASS = "password";

    public static void main(String[] args) {
        Connection conn = null;
        Statement stmt = null;
        try{
            //STEP 2: Register JDBC driver
            Class.forName("com.mysql.jdbc.Driver");

            //STEP 3: Open a connection
            System.out.println("Connecting to database...");
            conn = DriverManager.getConnection(DB_URL, USER, PASS);

            //STEP 4: Execute a query
            System.out.println("Creating database...");
            stmt = conn.createStatement();
```

```
String sql = "CREATE DATABASE STUDENTS";  
stmt.executeUpdate(sql);
```

```
System.out.println("Database created successfully...");  
}catch(SQLException se){  
    //Handle errors for JDBC  
    se.printStackTrace();  
}catch(Exception e){  
    //Handle errors for Class.forName  
    e.printStackTrace();  
}finally{  
    //finally block used to close resources  
    try{  
        if(stmt!=null)  
            stmt.close();  
    }catch(SQLException se2){  
    }// nothing we can do  
    try{  
        if(conn!=null)  
            conn.close();  
    }catch(SQLException se){  
        se.printStackTrace();  
    }//end finally try  
    }//end try  
    System.out.println("Goodbye!");  
}//end main  
}//end JDBCExample
```

```
// Create Table  
// STEP 1. Import required packages  
import java.sql.*;  
  
public class JDBCExample {  
    // JDBC driver name and database URL  
    static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";  
    static final String DB_URL = "jdbc:mysql://localhost/STUDENTS";  
  
    // Database credentials  
    static final String USER = "username";  
    static final String PASS = "password";  
  
    public static void main(String[] args) {  
        Connection conn = null;  
        Statement stmt = null;  
        try{  
            //STEP 2: Register JDBC driver  
            Class.forName("com.mysql.jdbc.Driver");  
  
            //STEP 3: Open a connection
```

```

System.out.println("Connecting to a selected database...");
conn = DriverManager.getConnection(DB_URL, USER, PASS);
System.out.println("Connected database successfully...");

//STEP 4: Execute a query
System.out.println("Creating table in given database...");
stmt = conn.createStatement();

String sql = "CREATE TABLE REGISTRATION " +
    "(id INTEGER not NULL, " +
    " first VARCHAR(255), " +
    " last VARCHAR(255), " +
    " age INTEGER, " +
    " PRIMARY KEY ( id ))";

stmt.executeUpdate(sql);
System.out.println("Created table in given database...");
}catch(SQLException se){
    //Handle errors for JDBC
    se.printStackTrace();
}catch(Exception e){
    //Handle errors for Class.forName
    e.printStackTrace();
}finally{
    //finally block used to close resources
    try{
        if(stmt!=null)
            conn.close();
    }catch(SQLException se){
    }// do nothing
    try{
        if(conn!=null)
            conn.close();
    }catch(SQLException se){
        se.printStackTrace();
    }//end finally try
}//end try
System.out.println("Goodbye!");
}//end main
}//end JDBCExample

```

```

// Insert Record / Data
// STEP 1. Import required packages
import java.sql.*;

public class JDBCExample {
    // JDBC driver name and database URL
    static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
    static final String DB_URL = "jdbc:mysql://localhost/STUDENTS";

```



```

// Database credentials
static final String USER = "username";
static final String PASS = "password";

public static void main(String[] args) {
    Connection conn = null;
    Statement stmt = null;
    try{
        //STEP 2: Register JDBC driver
        Class.forName("com.mysql.jdbc.Driver");

        //STEP 3: Open a connection
        System.out.println("Connecting to a selected database...");
        conn = DriverManager.getConnection(DB_URL, USER, PASS);
        System.out.println("Connected database successfully...");

        //STEP 4: Execute a query
        System.out.println("Inserting records into the table...");
        stmt = conn.createStatement();

        String sql = "INSERT INTO Registration " +
            "VALUES (100, 'Jhay', 'Villareal', 18)";
        stmt.executeUpdate(sql);
        sql = "INSERT INTO Registration " +
            "VALUES (101, 'Julz', 'Saga', 25)";
        stmt.executeUpdate(sql);
        sql = "INSERT INTO Registration " +
            "VALUES (102, 'Mark', 'Abaya', 30)";
        stmt.executeUpdate(sql);
        sql = "INSERT INTO Registration " +
            "VALUES (103, 'Frans', 'Cruz', 28)";
        stmt.executeUpdate(sql);
        System.out.println("Inserted records into the table...");

    }catch(SQLException se){
        //Handle errors for JDBC
        se.printStackTrace();
    }catch(Exception e){
        //Handle errors for Class.forName
        e.printStackTrace();
    }finally{
        //finally block used to close resources
        try{
            if(stmt!=null)
                conn.close();
        }catch(SQLException se){
            //do nothing
        }
        try{
            if(conn!=null)
                conn.close();
        }catch(SQLException se){

```

```

        se.printStackTrace();
    } //end finally try
} //end try
System.out.println("Goodbye!");
} //end main
} //end JDBCExample

```

```

// Insert Record / Data using PreparedStatement
// STEP 1. Import required packages
import java.sql.*;

public class JDBCExample {
    // JDBC driver name and database URL
    static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
    static final String DB_URL = "jdbc:mysql://localhost/STUDENTS";

    // Database credentials
    static final String USER = "username";
    static final String PASS = "password";

    // Declare variables
    static Scanner scan = new Scanner(System.in);
    static int idno, edad;
    static String first_name, last_name, answer = "yes";
    public static void main(String[] args) {
        Connection conn = null;
        Statement stmt = null;
        try{
            //STEP 2: Register JDBC driver
            Class.forName("com.mysql.jdbc.Driver");

            //STEP 3: Open a connection
            System.out.println("Connecting to a selected database...");
            conn = DriverManager.getConnection(DB_URL, USER, PASS);
            System.out.println("Connected database successfully...");

            //STEP 4: Execute a query
            System.out.println("Inserting records into the table...");

            String sql = String sql = "INSERT INTO Registration (id,first,last,age)" +
                "VALUES (?, ?, ?, ?)";
            while (answer == "yes")
            {
                System.out.print("Enter Id No.: ");
                idno = scan.nextInt();
                System.out.print("Enter first name.: ");
                first_name = scan.nextLine();
                scan.next();
                System.out.print("Enter last name.: ");
                last_name = scan.nextLine();
                System.out.print("Enter age: ");
            }
        }
    }
}

```

```

        edad = scan.nextInt();
        // create insert PreparedStatement
        prepStmt.setString(1,idno);
        prepStmt.setString(2,first_name);
        prepStmt.setString(3,last_name);
        prepStmt.setString(4,edad);

        // execute PreparedStatement
        prepStmt.execute();
        System.out.println("Inserted records into the table...");
        Sytem.out.print("Input new record?[yes/no]: ");
        answer = scan.nextLine();
    }
    conn.close();

} catch(SQLException se){
    //Handle errors for JDBC
    se.printStackTrace();
} catch(Exception e){
    //Handle errors for Class.forName
    e.printStackTrace();
} finally{
    //finally block used to close resources
    try{
        if(stmt!=null)
            conn.close();
    } catch(SQLException se){
    } // do nothing
    try{
        if(conn!=null)
            conn.close();
    } catch(SQLException se){
        se.printStackTrace();
    } //end finally try
} //end try
System.out.println("Goodbye!");
} //end main
} //end JDBCExample

```

```

// Select Data / Record
// STEP 1. Import required packages
import java.sql.*;

public class JDBCExample {
    // JDBC driver name and database URL
    static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
    static final String DB_URL = "jdbc:mysql://localhost/STUDENTS";

    // Database credentials

```

```

static final String USER = "username";
static final String PASS = "password";

public static void main(String[] args) {
    Connection conn = null;
    Statement stmt = null;
    try{
        //STEP 2: Register JDBC driver
        Class.forName("com.mysql.jdbc.Driver");

        //STEP 3: Open a connection
        System.out.println("Connecting to a selected database...");
        conn = DriverManager.getConnection(DB_URL, USER, PASS);
        System.out.println("Connected database successfully...");

        //STEP 4: Execute a query
        System.out.println("Creating statement...");
        stmt = conn.createStatement();

        String sql = "SELECT id, first, last, age FROM Registration";
        ResultSet rs = stmt.executeQuery(sql);
        //STEP 5: Extract data from result set
        while(rs.next()){
            //Retrieve by column name
            int id = rs.getInt("id");
            int age = rs.getInt("age");
            String first = rs.getString("first");
            String last = rs.getString("last");

            //Display values
            System.out.print("ID: " + id);
            System.out.print(", Age: " + age);
            System.out.print(", First: " + first);
            System.out.println(", Last: " + last);
        }
        rs.close();
    }catch(SQLException se){
        //Handle errors for JDBC
        se.printStackTrace();
    }catch(Exception e){
        //Handle errors for Class.forName
        e.printStackTrace();
    }finally{
        //finally block used to close resources
        try{
            if(stmt!=null)
                conn.close();
        }catch(SQLException se){
            //do nothing
        }
        try{
            if(conn!=null)

```

```

        conn.close();
    }catch(SQLException se){
        se.printStackTrace();
    }//end finally try
} //end try
System.out.println("Goodbye!");
} //end main
} //end JDBCExample

```

```

// Update Record / Data
// STEP 1. Import required packages
import java.sql.*;

public class JDBCExample {
    // JDBC driver name and database URL
    static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
    static final String DB_URL = "jdbc:mysql://localhost/STUDENTS";

    // Database credentials
    static final String USER = "username";
    static final String PASS = "password";

    public static void main(String[] args) {
        Connection conn = null;
        Statement stmt = null;
        try{
            //STEP 2: Register JDBC driver
            Class.forName("com.mysql.jdbc.Driver");

            //STEP 3: Open a connection
            System.out.println("Connecting to a selected database...");
            conn = DriverManager.getConnection(DB_URL, USER, PASS);
            System.out.println("Connected database successfully...");

            //STEP 4: Execute a query
            System.out.println("Creating statement...");
            stmt = conn.createStatement();
            String sql = "UPDATE Registration " +
                "SET age = 30 WHERE id in (100, 101)";
            stmt.executeUpdate(sql);

            // Now you can extract all the records
            // to see the updated records
            sql = "SELECT id, first, last, age FROM Registration";
            ResultSet rs = stmt.executeQuery(sql);

            while(rs.next()){
                //Retrieve by column name
                int id = rs.getInt("id");
                int age = rs.getInt("age");
            }
        }
    }
}

```

```

        String first = rs.getString("first");
        String last = rs.getString("last");

        //Display values
        System.out.print("ID: " + id);
        System.out.print(", Age: " + age);
        System.out.print(", First: " + first);
        System.out.println(", Last: " + last);
    }
    rs.close();
} catch (SQLException se) {
    //Handle errors for JDBC
    se.printStackTrace();
} catch (Exception e) {
    //Handle errors for Class.forName
    e.printStackTrace();
} finally {
    //finally block used to close resources
    try {
        if (stmt != null)
            conn.close();
    } catch (SQLException se) {
    } // do nothing
    try {
        if (conn != null)
            conn.close();
    } catch (SQLException se) {
        se.printStackTrace();
    } //end finally try
} //end try
System.out.println("Goodbye!");
} //end main
} //end JDBCExample
// Delete Record / Data
// STEP 1. Import required packages
import java.sql.*;

public class JDBCExample {
    // JDBC driver name and database URL
    static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
    static final String DB_URL = "jdbc:mysql://localhost/STUDENTS";

    // Database credentials
    static final String USER = "username";
    static final String PASS = "password";

    public static void main(String[] args) {
        Connection conn = null;
        Statement stmt = null;
        try {
            //STEP 2: Register JDBC driver

```

```

Class.forName("com.mysql.jdbc.Driver");

//STEP 3: Open a connection
System.out.println("Connecting to a selected database...");
conn = DriverManager.getConnection(DB_URL, USER, PASS);
System.out.println("Connected database successfully...");

//STEP 4: Execute a query
System.out.println("Creating statement...");
stmt = conn.createStatement();
String sql = "DELETE FROM Registration " +
             "WHERE id = 101";
stmt.executeUpdate(sql);

// Now you can extract all the records
// to see the remaining records
sql = "SELECT id, first, last, age FROM Registration";
ResultSet rs = stmt.executeQuery(sql);

while(rs.next()){
    //Retrieve by column name
    int id = rs.getInt("id");
    int age = rs.getInt("age");
    String first = rs.getString("first");
    String last = rs.getString("last");

    //Display values
    System.out.print("ID: " + id);
    System.out.print(", Age: " + age);
    System.out.print(", First: " + first);
    System.out.println(", Last: " + last);
}
rs.close();
}catch(SQLException se){
    //Handle errors for JDBC
    se.printStackTrace();
}catch(Exception e){
    //Handle errors for Class.forName
    e.printStackTrace();
}finally{
    //finally block used to close resources
    try{
        if(stmt!=null)
            conn.close();
    }catch(SQLException se){
        //do nothing
    }try{
        if(conn!=null)
            conn.close();
    }catch(SQLException se){
        se.printStackTrace();
    }
}

```

```
    } //end finally try
  } //end try
  System.out.println("Goodbye!");
} //end main
} //end JDBCExample
```

```
// Sorting Records / Data
// STEP 1. Import required packages
import java.sql.*;

public class JDBCExample {
    // JDBC driver name and database URL
    static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
    static final String DB_URL = "jdbc:mysql://localhost/STUDENTS";

    // Database credentials
    static final String USER = "username";
    static final String PASS = "password";

    public static void main(String[] args) {
        Connection conn = null;
        Statement stmt = null;
        try{
            //STEP 2: Register JDBC driver
            Class.forName("com.mysql.jdbc.Driver");

            //STEP 3: Open a connection
            System.out.println("Connecting to a selected database...");
            conn = DriverManager.getConnection(DB_URL, USER, PASS);
            System.out.println("Connected database successfully...");

            //STEP 4: Execute a query
            System.out.println("Creating statement...");
            stmt = conn.createStatement();

            // Extract records in ascending order by first name.
            System.out.println("Fetching records in ascending order...");
            String sql = "SELECT id, first, last, age FROM Registration" +
                " ORDER BY first ASC";
            ResultSet rs = stmt.executeQuery(sql);

            while(rs.next()){
                //Retrieve by column name
                int id = rs.getInt("id");
                int age = rs.getInt("age");
                String first = rs.getString("first");
                String last = rs.getString("last");

                //Display values
                System.out.print("ID: " + id);
```



```

        System.out.print(", Age: " + age);
        System.out.print(", First: " + first);
        System.out.println(", Last: " + last);
    }

    // Extract records in descending order by first name.
    System.out.println("Fetching records in descending order...");
    sql = "SELECT id, first, last, age FROM Registration" +
        " ORDER BY first DESC";
    rs = stmt.executeQuery(sql);

    while(rs.next()){
        //Retrieve by column name
        int id = rs.getInt("id");
        int age = rs.getInt("age");
        String first = rs.getString("first");
        String last = rs.getString("last");

        //Display values
        System.out.print("ID: " + id);
        System.out.print(", Age: " + age);
        System.out.print(", First: " + first);
        System.out.println(", Last: " + last);
    }
    rs.close();
} catch (SQLException se){
    //Handle errors for JDBC
    se.printStackTrace();
} catch (Exception e){
    //Handle errors for Class.forName
    e.printStackTrace();
} finally{
    //finally block used to close resources
    try{
        if(stmt!=null)
            conn.close();
    } catch (SQLException se){
        // do nothing
    }
    try{
        if(conn!=null)
            conn.close();
    } catch (SQLException se){
        se.printStackTrace();
    }
} //end finally try
} //end try
System.out.println("Goodbye!");
} //end main
} //end JDBCExample

```

Laboratory Exercise:

1. Create a database of salesman:
 - a. Table Description:
 - Salesman Number – Integer – Primary Key – Not Null
 - Salesman Name – VarChar
 - Quarterly Sales – Real (Float)
 - Note: input of sales per quarter (1st, 2nd, 3rd, 4th)
 - Total Sales - Double
 - Commission – Real (Float)
 - b. Compute the Total Sales
 - c. Compute the Commission based on the following:

<u>Total Sales</u>	<u>Commission</u>
<= 5000	10% of Total Sales
<= 10000	15% of Total Sales
<= 15000	20% of Total Sales
<= 20000	25% of Total Sales
> 20000	35% of Total Sales or 15000 whichever is higher
 - d. Search a salesman (input must be Salesman Number), determine if the salesman is existing in the database, otherwise update the record by inputting new values of Quarterly Sales.
 - e. Search a salesman (input must be Salesman Number), determine if the salesman is existing in the database, otherwise delete the record.
 - f. Display all the record of database.
2. Modify #1, input Salesman Number:
 - 2.1. Search if the record is existing or not;
 - 2.2. If exist, ask if you are going to update or delete the record:
 - 2.2.1. If update, change the values of Salesman Name and Quarterly Sales
 - 2.2.2. If delete, remove the record from the table;
 - 2.3. If not, display "Record not exists!" and input another ID number repeat 2.1 to 2.3;
 - 2.4. Display the new contents of table.

Bibliography

Farrel, Joyce. Java Programming Comprehensive. Philippine copyright 1999.

Cadenhead, R, Lemay, L. Sams Teach Yourself Java 2 in 21 Days. Sams Publishing 1999.

Perry, Greg. Teach Yourself Object-Oriented Programming with Turbo C++ in 21 days
Sams Publishing 1993.

Munn, Lee Chuk. Object Oriented Program with Java 1999.

Internet Resources:

1. <https://www.redbooks.ibm.com/>
2. <http://java.sun.com/javase/downloads>
3. <https://www.tutorialpoints.com/jdbc>
4. <https://www.w3schools.com/java>