



UIUCCL JAVA

Lecture Notes

ABSTRACT

This sheet is prepared for the learners of UIUCCL (UIU Computer Club) Java workshop, modified & updated from the notes of our honorable Dr. Dewan Md. Farid sir.

Prepared by

Khaled Hasan Rony [Vice President of UIUCCL]

Chapter 1 : Introduction

1.1 What is Java

Java is a powerful programming language and it is used to develop robust applications.

Java is a computer programming language that is used to build softwares. It is also an application, development, and deployment environment. Java is object oriented programming language to help us visualize the program in real-life terms. The syntax of Java is similar to C and C++ based language. Java applications are typically compiled to bytecode (class file) that can run on any Java Virtual Machine (JVM) regardless of computer architecture.

1.2 Brief History of Java

In 1990, Sun Microsystems began a research project named Green to develop the C and C++ based language. James Gosling, a network software designer was assigned to this project.



Gosling's solution to the problems of C++ was a new language called Oak after an oak tree outside his window at Sun. In May 1995, Sun Microsystems formally announced Java (Oak was renamed Java). Finally, in 1996, Java arrived in the market as a programming language.

With the advent of Java 2 (released initially as J2SE 1.2 in December 1998), new versions had multiple configurations built for different types of platforms. For example, J2EE targeted enterprise applications and the greatly stripped-down version J2ME for mobile

applications (Mobile Java). J2SE designated the Standard Edition. In 2006, for marketing purposes, Sun renamed new J2 versions as Java EE, Java ME, and Java SE, respectively.

On November 13, 2006, Sun released much of Java as free and open source software, (FOSS), under the terms of the GNU General Public License (GPL). On May 8, 2007, Sun finished the process, making all of Java's core code available under free software/open-source distribution terms.

1.3 Principles

There were 5 primary goals in the creation of the Java language :

1. it must be "simple, object-oriented and familiar"
2. it must be "robust and secure"
3. it must be "architecture-neutral and portable"
4. it must execute with "high performance"
5. it must be "interpreted, threaded and dynamic"

1.4 Versions

Major release versions of Java, along with their release dates :

Version	Release Date
JDK 1.0	January 21, 1996
JDK 1.1	February 19, 1997
J2SE 1.2	December 8, 1998
J2SE 1.3	May 8, 2000
J2SE 1.4	February 6, 2002
J2SE 5.0	September 30, 2004
Java SE 6	December 11, 2006
Java SE 7	July 28, 2011
Java SE 8	March 18, 2014

1.5 What is Java Applets

A Java applet is a small application which is written in Java and delivered to users in the form of bytecode. The user launches the Java applet from a web page, and the applet is then executed within a Java Virtual Machine (JVM) in a process separate from the web browser itself. A Java applet can appear in a frame of the web page, a new application window.

1.6 Java Virtual Machine (JVM)

A Java virtual machine is a software that is capable of executing Java bytecode. Code for the JVM is stored in **.class** or **.jar** files, each of which contains code for at most one public class. JVM enables the Java application to be platform independent. JVM is distributed along with a set of standard class libraries that implement the Java application programming interface (API).

JVM mainly performs 3 tasks :

Loads Code : The class loader loads all classes needed for the execution of a program.

Verifies Code : The byte code verifier tests format of code fragment and checks code fragments for illegal code, which is code that forges pointers, violates access rights on objects, or attempts to change object type.

Execute Code : Performed by the runtime interpreter.

The Java Runtime Environment (JRE) is an implementation of the Java Virtual Machine (JVM), which actually executes Java programs.

The Java Development Kit (JDK) is the original Java development environment for Java programmers. The JDK consists of a library of standard classes and a collection of utilities for building, testing, and documenting Java programs.

can modify bytecode at runtime based on frequent executed methods

1.7 Automatic Memory Management

Many programming languages like C and C++ permit the memory to be allocated dynamically at runtime. After the allocated memory is no longer required, the program or

runtime environment should de-allocate the memory. If the program does not de-allocate the memory that can crash eventually when there is no memory left for the system to allocate. These types of programs are said memory leaks.

Java overcome this problem by using garbage collection. It provides a system level thread that tracks each memory allocation. During idle cycles in the JVM, the garbage collection thread checks for and frees any memory that can be freed in the object lifecycle.

1.8 A Simple Java Program

This example program prints a string **Welcome To Java** in output.

```
1 public class Welcome
2 {
3     public static void main(String[] args)
4     {
5         System.out.println("Welcome to Java");
6     }
7 }
```

Write this code in Notepad for Windows or Text Editor for Linux and save the file as **Welcome.java** (Source files must be named after the public class). Now you are ready to compile and run the program. To compile the program, open a command window or terminal, and change to the directory where the file **Welcome.java** is stored and type **javac Welcome.java**. If the program contains no errors then a new file **Welcome.class** will be created in the same directory. Now you can run this bytecodes file **Welcome.class** by typing the command **java Welcome**

Line 1 declares the user-defined class named **Welcome**. Every Java program consists at least one public class. The **Welcome.java** file creates a **Welcome.class** file when the source file is compiled.

Line 2 beginning of Welcome class.

Line 3 declares the **main** method, where the program starts executing.

The following describes each element of Line 3 :

public The access modifier of main() method is public, because the main() method can be accessed by anything including the Java technology interpreter.

static The main() is static method because on instance of the class is needed to execute static methods and static method is a single copy.

void The return type of the main() method is void, because the main() method does not return any value.

String[] args The main() method declares the single parameter of String array named args. If the program is given any arguments on its command line, these are passed into the main() method through this args array of String type.

Line 4 beginning of main() method.

Line 5 displays or prints a line of text in the command window. System.out is known as the standard output object.

Line 6 end of the main() method.

Line 7 end of the Welcome class.

1.9 Another Simple Java Program

```
1  class Hello
2  {
3      public void display()
4      {
5          System.out.println("Welcome to Java");
6      }
7  }
8
9  public class Welcome2
10 {
11     public static void main(String[] args)
12     {
13         Hello hello = new Hello();
14
15         hello.display();
16     }
17 }
```

The name of the source file must be the same as the name of the public class declaration in that file. A source file can include more than one class declaration, but only one class can be declared public.

Line 13 creates an object of **Hello** class named **hello**.

Line 15 call the **display()** method of **Hello** class using the **hello** object with the dot notation.

Line 1 declares a user-defined class **Hello** with default access control.

Line 3 declares a user-defined method named **display()**.

1.10 Declaring Classes, Attributes, Constructors and Methods

A class is a software blueprint. A class defines the set of data elements (called attributes), as well as the set of behaviors or functions (called methods). Together, attributes and methods are called members of a class. In Java technology, class name usually starts with a capital letter, attribute and method name usually starts with a small letter. The access modifier for the classes in the Java technology can be **public** or **default**.

A constructor is a set of instructions designed to initialize an instance or object. The name of the constructor must always be the same as the class name. Constructor do not return any value. Valid modifiers for constructors are **public**, **protected**, and **private**.

The Default Constructor : Every Java class has at least one constructor. If you do not write a constructor in a class then Java technology provides a default constructor for you. This constructor takes no arguments and has an empty body.

1.11 Another Java Program

```
1  public class MyDate
2  {
3      private int day;
4      private int month;
5      private int year;
6
7      public MyDate()
8      {
9          day = 1;
10         month = 6;
11         year = 1979;
12     }
13
14     public void showDate()
15     {
16         System.out.println("Day: " + day);
17         System.out.println("Month: " + month);
18         System.out.println("Year: " + year);
19     }
20
21     public static void main(String[] args)
22     {
23         MyDate date = new MyDate();
24
25         date.showDate();
26     }
27 }
```

1.12 Source File Layout

A Java technology source file declares firstly package statement (only one package), secondly import statements, and finally classes. Which are following bellow :

1. *package* < topPackageName > .[< subPackageName >];
2. *import* < packageName > .[subPackageName]*;
3. < classDeclaration >*

The package statement is a way to group related classes. The package statement place at beginning of the source file and only one package statement declaration is permitted. If a Java technology source file does not contain a package declaration then the classes declared in the file belong to the default package. The package names are separated by dots. The package name to be entirely lower case.

The import statement is used to make classes in other packages accessible to the current class. Normally, the Java compiler places the .class files in the same directory as the source file present. We can reroute the class file to another directory using the -d option of the javac command (javac -d . ClassName.java).

1.13 Compile & Run Java Program from Command / Terminal

```
1  package bankingProject;
2
3  public class Account
4  {
5      public String accountName = "Savings Account";
6
7      public void showAccountName()
8      {
9          System.out.println(accountName);
10     }
11
12     public static void main(String[] args)
13     {
14         Account account = new Account();
15
16         account.showAccountName();
17     }
18 }
```

To compile this code, open a command window or terminal and change to the directory where the program is saved and type **`javac -d . Account.java`** then a package or folder will be create in the same directory named **bankingProject** and inside the directory there will be a .class file named **Account.class**. Now you can run this **Account.class** file in **bankingProject** package by typing the command **`java bankingProject.Account`**

```
1 package bankingProject;
2
3 //import bankingProject.Account;
4
5 public class Customer
6 {
7     public String name = "James Bond";
8
9     public static void main(String[] args)
10    {
11        Account account = new Account();
12        account.showAccountName();
13
14        Customer customer = new Customer();
15        System.out.println(customer.name);
16    }
17 }
```

Similarly, we can compile and run this code.

Line 3 we import the **Account** class from **bankingProject** package.

Line 11 we create object of **Account** class.

Line 12 we call the method of **Account** class.

1.14 Comments in Java

In Java Technology, we can add comments to Java source code in 3 ways which are shown below :

```
1  public class CommentsInJava
2  {
3      public static void main(String[] args)
4      {
5          System.out.println("Comments in Java");
6
7          //comment on one line
8
9          /*
10         * comment on one
11         * or more lines
12         *
13         */
14
15         /**
16         * documentation comment
17         * can also span one or more lines
18         */
19     }
20 }
```

1.15 Blocks

A block in Java is a group of one or more statements enclosed in braces. A block begins with an opening brace { and ends with a closing brace }. Between the opening and closing braces, you can code one or more statements.

In Java source code, block statements are executed first,

then constructor statements are executed,

and then method statements are executed.

```
1  public class BlockTest
2  {
3      public BlockTest()
4      {
5          System.out.println("constructor executed");
6      }
7
8      {
9          System.out.println("block executed");
10     }
11
12     {
13         System.out.println("outer statement in block executed");
14         {
15             System.out.println("inner statement in block executed");
16         }
17     }
18
19     public void method()
20     {
21         System.out.println("method executed");
22     }
23
24     public static void main(String[] args)
25     {
26         BlockTest bt = new BlockTest();
27
28         bt.method();
29     }
30 }
```

1.16 Practical Example of Blocks

```
1 public class BlockTestPractical
2 {
3     public BlockTestPractical()
4     {
5         //50 lines of code
6     }
7
8     public BlockTestPractical(int a)
9     {
10        //60 lines of code
11    }
12
13    public BlockTestPractical(int a, int b)
14    {
15        //70 lines of code
16    }
17
18    public BlockTestPractical(int a, int b, int c)
19    {
20        //80 lines of code
21    }
22
23    {
24        //10000 lines of code
25    }
26
27    public static void main(String[] args)
28    {
29        BlockTestPractical object1 = new BlockTestPractical();
30
31        BlockTestPractical object2 = new BlockTestPractical(10);
32
33        BlockTestPractical object3 = new BlockTestPractical(10, 20);
34
35        BlockTestPractical object4 = new BlockTestPractical(10, 20, 30);
36    }
37 }
```

Return to page no. 42 if you have come from that page.

1.17 Java Programming Language Keywords

Keywords have special meaning to the Java technology compiler. They identify a data type name or program construct name. Lists of keywords used in Java programming language are shown in table below.

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

1.18 Java Identifiers

In Java technology, identifier is a name given to a variable, class, or method. Java identifier start with a letter, underscore (_), or dollar sign (\$). Subsequence characters can be digits. Identifiers are case sensitive and have no maximum length. An identifier cannot be a keyword, but it can contain a keyword as part of its name.

Examples of some legal identifiers :

- identifier
- username
- _a
- \$b
- ___u_i_u
- _\$
- user789
- new_java_class
- this_is_a_large_identifier

Examples of some illegal identifiers :

- :a
- -b
- uiu#
- 7ok
- new
- class

1.19 this Reference

In Java technology, this keyword is used to resolved ambiguity between instance variables and parameters. It is also used to pass the current object as a parameter to another method.

```
1 public class AddNumbers
2 {
3     private int number1;
4     private int number2;
5     private int sum;
6
7     public AddNumbers(int number1, int number2)
8     {
9         this.number1 = number1;
10        this.number2 = number2;
11    }
12
13    public void add()
14    {
15        sum = (number1 + number2);
16
17        System.out.println("Sum = " + sum);
18    }
19
20    public static void main(String[] args)
21    {
22        AddNumbers addNumbers = new AddNumbers(10, 20);
23
24        addNumbers.add();
25    }
26 }
```

1.20 Data Types in Java

In Java technology, data are divided into 2 broad categories :

1. primitive types
2. class types

Primitive data are eight types in 4 categories :

1. Logical : boolean (true or false)
2. Textual : char (16 bits)
3. Integral :
 - a. byte (8 bits)
 - b. short (16 bits)
 - c. int (32 bits)
 - d. long (64 bits)
4. Floating point :
 - a. float (32 bits)
 - b. double (64 bits)

Class or reference data used to create objects which are 2 types :

1. Textual : String
2. All classes declared by us

1.21 Variable Initialization

In Java source code, the compiler will assign default value for the class variables, if the programmer does not initialize the class variables.

But local variables must be initialized manually before use, because Java compiler will not assign the local variables and local variable does not have any modifier.

The following table shows the default values of different variable types :

Variable Type	Default Value	Range
boolean	false	true, false
char	'\u0000'	'\u0000' (0) to '\uffff' (65,535)
byte	0	-128 to 127
short	0	-32,768 to 32,767
int	0	-2,14,74,83,648 to 2,14,74,83,647
long	0L	-92,233,72,03,685,47,75,808 to 92,233,72,03,685,47,75,807
float	0.0f	single precision 32-bit IEEE 754 floating point
double	0.0d	double precision 64-bit IEEE 754 floating point
String or any object	null	

1.22 Operators in Java Programming

The Java programming language operators are similar in style and function to those of C and C++.

1. Arithmetic operators (+, -, *, /, %, ++, --)
2. Relational operators (==, !=, <, >, <=, >=)
3. Bitwise operators (&, |, ^, ~, <<, >>, >>>)
4. Logical operators (&&, ||, !)

5. Assignment operators (=, +=, -=, *=, /=, %=, <<=, >>=, &=, |=, ^=)
6. Conditional operator (? :)
7. instanceof operator

1.23 String Concatenation in Java

The + operator performs a concatenation of String objects, producing a new String.

Example :

```
1  public class StringConcatenationTest
2  {
3      public static void main(String[] args)
4      {
5          String salutation = "Mr. ";
6
7          //String name = "Dewan Md. Farid";
8          String name = "Dewan " + "Md. " + "Farid";
9
10         String title = salutation + name;
11
12         System.out.println(title);
13     }
14 }
```

1.24 Casting

Casting means assigning a value of one type to a variable of another type.

```
1 public class CastTest
2 {
3     public static void main(String[] args)
4     {
5         long bigValue = 99L;
6         int smallValue = (int) bigValue; //Casting
7
8         System.out.println(smallValue);
9
10        smallValue = 50;
11        bigValue = smallValue; //Auto Casting
12
13        System.out.println(bigValue);
14    }
15 }
```

Chapter 2 : Flow Controls & Arrays

2.1 if else statement

```
1 public class IfElse
2 {
3     public static void main(String[] args)
4     {
5         int number = 10;
6
7         System.out.println(number);
8
9         if( number > 50 )
10        {
11            System.out.println("Number is greater than 50");
12        }
13        else if( number < 50 )
14        {
15            System.out.println("Number is less than 50");
16        }
17        else
18        {
19            System.out.println("Number is equal to 50");
20        }
21    }
22 }
```

2.2 switch case

```
1 public class SwitchCase
2 {
3     public void method(int number)
4     {
5         switch(number)
6         {
7             case 1: System.out.println("Number is equal to 1");
8             break;
9
10            case 2: System.out.println("Number is equal to 2");
11            break;
12
13            case 3: System.out.println("Number is equal to 3");
14            break;
15
16            default: System.out.println("Number");
17        }
18    }
19
20    public static void main(String[] args)
21    {
22        SwitchCase switchCase = new SwitchCase();
23
24        switchCase.method(3);
25    }
26 }
```

2.3 for loop

```
1 public class ForLoop
2 {
3     public static void main(String[] args)
4     {
5         for( int i = 1; i <= 10; i++ )
6         {
7             System.out.println(i);
8         }
9
10        //System.out.println(i);
11    }
12 }
```

2.4 Fibonacci Series

```
1 public class Fibonacci
2 {
3     public static void main(String[] args)
4     {
5         int a = 1;
6         int b = 0;
7
8         System.out.println("Fibonacci series:");
9
10        for( int i = 1; i <= 10; i++ )
11        {
12            System.out.println(i + ". " + a);
13
14            a = (a + b);
15            b = (a - b);
16        }
17    }
18 }
```

2.5 Array

An array is a group of variables of the same data type referable by a common name. The data type can be either a primitive data type or a class or reference type.

```
1 public class ArrayTest
2 {
3     public static void main(String[] args)
4     {
5         int[] myArray;//declaring array
6
7         myArray = new int[5];//creating array
8
9         //int[] myArray = new int[5];
10
11         for( int i = 0; i < 5; i++ )
12         {
13             myArray[i] = (100 + i);
14         }
15
16         for( int i = 0; i < 5; i++ )
17         {
18             System.out.println( myArray[i] );
19         }
20     }
21 }
```

```
1  class Bird
2  {
3      public void fly()
4      {
5          System.out.println("Bird can fly.");
6      }
7  }
8
9  public class BirdArrayTest
10 {
11     public static void main(String[] args)
12     {
13         Bird[] myArray = new Bird[3];
14
15         for( int i = 0; i < 3; i++ )
16         {
17             myArray[i] = new Bird();
18         }
19
20         for( int i = 0; i < 3; i++ )
21         {
22             System.out.print("Object : " + (i + 1) + ", ");
23
24             myArray[i].fly();
25         }
26     }
27 }
```



```
1  class Car
2  {
3      public String carName;
4      public String carModel;
5
6      public Car(String carName, String carModel)
7      {
8          this.carName = carName;
9          this.carModel = carModel;
10     }
11
12     public void display()
13     {
14         System.out.println("Name: " + carName + "\tModel: " + carModel);
15     }
16 }
17
18 public class CarArrayTest
19 {
20     public static void main(String[] args)
21     {
22         Car[] myArray = new Car[3];
23
24         myArray[0] = new Car("Ferrari", "Ferrari 488 GTB");
25         myArray[1] = new Car("Tesla", "Model 3");
26         myArray[2] = new Car("BMW", "BMW i8");
27
28         for( int i = 0; i < 3; i++ )
29         {
30             myArray[i].display();
31         }
32     }
33 }
```

2.6 Multidimensional Arrays in Java

The Java programming language provide multidimensional arrays, which is arrays of arrays.

```
1 public class MultiDimensionalArray1
2 {
3     public static void main(String[] args)
4     {
5         int[][] myArray = new int[3][3];
6
7         for( int i = 0; i < myArray.length; i++ )
8         {
9             for( int j = 0; j < myArray[i].length; j++ )
10            {
11                myArray[i][j] = (i + j + 10);
12            }
13        }
14
15        for( int i = 0; i < myArray.length; i++ )
16        {
17            for( int j = 0; j < myArray[i].length; j++ )
18            {
19                System.out.print(myArray[i][j] + " ");
20            }
21
22            System.out.println();
23        }
24    }
25 }
```

```
1 public class MultiDimensionalArray2
2 {
3     public static void main(String[] args)
4     {
5         /*int[][] myArray =
6         {
7             {10, 20, 30},
8             {40, 50, 60, 70},
9             {85, 90},
10        };*/
11
12        int[][] myArray = new int[3][];
13
14        myArray[0] = new int[5];
15        myArray[1] = new int[3];
16        myArray[2] = new int[4];
17
18        for( int i = 0; i < myArray.length; i++ )
19        {
20            for( int j = 0; j < myArray[i].length; j++ )
21            {
22                myArray[i][j] = (i + j + 10);
23            }
24        }
25
26        for( int i = 0; i < myArray.length; i++ )
27        {
28            for( int j = 0; j < myArray[i].length; j++ )
29            {
30                System.out.print(myArray[i][j] + " ");
31            }
32
33            System.out.println();
34        }
35    }
36 }
```

2.7 Matrix Multiplication

```
1  import java.util.Random;
2
3  public class MatrixMultiplication
4  {
5      public static void main(String[] args)
6      {
7          Random random = new Random();
8
9          int[][] a = new int[3][3];
10         int[][] b = new int[3][3];
11         int[][] c = new int[3][3];
12
13         for( int i = 0; i < 3; i++ )
14         {
15             for( int j = 0; j < 3; j++ )
16             {
17                 a[i][j] = random.nextInt(5);
18             }
19         }
20
21         for( int i = 0; i < 3; i++ )
22         {
23             for( int j = 0; j < 3; j++ )
24             {
25                 b[i][j] = random.nextInt(5);
26             }
27         }
28
29         System.out.println("Matrix a:");
30
31         for( int i = 0; i < 3; i++ )
32         {
33             for( int j = 0; j < 3; j++ )
34             {
35                 System.out.print(a[i][j] + " ");
36             }
37
38             System.out.println();
39         }
```

```
40
41     System.out.println("\nMatrix b:");
42
43     for( int i = 0; i < 3; i++ )
44     {
45         for( int j = 0; j < 3; j++ )
46         {
47             System.out.print(b[i][j] + " ");
48         }
49
50         System.out.println();
51     }
52
53     //Matrix Multiplication
54
55     for( int i = 0; i < 3; i++ )
56     {
57         for( int j = 0; j < 3; j++ )
58         {
59             c[i][j] = ( a[i][0] * b[0][j] ) + ( a[i][1] * b[1][j] ) +
60 ( a[i][2] * b[2][j] );
61         }
62     }
63
64
65     System.out.println("\nMatrix c:");
66
67     for( int i = 0; i < 3; i++ )
68     {
69         for( int j = 0; j < 3; j++ )
70         {
71             System.out.print(c[i][j] + " ");
72         }
73
74         System.out.println();
75     }
76 }
77 }
```

2.8 Array Bounds, Resizing, Copy in Java

In Java technology, all array indexes begin at 0. The number of elements in an array is stored as part of the array object in the length attribute. If an out-of-bounds runtime access occurs, then a runtime exception is thrown. After array is created, you cannot resize an array. However, you can use the same reference variable to refer to an entirely new array. Java provides a special method in the System class, `arraycopy()` to copy arrays.

```
1 public class ArrayBoundsResizingCopy
2 {
3     public static void main(String[] args)
4     {
5         int[] a = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
6         int[] b = {100, 200, 300, 400, 500};
7
8         System.arraycopy(b, 0, a, 0, b.length);
9
10        System.out.print("a = ");
11        for( int i = 0; i < a.length; i++ )
12        {
13            System.out.print(a[i]+ " ");
14        }
15        System.out.println();
16
17        System.out.print("b = ");
18        for( int i = 0; i < b.length; i++ )
19        {
20            System.out.print(b[i]+ " ");
21        }
22        System.out.println();
23
24
25        a = new int[5];
26
27        /*for( int i = 0; i < a.length; i++ )
28        {
29            a[i] = (i + 1);
30        }*/
31
32        System.out.print("a = ");
33        for( int i = 0; i < a.length; i++ )
34        {
35            System.out.print(a[i]+ " ");
36        }
37        System.out.println();
38    }
39 }
```

2.9 Enhanced for loop

```
1 public class EnhancedForLoop
2 {
3     public static void main(String[] args)
4     {
5         int[] myArray = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};
6
7         for( int i : myArray )
8         {
9             System.out.print(i + " ");
10        }
11
12        /*int[][] myArray =
13        {
14            {10, 20, 30},
15            {40, 50, 60, 70},
16            {85, 90},
17        };
18
19        for( int[] i : myArray )
20        {
21            for( int j : i )
22            {
23                System.out.print(j + " ");
24            }
25            System.out.println();
26        }*/
27    }
28 }
```

2.10 Random Number Generation

```
1 public class RandomNumberTest1
2 {
3     public static void main(String[] args)
4     {
5         int number;
6
7         for( int i = 1; i <= 10; i++ )
8         {
9             number = (int)( Math.random() * 100 );
10
11             System.out.println(i + ". " + number);
12         }
13     }
14 }
```

```
1 import java.util.Random;
2
3 public class RandomNumberTest2
4 {
5     public static void main(String[] args)
6     {
7         int number;
8
9         Random random = new Random();
10
11         for( int i = 1; i <= 10; i++ )
12         {
13             number = random.nextInt(100);
14
15             System.out.println(i + ". " + number);
16         }
17     }
18 }
```


2.11 Input from User (Using Scanner)

```
1  import java.util.Scanner;
2
3  public class ScannerTest
4  {
5      public static void main(String[] args)
6      {
7          String string;
8          int intNumber;
9          float floatNumber;
10         double doubleNumber;
11
12         Scanner scanner = new Scanner(System.in);
13
14         System.out.print("Enter a string: ");
15         //string = scanner.next();
16         string = scanner.nextLine();
17         System.out.println("You entered: " + string);
18
19         System.out.print("Enter an integer: ");
20         intNumber = scanner.nextInt();
21         System.out.println("You entered: " + intNumber);
22
23         System.out.print("Enter a float number: ");
24         floatNumber = scanner.nextFloat();
25         System.out.println("You entered: " + floatNumber);
26
27         System.out.print("Enter a double number: ");
28         doubleNumber = scanner.nextDouble();
29         System.out.println("You entered: " + doubleNumber);
30
31         scanner.close();
32     }
33 }
```

Chapter 3 : Object Oriented Programming

3.1 Access Control

Modifier	Same Class	Same Package	Sub Class	Universe
private	Yes			
default	Yes	Yes		
protected	Yes	Yes	Yes	
public	Yes	Yes	Yes	Yes

3.2 Wrapper Classes

The Java programming language provides wrapper classes to manipulate primitive data elements as object. Each Java primitive data type has a corresponding wrapper class in the java.lang package. Each wrapper class encapsulates a single primitive value. If we change the primitive data types to their object equivalents, which is called boxing, then we need to use the wrapper classes. From J2SE version 5.0 the autoboxing concept is introduced.

Primitive Data Type	Wrapper Class
boolean	Boolean
byte	Byte
char	Character
short	Short
int	Integer
long	Long
float	Float
double	Double

```

1  public class BoxingTest
2  {
3      public void boxing(int primitiveInteger)
4      {
5          Integer wrapperInteger = new Integer(primitiveInteger); //boxing
6
7          int number = wrapperInteger.intValue(); //unboxing
8          System.out.println(number);
9      }
10
11     public void autoboxing(int primitiveInteger)
12     {
13         Integer wrapperInteger = primitiveInteger; //autoboxing
14
15         int number = wrapperInteger; //autounboxing
16         System.out.println(number);
17     }
18
19     public static void main(String[] args)
20     {
21         BoxingTest boxingTest = new BoxingTest();
22
23         boxingTest.boxing(100);
24         boxingTest.autoboxing(200);
25     }
26 }

```

3.3 Key Features of Java

The Java technology programming language supports 4 key features of Object Oriented Programming :

1. *Encapsulation*
2. *Inheritance*
3. *Polymorphism*
4. *Abstraction*

3.4 Encapsulation

Encapsulation is the technique of hiding some members of a class from other classes but provides a public interface to access that members.

```
1  class MyNumber
2  {
3      private int number;
4
5      public void setNumber(int number)
6      {
7          this.number = number;
8      }
9
10     public int getNumber()
11     {
12         return number;
13     }
14 }
15
16 public class EncapsulationTest
17 {
18     public static void main(String[] args)
19     {
20         MyNumber myNumber = new MyNumber();
21
22         myNumber.setNumber(10);
23         System.out.println( myNumber.getNumber() );
24
25         myNumber.setNumber(30);
26         System.out.println( myNumber.getNumber() );
27     }
28 }
```

At line 3, attribute number is a private member of MyNumber class, so this attribute will not be accessible form other classes. But from the EncapsulationTest class we are accessing the number variable of MyNumber class using set and get methods of MyNumber class.

3.5 Inheritance

Inheritance is the process of sub-classing that we can create a child-class from a parent-class. Java programming language permits single inheritance, because in Java a class can extend one other class only. A child-class can inherit all of the members from the parent-class, but it does not inherit the constructor.

```
1  class Parent
2  {
3      //public int a;
4      //public int b;
5
6      public int a = 10;
7      public int b = 20;
8
9      public void show()
10     {
11         System.out.println("Printing from parent class:");
12
13         System.out.println("a = " + a);
14         System.out.println("b = " + b);
15     }
16 }
17
18 public class Child extends Parent
19 {
20     public static void main(String[] args)
21     {
22         Child child = new Child();
23
24         child.show();
25
26         System.out.println("Printing from child class:");
27         System.out.println("a = " + child.a);
28         System.out.println("b = " + child.b);
29     }
30 }
31
```

In the above code, Child class extends Parent class, so Child class become child class of Parent class.

3.6 Overriding Methods

If a method is defined in a sub-class so that the name, return type, and argument list must exactly those of a method in the parent class, then the new method is said to override the old one.

The overriding method can not be less accessible than the method it overrides

```
1  class A
2  {
3      public void show()
4      {
5          System.out.println("Bird can fly.");
6      }
7  }
8
9  public class B extends A
10 {
11     public void show()
12     {
13         System.out.println("Bird can fly in the sky.");
14     }
15
16     public static void main(String[] args)
17     {
18         B b = new B();
19
20         b.show();
21     }
22 }
```

Line 11 declares the method `show()`, which override the parent class `show()` method of line 3.

3.7 Invoking Overriding Methods

A sub-class method can invoke a super-class method using the super keyword.

```
1  class Employee
2  {
3      public String name = "Hasan";
4      public int salary = 100000;
5
6      public void showDetails()
7      {
8          System.out.println("Name: " + name + "\tSalary: " + salary);
9      }
10 }
11
12 public class Manager extends Employee
13 {
14     public String department = "Engineering";
15
16     public void showDetails()
17     {
18         super.showDetails();
19         System.out.println("Department: " + department);
20         //super.showDetails();
21     }
22
23     public static void main(String[] args)
24     {
25         Manager manager = new Manager();
26
27         manager.showDetails();
28     }
29 }
```

In the above code, line 18 invokes the parent class method showDetails().

3.8 Invoking Parent Class Constructor

A sub-class constructor can invoke a super-class constructor using the super keyword. In line 24, example of parent class constructor invoking is shown.

```
1  class Employee1
2  {
3      public String name;
4      public int salary;
5
6      public Employee1(String name, int salary)
7      {
8          this.name = name;
9          this.salary = salary;
10     }
11
12     public void showDetails()
13     {
14         System.out.println("Name: " + name + "\tSalary: " + salary);
15     }
16 }
17
18 public class Manager1 extends Employee1
19 {
20     public String department;
21
22     public Manager1(String department)
23     {
24         super("Hasan", 100000);
25         this.department = department;
26     }
27
28     public void showDetails()
29     {
30         super.showDetails();
31         System.out.println("Department: " + department);
32         //super.showDetails();
33     }
34
35     public static void main(String[] args)
36     {
37         Manager1 manager1 = new Manager1("Engineering");
38
39         manager1.showDetails();
40     }
41 }
```


3.9 Overloading Methods

We can declare several methods of same name in a class, which is known as methods overloading. For overloading methods **argument lists must be different** and **return type can be different**. In the following code, there are 3 methods of same name with different argument, which is an example of overloading methods.

```
1  public class ABC
2  {
3      public int a;
4      public int b;
5
6      public void setValue()
7      {
8          a = 1;
9          b = 2;
10     }
11     public void setValue(int a)
12     {
13         this.a = a;
14         b = 2;
15     }
16     public int setValue(int a, int b)
17     {
18         this.a = a;
19         this.b = b;
20
21         return (a + b);
22     }
23
24     public static void main(String[] args)
25     {
26         ABC abc = new ABC();
27
28         abc.setValue();
29         System.out.println("a = " + abc.a);
30         System.out.println("b = " + abc.b);
31
32         abc.setValue(10);
33         System.out.println("a = " + abc.a);
34         System.out.println("b = " + abc.b);
35
36         int sum = abc.setValue(10, 20);
37         System.out.println("a = " + abc.a);
38         System.out.println("b = " + abc.b);
39         System.out.println("sum = " + sum);
40     }
41 }
```

3.10 Overloading Constructors

We can declare several constructors with different arguments in a class, which is known as overloading constructors.

```
1  public class ABCD
2  {
3      public int a;
4      public int b;
5
6      public ABCD()
7      {
8          a = 1;
9          b = 2;
10     }
11     public ABCD(int a)
12     {
13         this.a = a;
14         b = 2;
15     }
16     public ABCD(int a, int b)
17     {
18         this.a = a;
19         this.b = b;
20     }
21
22     public static void main(String[] args)
23     {
24         ABCD object1 = new ABCD();
25         System.out.println("a = " + object1.a);
26         System.out.println("b = " + object1.b);
27
28         ABCD object2 = new ABCD(10);
29         System.out.println("a = " + object2.a);
30         System.out.println("b = " + object2.b);
31
32         ABCD object3 = new ABCD(10, 20);
33         System.out.println("a = " + object3.a);
34         System.out.println("b = " + object3.b);
35     }
36 }
```

Practical Example of Blocks -> go to page no. 13

3.11 Example of Encapsulation & Inheritance with Overriding, Overloading and Invoking

```
1  class Human
2  {
3      private int weight;
4
5      public Human(int weight)
6      {
7          this.weight = weight;
8      }
9
10     public void setWeight(int weight)
11     {
12         this.weight = weight;
13     }
14
15     public int getWeight()
16     {
17         return weight;
18     }
19
20     public void show()
21     {
22         System.out.println("Weight = " + weight);
23     }
24 }
```

```
25
26 public class ManOfSteel extends Human
27 {
28     public int power;
29
30     public ManOfSteel(int power)
31     {
32         super(70); //invoking parent class constructor
33         this.power = power;
34     }
35
36     public ManOfSteel(int weight, int power) //overloading constructor
37     {
38         super(weight); //invoking parent class constructor
39         this.power = power;
40     }
41
42     public void show() //overriding method
43     {
44         super.show(); //invoking parent class method
45         System.out.println("Power = " + power);
46     }
47
48     public void show(String string) //overloading method
49     {
50         System.out.println(string);
51     }
52
53     public static void main(String[] args)
54     {
55         ManOfSteel manOfSteel1 = new ManOfSteel(90);
56         manOfSteel1.show();
57
58         ManOfSteel manOfSteel2 = new ManOfSteel(50, 100);
59         manOfSteel2.show("Henry Cavill");
60     }
61 }
```

3.12 Polymorphism

Polymorphism is the technique of creating object of parent-class through the constructor of child-class. Using polymorphism we can call or execute the child-class overriding method by the parent-class object.

```
1  class Man
2  {
3      public void fly()
4      {
5          System.out.println("Man can not fly.");
6      }
7  }
8
9  class SuperMan extends Man
10 {
11     public void fly()
12     {
13         System.out.println("Superman can fly.");
14     }
15 }
16
17 class SpiderMan extends Man
18 {
19     public void fly()
20     {
21         System.out.println("Spiderman can not fly, but can jump.");
22     }
23 }
24
25 public class TestMan
26 {
27     public static void main(String[] args)
28     {
29         Man man1 = new Man();
30         man1.fly();
31
32         Man man2 = new SuperMan();//polymorphism
33         man2.fly();
34
35         Man man3 = new SpiderMan();//polymorphism
36         man3.fly();
37     }
38 }
```

3.13 Homogeneous Collection

Homogeneous collection is the collection of objects that have a common class.

```
1 public class TestHomogeneous
2 {
3     public static void main(String[] args)
4     {
5         TestHomogeneous[] myArray = new TestHomogeneous[3];
6
7         myArray[0] = new TestHomogeneous();
8         myArray[1] = new TestHomogeneous();
9         myArray[2] = new TestHomogeneous();
10    }
11 }
```

3.14 Heterogeneous Collection

Heterogeneous collection is a collection of dissimilar objects or classes.

```
1  class Man
2  {
3      public void fly()
4      {
5          System.out.println("Man can not fly.");
6      }
7  }
8
9  class SuperMan extends Man
10 {
11     public void fly()
12     {
13         System.out.println("Superman can fly.");
14     }
15 }
16
17 class SpiderMan extends Man
18 {
19     public void fly()
20     {
21         System.out.println("Spiderman can not fly, but can jump.");
22     }
23 }
24
25 public class TestHeterogeneous
26 {
27     public static void main(String[] args)
28     {
29         Man[] myArray = new Man[3];
30
31         myArray[0] = new Man();
32         myArray[1] = new SuperMan();
33         myArray[2] = new SpiderMan();
34
35         myArray[0].fly();
36         myArray[1].fly();
37         myArray[2].fly();
38     }
39 }
```

3.15 The Object Class

In the Java technology, the Object class is the root of all classes. If a class in Java programming does not extends any class then this class extends the Object class by default.

```
1 public class ObjectTest extends Object
2 {
3     public static void main(String[] args)
4     {
5         System.out.println("Hello");
6     }
7 }
8
9 /*
10 public class ObjectTest
11 {
12     public static void main(String[] args)
13     {
14         System.out.println("Hello");
15     }
16 }
17 */
```


3.16 The instanceof Operator

Using the instanceof operator, we can know the actual object of a class.

At line 35, we are passing object through argument list of test(Object object) method but we do not know this object is from which class. Then we use instanceof operator to know object is from which class by the conditions.

```
1  class Ferrari
2  {
3      public void show()
4      {
5          System.out.println("I am Ferrari");
6      }
7  }
8
9  class Tesla
10 {
11     public void show()
12     {
13         System.out.println("I am Tesla");
14     }
15 }
16
17 class BMW
18 {
19     public void show()
20     {
21         System.out.println("I am BMW");
22     }
23 }
24
25 class Toyota
26 {
27     public void show()
28     {
29         System.out.println("I am Toyota");
30     }
31 }
```

```
32
33 public class TestInstanceof1
34 {
35     public void test(Object object)
36     {
37         if( object instanceof Ferrari )
38         {
39             System.out.println("Object is of Ferrari class.");
40         }
41         else if( object instanceof Tesla )
42         {
43             System.out.println("Object is of Tesla class.");
44         }
45         else if( object instanceof BMW )
46         {
47             System.out.println("Object is of BMW class.");
48         }
49         else if( object instanceof Toyota )
50         {
51             System.out.println("Object is of Toyota class.");
52         }
53         else
54         {
55             System.out.println("Object is none of Ferrari, Tesla,
BMW, Toyota class.");
56         }
57     }
58
59     public static void main(String[] args)
60     {
61         Ferrari object1 = new Ferrari();
62         Tesla object2 = new Tesla();
63         BMW object3 = new BMW();
64         Toyota object4 = new Toyota();
65
66         TestInstanceof1 t = new TestInstanceof1();
67
68         t.test(object1);
69         t.test(object2);
70         t.test(object3);
71         t.test(object4);
72     }
73 }
```

```
1 public class TestInstanceof2
2 {
3     public static void main(String[] args)
4     {
5         Ferrari object1 = new Ferrari();
6         Tesla object2 = new Tesla();
7         BMW object3 = new BMW();
8         Toyota object4 = new Toyota();
9
10        if( (Object)object1 instanceof Ferrari )
11        {
12            System.out.println("object is of Ferrari class.");
13        }
14        else if( (Object)object1 instanceof Tesla )
15        {
16            System.out.println("object is of Tesla class.");
17        }
18        else if( (Object)object1 instanceof BMW )
19        {
20            System.out.println("object is of BMW class.");
21        }
22        else if( (Object)object1 instanceof Toyota )
23        {
24            System.out.println("object is of Toyota class.");
25        }
26        else
27        {
28            System.out.println("object is none of Ferrari, Tesla,
29            BMW, Toyota class.");
30        }
31        //check for object2
32        //check for object3
33        //check for object4
34    }
35 }
```

3.17 The equals Method

The method `public boolean equals(Object object)` of `Object` class in the `java.lang` package compares two objects for equality. This method returns `true`, only if the two objects being compared refer to the same object. And, `==` operator performs an equivalent comparison. If `x == y` returns `true`, that means `x` and `y` refer to the same object.

```
1 public class TestEquals1
2 {
3     public int a;
4     public int b;
5
6     public TestEquals1(int a, int b)
7     {
8         this.a = a;
9         this.b = b;
10    }
11
12    public static void main(String[] args)
13    {
14        TestEquals1 t1 = new TestEquals1(5, 10);
15        TestEquals1 t2 = new TestEquals1(5, 10);
16
17        if( t1 == t2 )
18        {
19            System.out.println("t1 is identical to t2");
20        }
21        else
22        {
23            System.out.println("t1 is not identical to t2");
24        }
25        //output: t1 is not identical to t2
26
27        if( t1.equals(t2) )
28        {
29            System.out.println("t1 is equal to t2");
30        }
31        else
32        {
33            System.out.println("t1 is not equal to t2");
34        }
35        //output: t1 is not equal to t2
36
37        System.out.println("Setting: t1 = t2");
38        t1 = t2;
```

```
39
40     if( t1 == t2 )
41     {
42         System.out.println("t1 is identical to t2");
43     }
44     else
45     {
46         System.out.println("t1 is not identical to t2");
47     }
48     //output: t1 is identical to t2
49
50     if( t1.equals(t2) )
51     {
52         System.out.println("t1 is equal to t2");
53     }
54     else
55     {
56         System.out.println("t1 is not equal to t2");
57     }
58     //output: t1 is equal to t2
59 }
60 }
```

3.18 Difference between "==" Operator and equals() Method in Java

"==" operator	equals() method
It is a binary operator in java.	It is a public method of java.lang.Object class.
It compares the two objects based on their location in the memory.	The default version of equals method also does the comparison of two objects based on their location in the memory. But, you can override the equals method so that it performs the comparison of two objects on some condition.
It can be used on both primitive types as well as on derived types.	It can be used only on derived types.
It is the best suitable for primitive types.	It is the best suitable for derived types.
You can't override the "==" operator. It behaves same for all objects.	You can override the equals method according to your business requirements.

Reference :

<http://javaconceptoftheday.com/difference-between-operator-and-equals-method-in-java/>

```
1 public class TestEquals2
2 {
3     public int a;
4     public int b;
5
6     public TestEquals2(int a, int b)
7     {
8         this.a = a;
9         this.b = b;
10    }
11
12    public boolean equals(Object object)
13    {
14        boolean result = false;
15
16        if( (object != null) && (object instanceof TestEquals2) )
17        {
18            TestEquals2 t = (TestEquals2)object;
19
20            if( (a == t.a) && (b == t.b) )
21            {
22                result = true;
23            }
24        }
25
26        return result;
27    }
28
29    public static void main(String[] args)
30    {
31        TestEquals2 t1 = new TestEquals2(5, 10);
32        TestEquals2 t2 = new TestEquals2(5, 10);
33
34        if( t1 == t2 )
35        {
36            System.out.println("t1 is identical to t2");
37        }
38        else
39        {
40            System.out.println("t1 is not identical to t2");
41        }
42        //output: t1 is not identical to t2
```

```
43
44     if( t1.equals(t2) )
45     {
46         System.out.println("t1 is equal to t2");
47     }
48     else
49     {
50         System.out.println("t1 is not equal to t2");
51     }
52     //output: t1 is equal to t2
53
54     System.out.println("Setting: t1 = t2");
55     t1 = t2;
56
57     if( t1 == t2 )
58     {
59         System.out.println("t1 is identical to t2");
60     }
61     else
62     {
63         System.out.println("t1 is not identical to t2");
64     }
65     //output: t1 is identical to t2
66
67     if( t1.equals(t2) )
68     {
69         System.out.println("t1 is equal to t2");
70     }
71     else
72     {
73         System.out.println("t1 is not equal to t2");
74     }
75     //output: t1 is equal to t2
76 }
77 }
```



```
1 public class TestEqualsString
2 {
3     public static void main(String[] args)
4     {
5         String string1 = new String("UIU");
6         String string2 = new String("UIU");
7
8         if( string1 == string2 )
9         {
10            System.out.println("string1 is identical to string2");
11        }
12        else
13        {
14            System.out.println("string1 is not identical to string2");
15        }
16        //output: string1 is not identical to string2
17
18        if( string1.equals(string2) )
19        {
20            System.out.println("string1 is equal to string2");
21        }
22        else
23        {
24            System.out.println("string1 is not equal to string2");
25        }
26        //output: string1 is equal to string2
27
28        System.out.println("Setting: string1 = string2");
29        string1 = string2;
30
31        if( string1 == string2 )
32        {
33            System.out.println("string1 is identical to string2");
34        }
35        else
36        {
37            System.out.println("string1 is not identical to string2");
38        }
39        //output: string1 is identical to string2
40
41        if( string1.equals(string2) )
42        {
43            System.out.println("string1 is equal to string2");
44        }
45        else
46        {
47            System.out.println("string1 is not equal to string2");
48        }
49        //output: string1 is equal to string2
```

```
50
51
52
53     String string3 = new String("UIU");
54     String string4 = new String("CSE");
55
56     if( string3 == string4 )
57     {
58         System.out.println("string3 is identical to string4");
59     }
60     else
61     {
62         System.out.println("string3 is not identical to string4");
63     }
64     //output: string3 is not identical to string4
65
66     if( string3.equals(string4) )
67     {
68         System.out.println("string3 is equal to string4");
69     }
70     else
71     {
72         System.out.println("string3 is not equal to string4");
73     }
74     //output: string3 is not equal to string4
75
76     System.out.println("Setting: string3 = string4");
77     string3 = string4;
78
79     if( string3 == string4 )
80     {
81         System.out.println("string3 is identical to string4");
82     }
83     else
84     {
85         System.out.println("string3 is not identical to string4");
86     }
87     //output: string3 is identical to string4
88
89     if( string3.equals(string4) )
90     {
91         System.out.println("string3 is equal to string4");
92     }
93     else
94     {
95         System.out.println("string3 is not equal to string4");
96     }
97     //output: string3 is equal to string4
```

```
98
99
100
101     String string5 = "UIU";
102     String string6 = "UIU";
103
104     if( string5 == string6 )
105     {
106         System.out.println("string5 is identical to string6");
107     }
108     else
109     {
110         System.out.println("string5 is not identical to string6");
111     }
112     //output: string5 is identical to string6
113
114     if( string5.equals(string6) )
115     {
116         System.out.println("string5 is equal to string6");
117     }
118     else
119     {
120         System.out.println("string5 is not equal to string6");
121     }
122     //output: string5 is equal to string6
123
124     System.out.println("Setting: string5 = string6");
125     string5 = string6;
126
127     if( string5 == string6 )
128     {
129         System.out.println("string5 is identical to string6");
130     }
131     else
132     {
133         System.out.println("string5 is not identical to string6");
134     }
135     //output: string5 is identical to string6
136
137     if( string5.equals(string6) )
138     {
139         System.out.println("string5 is equal to string6");
140     }
141     else
142     {
143         System.out.println("string5 is not equal to string6");
144     }
145     //output: string5 is equal to string6
```

```
146
147
148
149     String string7 = "UIU";
150     String string8 = "CSE";
151
152     if( string7 == string8 )
153     {
154         System.out.println("string7 is identical to string8");
155     }
156     else
157     {
158         System.out.println("string7 is not identical to string8");
159     }
160     //output: string7 is not identical to string8
161
162     if( string7.equals(string8) )
163     {
164         System.out.println("string7 is equal to string8");
165     }
166     else
167     {
168         System.out.println("string7 is not equal to string8");
169     }
170     //output: string7 is not equal to string8
171
172     System.out.println("Setting: string7 = string8");
173     string7 = string8;
174
175     if( string7 == string8 )
176     {
177         System.out.println("string7 is identical to string8");
178     }
179     else
180     {
181         System.out.println("string7 is not identical to string8");
182     }
183     //output: string7 is identical to string8
184
185     if( string7.equals(string8) )
186     {
187         System.out.println("string7 is equal to string8");
188     }
189     else
190     {
191         System.out.println("string7 is not equal to string8");
192     }
193     //output: string7 is equal to string8
194 }
}
```

```
1 public class TestEqualsInteger
2 {
3     public static void main(String[] args)
4     {
5         Integer number1 = 10;
6         Integer number2 = 10;
7
8         if( number1 == number2 )
9         {
10            System.out.println("number1 is identical to number2");
11        }
12        else
13        {
14            System.out.println("number1 is not identical to number2");
15        }
16        //output: number1 is identical to number2
17
18        if( number1.equals(number2) )
19        {
20            System.out.println("number1 is equal to number2");
21        }
22        else
23        {
24            System.out.println("number1 is not equal to number2");
25        }
26        //output: number1 is equal to number2
27
28        System.out.println("Setting: number1 = number2");
29        number1 = number2;
30
31        if( number1 == number2 )
32        {
33            System.out.println("number1 is identical to number2");
34        }
35        else
36        {
37            System.out.println("number1 is not identical to number2");
38        }
39        //output: number1 is identical to number2
40
41        if( number1.equals(number2) )
42        {
43            System.out.println("number1 is equal to number2");
44        }
45        else
46        {
47            System.out.println("number1 is not equal to number2");
48        }
49        //output: number1 is equal to number2
```

```
50
51
52
53     Integer number3 = 10;
54     Integer number4 = 20;
55
56     if( number3 == number4 )
57     {
58         System.out.println("number3 is identical to number4");
59     }
60     else
61     {
62         System.out.println("number3 is not identical to number4");
63     }
64     //output: number3 is not identical to number4
65
66     if( number3.equals(number4) )
67     {
68         System.out.println("number3 is equal to number4");
69     }
70     else
71     {
72         System.out.println("number3 is not equal to number4");
73     }
74     //output: number3 is not equal to number4
75
76     System.out.println("Setting: number3 = number4");
77     number3 = number4;
78
79     if( number3 == number4 )
80     {
81         System.out.println("number3 is identical to number4");
82     }
83     else
84     {
85         System.out.println("number3 is not identical to number4");
86     }
87     //output: number3 is identical to number4
88
89     if( number3.equals(number4) )
90     {
91         System.out.println("number3 is equal to number4");
92     }
93     else
94     {
95         System.out.println("number3 is not equal to number4");
96     }
97     //output: number3 is equal to number4
98 }
99 }
```

3.19 The toString Method

The toString() method of Object class convert an object to a String representation, which returns the class name and its reference address. Many classes override toString to provide more useful information.

```
1 public class TesttoString
2 {
3     public static void main(String[] args)
4     {
5         TesttoString t = new TesttoString();
6
7         System.out.println(t);
8
9         System.out.println( t.toString() );
10    }
11 }
```

3.20 The static keyword

In Java technology, members (attributes, methods, and nested classes) of a class can be declared with static keyword that are associated with the class rather than the instances of the class. The static variable sometime called class variable and static method sometime called class method.

A static variable is similar to global variable in other programming languages. We can use the static members of a class without creating the object or instance of that class.

The static methods can only access the local attributes, static class attributes, and its parameters. Attempting to access non-static class attributes in a static methods will cause a compilation error.

We can not override static method.

The main() method is a static method because the Java Virtual Machine (JVM) does not create an instance of the class when executing the main method. The static block code executed once when the class is loaded.

```
1  public class TestStatic
2  {
3      public static int counter = 10;
4
5      public static void incrementCounter()
6      {
7          counter++;
8      }
9
10     public static void main(String[] args)
11     {
12         System.out.println("counter = " + counter);
13         incrementCounter();
14         System.out.println("counter = " + counter);
15
16         TestStatic t1 = new TestStatic();
17         TestStatic t2 = new TestStatic();
18
19         System.out.println( t1.toString() );
20         System.out.println( t2.toString() );
21
22         System.out.println( "counter = " + t2.counter );
23         t2.counter++;
24         //counter++;
25         System.out.println( "counter = " + t1.counter );
26     }
27 }
```


3.21 The final keyword

In the Java technology, the final keyword can be applied to the classes, methods, and variables. In Java, final classes can not be inherited, final methods can not be overridden but can be overloaded, final variables are constant. Any attempt to change the value of a final variable causes a compilation error. A blank final variable is a final variable that is not initialized in its declaration, but it can be initialized later once only.

```
1 public class TestFinal
2 {
3     public final int a = 10;
4     public final int b;
5
6     public TestFinal(int b)
7     {
8         this.b = b;
9     }
10
11    public final void show()
12    {
13        System.out.println("Final method can not be overridden");
14    }
15    public final void show(int number)
16    {
17        System.out.println("Final method can be overloaded");
18    }
19
20    public static void main(String[] args)
21    {
22        TestFinal testFinal = new TestFinal(20);
23
24        System.out.println("a = " + testFinal.a);
25        System.out.println("b = " + testFinal.b);
26
27        testFinal.show();
28        testFinal.show(10);
29
30        //testFinal.b = 30;
31
32        final int number;
33
34        number = 100;
35
36        System.out.println("number = " + number);
37
38        //number = 200;
39    }
40 }
```

3.22 Declaring Abstract Class

In Java technology, we can declare a method in a class which have no body, this method is called abstract method, if a class contains an abstract method then the class will be abstract class. An abstract class must have at least one abstract method and can never be instantiated.

```
1  abstract class Vehicle
2  {
3      public String model = "Tesla Model 3";
4      public String year = "2017";
5
6      public abstract void goFast();
7
8      public void show()
9      {
10         System.out.println("Model: " + model);
11         System.out.println("Year: " + year);
12     }
13 }
14
15 public class Car extends Vehicle
16 {
17     public void goFast()
18     {
19         System.out.println("Car can go fast.");
20     }
21
22     public static void main(String[] args)
23     {
24         //Vehicle vehicle = new Vehicle();//compilation error
25
26         Car car = new Car();
27
28         car.show();
29         car.goFast();
30     }
31 }
```

3.23 Declaring Interface

In Java interfaces are declaring only the contract and no implementation, like a 100% abstract superclass. All methods declared in an interface are public and abstract (do not need to actually type the public and abstract modifiers in the method declaration, but the method is still always public and abstract). Interface methods must not be static. Because interface methods are abstract, they cannot be marked final, strictfp, or native. All variables in an interface are public, static, and final in interfaces. An interface can extend one or more other interfaces. An interface cannot implement another interface or class.

```
1  interface Bounceable
2  {
3      public abstract void bounce();
4
5      void setBounce(int bounce);
6  }
7
8  public class Tyre implements Bounceable
9  {
10     public void bounce()
11     {
12         System.out.println("Tyre is bounceable.");
13     }
14
15     public void setBounce(int bounce)
16     {
17         int a = bounce;
18
19         System.out.println(a);
20     }
21
22     public static void main(String[] args)
23     {
24         Tyre tyre = new Tyre();
25
26         tyre.bounce();
27         tyre.setBounce(15);
28     }
29 }
```

```
1 interface AB
2 {
3     public abstract void printHello();
4
5     void printGoodbye();
6 }
7
8 interface Bounceable extends AB
9 {
10    public abstract void bounce();
11
12    void setBounce(int bounce);
13 }
14
15 public class Tyre implements Bounceable
16 {
17     public void printHello()
18     {
19         System.out.println("Hello");
20     }
21
22     public void printGoodbye()
23     {
24         System.out.println("Goodbye");
25     }
26
27     public void bounce()
28     {
29         System.out.println("Tyre is bounceable.");
30     }
31
32     public void setBounce(int bounce)
33     {
34         int a = bounce;
35
36         System.out.println("Bounce value = " + a);
37     }
38
39     public static void main(String[] args)
40     {
41         Tyre tyre = new Tyre();
42
43         tyre.printHello();
44         tyre.printGoodbye();
45         tyre.bounce();
46         tyre.setBounce(15);
47     }
48 }
```

3.24 Example of abstract & interface

In Java technology, a java class can only extend another one class, but can implement one or more interfaces.

```
1  abstract class Animal
2  {
3      public void type()
4      {
5          System.out.println("Animal");
6      }
7
8      public abstract void name();
9  }
10
11 interface Cat
12 {
13     public abstract void jump();
14
15     void run();
16 }
17
18 interface Bird
19 {
20     void fly();
21 }
```

```
22
23 public class Tiger extends Animal implements Cat, Bird
24 {
25     public void name()
26     {
27         System.out.println("Tiger");
28     }
29
30     public void jump()
31     {
32         System.out.println("Tiger can jump.");
33     }
34
35     public void run()
36     {
37         System.out.println("Tiger can run.");
38     }
39
40     public void fly()
41     {
42         //
43     }
44
45     public static void main(String[] args)
46     {
47         Tiger tiger = new Tiger();
48
49         tiger.type();
50         tiger.name();
51         tiger.jump();
52         tiger.run();
53         tiger.fly();
54     }
55 }
```

3.25 Exceptions

Exceptions are mechanism used by many programming languages to describe what to do when errors occurs.

There are 2 types of exceptions in Java programming, known as **checked** and **unchecked** exceptions.

Checked exceptions are those that the programmer can easily handle this type of exceptions like: file not found, and network failure etc.

Unchecked exceptions are arisen from the conditions that are difficult for programmers to handle. Unchecked exceptions are called runtime exceptions.

In Java, Exception class is the base class that represents checked and unchecked exceptions and RuntimeException class is the base class that is used for the unchecked exceptions.

```
1 public class TestExceptionA
2 {
3     public static void main(String[] args)
4     {
5         int sum = 0;
6
7         for( int i = 0; i < args.length; i++ )
8         {
9             sum += Integer.parseInt(args[i]);
10        }
11
12        System.out.println("sum = " + sum);
13    }
14 }
```

This program works fine if all of the command-line arguments are integers.

Compile : javac TestExceptionA.java

Run : java TestExceptionA 2 4 6 8

Output : 20

But this program fails if any of the arguments are not integers.

Run : java TestExceptionA 2 4 six 8

Output : NumberFormatException

```
Command Prompt

D:\Java\jdk1.8.0_31\bin>javac TestExceptionA.java

D:\Java\jdk1.8.0_31\bin>java TestExceptionA 2 4 6 8
sum = 20

D:\Java\jdk1.8.0_31\bin>java TestExceptionA 2 4 six 8
Exception in thread "main" java.lang.NumberFormatException: For input string: "six"
    at java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
    at java.lang.Integer.parseInt(Integer.java:580)
    at java.lang.Integer.parseInt(Integer.java:615)
    at TestExceptionA.main(TestExceptionA.java:9)

D:\Java\jdk1.8.0_31\bin>
```

3.26 try catch statement

To avoid this problem, we can use the try-catch statement. If any exception or error occurs in the try block then the catch block will execute. If try block execute without any error, then catch block will not execute.

```
1 public class TestExceptionB
2 {
3     public static void main(String[] args)
4     {
5         int sum = 0;
6
7         for( int i = 0; i < args.length; i++ )
8         {
9             try
10            {
11                sum += Integer.parseInt(args[i]);
12            }
13            catch(Exception ex)
14            {
15                System.out.println("index: " + i + " is not integer");
16                System.out.println(ex);
17            }
18        }
19
20        System.out.println("sum = " + sum);
21    }
22 }
```


This program works well if all or any of the command-line arguments are not integers.

Compile : `javac TestExceptionB.java`

Run : `java TestExceptionB 2 4 6 8`

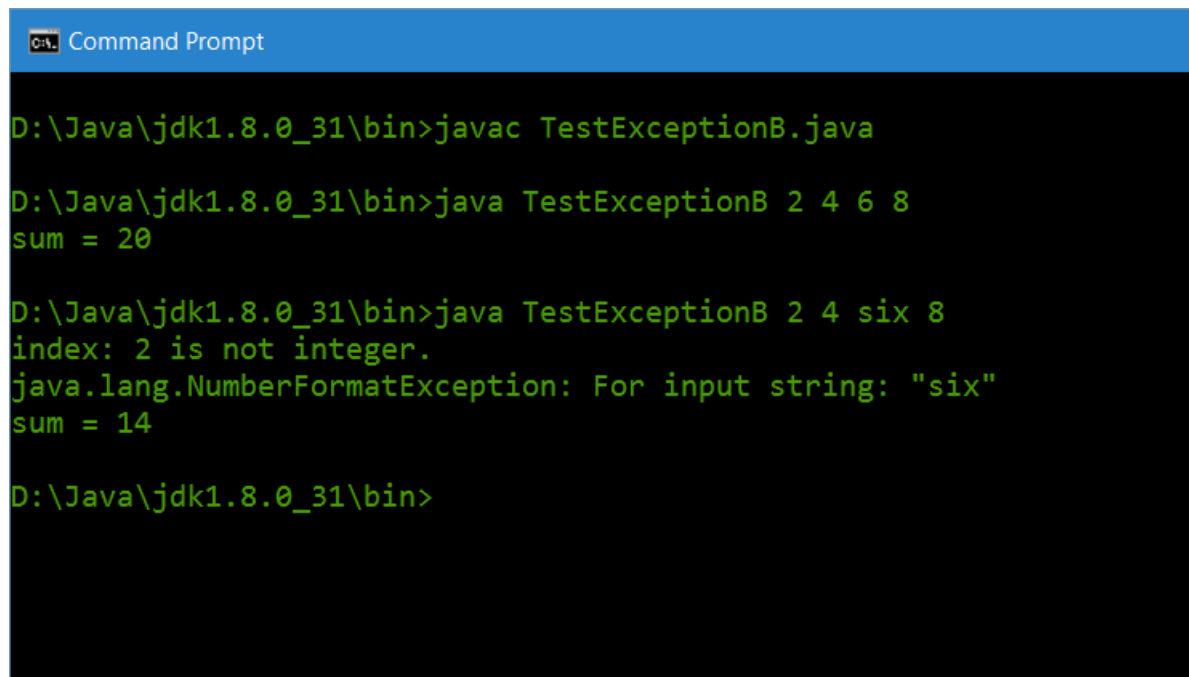
Output : 20

Run : `java TestExceptionB 2 4 six 8`

Output : index: 2 is not integer

`java.lang.NumberFormatException: For input string: "six"`

`sum = 14`



```
Command Prompt

D:\Java\jdk1.8.0_31\bin>javac TestExceptionB.java

D:\Java\jdk1.8.0_31\bin>java TestExceptionB 2 4 6 8
sum = 20

D:\Java\jdk1.8.0_31\bin>java TestExceptionB 2 4 six 8
index: 2 is not integer.
java.lang.NumberFormatException: For input string: "six"
sum = 14

D:\Java\jdk1.8.0_31\bin>
```

3.27 Using Multiple catch Clauses

In Java , there can be multiple catch blocks after a try block, and each catch block handles a different exception type.

```
1 public class MultiCatch
2 {
3     public static void main(String[] args)
4     {
5         int number;
6
7         try
8         {
9             number = (9 / 0);
10            System.out.println(number);
11        }
12        catch(ArithmeticException ex)
13        {
14            System.out.println("Arithmetic Exception");
15        }
16        catch(Exception ex)
17        {
18            System.out.println("Exception");
19        }
20    }
21 }
```

If we place ArithmeticException catch block after Exception catch block, it will be compilation error because ArithmeticException is the child class of Exception class, that means ArithmeticException is already been caught in the Exception class. So we do not need ArithmeticException catch block after we have declared Exception catch block.

3.28 Call Stack Mechanism

If a statement throws an exception, and that exception is not handled in the immediately enclosing method, then that exception is thrown to the calling method. If the exception is not handled in the calling method, it is thrown to the caller of that method. This process continues. If the exception is still not handled by the time it gets back to the main() method and main() does not handle it, the exception terminates the program abnormally.

3.29 try catch finally statements

The finally clause defines a block of code that always executes. If try block executes properly then catch block will not execute, and if try block will not execute properly then catch block will execute, but the finally block must execute.

```
1 public class TestTryCatchFinally
2 {
3     public static void main(String[] args)
4     {
5         int number;
6
7         try
8         {
9             number = (9 / 0);
10            System.out.println(number);
11        }
12        catch(ArithmeticException ex)
13        {
14            System.out.println("Arithmetic Exception");
15        }
16        catch(Exception ex)
17        {
18            System.out.println("Exception");
19        }
20        finally
21        {
22            System.out.println("finally block must execute");
23        }
24    }
25 }
```

3.30 Exception Declaring Rules

In Java, we can declare exceptions by following :

1. try catch finally
2. void methodB() throws IOException{ }
3. void methodC() throws FileNotFoundException, SecurityException{ }

```
1 import java.io.FileNotFoundException;
2 import java.io.IOException;
3
4 public class DeclareException
5 {
6     public void methodA()
7     {
8         try
9         {
10             throw new SecurityException();
11         }
12         catch(SecurityException ex)
13         {
14             System.out.println("Security Exception");
15         }
16         catch(Exception ex)
17         {
18             System.out.println("Exception");
19         }
20         finally
21         {
22             System.out.println("finally block must execute");
23         }
24     }
25
26     public void methodB() throws IOException
27     {
28         //
29     }
30
31     public void methodC() throws FileNotFoundException, SecurityException
32     {
33         //
34     }
35
36     public static void main(String[] args)
37     {
38         DeclareException declareException = new DeclareException();
39
40         declareException.methodA();
41         //declareException.methodB();
42         //declareException.methodC();
43     }
44 }
```

3.31 Method Overriding & Exception

The overriding method can declare only exceptions that are either the same class or a subclass of the exception.

For example, if the superclass method throws an `IOException`, then the overriding method of superclass method can throw an `IOException`, and also can throw `FileNotFoundException`, `EOFException`, which are the subclasses of `IOException`, but can not throw `Exception`, which is the superclass of `IOException`.

```
1  import java.io.EOFException;
2  import java.io.IOException;
3
4  class TestA
5  {
6      public void methodA() throws IOException
7      {
8          //
9      }
10 }
11
12 class TestB extends TestA
13 {
14     public void methodA() throws EOFException
15     {
16         //
17     }
18     //valid because EOFException is the sub class of IOException
19 }
20
21 public class TestC extends TestA
22 {
23     public void methodA() throws Exception
24     {
25         //
26     }
27     //invalid because Exception is the superclass of IOException
28 }
```

3.32 Creating and Throwing User Defined Exception

By extending Exception class we can create our own Exception class.

```
1 public class MyException extends Exception
2 {
3     public MyException(String message)
4     {
5         super(message);
6     }
7
8     public static void main(String[] args)
9     {
10        try
11        {
12            throw new MyException("UserDefinedException");
13        }
14        catch(MyException ex)
15        {
16            System.out.println(ex);
17        }
18    }
19 }
```

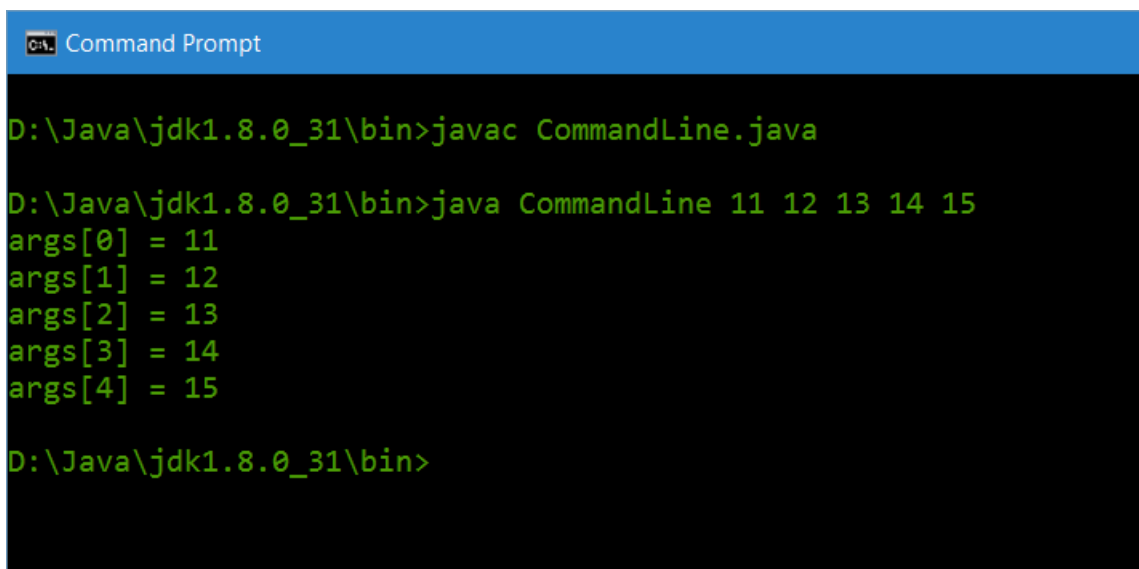
Chapter 4 :

Text Based and GUI Based Applications

4.1 Command Line Arguments

In Java programming, we can provide zero or more command line arguments of String type from a console / terminal window. The sequence of arguments follows the name of the program class and is stored in an array of String objects passed to the public static void main(String[] args) method.

```
1 public class CommandLine
2 {
3     public static void main(String[] args)
4     {
5         for( int i = 0; i < args.length; i++ )
6         {
7             System.out.println("args[" + i + "] = " + args[i]);
8         }
9     }
10 }
```



The screenshot shows a Windows Command Prompt window with a blue title bar labeled "Command Prompt". The command prompt is at the directory "D:\Java\jdk1.8.0_31\bin". The user has entered the command "javac CommandLine.java" to compile the program. Then, they entered "java CommandLine 11 12 13 14 15" to run the program with five command-line arguments. The output of the program is displayed as five lines: "args[0] = 11", "args[1] = 12", "args[2] = 13", "args[3] = 14", and "args[4] = 15". The prompt is now ready for another command.

```
Command Prompt
D:\Java\jdk1.8.0_31\bin>javac CommandLine.java
D:\Java\jdk1.8.0_31\bin>java CommandLine 11 12 13 14 15
args[0] = 11
args[1] = 12
args[2] = 13
args[3] = 14
args[4] = 15
D:\Java\jdk1.8.0_31\bin>
```

4.2 Console Input / Output

Java support console I/O with 3 public variables in the `java.lang.System` class :

1. `System.in` (`System.in` is a `InputStream` object)
2. `System.out` (`System.out` is a `PrintStream` object)
3. `System.err` (`System.err` is a `PrintStream` object)

Following code provides an example of keyboard input, which import `java.io.*` package. In line 9, `InputStreamReader` reads characters and converts the raw bytes into Unicode characters. In line 11, `BufferedReader` provides the `readLine()` method (in line 17) which enables the program to read from standard input (keyboard) one line at a time. The `readLine()` method can throw `IOException`, so this code is in try catch block.

```
1  import java.io.*;
2
3  public class ConsoleInput
4  {
5      public static void main(String[] args)
6      {
7          String string;
8
9          InputStreamReader isr = new InputStreamReader(System.in);
10
11         BufferedReader in = new BufferedReader(isr);
12
13         try
14         {
15             System.out.print("Write something: ");
16
17             string = in.readLine();
18
19             System.out.println("Read: " + string);
20
21             in.close();
22             isr.close();
23         }
24         catch(Exception ex)
25         {
26             System.out.println(ex);
27         }
28     }
29 }
```


4.3 Scanner

The scanner class provides formatted input functionality. It is a part of the java.util package.

```
1  import java.util.Scanner;
2
3  public class ScannerTest
4  {
5      public static void main(String[] args)
6      {
7          String string;
8          int intNumber;
9          float floatNumber;
10         double doubleNumber;
11
12         Scanner scanner = new Scanner(System.in);
13
14         System.out.print("Enter a string: ");
15         //string = scanner.next();
16         string = scanner.nextLine();
17         System.out.println("You entered: " + string);
18
19         System.out.print("Enter an integer: ");
20         intNumber = scanner.nextInt();
21         System.out.println("You entered: " + intNumber);
22
23         System.out.print("Enter a float number: ");
24         floatNumber = scanner.nextFloat();
25         System.out.println("You entered: " + floatNumber);
26
27         System.out.print("Enter a double number: ");
28         doubleNumber = scanner.nextDouble();
29         System.out.println("You entered: " + doubleNumber);
30
31         scanner.close();
32     }
33 }
```

4.4 Difference Between Scanner and BufferedReader Class in Java

<http://www.geeksforgeeks.org/difference-between-scanner-and-bufferreader-class-in-java/>

4.5 File in Java

To access a physical file, we have to create a File object, which contains the name and address of the file. The FileReader class is used to read characters from a file and the FileWriter class is used to write characters to a file. The PrintWriter class uses print() and println() methods.

4.6 Write String to a File

```
1  import java.io.*;
2
3  public class WriteFile
4  {
5      public static void main(String[] args)
6      {
7          String string;
8
9          File file = new File("F:", "MyText.txt");
10
11         try
12         {
13             InputStreamReader isr = new InputStreamReader(System.in);
14
15             BufferedReader in = new BufferedReader(isr);
16
17             PrintWriter out = new PrintWriter(new FileWriter(file));
18
19
20             System.out.print("Write something: ");
21
22             string = in.readLine();
23
24             out.print(string);
25
26             in.close();
27             isr.close();
28             out.close();
29         }
30         catch(IOException ex)
31         {
32             System.out.println(ex);
33         }
34     }
35 }
```

4.7 Read String from a File

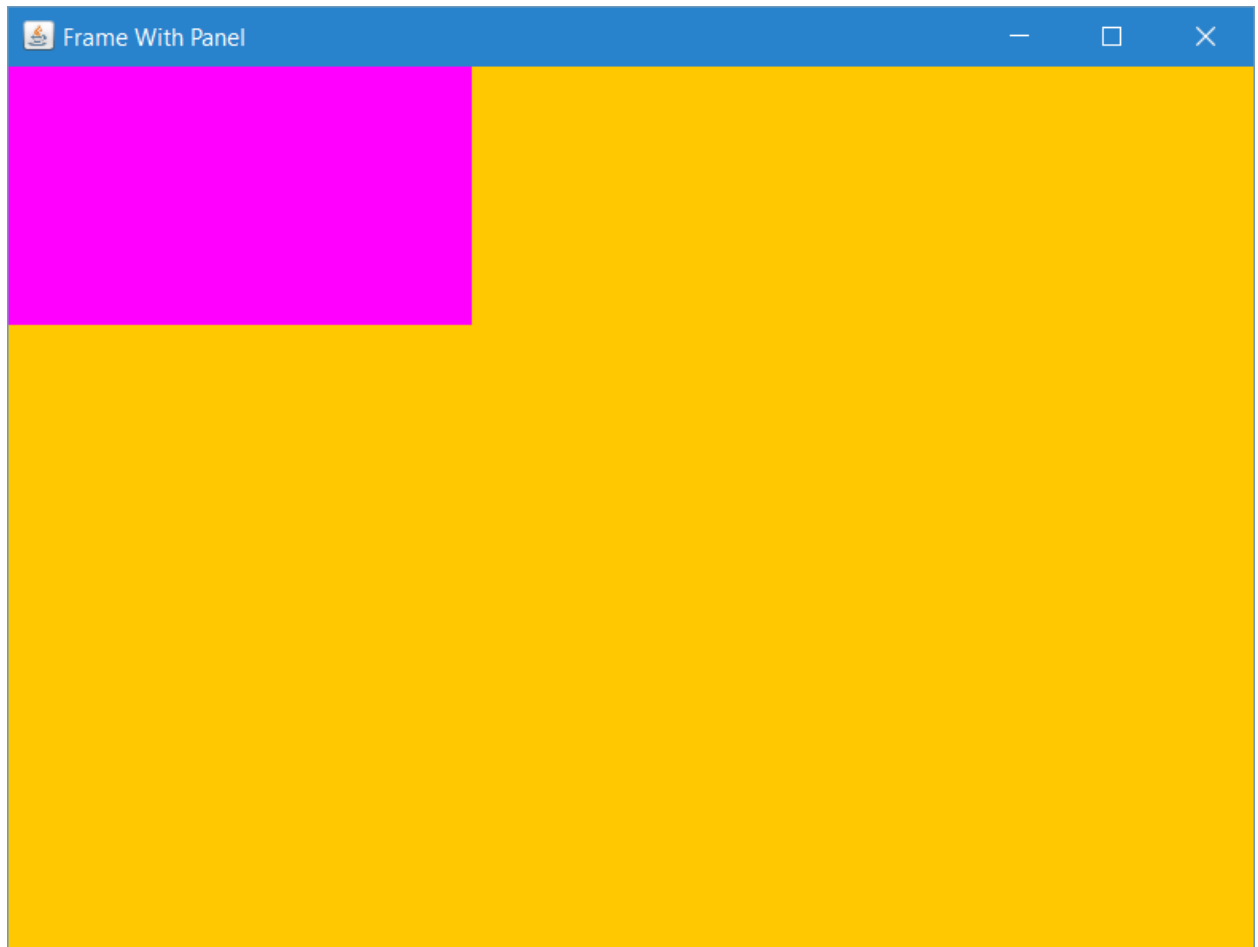
```
1  import java.io.*;
2
3  public class ReadFile
4  {
5      public static void main(String[] args)
6      {
7          String string;
8
9          try
10         {
11             File file = new File("F:", "MyText.txt");
12
13             BufferedReader in = new BufferedReader( new FileReader(file) );
14
15             string = in.readLine();
16
17             while( string != null )
18             {
19                 System.out.println("Read: " + string);
20
21                 string = in.readLine();
22             }
23
24             in.close();
25         }
26         catch(FileNotFoundException ex)
27         {
28             System.out.println("File Not Found");
29         }
30         catch(IOException ex)
31         {
32             System.out.println("Input Output Problem");
33         }
34     }
35 }
```

4.8 Abstract Window Toolkit (AWT)

AWT is the basic GUI (Graphics User Interface) used for Java applications and applets. Every GUI component that appears on the screen is a subclass of the abstract class Component or MenuComponent. The class Container is an abstract subclass of Component class, which permits other components to be nested inside it. There are two types of containers : Window and Panel. A Window is a free-standing native window on the display that is independent of other containers. There are two important types of Window containers : Frame and Dialog. A Frame is a window with a title and corners that you can resize. A Dialog is a simple window and cannot have a menu bar, you cannot resize it, but you can move it. A Panel must be contained within another Container, or inside a web browser's window.

```
1  import java.awt.*;
2  import java.awt.event.*;
3
4  public class FrameWithPanel implements WindowListener
5  {
6      private Frame frame;
7      private Panel panel;
8
9      public FrameWithPanel()
10     {
11         frame = new Frame("Frame With Panel");
12         panel = new Panel();
13     }
14
15     public void show()
16     {
17         frame.addWindowListener(this);
18         frame.setSize(800, 600);
19         frame.setBackground(Color.ORANGE);
20         frame.setLayout(null);
21
22         panel.setSize(300, 200);
23         panel.setBackground(Color.MAGENTA);
24
25         frame.add(panel);
26         frame.setResizable(true);
27         frame.setVisible(true);
28     }
29
```

```
30     @Override
31     public void windowOpened(WindowEvent e)
32     {
33     }
34
35     @Override
36     public void windowClosing(WindowEvent e)
37     {
38         System.exit(0);
39     }
40
41     @Override
42     public void windowClosed(WindowEvent e)
43     {
44     }
45
46     @Override
47     public void windowIconified(WindowEvent e)
48     {
49     }
50
51     @Override
52     public void windowDeiconified(WindowEvent e)
53     {
54     }
55
56     @Override
57     public void windowActivated(WindowEvent e)
58     {
59     }
60
61     @Override
62     public void windowDeactivated(WindowEvent e)
63     {
64     }
65
66     public static void main(String[] args)
67     {
68         FrameWithPanel frameWithPanel = new FrameWithPanel();
69         frameWithPanel.show();
70     }
71 }
72 }
```

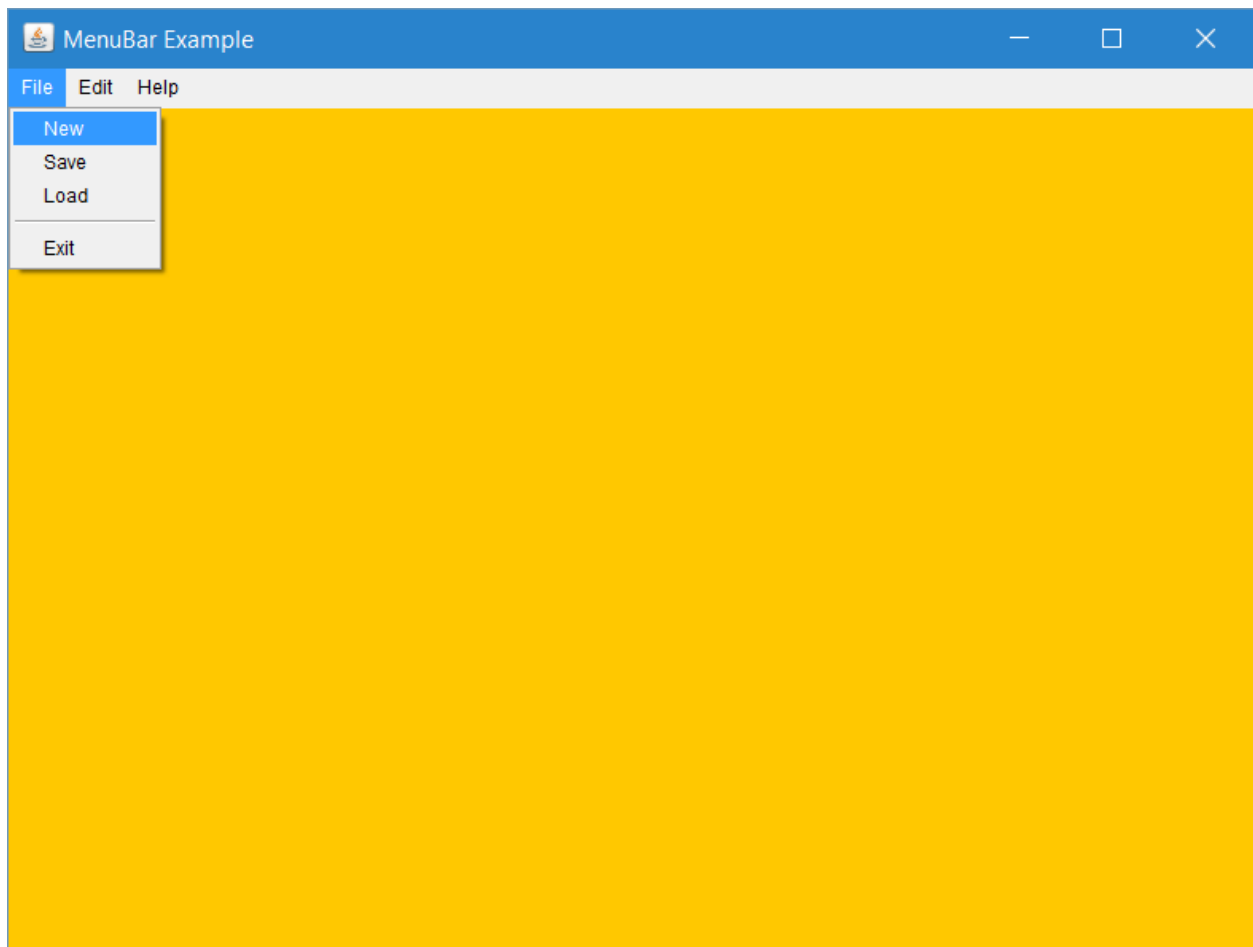


```
1 import java.awt.*;
2 import java.awt.event.*;
3
4 public class TestMenuBar implements WindowListener, ActionListener
5 {
6     private Frame frame;
7
8     private MenuBar menuBar;
9
10    private Menu menu1;
11    private Menu menu2;
12    private Menu menu3;
13
14    private MenuItem menuItem1;
15    private MenuItem menuItem2;
16    private MenuItem menuItem3;
17    private MenuItem menuItem4;
18
19    public TestMenuBar()
20    {
21        frame = new Frame("MenuBar Example");
22
23        menuBar = new MenuBar();
24
25        menu1 = new Menu("File");
26        menu2 = new Menu("Edit");
27        menu3 = new Menu("Help");
28
29        menuItem1 = new MenuItem("New");
30        menuItem2 = new MenuItem("Save");
31        menuItem3 = new MenuItem("Load");
32        menuItem4 = new MenuItem("Exit");
33    }
34
35    public void show()
36    {
37        menuItem1.addActionListener(this);
38        //menuItem1.setActionCommand("New menuItem clicked");
39
40        menuItem4.addActionListener(this);
41        //menuItem4.setActionCommand("Exit menuItem clicked");
42
43        menu1.add(menuItem1);
44        menu1.add(menuItem2);
45        menu1.add(menuItem3);
46
47        menu1.addSeparator();
48    }
```

```
49         menu1.add(menuItem4);
50
51         menuBar.add(menu1);
52         menuBar.add(menu2);
53         menuBar.setHelpMenu(menu3);
54
55         frame.setMenuBar(menuBar);
56
57         frame.addWindowListener(this);
58         frame.setSize(800, 600);
59         frame.setBackground(Color.ORANGE);
60         frame.setLayout(null);
61         frame.setResizable(true);
62         frame.setVisible(true);
63     }
64
65     @Override
66     public void windowOpened(WindowEvent e)
67     {
68     }
69
70     @Override
71     public void windowClosing(WindowEvent e)
72     {
73         System.exit(0);
74     }
75
76     @Override
77     public void windowClosed(WindowEvent e)
78     {
79     }
80
81     @Override
82     public void windowIconified(WindowEvent e)
83     {
84     }
85
86     @Override
87     public void windowDeiconified(WindowEvent e)
88     {
89     }
90
91     @Override
92     public void windowActivated(WindowEvent e)
93     {
94     }
95
```



```
96         @Override
97         public void windowDeactivated(WindowEvent e)
98         {
99         }
100
101         @Override
102         public void actionPerformed(ActionEvent e)
103         {
104             if( e.getSource() == menuItem1 )
105             {
106                 TestMenuBar t = new TestMenuBar();
107                 t.show();
108
109                 frame.setVisible(false);
110             }
111             else if( e.getSource() == menuItem4 )
112             {
113                 System.exit(0);
114             }
115
116             /*if( e.getActionCommand() == "New menuItem clicked" )
117             {
118                 TestMenuBar t = new TestMenuBar();
119                 t.show();
120
121                 frame.setVisible(false);
122             }
123             else if( e.getActionCommand() == "Exit menuItem clicked" )
124             {
125                 System.exit(0);
126             }*/
127         }
128
129         public static void main(String[] args)
130         {
131             TestMenuBar testMenuBar = new TestMenuBar();
132
133             testMenuBar.show();
134         }
135     }
```



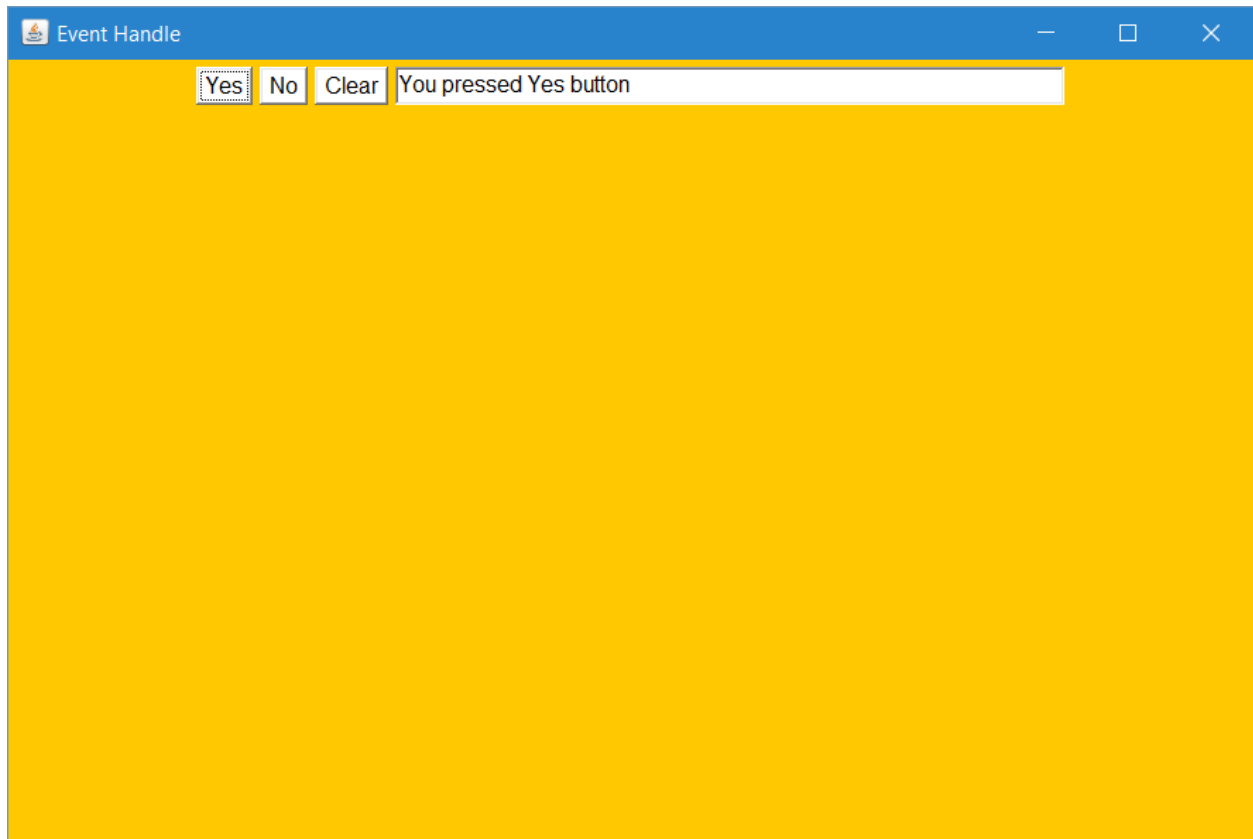
4.9 J2SE Event Model

An event is issued, when the user performs an action at the user interface level like : clicks a mouse or presses a key.

```
1  import java.awt.*;
2  import java.awt.event.*;
3
4  public class EventHandle implements WindowListener, ActionListener
5  {
6      private Frame frame;
7
8      private Button button1;
9      private Button button2;
10     private Button button3;
11
12     private TextField textField;
13
14     public EventHandle()
15     {
16         frame = new Frame("Event Handle");
17
18         button1 = new Button();
19         button1.setLabel("Yes");
20
21         button2 = new Button("No");
22
23         button3 = new Button("Clear");
24
25         textField = new TextField(50);
26     }
27
28     public void show()
29     {
30         button1.addActionListener(this);
31         button1.setBackground(Color.WHITE);
32         button1.setForeground(Color.BLACK);
33         button1.setFont( new Font("Consolas", Font.PLAIN, 16) );
34
35         button2.addActionListener(this);
36         button2.setBackground(Color.WHITE);
37         button2.setForeground(Color.BLACK);
38         button2.setFont( new Font("Consolas", Font.PLAIN, 16) );
39     }
```

```
40     button3.addActionListener(this);
41     button3.setBackground(Color.WHITE);
42     button3.setForeground(Color.BLACK);
43     button3.setFont( new Font("Consolas", Font.PLAIN, 16) );
44
45     textField.setBackground(Color.WHITE);
46     textField.setForeground(Color.BLACK);
47     textField.setFont( new Font("Consolas", Font.PLAIN, 16) );
48     textField.setEditable(false);
49     textField.setFocusable(true);
50
51     frame.add(button1);
52     frame.add(button2);
53     frame.add(button3);
54     frame.add(textField);
55
56     frame.addWindowListener(this);
57     frame.setSize(900, 600);
58     frame.setBackground(Color.ORANGE);
59     frame.setLayout( new FlowLayout() );
60     frame.setResizable(true);
61     frame.setVisible(true);
62 }
63
64 @Override
65 public void windowOpened(WindowEvent e)
66 {
67 }
68
69 @Override
70 public void windowClosing(WindowEvent e)
71 {
72     System.exit(0);
73 }
74
75 @Override
76 public void windowClosed(WindowEvent e)
77 {
78 }
79
80 @Override
81 public void windowIconified(WindowEvent e)
82 {
83 }
84
85 @Override
86 public void windowDeiconified(WindowEvent e)
87 {
88 }
89
```

```
90         @Override
91         public void windowActivated(WindowEvent e)
92         {
93         }
94
95         @Override
96         public void windowDeactivated(WindowEvent e)
97         {
98         }
99
100        @Override
101        public void actionPerformed(ActionEvent e)
102        {
103            String string = null;
104
105            if( e.getSource() == button1 )
106            {
107                string = "You pressed Yes button";
108            }
109            else if( e.getSource() == button2 )
110            {
111                string = "You pressed No button";
112            }
113            else if( e.getSource() == button3 )
114            {
115                string = "You pressed Clear button";
116            }
117
118            textField.setText(string);
119        }
120
121        public static void main(String[] args)
122        {
123            EventHandle eventHandle = new EventHandle();
124
125            eventHandle.show();
126        }
127    }
```



4.10 Java Color Class Static Constants and RGB Values

Color Constant	Color	RGB Value
public final static Color black	Black	0, 0, 0
public final static Color white	White	255, 255, 255
public final static Color orange	Orange	255, 200, 0
public final static Color pink	Pink	255, 175, 175
public final static Color cyan	Cyan	0, 255, 255
public final static Color magenta	Magenta	255, 0, 255
public final static Color yellow	Yellow	255, 255, 0
public final static Color gray	Gray	128, 128, 128
public final static Color lightGray	Light Gray	192, 192, 192
public final static Color darkGray	Dark Gray	64, 64, 64
public final static Color red	Red	255, 0, 0
public final static Color green	Green	0, 255, 0
public final static Color blue	Blue	0, 0, 255

4.11 Colors in Java

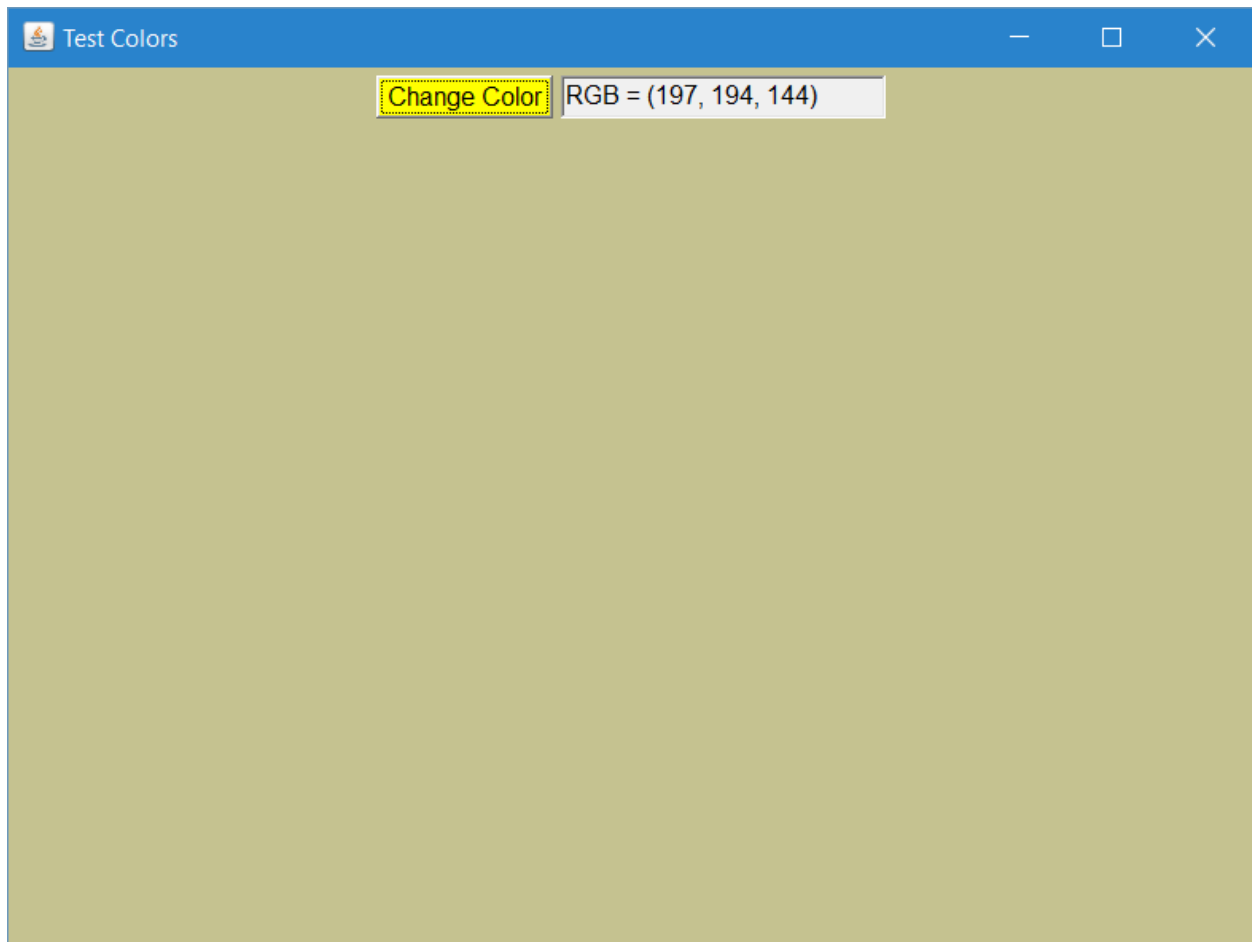
In Java, we can control the colors used for the foreground using `setForeground()` method and the background using `setBackground()` method of AWT components. The `setForeground()` and `setBackground()` methods take an argument that is an instance of `java.awt.Color` class. We can use the constant colors referred to as `Color.red`, `Color.blue`, `Color.green`, `Color.yellow`, and so on. We can also construct a specific `Color` object by specifying the color by a combination of three byte-sized integers (0 - 255), one for each primary color : red, green and blue (RGB).

```
1 import java.awt.*;
2 import java.awt.event.*;
3 import java.util.Random;
4
5 public class TestColors implements WindowListener, ActionListener
6 {
7     private Frame frame;
8     private Button button;
9     private TextField textField;
10
11     private Random random;
12
13     public TestColors()
14     {
15         frame = new Frame("Test Colors");
16         textField = new TextField(20);
17         button = new Button("Change Color");
18
19         random = new Random();
20     }
21
22     private void show()
23     {
24         button.addActionListener(this);
25         button.setBackground(Color.YELLOW);
26         button.setForeground(Color.BLACK);
27         button.setFont( new Font("Consolas", Font.PLAIN, 16) );
28         button.setEnabled(true);
29
30         textField.setFocusable(true);
31         textField.setEnabled(true);
32         textField.setEditable(false);
33         textField.setFont( new Font("Consolas", Font.PLAIN, 16) );
34
35         frame.add(button);
36         frame.add(textField);
37         frame.addWindowListener(this);
38         frame.setSize(800, 600);
39         frame.setBackground(Color.GREEN);
40         frame.setLayout( new FlowLayout() );
41         frame.setResizable(true);
42         frame.setVisible(true);
43     }
44 }
```

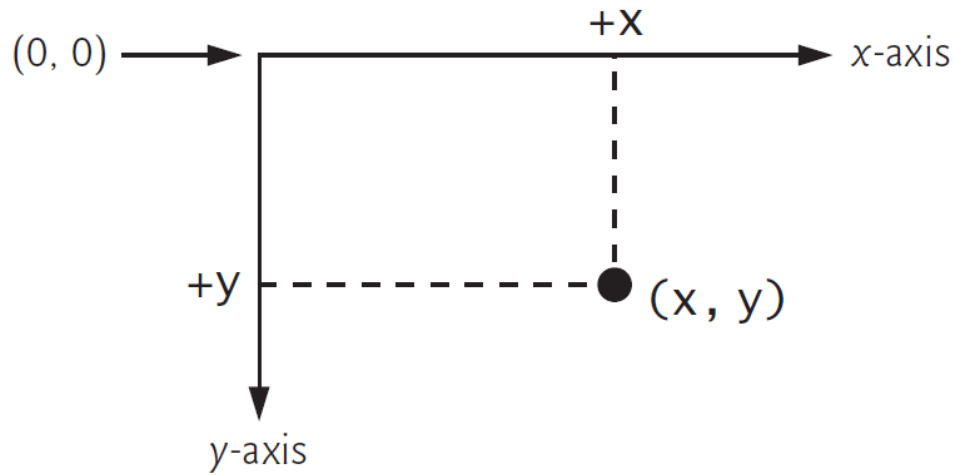


```
45     @Override
46     public void actionPerformed(ActionEvent e)
47     {
48         int r;
49         int g;
50         int b;
51         String string;
52
53         if( e.getSource() == button )
54         {
55             r = random.nextInt(256);
56             g = random.nextInt(256);
57             b = random.nextInt(256);
58
59             Color color = new Color(r, g, b);
60
61             string = "RGB = (" + r + ", " + g + ", " + b + ")";
62
63             frame.setBackground(color);
64             textField.setText(string);
65         }
66     }
67
68     @Override
69     public void windowActivated(WindowEvent e)
70     {
71     }
72
73     @Override
74     public void windowClosed(WindowEvent e)
75     {
76     }
77
78     @Override
79     public void windowClosing(WindowEvent e)
80     {
81         System.exit(0);
82     }
83
84     @Override
85     public void windowDeactivated(WindowEvent e)
86     {
87     }
88
89     @Override
90     public void windowDeiconified(WindowEvent e)
91     {
92     }
93
```

```
94         @Override
95         public void windowIconified(WindowEvent e)
96         {
97         }
98
99         @Override
100        public void windowOpened(WindowEvent e)
101        {
102        }
103
104        public static void main(String[] args)
105        {
106            TestColors testColors = new TestColors();
107
108            testColors.show();
109        }
110    }
```



4.12 Java Coordinate System



4.13 Layout Managers

In Java programming, the layout manager manages the layout of components in a container, which we can change by calling `setLayout()` method. The layout manager is responsible for deciding the layout policy and size of each of its container's child components. The following layout managers are included with the Java programming language :

1. `FlowLayout`
2. `BorderLayout`
3. `GridLayout`
4. `BoxLayout`
5. `CardLayout`
6. `GridBagLayout`
7. `GroupLayout`
8. `SpringLayout`

A Visual Guide to Layout Managers :

<https://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html>

4.14 FlowLayout

FlowLayout manager uses line-by-line technique to place the components in a container. Each time a line is filled, a new line is started. It does not consider the size of components.

```
1  import java.awt.*;
2  import java.awt.event.*;
3
4  public class FlowLayoutExample implements WindowListener
5  {
6      private Frame frame;
7      private Button[] button;
8
9      public FlowLayoutExample()
10     {
11         frame = new Frame("FlowLayout Example");
12
13         button = new Button[5];
14
15         for(int i = 0; i < button.length; i++)
16         {
17             button[i] = new Button("Button " + (i + 1));
18             button[i].setFont( new Font("Consolas", Font.PLAIN, 16) );
19         }
20     }
21
22     private void show()
23     {
24         frame.setLayout( new FlowLayout() );
25         //frame.setLayout( new FlowLayout(FlowLayout.LEFT) );
26         //frame.setLayout( new FlowLayout(FlowLayout.RIGHT) );
27         //frame.setLayout( new FlowLayout(FlowLayout.CENTER) );
28         //frame.setLayout( new FlowLayout(FlowLayout.CENTER, 50, 150) );
29         //frame.setLayout( new FlowLayout(FlowLayout.LEADING) );
30         //frame.setLayout( new FlowLayout(FlowLayout.TRAILING) );
31
32         for(int i = 0; i < button.length; i++)
33         {
34             frame.add(button[i]);
35         }
36
37         frame.addWindowListener(this);
38         frame.setSize(800, 600);
39         frame.setResizable(true);
40         frame.setBackground(Color.ORANGE);
41         frame.setVisible(true);
42     }
43 }
```

```
44     @Override
45     public void windowActivated(WindowEvent e)
46     {
47     }
48
49     @Override
50     public void windowClosed(WindowEvent e)
51     {
52     }
53
54     @Override
55     public void windowClosing(WindowEvent e)
56     {
57         System.exit(0);
58     }
59
60     @Override
61     public void windowDeactivated(WindowEvent e)
62     {
63     }
64
65     @Override
66     public void windowDeiconified(WindowEvent e)
67     {
68     }
69
70     @Override
71     public void windowIconified(WindowEvent e)
72     {
73     }
74
75     @Override
76     public void windowOpened(WindowEvent e)
77     {
78     }
79
80     public static void main(String[] args)
81     {
82         FlowLayoutExample flowLayoutExample = new FlowLayoutExample();
83
84         flowLayoutExample.show();
85     }
86 }
```



FlowLayout Example



Button 1

Button 2

Button 3

Button 4

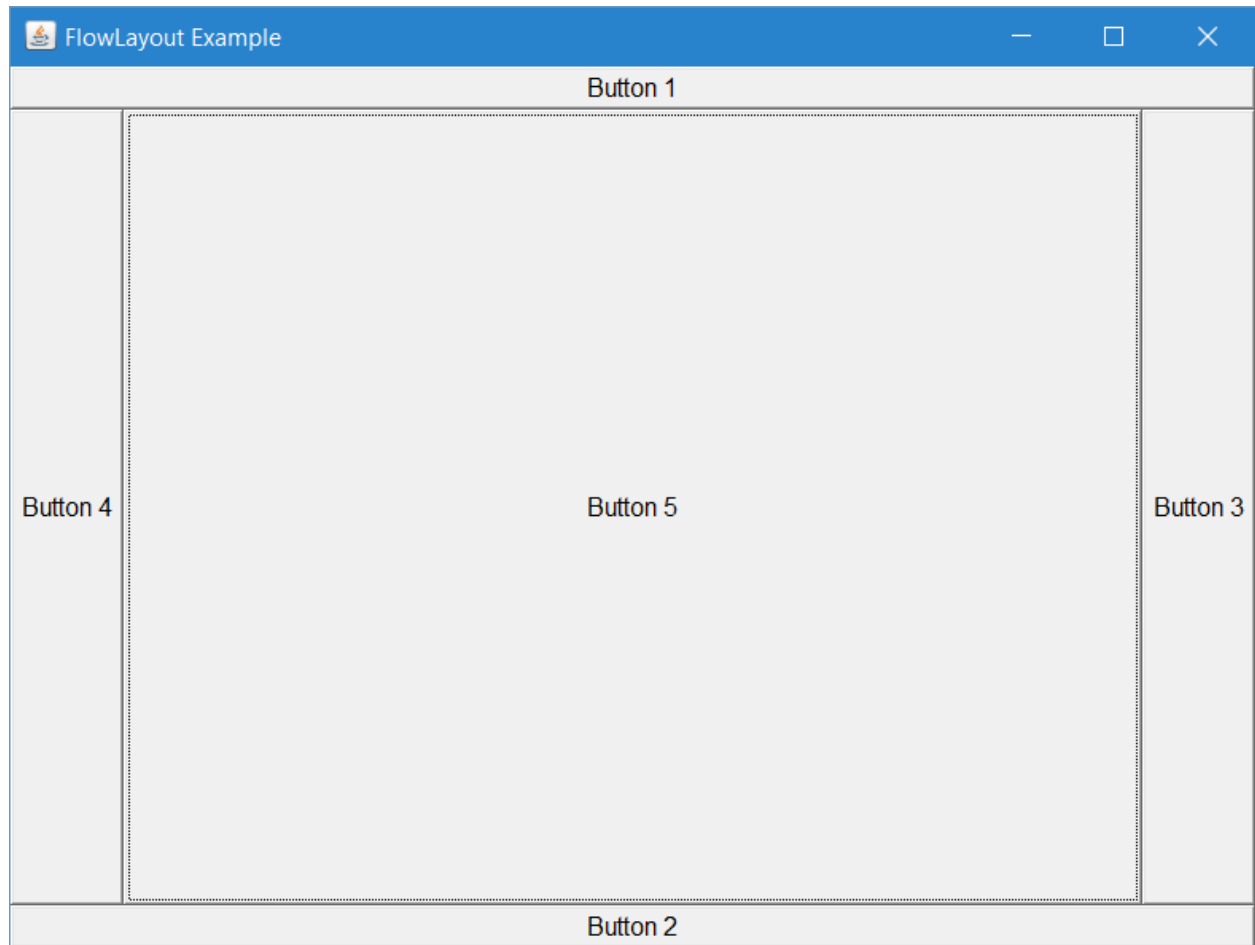
Button 5

4.15 BorderLayout

The BorderLayout manager contains five distinct areas: NORTH, SOUTH, EAST, WEST, and CENTER, indicated by BorderLayout.NORTH, and so on.

```
1  import java.awt.*;
2  import java.awt.event.*;
3
4  public class BorderLayoutExample implements WindowListener
5  {
6      private Frame frame;
7      private Button[] button;
8
9      public BorderLayoutExample()
10     {
11         frame = new Frame("FlowLayout Example");
12
13         button = new Button[5];
14
15         for(int i = 0; i < button.length; i++)
16         {
17             button[i] = new Button("Button " + (i + 1));
18             button[i].setFont( new Font("Consolas", Font.PLAIN, 16) );
19         }
20     }
21
22     private void show()
23     {
24         frame.setLayout( new BorderLayout() );
25         //frame.setLayout( new BorderLayout(10, 25) );
26
27         frame.add(button[0], BorderLayout.NORTH);
28         frame.add(button[1], BorderLayout.SOUTH);
29         frame.add(button[2], BorderLayout.EAST);
30         frame.add(button[3], BorderLayout.WEST);
31         frame.add(button[4], BorderLayout.CENTER);
32
33         frame.addWindowListener(this);
34         frame.setSize(800, 600);
35         frame.setResizable(true);
36         frame.setBackground(Color.ORANGE);
37         frame.setVisible(true);
38     }
39
```

```
40     @Override
41     public void windowOpened(WindowEvent e)
42     {
43     }
44
45     @Override
46     public void windowClosing(WindowEvent e)
47     {
48         System.exit(0);
49     }
50
51     @Override
52     public void windowClosed(WindowEvent e)
53     {
54     }
55
56     @Override
57     public void windowIconified(WindowEvent e)
58     {
59     }
60
61     @Override
62     public void windowDeiconified(WindowEvent e)
63     {
64     }
65
66     @Override
67     public void windowActivated(WindowEvent e)
68     {
69     }
70
71     @Override
72     public void windowDeactivated(WindowEvent e)
73     {
74     }
75
76     public static void main(String[] args)
77     {
78         BorderLayoutExample borderLayoutExample = new BorderLayoutExample();
79
80         borderLayoutExample.show();
81     }
82 }
```

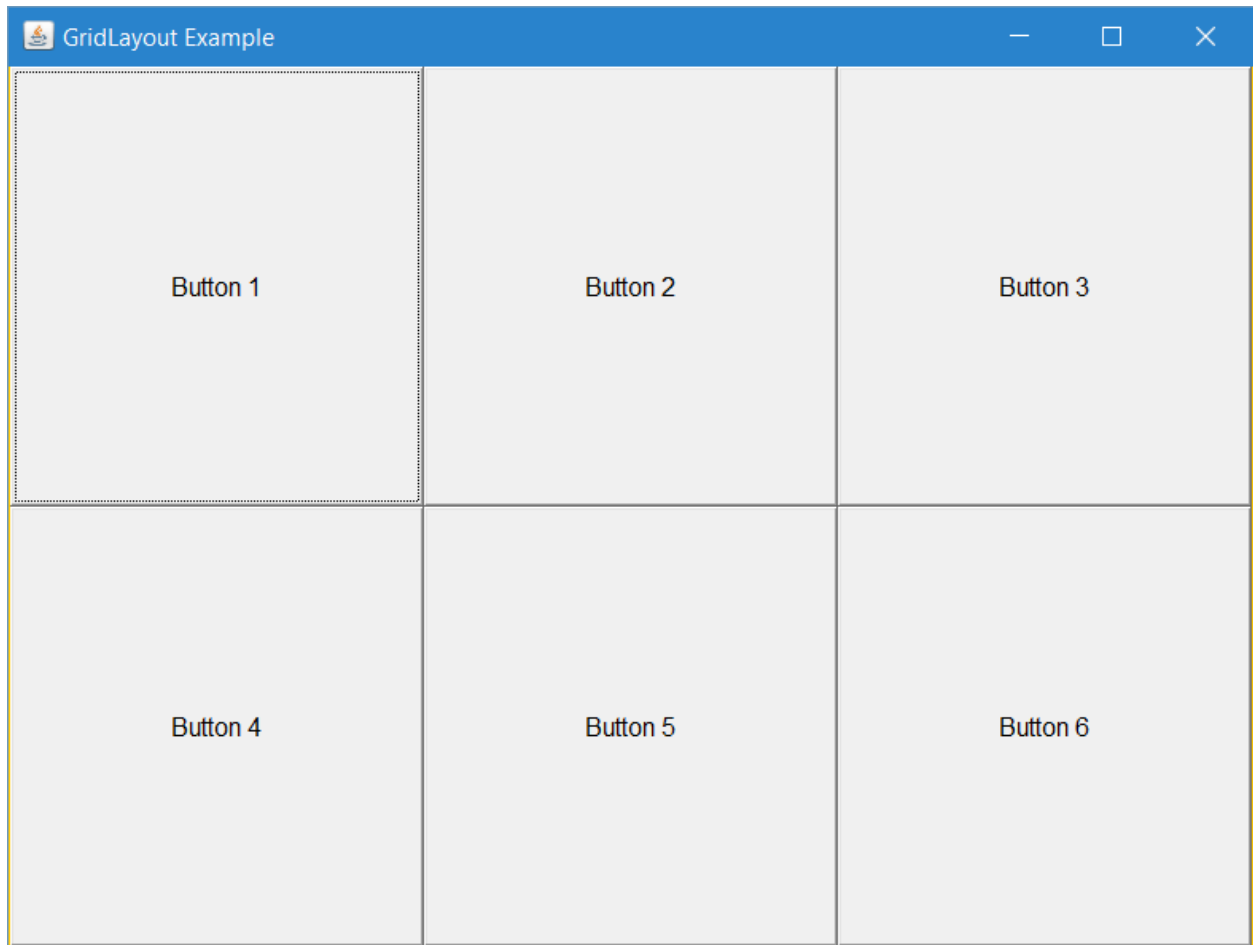
4.16 GridLayout

The GridLayout manager places components with a number of rows and columns. The following constructor creates a GridLayout with the specified size :

```
new GridLayout(int rows, int columns);
```

```
1  import java.awt.*;
2  import java.awt.event.*;
3
4  public class GridLayoutExample implements WindowListener
5  {
6      private Frame frame;
7      private Button[] button;
8
9      private GridLayoutExample()
10     {
11         frame = new Frame("GridLayout Example");
12
13         button = new Button[6];
14
15         for(int i = 0; i < button.length; i++)
16         {
17             button[i] = new Button("Button " + (i + 1));
18             button[i].setFont( new Font("Consolas", Font.PLAIN, 16) );
19         }
20     }
21
22     private void show()
23     {
24         frame.setLayout( new GridLayout(2, 3) );
25         //frame.setLayout( new GridLayout(3, 2, 20, 20) );
26
27         for(int i = 0; i < button.length; i++)
28         {
29             frame.add(button[i]);
30         }
31
32         frame.addWindowListener(this);
33         frame.setSize(800, 600);
34         frame.setResizable(false);
35         frame.setBackground(Color.ORANGE);
36         frame.setVisible(true);
37     }
38 }
```

```
39     @Override
40     public void windowActivated(WindowEvent e)
41     {
42     }
43
44     @Override
45     public void windowClosed(WindowEvent e)
46     {
47     }
48
49     @Override
50     public void windowClosing(WindowEvent e)
51     {
52         System.exit(0);
53     }
54
55     @Override
56     public void windowDeactivated(WindowEvent e)
57     {
58     }
59
60     @Override
61     public void windowDeiconified(WindowEvent e)
62     {
63     }
64
65     @Override
66     public void windowIconified(WindowEvent e)
67     {
68     }
69
70     @Override
71     public void windowOpened(WindowEvent e)
72     {
73     }
74
75     public static void main(String[] args)
76     {
77         GridLayoutExample gridLayoutExample = new GridLayoutExample();
78
79         gridLayoutExample.show();
80     }
81 }
```



4.17 BorderLayout

BoxLayout either stacks its components on top of each other or places them in a row.

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.BoxLayout;
4
5  public class BorderLayoutExample implements WindowListener
6  {
7      private Frame frame;
8      private Button[] button;
9
10     public BorderLayoutExample()
11     {
12         frame = new Frame("BoxLayout Example");
13
14         button = new Button[5];
15
16         for( int i = 0; i < button.length; i++ )
17         {
18             button[i] = new Button("Button " + (i + 1));
19             button[i].setFont( new Font("Consolas", Font.PLAIN, 16) );
20         }
21     }
22
23     private void show()
24     {
25         frame.addWindowListener(this);
26         //frame.setLayout( new BoxLayout(frame, BoxLayout.X_AXIS) );
27         frame.setLayout( new BoxLayout(frame, BoxLayout.Y_AXIS) );
28
29         for( int i = 0; i < button.length; i++ )
30         {
31             frame.add(button[i]);
32         }
33
34         frame.setSize(800, 600);
35         frame.setResizable(true);
36         frame.setBackground(Color.ORANGE);
37         frame.setVisible(true);
38     }
39
40     @Override
41     public void windowActivated(WindowEvent e)
42     {
43     }
44 }
```

```
45     @Override
46     public void windowClosed(WindowEvent e)
47     {
48     }
49
50     @Override
51     public void windowClosing(WindowEvent e)
52     {
53         System.exit(0);
54     }
55
56     @Override
57     public void windowDeactivated(WindowEvent e)
58     {
59     }
60
61     @Override
62     public void windowDeiconified(WindowEvent e)
63     {
64     }
65
66     @Override
67     public void windowIconified(WindowEvent e)
68     {
69     }
70
71     @Override
72     public void windowOpened(WindowEvent e)
73     {
74     }
75
76     public static void main(String[] args)
77     {
78         BoxLayoutExample boxLayoutExample = new BoxLayoutExample();
79
80         boxLayoutExample.show();
81     }
82 }
```



BoxLayout Example



Button 1

Button 2

Button 3

Button 4

Button 5

4.18 Example of Different Layouts Together

```
1 import java.awt.*;
2 import java.awt.event.*;
3 import javax.swing.BoxLayout;
4
5 public class TestDifferentLayouts implements WindowListener
6 {
7     private Frame frame;
8     private Panel[] panel;
9     private Button[][] button;
10
11     private TestDifferentLayouts()
12     {
13         frame = new Frame("Test Different Layouts");
14         panel = new Panel[4];
15
16         for( int i = 0; i < panel.length; i++ )
17         {
18             panel[i] = new Panel();
19         }
20
21         button = new Button[4][4];
22
23         for( int counter = 1, i = 0; i < button.length; i++ )
24         {
25             for( int j = 0; j < button[i].length; j++ )
26             {
27                 button[i][j] = new Button("Button " + counter);
28                 button[i][j].setFont( new Font("Consolas", Font.PLAIN, 16) );
29                 counter++;
30             }
31         }
32     }
33
34     private void show()
35     {
36         panel[0].setBackground(Color.ORANGE);
37         panel[0].setLayout( new FlowLayout() );
38         panel[0].add(button[0][0]);
39         panel[0].add(button[0][1]);
40         panel[0].add(button[0][2]);
41         panel[0].add(button[0][3]);
42     }
```



```

43     panel[1].setBackground(Color.GREEN);
44     panel[1].setLayout( new BorderLayout() );
45     panel[1].add(button[1][0], BorderLayout.NORTH);
46     panel[1].add(button[1][1], BorderLayout.SOUTH);
47     panel[1].add(button[1][2], BorderLayout.EAST);
48     panel[1].add(button[1][3], BorderLayout.WEST);
49
50     panel[2].setBackground(Color.BLUE);
51     panel[2].setLayout( new BoxLayout(panel[2], BoxLayout.X_AXIS) );
52     //panel[2].setLayout( new BoxLayout(panel[2], BoxLayout.Y_AXIS) );
53     panel[2].add(button[2][0]);
54     panel[2].add(button[2][1]);
55     panel[2].add(button[2][2]);
56     panel[2].add(button[2][3]);
57
58     panel[3].setBackground(Color.YELLOW);
59     panel[3].setLayout( new GridLayout(2, 2, 10, 20) );
60     panel[3].add(button[3][0]);
61     panel[3].add(button[3][1]);
62     panel[3].add(button[3][2]);
63     panel[3].add(button[3][3]);
64
65     frame.add(panel[0]);
66     frame.add(panel[1]);
67     frame.add(panel[2]);
68     frame.add(panel[3]);
69
70     frame.addWindowListener(this);
71     frame.setLayout( new GridLayout(2, 2, 10, 10) );
72     frame.setBackground(Color.MAGENTA);
73     frame.setSize(800, 800);
74     frame.setResizable(true);
75     frame.setVisible(true);
76 }
77
78 @Override
79 public void windowActivated(WindowEvent e)
80 {
81 }
82
83 @Override
84 public void windowClosed(WindowEvent e)
85 {
86 }
87
88 @Override
89 public void windowClosing(WindowEvent e)
90 {
91     System.exit(0);
92 }
93

```

```
94         @Override
95         public void windowDeactivated(WindowEvent e)
96         {
97         }
98
99         @Override
100        public void windowDeiconified(WindowEvent e)
101        {
102        }
103
104        @Override
105        public void windowIconified(WindowEvent e)
106        {
107        }
108
109        @Override
110        public void windowOpened(WindowEvent e)
111        {
112        }
113
114        public static void main(String[] args)
115        {
116            TestDifferentLayouts testLayouts = new TestDifferentLayouts();
117            testLayouts.show();
118        }
119    }
120 }
```



4.19 Example of MouseListener

```
1 import java.awt.*;
2 import java.awt.event.*;
3
4 public class MouseListenerExample implements WindowListener, MouseListener
5 {
6     private Frame frame;
7     private TextField textField;
8
9     public MouseListenerExample()
10    {
11        frame = new Frame("MouseListener Example");
12        textField = new TextField(30);
13    }
14
15    private void show()
16    {
17        Label label = new Label("This is a label");
18        label.setFont( new Font("Consolas", Font.PLAIN, 20) );
19
20        label.setAlignment(Label.CENTER);
21        //label.setAlignment(Label.LEFT);
22        //label.setAlignment(Label.RIGHT);
23
24        textField.setEditable(false);
25        textField.setFocusable(false);
26        textField.setFont( new Font("Consolas", Font.PLAIN, 16) );
27
28        frame.setLayout( new BorderLayout() );
29        frame.add(label, BorderLayout.NORTH);
30        frame.add(textField, BorderLayout.SOUTH);
31
32        frame.addWindowListener(this);
33        frame.addMouseListener(this);
34        frame.setSize(800, 600);
35        frame.setResizable(true);
36        frame.setBackground(Color.ORANGE);
37        frame.setVisible(true);
38    }
39
40    @Override
41    public void mouseClicked(MouseEvent e)
42    {
43        String string = "Mouse clicked at: " + e.getX() + ", " + e.getY();
44
45        textField.setText(string);
46    }
```

```
47
48     @Override
49     public void mouseEntered(MouseEvent e)
50     {
51         String string = "Mouse entered the frame";
52
53         textField.setText(string);
54     }
55
56     @Override
57     public void mouseExited(MouseEvent e)
58     {
59         String string = "Mouse left the frame";
60
61         textField.setText(string);
62     }
63
64     @Override
65     public void mousePressed(MouseEvent e)
66     {
67         String string = "Mouse pressed at: " + e.getX() + ", " + e.getY();
68
69         textField.setText(string);
70     }
71
72     @Override
73     public void mouseReleased(MouseEvent e)
74     {
75         String string = "Mouse released at: " + e.getX() + ", " + e.getY();
76
77         textField.setText(string);
78     }
79
80     @Override
81     public void windowActivated(WindowEvent arg0)
82     {
83     }
84
85     @Override
86     public void windowClosed(WindowEvent arg0)
87     {
88     }
89
90     @Override
91     public void windowClosing(WindowEvent arg0)
92     {
93         System.exit(0);
94     }
95
```

```
96         @Override
97         public void windowDeactivated(WindowEvent arg0)
98         {
99         }
100
101         @Override
102         public void windowDeiconified(WindowEvent arg0)
103         {
104         }
105
106         @Override
107         public void windowIconified(WindowEvent arg0)
108         {
109         }
110
111         @Override
112         public void windowOpened(WindowEvent arg0)
113         {
114         }
115
116         public static void main(String[] args)
117         {
118             MouseListenerExample object = new MouseListenerExample();
119
120             object.show();
121         }
122     }
```



MouseListener Example



This is a label



Mouse clicked at: 108, 124

4.20 Example of MouseMotionListener

```
1  import java.awt.*;
2  import java.awt.event.*;
3
4  public class MouseMotionListenerExample implements WindowListener,
    MouseMotionListener
5  {
6      private Frame frame;
7      private TextField textField;
8
9      public MouseMotionListenerExample()
10     {
11         frame = new Frame("MouseMotionListener Example");
12         textField = new TextField(30);
13     }
14
15     private void show()
16     {
17         Label label = new Label("Click and Drag the Mouse");
18         label.setFont( new Font("Consolas", Font.PLAIN, 20) );
19
20         label.setAlignment(Label.CENTER);
21         //label.setAlignment(Label.LEFT);
22         //label.setAlignment(Label.RIGHT);
23
24         textField.setEditable(false);
25         textField.setFocusable(false);
26         textField.setFont( new Font("Consolas", Font.PLAIN, 16) );
27
28         frame.setLayout( new BorderLayout() );
29         frame.add(label, BorderLayout.NORTH);
30         frame.add(textField, BorderLayout.SOUTH);
31
32         frame.addWindowListener(this);
33         frame.addMouseMotionListener(this);
34         frame.setSize(800, 600);
35         frame.setResizable(true);
36         frame.setBackground(Color.ORANGE);
37         frame.setVisible(true);
38     }
39
```



```
40     @Override
41     public void mouseDragged(MouseEvent e)
42     {
43         String string = "Mouse dragged at: " + e.getX() + ", " + e.getY();
44
45         textField.setText(string);
46     }
47
48     @Override
49     public void mouseMoved(MouseEvent e)
50     {
51         String string = "Mouse moved at: " + e.getX() + ", " + e.getY();
52
53         textField.setText(string);
54     }
55
56     @Override
57     public void windowActivated(WindowEvent arg0)
58     {
59     }
60
61     @Override
62     public void windowClosed(WindowEvent arg0)
63     {
64     }
65
66     @Override
67     public void windowClosing(WindowEvent arg0)
68     {
69         System.exit(0);
70     }
71
72     @Override
73     public void windowDeactivated(WindowEvent arg0)
74     {
75     }
76
77     @Override
78     public void windowDeiconified(WindowEvent arg0)
79     {
80     }
81
82     @Override
83     public void windowIconified(WindowEvent arg0)
84     {
85     }
86
```

```
87     @Override
88     public void windowOpened(WindowEvent arg0)
89     {
90     }
91
92     public static void main(String[] args)
93     {
94         MouseMotionListenerExample a = new MouseMotionListenerExample();
95
96         a.show();
97     }
98 }
```



4.21 Fonts in Java

In Java, Font class manages the font of Strings. The constructor of Font class takes 3 arguments : the font name, font style, and font size. The font name is any font currently supported by the system where the program is running such as standard Java fonts Monospaced, SansSerif, and Serif. The font style is Font.PLAIN, Font.ITALIC, or Font.BOLD. Font styles can be used in combination such as : Font.ITALIC + Font.BOLD

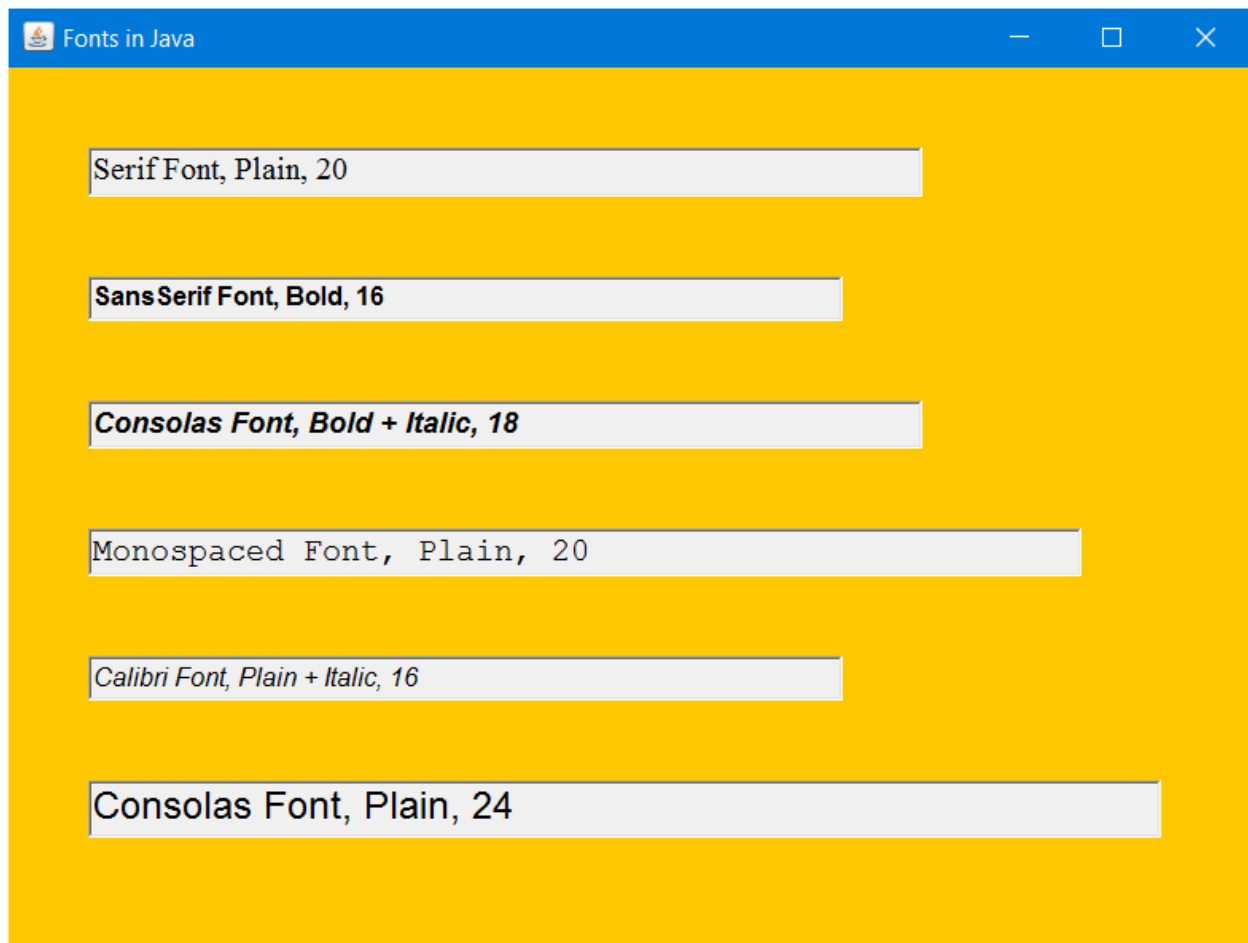
```
1  import java.awt.*;
2  import java.awt.event.*;
3
4  public class TestFonts implements WindowListener
5  {
6      private Frame frame;
7      private TextField[] textField;
8      private Font[] font;
9
10     public TestFonts()
11     {
12         frame = new Frame("Fonts in Java");
13
14         textField = new TextField[6];
15         font = new Font[6];
16
17         for( int i = 0; i < textField.length; i++ )
18         {
19             textField[i] = new TextField(50);
20             textField[i].setEditable(false);
21             textField[i].setFocusable(false);
22         }
23
24         font[0] = new Font("Serif", Font.PLAIN, 20);
25         textField[0].setText("Serif Font, Plain, 20");
26
27         font[1] = new Font("SansSerif", Font.BOLD, 16);
28         textField[1].setText("SansSerif Font, Bold, 16");
29
30         font[2] = new Font("Consolas", Font.BOLD + Font.ITALIC, 18);
31         textField[2].setText("Consolas Font, Bold + Italic, 18");
32
33         font[3] = new Font("Monospaced", Font.PLAIN, 20);
34         textField[3].setText("Monospaced Font, Plain, 20");
35
36         font[4] = new Font("Calibri", Font.PLAIN + Font.ITALIC, 16);
37         textField[4].setText("Calibri Font, Plain + Italic, 16");
38     }
```

```
39         font[5] = new Font("Consolas", Font.PLAIN, 24);
40         textField[5].setText("Consolas Font, Plain, 24");
41     }
42
43     private void show()
44     {
45         for( int i = 0; i < textField.length; i++ )
46         {
47             textField[i].setFont( font[i] );
48         }
49
50         frame.addWindowListener(this);
51         frame.setLayout( new FlowLayout(FlowLayout.LEFT, 50, 50) );
52
53         for( int i = 0; i < textField.length; i++ )
54         {
55             frame.add(textField[i]);
56         }
57
58         frame.setSize(800, 600);
59         frame.setResizable(true);
60         frame.setBackground(Color.ORANGE);
61         frame.setVisible(true);
62     }
63
64     @Override
65     public void windowActivated(WindowEvent e)
66     {
67     }
68
69     @Override
70     public void windowClosed(WindowEvent e)
71     {
72     }
73
74     @Override
75     public void windowClosing(WindowEvent e)
76     {
77         System.exit(0);
78     }
79
80     @Override
81     public void windowDeactivated(WindowEvent e)
82     {
83     }
84
85     @Override
86     public void windowDeiconified(WindowEvent e)
87     {
88     }
89
```

```

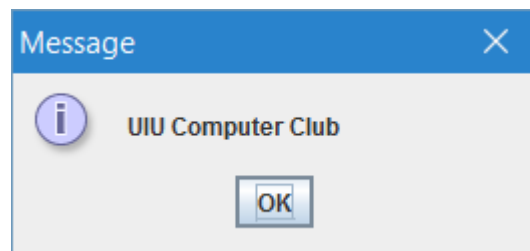
90         @Override
91         public void windowIconified(WindowEvent e)
92         {
93         }
94
95         @Override
96         public void windowOpened(WindowEvent e)
97         {
98         }
99
100        public static void main(String[] args)
101        {
102            TestFonts testFonts = new TestFonts();
103
104            testFonts.show();
105        }
106    }

```



4.22 Dialog Example

```
1 import javax.swing.JOptionPane;
2
3 public class DialogTest
4 {
5     public static void main(String[] args)
6     {
7         JOptionPane.showMessageDialog(null, "UIU Computer Club");
8     }
9 }
```



Chapter 5 : Thread, Collection API, Socket

5.1 What is Thread

Modern computers perform multiple tasks at the same time. Thread is the process of multi tasking using CPU, which performs computations.

A thread or execution context is composed of 3 main parts :

1. a virtual CPU
2. the code that the CPU executes
3. the data on which the code works

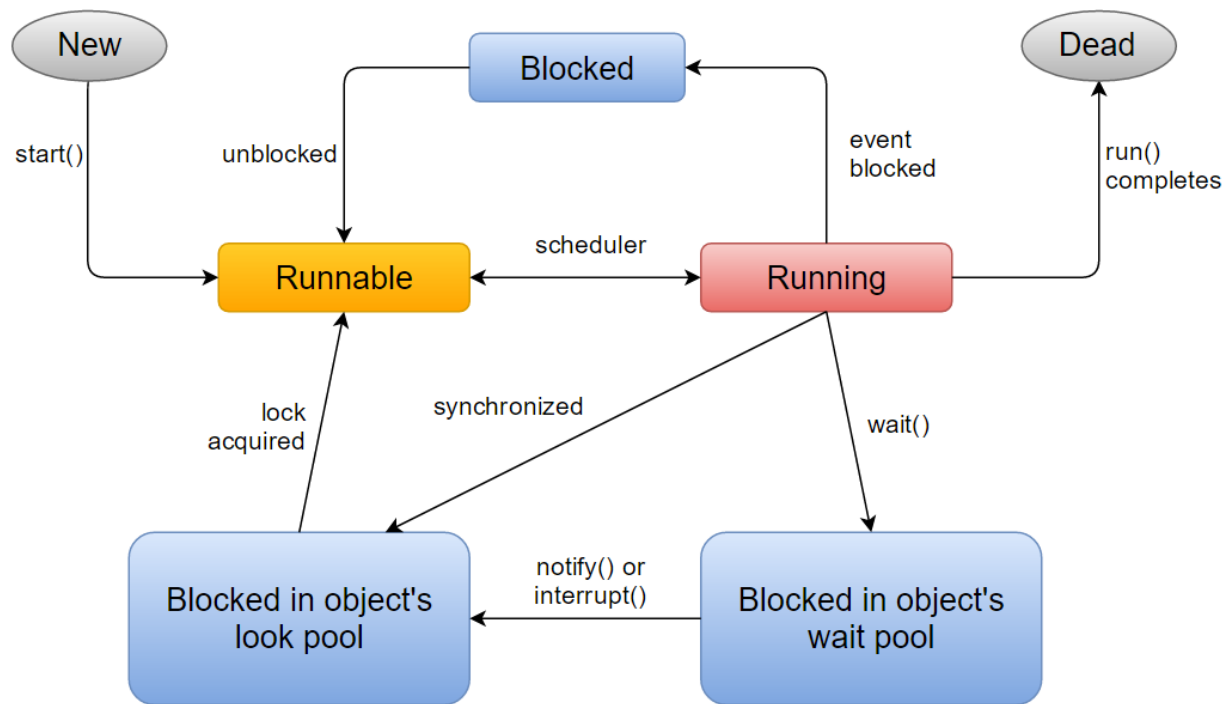


Figure: Thread State Diagram with wait() and notify()

5.2 Thread in Java

The class `java.lang.Thread` enables us to create and control threads.

A process is a program in execution.

One or more threads can be run under a process. A thread is composed of CPU, code and data. Code can be shared with multiple threads, two threads can share the same code.

We can create thread by :

1. implementing `Runnable` interface
2. extending `Thread` class

`Thread` class implements the `Runnable` interface itself. The `Runnable` interface provides the `public void run()` method. We override the `run()` method, which contains the code for CPU execution.

A newly created thread does not start running automatically, we must call its `start()` method.


```
1  import java.util.concurrent.TimeUnit;
2
3  public class ThreadTest1 implements Runnable
4  {
5      @Override
6      public void run()
7      {
8          for( int i = 1; i <= 100; i++ )
9          {
10             System.out.println("i = " + i);
11
12             try
13             {
14                 TimeUnit.MILLISECONDS.sleep(20);
15                 //Thread.sleep(20);
16             }
17             catch(InterruptedException ex)
18             {
19                 System.out.println(ex);
20             }
21         }
22     }
23
24     public static void main(String[] args)
25     {
26         ThreadTest1 object = new ThreadTest1();
27
28         Thread thread1 = new Thread(object);
29         Thread thread2 = new Thread(object);
30         Thread thread3 = new Thread(object);
31
32         thread1.start();
33         thread2.start();
34         thread3.start();
35     }
36 }
```

```
1  import java.util.concurrent.TimeUnit;
2
3  public class MyThread1 extends Thread
4  {
5      public void run()
6      {
7          for( int i = 1; i <= 100; i++ )
8          {
9              System.out.println("i = " + i);
10
11              try
12              {
13                  TimeUnit.MILLISECONDS.sleep(20);
14                  //Thread.sleep(20);
15              }
16              catch(InterruptedException ex)
17              {
18                  System.out.println(ex);
19              }
20          }
21      }
22
23      public static void main(String[] args)
24      {
25          Thread thread1 = new MyThread1();
26          Thread thread2 = new MyThread1();
27          Thread thread3 = new MyThread1();
28
29          thread1.start();
30          thread2.start();
31          thread3.start();
32      }
33 }
```

```
1  import java.util.concurrent.TimeUnit;
2
3  public class ThreadTest2 implements Runnable
4  {
5      private String name;
6
7      public ThreadTest2(String name)
8      {
9          this.name = name;
10     }
11
12     @Override
13     public void run()
14     {
15         for( int i = 1; i <= 100; i++ )
16         {
17             System.out.println(name + ": i = " + i);
18
19             try
20             {
21                 TimeUnit.MILLISECONDS.sleep(20);
22                 //Thread.sleep(20);
23             }
24             catch(InterruptedException ex)
25             {
26                 System.out.println(ex);
27             }
28         }
29     }
30
31     public static void main(String[] args)
32     {
33         ThreadTest2 object1 = new ThreadTest2("SuperMan");
34         ThreadTest2 object2 = new ThreadTest2("SpiderMan");
35         ThreadTest2 object3 = new ThreadTest2("IronMan");
36
37         Thread thread1 = new Thread(object1);
38         Thread thread2 = new Thread(object2);
39         Thread thread3 = new Thread(object3);
40
41         thread1.start();
42         thread2.start();
43         thread3.start();
44     }
45 }
```

```
1  import java.util.concurrent.TimeUnit;
2
3  public class MyThread2 extends Thread
4  {
5      private String name;
6
7      public MyThread2(String name)
8      {
9          this.name = name;
10     }
11
12     public void run()
13     {
14         for( int i = 1; i <= 100; i++ )
15         {
16             System.out.println(name + ": i = " + i);
17
18             try
19             {
20                 TimeUnit.MILLISECONDS.sleep(20);
21                 //Thread.sleep(20);
22             }
23             catch(InterruptedException ex)
24             {
25                 System.out.println(ex);
26             }
27         }
28     }
29
30     public static void main(String[] args)
31     {
32         Thread thread1 = new MyThread2("SuperMan");
33         Thread thread2 = new MyThread2("SpiderMan");
34         Thread thread3 = new MyThread2("IronMan");
35
36         thread1.start();
37         thread2.start();
38         thread3.start();
39     }
40 }
```

```
1 import java.util.Random;
2 import java.util.concurrent.TimeUnit;
3
4 public class ThreadTest3 implements Runnable
5 {
6     private String name;
7     private Random random;
8
9     public ThreadTest3(String name)
10    {
11        this.name = name;
12
13        random = new Random();
14    }
15
16    @Override
17    public void run()
18    {
19        for( int i = 1; i <= 100; i++ )
20        {
21            System.out.println(name + ": i = " + i);
22
23            try
24            {
25                TimeUnit.MILLISECONDS.sleep( random.nextInt(20) );
26            }
27            catch(InterruptedException ex)
28            {
29                System.out.println(ex);
30            }
31        }
32    }
33
34    public static void main(String[] args)
35    {
36        ThreadTest3 object1 = new ThreadTest3("SuperMan");
37        ThreadTest3 object2 = new ThreadTest3("SpiderMan");
38        ThreadTest3 object3 = new ThreadTest3("IronMan");
39
40        Thread thread1 = new Thread(object1);
41        Thread thread2 = new Thread(object2);
42        Thread thread3 = new Thread(object3);
43
44        thread1.start();
45        thread2.start();
46        thread3.start();
47    }
48 }
```

```
1 import java.util.Random;
2 import java.util.concurrent.TimeUnit;
3
4 class MyStack
5 {
6     private int index;
7     private int size;
8     private char[] data;
9
10    public MyStack(int size)
11    {
12        this.size = size;
13        index = -1;
14
15        data = new char[size];
16    }
17
18    public synchronized void push(char ch)
19    {
20        this.notify();
21
22        if( index < (size - 1) )
23        {
24            index++;
25            data[index] = ch;
26        }
27    }
28
29    public synchronized char pop()
30    {
31        if( index == -1 )
32        {
33            try
34            {
35                this.wait();
36            }
37            catch(InterruptedException ex)
38            {
39                System.out.println(ex);
40            }
41        }
42
43        return ( data[index--] );
44    }
45 }
46
```

```
47 class Producer implements Runnable
48 {
49     private MyStack myStack;
50     private Random random;
51
52     public Producer(MyStack myStack)
53     {
54         this.myStack = myStack;
55
56         random = new Random();
57     }
58
59     @Override
60     public void run()
61     {
62         char ch;
63
64         for( int i = 1; i <= 10; i++ )
65         {
66             ch = (char)(65 + random.nextInt(26) );
67
68             myStack.push(ch);
69
70             System.out.println(i + ". Producer: " + ch);
71
72             try
73             {
74                 TimeUnit.MILLISECONDS.sleep( random.nextInt(100) );
75             }
76             catch(InterruptedException ex)
77             {
78                 System.out.println(ex);
79             }
80         }
81     }
82 }
83
```

```

84     class Consumer implements Runnable
85     {
86         private MyStack myStack;
87         private Random random;
88
89         public Consumer(MyStack myStack)
90         {
91             this.myStack = myStack;
92
93             random = new Random();
94         }
95
96         @Override
97         public void run()
98         {
99             char ch;
100
101             for( int i = 1; i <= 10; i++ )
102             {
103                 ch = myStack.pop();
104
105                 System.out.println(i + ". Consumer: " + ch);
106
107                 try
108                 {
109                     TimeUnit.MILLISECONDS.sleep( random.nextInt(100) );
110                 }
111                 catch(InterruptedException ex)
112                 {
113                     System.out.println(ex);
114                 }
115             }
116         }
117     }
118
119     public class StackTest
120     {
121         public static void main(String[] args)
122         {
123             MyStack myStack = new MyStack(5);
124
125             Producer producer = new Producer(myStack);
126             Thread thread1 = new Thread(producer);
127
128             thread1.start();
129
130             Consumer consumer = new Consumer(myStack);
131             Thread thread2 = new Thread(consumer);
132
133             thread2.start();
134         }
135     }

```


5.3 The Collection API

A collection is a single object representing a group of objects. The objects in the collection are called elements. Implementation of collection determine whether there is specific ordering and whether duplicates are permitted.

1. Set - an unordered collection, where no duplicates are permitted
2. List - an ordered collection, where duplicates are permitted

```
1  import java.util.HashSet;
2  import java.util.Set;
3
4  public class SetExample
5  {
6      public static void main(String[] args)
7      {
8          Set hashSet = new HashSet();
9
10         hashSet.add("One");
11         hashSet.add("2nd");
12         hashSet.add("3rd");
13         hashSet.add(4);
14         hashSet.add(5.1);
15         hashSet.add( new Integer(6) );
16         hashSet.add( new Double(7.2) );
17         hashSet.add("One");//duplicate is not added
18         hashSet.add(null);
19
20         System.out.println(hashSet);
21     }
22 }
```

```
1 import java.util.ArrayList;
2 import java.util.List;
3
4 public class ListExample
5 {
6     public static void main(String[] args)
7     {
8         List list = new ArrayList();
9
10        list.add("One");
11        list.add("2nd");
12        list.add("3rd");
13        list.add(4);
14        list.add(5.1);
15        list.add( new Integer(6) );
16        list.add( new Double(7.2) );
17        list.add("One");//duplicate is added
18        list.add(null);
19
20        System.out.println(list);
21    }
22 }
```

```
1 import java.util.LinkedList;
2 import java.util.Random;
3
4 public class QuickSortTest
5 {
6     private int[] myArray;
7
8     private LinkedList linkedList;
9
10    public QuickSortTest(int size)
11    {
12        myArray = new int[size];
13    }
14
15    public void initializeWithWorstCaseNumbers()
16    {
17        for( int i = 0; i < myArray.length; i++ )
18        {
19            myArray[i] = (myArray.length - i);
20        }
21    }
22
23    public void initializeWithRandomNumbers()
24    {
25        Random random = new Random();
26
27        for( int i = 0; i < myArray.length; i++ )
28        {
29            myArray[i] = random.nextInt(100);
30        }
31    }
32
33    public void quickSortMain()
34    {
35        int temp;
36
37        int first = (int)linkedList.removeFirst();
38        int last = (int)linkedList.removeFirst();
39
40        if( first >= last )
41        {
42            return;
43        }
44
45        int i = (first + 1);
46        int j = last;
47    }
```

```

48         while( i < last && myArray[first] >= myArray[i] )
49         {
50             i++;
51         }
52         while( first < j && myArray[first] <= myArray[j] )
53         {
54             j--;
55         }
56
57         while( i < j )
58         {
59             temp = myArray[i];
60             myArray[i] = myArray[j];
61             myArray[j] = temp;
62
63             i++;
64             j--;
65
66             while( i < last && myArray[first] >= myArray[i] )
67             {
68                 i++;
69             }
70             while( first < j && myArray[first] <= myArray[j] )
71             {
72                 j--;
73             }
74         }
75
76         temp = myArray[first];
77         myArray[first] = myArray[j];
78         myArray[j] = temp;
79
80         linkedList.add(first);
81         linkedList.add(j - 1);
82
83         linkedList.add(j + 1);
84         linkedList.add(last);
85     }
86
87     public void quickSort()
88     {
89         linkedList = new LinkedList();
90
91         linkedList.add(0);
92         linkedList.add( this.myArray.length - 1 );
93
94         while( linkedList.isEmpty() == false )
95         {
96             quickSortMain();
97         }
98     }

```

```
99
100     public static void main(String[] args)
101     {
102         QuickSortTest t1 = new QuickSortTest(20);
103
104         t1.initializeWithWorstCaseNumbers();
105         //t1.initializeWithRandomNumbers();
106
107         System.out.print("Before sorting: ");
108         for( int i : t1.myArray )
109         {
110             System.out.print(i + " ");
111         }
112         System.out.println();
113
114         t1.quickSort();
115
116         System.out.print("After sorting: ");
117         for( int i : t1.myArray )
118         {
119             System.out.print(i + " ");
120         }
121         System.out.println();
122     }
123 }
```

5.4 Networking in Java

Socket is the Java programming model to establish the communication link between two processes. A socket can hold input and output stream. A process sends data to another process through the network by writing to the output stream associated with the socket. A process reads data written the another process by reading from the input stream associated with the socket.

```
1 import java.io.PrintWriter;
2 import java.net.ServerSocket;
3 import java.net.Socket;
4 import java.util.Random;
5
6 public class Server
7 {
8     private ServerSocket serverSocket;
9     private Socket socket;
10    private PrintWriter writer;
11
12    private Random random;
13    private int number;
14
15    public Server()
16    {
17        try
18        {
19            serverSocket = new ServerSocket(5000);
20        }
21        catch(Exception ex)
22        {
23            System.out.println(ex);
24        }
25
26        random = new Random();
27    }
28
29    public int getNumber()
30    {
31        return ( 65 + random.nextInt(26) );
32    }
33
```

```
34     public void go()
35     {
36         System.out.println("Server is running.");
37         System.out.println("Waiting for connection...");
38
39         try
40         {
41             while( true )
42             {
43                 socket = serverSocket.accept();
44
45                 writer = new PrintWriter( socket.getOutputStream() );
46
47                 number = getNumber();
48
49                 writer.println(number);
50
51                 writer.close();
52
53                 System.out.println("Sent: " + (char)number);
54             }
55         }
56         catch(Exception ex)
57         {
58             System.out.println(ex);
59         }
60     }
61
62     public static void main(String[] args)
63     {
64         Server server = new Server();
65         server.go();
66     }
67 }
```

```
1  import java.net.Socket;
2  import java.util.Scanner;
3
4  public class Client
5  {
6      private Socket socket;
7      private Scanner scanner;
8
9      private char ch;
10
11     public Client()
12     {
13         try
14         {
15             socket = new Socket("127.0.0.1", 5000);
16
17             scanner = new Scanner( socket.getInputStream() );
18         }
19         catch(Exception ex)
20         {
21             System.out.println(ex);
22         }
23     }
24
25     public void go()
26     {
27         try
28         {
29             ch = (char)scanner.nextInt();
30
31             scanner.close();
32
33             System.out.println("Received: " + ch);
34         }
35         catch(Exception ex)
36         {
37             System.out.println(ex);
38         }
39     }
40
41     public static void main(String[] args)
42     {
43         Client client = new Client();
44         client.go();
45     }
46 }
```


Chapter 6 : Extras

6.1 Enum

Enum in Java is a data type that contains fixed set of constants. It can be used for days of the week (SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY and SATURDAY), directions (NORTH, SOUTH, EAST and WEST) etc. The Java enum constants are static and final implicitly. It is available from JDK 1.5. Java Enums can be thought of as classes that have fixed set of constants.

```
1  enum Day
2  {
3      SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY
4  }
5
6  public class EnumTest
7  {
8      public Day myDay;
9
10     public EnumTest(Day myDay)
11     {
12         this.myDay = myDay;
13     }
14
15     public static void main(String[] args)
16     {
17         EnumTest enumTest = new EnumTest(Day.FRIDAY);
18
19         System.out.println("Today is " + enumTest.myDay);
20
21         enumTest.myDay = Day.MONDAY;
22
23         System.out.println("Today is " + enumTest.myDay);
24     }
25 }
```

6.2 Variable or Array

```
1 public class VariableOrArrayTest1
2 {
3     public void show(int a)
4     {
5         System.out.println(a);
6     }
7
8     public void show(int[] a)
9     {
10        for( int i : a )
11        {
12            System.out.print(i + " ");
13        }
14        System.out.println();
15    }
16
17    public static void main(String[] args)
18    {
19        int a = 5;
20
21        int[] b = {10, 20, 30, 40, 50};
22
23        VariableOrArrayTest1 t1 = new VariableOrArrayTest1();
24
25        t1.show(a);
26        t1.show(b);
27    }
28 }
```

```
1 public class VariableOrArrayTest2
2 {
3     public void show(int ... a)
4     {
5         for( int i : a )
6         {
7             System.out.print(i + " ");
8         }
9         System.out.println();
10    }
11
12    public static void main(String[] args)
13    {
14        int a = 5;
15
16        int[] b = {10, 20, 30, 40, 50};
17
18        VariableOrArrayTest2 t2 = new VariableOrArrayTest2();
19
20        t2.show(a);
21        t2.show(b);
22    }
23 }
```

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import java.io.*;
4  import java.util.Random;
5
6  public class OOPLabFinal1 implements WindowListener, ActionListener
7  {
8      private Frame frame;
9
10     private MenuBar menuBar;
11     private Menu menu1, menu2;
12     private MenuItem menuItem1, menuItem2;
13
14     private Panel panel1, panel2;
15     private Color color1, color2;
16
17     private TextField textField1, textField2;
18     private Button button1, button2;
19
20     private OOPLabFinal1()
21     {
22         frame = new Frame("Test Program 1");
23
24         menuBar = new MenuBar();
25
26         menu1 = new Menu("File");
27         menu2 = new Menu("Color");
28
29         menuItem1 = new MenuItem("Exit");
30         menuItem2 = new MenuItem("Change Background Colors");
31
32         panel1 = new Panel();
33         panel2 = new Panel();
34
35         textField1 = new TextField(50);
36         textField2 = new TextField(50);
37
38         button1 = new Button("Write to file");
39         button2 = new Button("Read from file");
40
41         color1 = Color.GREEN;
42         color2 = Color.ORANGE;
43     }
44
45     private void show()
46     {
47         menuItem1.addActionListener(this);
48         menuItem2.addActionListener(this);
49     }
```

```
50     button1.addActionListener(this);
51     button2.addActionListener(this);
52     button1.setFont( new Font("Consolas", Font.PLAIN, 16) );
53     button2.setFont( new Font("Consolas", Font.PLAIN, 16) );
54
55     menu1.add(menuItem1);
56     menu2.add(menuItem2);
57
58     menuBar.add(menu1);
59     menuBar.add(menu2);
60
61     textField1.setEditable(true);
62     textField2.setEditable(false);
63     textField1.setFont( new Font("Consolas", Font.PLAIN, 16) );
64     textField2.setFont( new Font("Consolas", Font.PLAIN, 16) );
65
66     panel1.setBackground(color1);
67     panel1.setLayout( new FlowLayout() );
68     panel1.add(textField1);
69     panel1.add(button1);
70
71     panel2.setBackground(color2);
72     panel2.setLayout( new FlowLayout() );
73     panel2.add(textField2);
74     panel2.add(button2);
75
76     frame.addWindowListener(this);
77     frame.setMenuBar(menuBar);
78     frame.setLayout( new GridLayout(2, 1) );
79     frame.add(panel1);
80     frame.add(panel2);
81     frame.setSize(800, 600);
82     frame.setResizable(true);
83     frame.setVisible(true);
84 }
85
86 @Override
87 public void actionPerformed(ActionEvent e)
88 {
89     if( e.getSource() == menuItem1 )
90     {
91         System.exit(0);
92     }
93     else if( e.getSource() == menuItem2 )
94     {
95         int r;
96         int g;
97         int b;
98     }
```

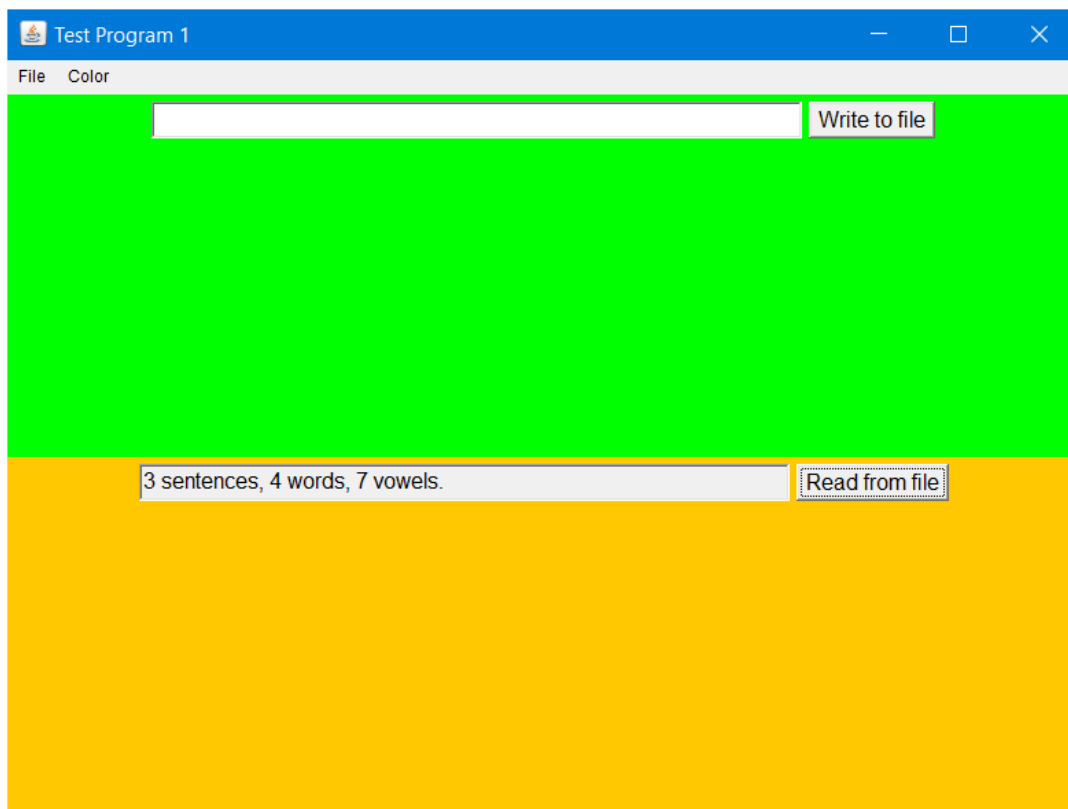
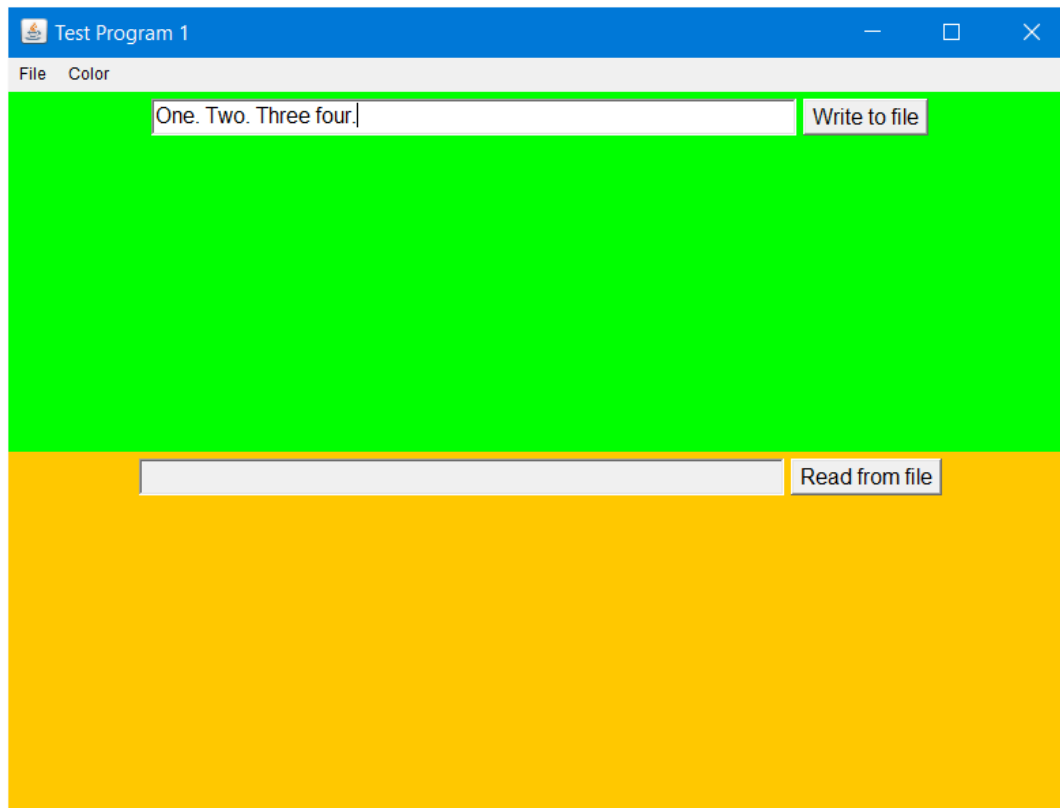
```
99         Random random = new Random();
100
101         r = random.nextInt(256);
102         g = random.nextInt(256);
103         b = random.nextInt(256);
104
105         color1 = new Color(r, g, b);
106
107         r = random.nextInt(256);
108         g = random.nextInt(256);
109         b = random.nextInt(256);
110
111         color2 = new Color(r, g, b);
112
113         panel1.setBackground(color1);
114         panel2.setBackground(color2);
115     }
116     else if( e.getSource() == button1 )
117     {
118         String string;
119
120         try
121         {
122             File file = new File("F:", "MyText.txt");
123
124             PrintWriter out = new PrintWriter( new FileWriter(file) );
125
126             string = textField1.getText();
127
128             out.print(string);
129
130             out.close();
131
132             textField1.setText("");
133         }
134         catch(FileNotFoundException ex)
135         {
136             System.out.println(ex);
137         }
138         catch(IOException ex)
139         {
140             System.out.println(ex);
141         }
142     }
143     else if( e.getSource() == button2 )
144     {
145         String string = null;
146     }
```

```

147         int word = 0;
148         int sentence = 0;
149         int vowel = 0;
150
151         try
152         {
153             File file = new File("F:", "MyText.txt");
154
155             BufferedReader in = new BufferedReader( new
156                 FileReader(file) );
157
158             string = in.readLine();
159
160             while( string != null )
161             {
162                 System.out.println("Read: " + string);
163
164                 sentence += string.length() -
165                     string.replace(".", "").length();
166
167                 word += string.length() - string.replace("
168                     ", "").length();
169                 word++;
170
171                 vowel += string.length() -
172                     string.toLowerCase().replaceAll("a|e|i|o|u", "").length();
173
174                 string = in.readLine();
175             }
176
177             in.close();
178
179             catch(FileNotFoundException ex)
180             {
181                 System.out.println(ex);
182             }
183
184             catch(IOException ex)
185             {
186                 System.out.println(ex);
187             }
188
189             finally
190             {
191                 textField2.setText(sentence + " sentences, " +
192                     word + " words, " + vowel + " vowels.");
193             }
194         }
195     }
196 }

```

```
191     @Override
192     public void windowActivated(WindowEvent e)
193     {
194     }
195
196     @Override
197     public void windowClosed(WindowEvent e)
198     {
199     }
200
201     @Override
202     public void windowClosing(WindowEvent e)
203     {
204         System.exit(0);
205     }
206
207     @Override
208     public void windowDeactivated(WindowEvent e)
209     {
210     }
211
212     @Override
213     public void windowDeiconified(WindowEvent e)
214     {
215     }
216
217     @Override
218     public void windowIconified(WindowEvent e)
219     {
220     }
221
222     @Override
223     public void windowOpened(WindowEvent e)
224     {
225     }
226
227     public static void main(String[] args)
228     {
229         OOPLabFinal1 object = new OOPLabFinal1();
230
231         object.show();
232     }
233 }
```

```
1 import java.awt.*;
2 import java.awt.event.*;
3 import java.util.Random;
4 import java.util.concurrent.TimeUnit;
5
6 public class OOPLabFinal2 implements WindowListener, ActionListener, Runnable
7 {
8     private Frame frame;
9     private TextField textField;
10
11     private Button startButton;
12     private Button stopButton;
13
14     private Color color;
15
16     private Random random;
17
18     private boolean first_start_button_click;
19
20     private Thread thread;
21
22     public OOPLabFinal2()
23     {
24         frame = new Frame("Test Program 2");
25
26         textField = new TextField(50);
27
28         startButton = new Button("Start");
29         stopButton = new Button("Stop");
30
31         color = Color.GREEN;
32
33         random = new Random();
34
35         first_start_button_click = false;
36     }
37
38     @Override
39     public void run()
40     {
41         int r;
42         int g;
43         int b;
44
45         while( true )
46         {
```

```

47         r = random.nextInt(256);
48         g = random.nextInt(256);
49         b = random.nextInt(256);
50
51         String string = "RGB = (" + r + ", " + g + ", " + b + ")";
52
53         color = new Color(r, g, b);
54         frame.setBackground(color);
55
56         textField.setText(string);
57
58         try
59         {
60             TimeUnit.MILLISECONDS.sleep(500);
61         }
62         catch (InterruptedException ex)
63         {
64             System.out.println(ex);
65         }
66     }
67 }
68
69 private void show()
70 {
71     textField.setEditable(false);
72     textField.setFont( new Font("Consolas", Font.PLAIN, 16) );
73
74     startButton.addActionListener(this);
75     startButton.setFont( new Font("Consolas", Font.PLAIN, 16) );
76
77     stopButton.addActionListener(this);
78     stopButton.setEnabled(false);
79     stopButton.setFont( new Font("Consolas", Font.PLAIN, 16) );
80
81     frame.addWindowListener(this);
82     frame.setSize(800, 600);
83     frame.setResizable(true);
84     frame.setBackground(Color.ORANGE);
85     frame.setLayout( new FlowLayout() );
86     frame.add(startButton);
87     frame.add(stopButton);
88     frame.add(textField);
89     frame.setVisible(true);
90 }
91
92 @Override
93 public void actionPerformed(ActionEvent e)
94 {

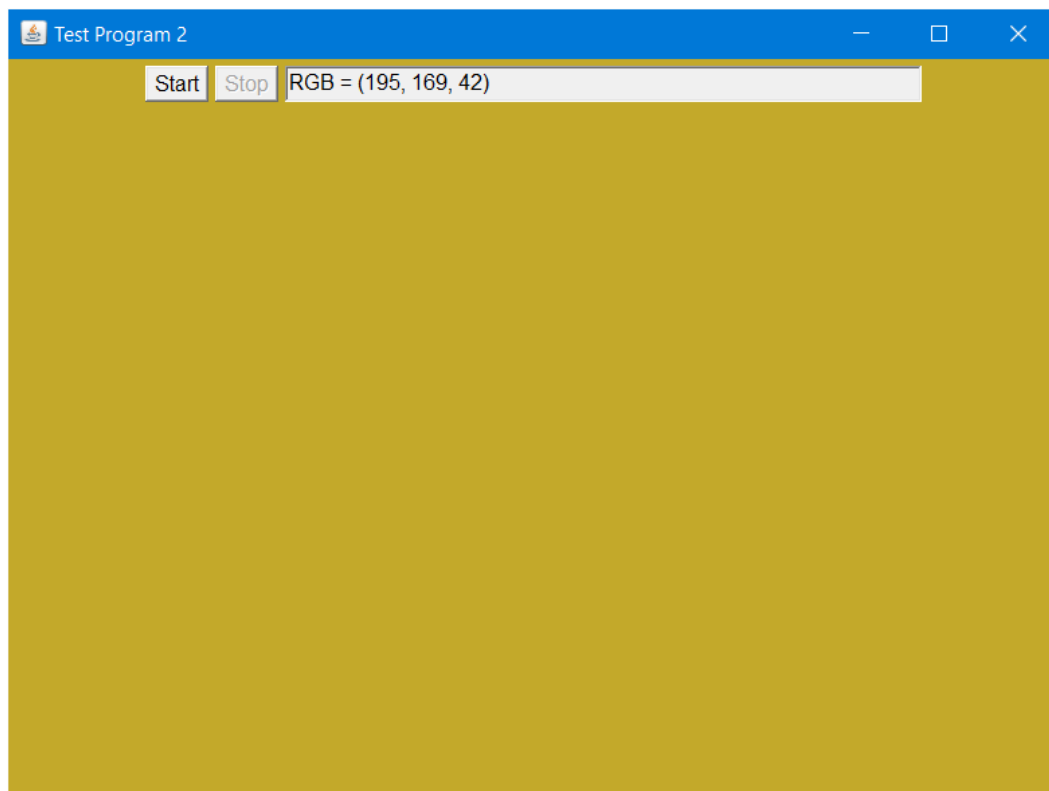
```

```

95         if( e.getSource() == startButton )
96         {
97             startButton.setEnabled(false);
98             stopButton.setEnabled(true);
99
100            if( first_start_button_click == false )
101            {
102                first_start_button_click = true;
103
104                thread = new Thread(this);
105
106                thread.start();
107            }
108
109            thread.resume();
110        }
111        else if( e.getSource() == stopButton )
112        {
113            startButton.setEnabled(true);
114            stopButton.setEnabled(false);
115
116            thread.suspend();
117        }
118    }
119
120
121    @Override
122    public void windowActivated(WindowEvent e)
123    {
124    }
125
126    @Override
127    public void windowClosed(WindowEvent e)
128    {
129    }
130
131    @Override
132    public void windowClosing(WindowEvent e)
133    {
134        System.exit(0);
135    }
136
137    @Override
138    public void windowDeactivated(WindowEvent e)
139    {
140    }
141

```

```
142         @Override
143         public void windowDeiconified(WindowEvent e)
144         {
145         }
146
147         @Override
148         public void windowIconified(WindowEvent e)
149         {
150         }
151
152         @Override
153         public void windowOpened(WindowEvent e)
154         {
155         }
156
157         public static void main(String[] args)
158         {
159             OOPLabFinal2 object = new OOPLabFinal2();
160
161             object.show();
162         }
163     }
```



6.3 Matrix Rotation

```
1  /*
2  Before rotation:
3  11 12 13 14
4  15 16 17 18
5  19 20 21 22
6  23 24 25 26
7
8  After rotation:
9  14 18 22 26
10 13 17 21 25
11 12 16 20 24
12 11 15 19 23
13 */
14 public class MatrixRotation
15 {
16     public int[][] myArray;
17
18     public MatrixRotation(int row, int column)
19     {
20         myArray = new int[row][column];
21
22         int counter = 11;
23
24         for( int i = 0; i < myArray.length; i++ )
25         {
26             for( int j = 0; j < myArray[i].length; j++ )
27             {
28                 myArray[i][j] = counter;
29                 counter++;
30             }
31         }
32     }
33
34     public void printMatrix()
35     {
36         for( int i = 0; i < myArray.length; i++ )
37         {
38             for( int j = 0; j < myArray[i].length; j++ )
39             {
40                 System.out.print( myArray[i][j] + " ");
41             }
42             System.out.println();
43         }
44     }
45 }
```

```

46     public void rotateMatrix()
47     {
48         int temp;
49
50         for( int i = 0; i < (myArray.length / 2); i++ )
51         {
52             for( int j = i; j < (myArray.length - i - 1); j++ )
53             {
54                 temp = myArray[i][j];
55
56                 myArray[i][j] = myArray[j][myArray.length - i - 1];
57
58                 myArray[j][myArray.length - i - 1] =
myArray[myArray.length - i - 1][myArray.length - j - 1];
59
60                 myArray[myArray.length - i - 1][myArray.length - j -
1] = myArray[myArray.length - j - 1][i];
61
62                 myArray[myArray.length - j - 1][i] = temp;
63             }
64         }
65     }
66
67     public static void main(String[] args)
68     {
69         //new MatrixRotation(4, 4).printMatrix();
70
71         MatrixRotation matrixRotation = new MatrixRotation(4, 4);
72
73         System.out.println("Before rotation:");
74         matrixRotation.printMatrix();
75
76         matrixRotation.rotateMatrix();
77
78         System.out.println("After rotation:");
79         matrixRotation.printMatrix();
80     }
81 }

```

Java Reference Books :

<https://drive.google.com/open?id=0BzlrQNpxfFtgWV9hS3NtcERkaDg>

Java Tutorials :

<http://www.w3schools.in/java/>

<http://www.tutorialspoint.com/java/index.htm>

Java Discussion Portals :

<http://www.geeksforgeeks.org/java/>

<http://stackoverflow.com/questions/tagged/java/>