Chapter 1

CSI 211, Object Oriented Programming

1.1 Faculty Details

Dr. Dewan Md. Farid
Assistant Professor
Department of Computer Science and Engineering
United International University
Room: 401 at UIU Bhaban
Cell: 01715-833499 and Email: dewanfarid@cse.uiu.ac.bd

1.1.1 Short Biography of Dr. Farid

Dr. Dewan Md. Farid is an Assistant Professor in the Department of Computer Science and Engineering, United International University, Bangladesh. He was a Post Doctoral Fellow in the Computational Intelligence Group (CIG), Department of Computer Science and Digital Technology, University of Northumbria at Newcastle, UK in 2013. He has received a Ph.D. degree in Computer Science and Engineering from Jahangirnagar University, Bangladesh in 2012. Part of his Ph.D. research has been done at ERIC Laboratory, University Lumire Lyon 2, France. He has received numerous awards for his contribution on research into knowledge discovery and data mining, including Senior Fellowship I & II awarded by National Science & Information and Communication Technology (NSICT), Ministry of Science & Information and Communication Technology, Govt. of Bangladesh in 2008 and 2011 respectively, and eLink (east west Link for Innovation, Networking and Knowledge exchange) & cLink (Centre of excellence for Learning, Innovation, Networking and Knowledge) Erasmus Mundus scholarship awarded by European Union in 2009 and 2013 respectively. He has published a book chapter, 13 international journals and 17 international conference papers in the field of data mining and machine learning. He has participated and presented his papers in international conferences at Malaysia, Portugal, Italy, and France. He is a member of IEEE and IEEE Computer Society.

1.2 What is Java?

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.

1.2.1 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 What is Java Applets?

Applets are Java programs that reside on web servers, download by a browser to a client's computer and run by that browser. Applets are usually small in size to minimize download time and are invoked by a Hypertext Markup Language (HTML) web page.

Table 1.1: Major release versions of Java.

Version	Date
JDK 1.0	January 23, 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

1.4 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 three 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.

1.5 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-allocated 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 the can be freed in the object lifecycle.

1.6 A Simple Java Application

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

1.6.1 Simple Java Program for beginners

This short example shows how to write first java application and compile and run it, which prints a String **Welcome in Java World** in output.

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

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 program 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 being complied.

Line 2 declares the **main** method, where the program starts to execute. The following describes each element of Line 2:

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

Line 4 end of the main() method.

Line 5 end of the Welcome class.

1.6.2 Another Java Program for beginners

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 then one class declaration, but only one class can be declared public.

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

Line 3 creates an object of **Hello** class named **hello**

Line 4 call the **display()** method of **Hello** class using the **hello** object with the dot notation.

Line 8 declares a user-defined class **Hello** with default access control.

Line 9 declares a user-defined method named **display()**.

1.7 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. The Java technology class, attribute, constructor, and method declaration takes the following forms:

```
<modifier>* class <class_name>{
   <modifier>* <data_type>* <attribute_name>;
   <modifier>* <data_type>* <attribute_name>[=<value>];

   [<modifier>] <class_name>(<argument>*){
       <statement>*
   }

   <modifier>* <return_type>* <method_name>(<argument>*){
       <statement>*
   }
}
```

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.

```
public class MyDate{
   private int day;
   private int month;
   private int year;

   public MyDate(){
      day = 1;
      month = 6;
      year = 1979;
   }

   public void showDate(){
      System.out.println("Day: " + day);
      System.out.println("Month: " + month);
      System.out.println("Year: " + year);
   }

   public static void main(String[] args){
        MyDate date = new MyDate();
        date.showDate();
    }
}
```

1.8 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 hierarchical and 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).

```
package bankingPrj;

public class Account{
    public String accountName = "Saving Account";

    public void showAccName(){
        System.out.println(accountName);
    }

    public static void main(String[] args){
        Account acc = new Account();
        acc.showAccName();
    }
}
```

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 **bankingPrj**. Now you can run this **Account.class** file in **bankingPrj** package by typing the command **java bankingPrj.Account**

```
1  package bankingPrj;
2
3  import bankingPrj.Account;
4
5  public class Coustomer{
6      public String name = "James Bond";
7
8      public static void main(String[] args){
9          Account acc = new Account();
10         acc.showAccName();
11         Coustomer cous = new Coustomer();
12         System.out.println(cous.name);
13     }
14 }
```

Similarly, we can compile and run this code.

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

Line 9 we create object of **Account** class.

Line 10 we call the method of **Account** class.

1.9 Comments and Semicolons

1.9.1 Java Comments

In Java Technology, we can add comments to Java source code in three ways which are given bellow:

```
public class TestComment{
   // comment on one line
   public static void main(String[] args){
      /* comment on one
       * or more lines
       */
      System.out.println("Comments in Java");
   }
   /** documentation comment
    *  can also span one or more lines
    */
}
```

1.9.2 Semicolons

In the Java source code a statement is one or more lines of code terminated with a semicolons (;).

```
public class TestSemicolon{
  public static void main(String[] args){
     int a=10, b=20, c;
     c = a+b;
     System.out.println(c);
  }
}
```

1.10 Blocks

A block as a group of statements that are collected together called a compound statement. It is bound by opening and closing braces { }. A class definition is contained in a block. A block statement can be nested withing another block. In Java source code, block statements are executed first, then constructor statements are executed, and then method statements are executed.

```
public class BlockTest{
   public String info;

   public BlockTest(){
      info = "Constructor: Executed in 2nd step";
      System.out.println(info);
   }

   {
      info = "Block: Executed in 1st step";
      System.out.println(info);
   }

   public void show(){
      info = "Method: Executed in 3rd step";
      System.out.println(info);
   }

   public static void main(String[] args){
        BlockTest bt = new BlockTest();
        bt.show();
    }
}
```

1.11 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

1.12 The 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.

```
public class AddNumbers{
    private int num1, num2, result;

    public AddNumbers(int num1, int num2){
        this.num1 = num1;
        this.num2 = num2;
    }

    public void add(){
        result = num1 + num2;
        System.out.println("Result is: " + result);
    }

    public static void main(String[] args){
        AddNumbers addnum = new AddNumbers(10, 20);
        addnum.add();
    }
}
```

Table 1.2: Java Programming Language Keywords.

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				

1.13 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 that are used in Java programming language shown in tabel 1.2.

1.14 Data Types in Java

In Java technology, data are divided into two broad categories: primitive types and class types.

Primitive data are eight types in four categories:

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

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

1. Textual: String
2. All classes that declare by ourself

1.14.1 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.

Table 1.3: Default value of class variables.

Class Variable Types	**Value**
boolean	false
char	’0̆000’
byte	0
short	0
int	0
long	0L
float	0.0F
double	0.0D
all class types	null

1.15 Operators in Java Programming

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

1. Assignment operator (=)
2. Arithmetic operator (+, -, *, /, %, ++, -)
3. Relational operator
4. Logical operator
5. Bitwise operator
6. What operator ((boolean expression)? true_exp: false_exp;)
7. instanceof operator

1.16 String Concatenation with +

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

```
public class StringConTest{
  public static void main(String[] args){
     String salutation = "Mr. ";
     String name = "Dewan " + " Md. " + "Farid";
     String title = salutation + name;
     System.out.println(title);
  }
}
```

1.17 Casting

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

```
public class CastTest{
  public static void main(String[] args){
     long bigValue = 99L;
     int smallValue = (int) bigValue; // Casting
     System.out.println(smallValue);
     smallValue = 50;
     bigValue = smallValue; // Auto Casting
     System.out.println(bigValue);
  }
}
```

Chapter 2

Flow Controls, and Arrays

2.1 Branching Statements

2.1.1 The if-else Statement

The general form of the if-else statement is:

```
if(<boolean_expression>)
   <statement_or_block>
else
   <statement_or_block>
```

```
public class IfTest{
   public static void main(String[] args){
      int x = (int) (Math.random() * 100);
      System.out.println(x);
      if(x>100){
          System.out.println("WHITE");
      }
      else{
          System.out.println("BLACK");
      }
   }
}
```

2.1.2 The switch Statement

The switch statement syntax is:

```
switch(<expression>){
    case <constant>: <statement_or_block>*
                     [break;]
    case <constant>: <statement_or_block>*
```

```
                    [break;]
   case <constant>: <statement_or_block>*
                    [break;]
   default: <statement_or_block>*
}
```

```
public class SwitchTest{
  public void mySwitch(int x){
     switch(x){
        case 1: System.out.println("RED");
                break;
        case 2: System.out.println("GREEN");
                break;
        case 3: System.out.println("BLUE");
                break;
        default: System.out.println("WHITE");
     }
  }

  public static void main(String[] args){
     SwitchTest st = new SwitchTest();
     st.mySwitch(2);
  }
}
```

2.1.3 Loop Flow Control

break; Use the break statement to exit from switch, loop, and block.

continue; Use the continue to jump at the end of the loop body, and then return to the loop control.

2.2 Looping statements

Looping statements enable us to execute blocks of statements repeatedly. Java supports three of loop constructs: for, while, and do-while loops. The for and while loops test the loop condition before execution of loop body. And do-while loops check the loop condition after executing the loop body.

2.2.1 The for Loop

The for loop syntax is:

```
for(initialization; condition; increment/decrement){
  statements;
}
```

```
public class Loop_for{
  public static void main(String[] args){
    for(int i=1; i<=50; i++){
      System.out.println(i);
    }
  }
}
```

Write a Java program that loops 1-50 and print foo for every multiple of 3, bar for every multiple of 5 and baz for every multiple of 3 and 5.

```
public class Series{
  public static void main(String[] args){
     for(int i=1; i<=50; i++){
         System.out.print(i);
         if(i%3==0 && i%5==0)
            System.out.print(" - baz");
         else if(i%3==0)
            System.out.print(" - foo");
         else if(i%5==0)
            System.out.print(" - bar");
         System.out.println();
     }
  }
}
```

Write a Java program that will display the Fibonacci sequence is generated by adding the previous two terms. By starting with 1, 1, and 2, and continue up to the first 10 terms.

```
public class Fibonacci{
  public static void main(String[] args){
     int x=1, y=1, z=0;
     for(int i=1; i<=10; i++){
        if(i==1){
           System.out.print(x+" "+y);
        }
        z=x+y;
        System.out.print(" "+z);
        x=y;
        y=z;
     }
  }
}
```

Write a Java program to show the following output:

```
                1
              1 2 1
            1 2 3 2 1
          1 2 3 4 3 2 1
        1 2 3 4 5 4 3 2 1
      1 2 3 4 5 6 5 4 3 2 1
    1 2 3 4 5 6 7 6 5 4 3 2 1
  1 2 3 4 5 6 7 8 7 6 5 4 3 2 1
1 2 3 4 5 6 7 8 9 8 7 6 5 4 3 2 1
```

```java
public class Pyramid{
  public static void main(String[] args){
     int n=9;
     for(int i=1; i<=n; i++){
         for(int j=n-i; j>=1; j--)
             System.out.print("   ");
         for(int k=1; k<=i; k++)
             System.out.print(k+" ");
         for(int p=i-1; p>=1; p--)
             System.out.print(p+" ");
         System.out.println();
     }
  }
}
```

2.2.2 The while Loop

The while loop syntax is:

```
initialization;
while(condition){
  statements;
  increment/decrement;
}
```

```java
public class Loop_while{
  public static void main(String[] args){
    int i=1;
    while(i<=50){
      System.out.println(i);
      i++;
    }
  }
}
```

2.2.3 The do-while Loop

The do-while loop syntax is:

```
initialization;
do{
  statements;
  increment/decrement;
}while(condition);
```

```
public class Loop_do{
  public static void main(String[] args){
    int i=1;
    do{
      System.out.println(i);
      i++;
    }while(i<=50);
  }
}
```

2.3 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 an class or reference type.

```
public class ArrayTest{
  public static void main(String[] args){
    int[] myArray; // declaring an array
    myArray = new int[5]; // creating array
    // int[] myArray = new int[5];
    for(int i=0; i<5; i++)
       myArray[i]= 100+i; // initializing array

    for(int i=0; i<=4; i++)
       System.out.println(myArray[i]);
  }
}
```

```
class Bird{
   public void fly(){
      System.out.println("Bird can fly");
   }
}

```

```
public class ClassArray{
    public static void main(String[] args){
        Bird[] myArray = new Bird[3];
        for(int i=0; i<3; i++)
            myArray[i] = new Bird();
        for(int i=0; i<3; i++)
            myArray[i].fly();
    }
}
```

Write a Java program to create a class named ArrayExample, in this class declare an array named myArray of Car type. Now initialize the myAarray and call the display method for each index of myArray in the main method. In the Car class there are two class variables: carName and carModel, constructor (initialized the variables), and display method (display the value of carName and carModel).

```
class Car{
   public String carName;
   public String carModel;

   public Car(String carName, String carModel){
       this.carName = carName;
       this.carModel = carModel;
   }

   public void display(){
       System.out.println("Name: "+carName+", and Model: "+carModel);
   }
} // end of Car class;

public class ArrayExample{
   public static void main(String[] args){
        Car[] myArray = new Car[3];
        myArray[0] = new Car("BMW", "Road Track 509");
        myArray[1] = new Car("Toyota", "X corola 2012");
        myArray[2] = new Car("Honda", "M 2011");

        for(int i=0; i<3; i++)
        {
            myArray[i].display();
        } // end of for loop;
   } // end of the main();
} // end of the class;
```

2.4 Multidimensional Arrays in Java

The Java programming language provide multidimensional arrays, which is arrays of arrays. Line 4 and line 16 of code 2-9 show the example of two-dimensional or multidimensional array. Line 17 to 20 show the creation of non-rectangular arrays of arrays.

```
1   class MultiDim{
2      public void matrix(){
3         // declaration and creation of Array
4         int[][] twoDim = new int[3][3];
5         // initializing array using for loop
6         for(int row=0; row<3; row++)
7            for(int col=0; col<3; col++)
8               twoDim[row][col]= row+col+10;
9         // use of array
10        for(int row=0; row<=2; row++)
11           for(int col=0; col<=2; col++)
12              System.out.println(twoDim[row][col]);
13     }
14
15     public void mulArray(){
16        int [][] myArray = new int[4][];
17        myArray[0] = new int[2];
18        myArray[1] = new int[4];
19        myArray[2] = new int[6];
20        myArray[3] = new int[8];
21     }
22  }
```

```
public class Magic{
  public static void main(String[] args){
    int[][] rat = new int [3][];
    rat[0] = new int[2];
    rat[1] = new int[5];
    rat[2] = new int[3];

    for(int row=0; row<=2; row++){
       switch(row){
          case 0: for(int j=0; j<=1; j++)
                     rat[row][j] = 5+j;
                  break;
          case 1: for(int j=0; j<=4; j++)
                     rat[row][j] = 50+j;
                  break;
```

```
            case 2: for(int j=0; j<=2; j++)
                         rat[row][j] = 100+j;
         }
      }

      for(int row=0; row<=2; row++){
         switch(row){
           case 0: for(int j=0; j<=1; j++)
                        System.out.print(rat[row][j]+"\t");
                      System.out.print("\n");
                      break;
           case 1: for(int j=0; j<=4; j++)
                        System.out.print(rat[row][j]+"\t");
                      System.out.print("\n");
                      break;
           case 2: for(int j=0; j<=2; j++)
                        System.out.print(rat[row][j]+"\t");
                      System.out.print("\n");
         }
      }
   }
}
```

2.4.1 Matrix Multiplication

```
public class Matrix_Mult{

   public static void main(String[] args){
      int[][] a = new int[3][3];
      int[][] b = new int[3][3];
      int[][] c = new int[3][3];

      for(int i=0; i<=2; i++)
         for(int j=0; j<=2; j++)
         {
            a[i][j] = 10+i+j;
            b[i][j] = 20+i+j;
         }
      System.out.println("Matrix A:");
      for(int i=0; i<=2; i++)
      {
         for(int j=0; j<=2; j++)
           System.out.print("\t"+a[i][j]);
         System.out.print("\n");
      }
      System.out.println("Matrix B:");
      for(int i=0; i<=2; i++)
      {
         for(int j=0; j<=2; j++)
            System.out.print("\t"+b[i][j]);
         System.out.print("\n");
      }

      // Matrix Multiplication
      for(int i=0; i<=2; i++)
         for(int j=0; j<=2; j++)
           c[i][j]=(a[i][0]*b[0][j])
                     +(a[i][1]*b[1][j])+(a[i][2]*b[2][j]);

      System.out.println("Matrix C:");
      for(int i=0; i<=2; i++)
      {
         for(int j=0; j<=2; j++)
           System.out.print("\t"+c[i][j]);
         System.out.print("\n");
      }
   }
}
```

2.5 Array Bounds, Resizing, and 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 id 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.

```
public class ArrayExample{
  public void arrayElement(int[] myArray){
     for(int i=0; i<myArray.length; i++)
        System.out.println(myArray[i]);
  }

  public void arrayResizing(){
     int[] arr = new int[10];
     arr = new int[5];
  }
```

```

    public void arrayCopy(){
        int[] x = {1,2,3,4,5,6};
        int[] y = {11,12,13,14,15,16,17,18,19,20};
        System.arraycopy(x, 0, y, 0, x.length);
    }
}
```

2.6 Enhanced for loop

The Java 2 Platform Standard Edition (J2SE) version 5.0 added enhanced for loop.

```
public class EnhanFor{
    public static void main(String[] args){
        int[] myArray = new int[5];
        for(int i=0; i<5; i++)
            myArray[i]= i+11;
        // Enhanced for loop
        for(int element : myArray){
            System.out.println(element);
        }
    }
}
```

Chapter 3

Object Oriented Programming

3.1 Access Control

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 introduce the autoboxing concept.

```
public class BoxingTest{

   public void boxing(){
       int pInt = 420;
       Integer wInt = new Integer(pInt); // boxing
       int p2 = wInt.intValue(); // unboxing
   }

   public void autoboxing(){
```

Table 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

Table 3.2: Wrapper Classes.

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

```
        int pInt = 420;
        Integer wInt = pInt; // autoboxing
        int p2 = wInt; // autounboxing
    }
}
```

3.3 Key Features of Java

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

1. Encapsulation
2. Inheritance
3. Polymorphism

3.3.1 Encapsulation

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

```
class MyNumber{
  private int number;

  public void setNumber(int number){
    this.number = number;
  }

  public int getNumber(){
    return number;
  }
11 }
12
13 public class EncapTest{
14   public static void main(String[] args){
15     MyNumber my = new MyNumber();
16     my.setNumber(45);
17     System.out.println(my.getNumber());
18   }
19 }
```

At line 2, attribute number is a private member of MyNumber class, so this attribute will not be accessible form other classes. But from the EncapTest class we are accessing the number variable of MyNumber class using set and get methods of MyNumber class.

3.3.2 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 inherited all of the members from the parent-class, but it does not inherit the constructor.

```
class Cat{
    public int x = 10;
    public int y = 20;

    public void show(){
      System.out.println(x+" "+y);
    }
}

public class Rat extends Cat{
    public int z = 30;

    public static void main(String[] args){
        Rat r = new Rat();
        r.show();
        System.out.println(r.x+" "+r.y+" "+r.z);
    }
}
```

In the above code, Rat class extends Cat class, so Rat class become child class of Cat class.

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

In line 8 declare the method show(), which override the parent class show() method of line 2.

Invoking Overriding Methods

A sub-class method can invoke a super-class method using the super keyword. In the following code, line 14 invokes the parent class method showDetails().

```
1  class Employee{
2     public String name = "Mahmud";
3     public float salary = 50000;
4
5     public void showDetails(){
6        System.out.println(name+" "+salary);
7     }
8  }
9
10 public class Manager extends Employee{
11    public String department = "Engineering";
12
13    public void showDetails(){
        super.showDetails();
        System.out.println(" "+department);
    }

    public static void main(String[] args){
        Manager m = new Manager();
        m.showDetails();
    }
}
```

Invoking Parent Class Constructor

A sub-class constructor can invoke a super-class constructor using the super keyword. In line 19, show the parent class constructor invoking.

```
1  class Employee1{
2    public String name;
3    public float salary;
4
5    public Employee1(String name, float salary){
6       this.name = name;
7       this.salary = salary;
8    }
9
10   public void showDetails(){
11      System.out.println("Name: "+name+" Salary: "+salary);
12   }
13 }
14
15 public class Manager1 extends Employee1{
16   public String department;
17
18   public Manager1(String department){
19      super("Mahmud", 50000F);
20      this.department = department;
21   }
22
23   public void showDetails(){
24      super.showDetails();
25      System.out.println("Department: "+department);
26   }
27
28   public static void main(String[] args){
29      Manager1 m = new Manager1("Engineering");
```

```
        m.showDetails();
    }
}
```

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 four methods of same name with different argument, which is an example of overloading methods.

```
public class ABC{
    int a, b, c;

    public void setValue(){
        a=2;
        b=4;
        c=6;
    }

    public void setValue(int a){
        this.a=a;
        b=4;
        c=6;
    }

    public void setValue(int a, int b){
        this.a=a;
        this.b=b;
        c=6;
    }

    public int setValue(int a, int b, int c){
        this.a=a;
        this.b=b;
        this.c=c;
        int z = a+b+c;
        return z;
    }
}
```

Overloading Constructors

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

```
public class Light{
    int a, b, c;

    public Light(){
        a=2;
        b=4;
        c=6;
    }

    public Light(int a){
        this.a=a;
        b=4;
        c=6;
    }

    public Light(int a, int b){
        this.a=a;
        this.b=b;
        c=6;
    }

    public Light(int a, int b, int c){
        this.a=a;
        this.b=b;
        this.c=c;
    }
}
```

Example of Encapsulation, and Inheritance

Example with overriding, overloading, and Invoking

```
class Man{
   private int weight;

   public Man(int weight){
      this.weight=weight;
   }

   public void setWeight(int weight){ //seter method
```

```
        if(weight>=50 && weight<=100){
           this.weight=weight;
        }
  }

  public int getWeight(){ //geter method
     return weight;
  }

  public void show(){
     System.out.println(weight);
  }
}

public class SuperMan extends Man{
  public int power;

  public SuperMan(int power){
     super(55); //invoking parent class constructor
     this.power=power;
  }

  public SuperMan(int weight, int power){ //Overloading
     super(weight); //invoking parent class constructor
     this.power=power;
  }

  public void show(){ //overriding method
     super.show(); //invoking parent class method
     System.out.println(power);
  }

  public void show(String abc){ //overloading method
     System.out.println(abc);
  }

  public static void main(String[] args){
     SuperMan s = new SuperMan(10);
     s.show();
     SuperMan s1 = new SuperMan(50, 100);
     s1.show("Tiger");
  }
}
```

3.4 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.

```
class Man{
    public void fly(){
      System.out.println("Man can not fly");
    }
}

class SuperMan extends Man{
    public void fly(){
      System.out.println("Superman can fly");
    }
}

public class TestMan{
    public static void main(String[] args){
       Man m = new SuperMan(); // polymorphism
       m.fly();
    }
}
```

3.5 Homogeneous Collection

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

```
public class TestHomogeneous{
  public static void main(String[] args){
      TestHomogeneous[] arr = new TestHomogeneous[3];
      arr[0] = new TestHomogeneous();
      arr[1] = new TestHomogeneous();
      arr[2] = new TestHomogeneous();
  }
}
```

3.6 Heterogeneous Collection

Heterogeneous collection is a collection of dissimilar objects or classes.

```
class Man{
    public void fly(){
      System.out.println("Man can not fly");
    }
}

class SuperMan extends Man{
    public void fly(){
      System.out.println("Superman can fly");
    }
}

class SpiderMan extends Man{
    public void fly(){
      System.out.println("Spiderman can't fly, but can jump");
    }
}

public class TestHeterogeneous{
    public static void main(String[] args){
        Man[] arr = new Man[3];
        arr[0] = new Man();
        arr[1] = new SuperMan();
        arr[2] = new SpiderMan();
    }
}
```

3.7 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.

```
public class Car{
}
// or we can declare the Car class by
// extending Object class. Both have same mining.
 public class Car extends Object{
}
```

3.8 The instanceof Operator

Using the instanceof operator, we can know the actual object of a class. At line 22, we are passing object through argument list of test(Object x) method but we do not know this x object is from which class. Then we use instanceof operator to know object x is from which class by the conditions.

```
class Car{
  public void abc(){
     System.out.println("Car");
  }
}

class Bus{
  public void xyz(){
     System.out.println("Bus");
  }
}

public class Testinstanceof{
  public static void main(String[] args){
    Car c = new Car();
    Bus b = new Bus();
    Testinstanceof t = new Testinstanceof();
    t.test(c);
    t.test(b);
  }

  public void test(Object obj){
    if( obj instanceof Car){
       System.out.println("Object is of Car class");
    }else if( obj instanceof Bus){
       System.out.println("Object is of Bus class");
    }else{
       System.out.println("Object is not of Car/Bus class");
    }
  }
}
```

3.9 The equals Method

The method public boolean equals(Object obj) of Object class in the java.lang package compares two objcets for equality. This method returns true, only if the two objects being compared refer to the same object. On the other hand, == operator performs an equivalent comparison. If x==y returns true, that means x and y refer to the same object. We should override the public int hashCode() method of Object class whenever we override the equals method. A simple implementation could use a bit wise XOR on the hash codes of the elements tested for equality.

```
public class TestEquals{
  public int x;
  public int y;

  public TestEquals(int x, int y){
     this.x=x;
     this.y=y;
  }

  public boolean equals(Object obj){
     boolean result = false;
     if((obj!=null)&&(obj instanceof TestEquals)){
        TestEquals t = (TestEquals) obj;
        if((x==t.x)&&(y==t.y)){
           result = true;
        }
     }
     return result;
  }

  public int hashCode(){
     return (x^y);
  }

  public static void main(String[] args){
     TestEquals t1 = new TestEquals(5, 10);
     TestEquals t2 = new TestEquals(5, 10);
     if(t1==t2){
        System.out.println("t1 is identical to t2");
     }else{
        System.out.println("t1 is not identical to t2");
     }
     if(t1.equals(t2)){
        System.out.println("t1 is equal to t2");
     }else{
        System.out.println("t1 is not equal to t2");
     }
     System.out.println("Set t2=t1");
     t2=t1;
     if(t1==t2){
        System.out.println("t1 is identical to t2");
        }else{
            System.out.println("t1 is not identical to t2");
        }
    }
}
```

3.10 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.

```
public class TesttoString{
  public static void main(String[] args){
      TesttoString now = new TesttoString();
      System.out.println(now);
      // is equivalent to:
      System.out.println(now.toString());
  }
}
```

3.11 The Static keyword

In Java technology, members (attributes, methods, and nested classes) of a class can be declare 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 it's parameters. Attempting to access non-static class attributes in a static methods will cause a compiler error. We can not override the static method. The main() method is a static method because the 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.

```
public class TestStatic{
  public static int count = 10;

  public static void incrementCount(){
      count++;
  }

```

```
    public static void main(String[] args){
        for(int i=0; i<3; i++){
            System.out.println("Count is: "+TestStatic.count);
            TestStatic.incrementCount();
        }
    }
}
```

3.12 The final Keyword

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

```
public final class TestFinal{
  public final int studentID = 10;

  public final void diplay(){
     System.out.println("Final method can not be override");
  }
}
```

3.13 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 contain 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. In code 3-18, Vehicle class is an abstract class because it contain an abstract method goFast() in line 5. This goFast() method is implemented in the sub-class Car at line 13.

```
1  abstract class Vehicle{
2     public String model = "E class";
3     public String year = "2008";
4
5     public abstract void goFast();
6
7     public void show(){
8        System.out.println("Model: "+model+" Year: "+year);
9     }
}

public class Car extends Vehicle{
  public void goFast(){
     System.out.println("Car can go fast");
  }

  public static void main(String[] args){
     // Vehcle v = new Vehicle(); // compiler error
     Car c = new Car();
     c.show();
     c.goFast();
  }
}
```

3.14 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.

```
interface Bounceable{
   public abstract void bounce();
   void setBounce(int b);
}

public class Tire implements Bounceable{
   public void bounce(){
      System.out.println("Love");
   }

   public  void setBounce(int b){
      int a = b;
      System.out.println(a);
   }

   public static void main(String[] args){
      Tire t = new Tire();
```

```
        t.bounce();
        t.setBounce(15);
    }
}
```

3.14.1 Example of abstract and interface

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

```
abstract class Animal{

    public void type(){
        System.out.println("Animal");
    }

    public abstract void name();

}

interface Cat{
    public abstract void jump();
    void run();  // it will be automaticly public and abstract
}

interface Bird{
    void fly();
}

public class Tiger extends Animal implements Cat, Bird{

    public void fly(){
        System.out.println("Tiger can't fly");
    }

    public void jump(){
        System.out.println("Tiger can jump");
    }

    public void run(){
        System.out.println("Tiger can run");
    }

    public void name(){
        System.out.println("Tiger");
    }

    public static void main(String[] args){
        Tiger t = new Tiger();
        t.name();
        t.jump();
        t.run();
        t.fly();
    }
}
```

3.15 Exceptions

Exceptions are a mechanism used by many programming languages to describe what to do when errors happens. There are two 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 arises 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.

```
public class TestExceptionA{
  public static void main(String[] args){
     int sum = 0;
     for(int i=0; i<args.length; i++){
       sum += Integer.parseInt(args[i]);
     }
     System.out.println("Sum: " + sum);
  }
}
```

This program works 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: Runtime Exception
```

3.15.1 The try-catch Statement

To avoid the problem of code 3-20, we can use the try-catch statement. If any exception or error occurred in the try block then the catch block will execute. If try block execute without any error, then catch block will not execute.

```
public class TestExceptionB{
   public static void main(String[] args){
       int sum = 0;
       for(int i=0; i<args.length; i++){
           try{
               sum += Integer.parseInt(args[i]);
           }catch(NumberFormatException e){
               System.out.println("Index: "+i+ "is not integer. "+e);
           }
       }
       System.out.println("Sum: " + sum);
   }
}
```

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

```
Compile: javac TestExceptionB.java
Run: java TestExceptionA 2 4 six 8
```

Output: Index value of array 2 is not integer. java.lang.NumberFormatException: invalid character at position 1 in six, Sum: 14

3.15.2 Using Multiple catch Clauses

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

```
public class Multi_catch{
   public static void main(String[] args){
       try{
           int i = 9/0;
           System.out.println(i);
       }catch(ArithmeticException e1){
           System.out.println("Arithmetic Exception." + e1);
       }catch(Exception e2){
           System.out.println("Exception." + e2);
       }
   }
}
```

3.15.3 Call Stack Mechanism

If a statement throws an exception, and that exception is not handled in the immediately enclosing method, then that exception is throws 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.

3.15.4 The try-catch-finally statement

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 executes.

```
public class Test_finally{
  public static void main(String[] args){
    try{
       int i = 9/0;
       System.out.println(i);
    }catch(ArithmeticException e1){
       System.out.println("Arithmetic Exception." + e1);
    }finally{
       System.out.println("finally block must executes");
    }
  }
}
```

```
The Exception Declare Rules
In Java, we can declare exceptions by following:
1.try-catch-finally
2.void methodA() throws IOException{}
3.void methodB() throws IOException, OtherException{}
```

```
public class DeclareException{
  public void methodA(){
     try{
     }catch(Exception e){
        System.out.println("If error in try then catch execute");
     }finally{
        System.out.println("finally must executes");
     }
  }
  public void methodB() throws SecurityException{
}
```

```
    public void methodC() throws SecurityException, Exception{
}
}
```

3.15.5 Method Overriding and 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, a FileNotFoundException, which is the subclass of IOException, but not Exception, which is the superclass of IOException.

```
public class A{
  public void methodA() throws IOException{  }
}

class B extends A{
  public void methodA() throws EOFException{
    // Legal, because EOFException is the subclass of IOException;
  }
}

class C extends A{
  public void methodA() throws Exception{
    // Illegal, because Exception is the superclass of IOException;
  }
}
```

3.15.6 Creating and Throwing User Defined Exception

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

```
public class MyException extends Exception{
  public MyException(String massage){
     super(massage);
  }

  public static void main(String[] args){
     try{
        throw new MyException("User defined exception");
     }catch(MyException e){
        System.out.println(e);
     }
  }
}
```

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 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.

```
public class CommandLine{
  public static void main(String[] args){
    for(int i=0; i<args.length; i++)
      System.out.println("Value of args in "+i+" index is: "+args[i]);
  }
}
```

```
[farid@localhost MyJavaBook]\$ javac CommandLine.java
[farid@localhost MyJavaBook]\$ java CommandLine 13 15 17
Value of args in 0 index is: 13
Value of args in 1 index is: 15
Value of args in 2 index is: 17
```

4.2 Console Input/Output

Java 2 SDK support console I/O with three public variables in the java.lang.System class:

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

3. System.err (System.err is a PrintStream onject)

Following code provides an example of keybord input, which import the java.io.*; package. In line 6, InputStreamReader reads characters and converts the row bytes into Unicode characters. In line 7, BufferedReader provides the readLine() method (in line 10), which enables the program to read from standard input (keyboard) one line at a time. Line 12, close the buffered reader. The readLine() method can throw an I/O exception, so this code is in a try-catch block.

```
1  import java.io.*;
2
3  public class ConsoleInput{
4    public static void main(String[] args){
5      String s;
6      InputStreamReader isr = new InputStreamReader(System.in);
7      BufferedReader in = new BufferedReader(isr);
8      try{
9         System.out.println(”Write something: ”);
10        s = in.readLine();
11        System.out.println(”Read: ” + s);
12        in.close();
13     }catch(IOException e){
14        e.printStackTrace();
15     }
16   }
17 }
```

4.3 Scanner

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

```
import java.io.*;
import java.util.Scanner;

public class ScannerTest{
  public static void main(String[] args){
       Scanner s = new Scanner(System.in);
       String str = s.next();
       System.out.println(str);
       int num = s.nextInt();
       System.out.println(num);
       s.close();
  }
}
```

4.4 File in Java

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

4.4.1 Write String to a File

```
import java.io.*;

public class WriteFile{
  public static void main(String[] args){
     File file = new File("/home/farid/MyJavaBook", "MyText.txt");
     try{
        InputStreamReader isr = new InputStreamReader(System.in);
        BufferedReader in = new BufferedReader(isr);
        PrintWriter out = new PrintWriter(new FileWriter(file));
        System.out.println("Write String: ");
        String str = in.readLine();
        out.println(str);
        in.close();
        out.close();
     }catch(IOException e){
        e.printStackTrace();
     }
  }
}
```

4.4.2 Read String from a File

```
import java.io.*;

public class ReadFile{
  public static void main(String[] args){
     File file = new File("/home/farid/MyJavaBook", "MyText.txt");
     try{
        BufferedReader in = new BufferedReader(new FileReader(file));
        String str = in.readLine();
        while(str != null){
            System.out.println("Read: " + str);
            str = in.readLine();
            }
            in.close();
        }catch(FileNotFoundException e1){
            System.out.println("File not found");
        }catch(IOException e2){
            System.out.println("Input/ output problem");
        }
    }
}
```

4.5 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.

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

public class FrameWithPanel implements WindowListener{
    private Frame f;
    private Panel p;

    public FrameWithPanel(){
        f = new Frame("Frame Title");
        p = new Panel();
    }

    public void launchFrame(){
        f.addWindowListener(this);
        f.setSize(400,400);
        f.setBackground(Color.red);
        f.setLayout(null);

        p.setSize(150,150);
        p.setBackground(Color.green);
        f.add(p);
        f.setVisible(true);
    }

    public void windowClosing(WindowEvent e){
        System.exit(0);
    }

    public void windowOpened(WindowEvent e){}
    public void windowIconified(WindowEvent e){}
    public void windowDeiconified(WindowEvent e){}
    public void windowClosed(WindowEvent e){}
    public void windowActivated(WindowEvent e){}
    public void windowDeactivated(WindowEvent e){}

    public static void main(String[] args){
        FrameWithPanel fp = new FrameWithPanel();
        fp.launchFrame();
    }
}
```

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

public class TestMenuBar implements WindowListener, ActionListener{
    private Frame f;
    private MenuBar mb;
    private Menu m1, m2, m3;
    private MenuItem mi1, mi2, mi3, mi4;

    public TestMenuBar(){
        f = new Frame("MenuBar Example");
        mb = new MenuBar();
        m1 = new Menu("File");
        m2 = new Menu("Edit");
        m3 = new Menu("Help");
        mi1 = new MenuItem("New");
        mi2 = new MenuItem("Save");
        mi3 = new MenuItem("Load");
        mi4 = new MenuItem("Quit");
    }

    public void launchFrame(){
        mi4.addActionListener(this);
        m1.add(mi1);
        m1.add(mi2);
        m1.add(mi3);
        m1.addSeparator();
        m1.add(mi4);

        mb.add(m1);
        mb.add(m2);
        mb.setHelpMenu(m3);
        f.setMenuBar(mb);

        f.addWindowListener(this);
        f.setSize(400,400);
        f.setBackground(Color.red);
        f.setLayout(null);
        f.setVisible(true);
    }

    public void actionPerformed(ActionEvent e){
        System.exit(0);
    }

    public void windowClosing(WindowEvent e){
        System.exit(0);
    }

    public void windowOpened(WindowEvent e){}
    public void windowIconified(WindowEvent e){}
    public void windowDeiconified(WindowEvent e){}
    public void windowClosed(WindowEvent e){}
    public void windowActivated(WindowEvent e){}
    public void windowDeactivated(WindowEvent e){}

    public static void main(String[] args){
        TestMenuBar tmb = new TestMenuBar();
        tmb.launchFrame();
    }
}
```

4.6 J2SE Event Model

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

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

public class EventHandle implements WindowListener, ActionListener{
  private Frame f;
  private Button b1, b2, b3;
  private TextField tf;

  public EventHandle(){
    f = new Frame("Button Handling");
    b1 = new Button("YES");
    b1.setActionCommand("yes button");
    b2 = new Button("NO");
    b2.setActionCommand("no button");
    b3 = new Button("Clear");
    b3.setActionCommand("clear button");
    tf = new TextField(30);
  }

  public void launchFrame(){
    b1.addActionListener(this);
    b1.setForeground(Color.white);
    b1.setBackground(Color.blue);

    b2.addActionListener(this);
    b2.setForeground(Color.red);
    b2.setBackground(Color.green);

    b3.addActionListener(this);
    b3.setForeground(Color.blue);
    b3.setBackground(Color.yellow);

    tf.setForeground(Color.blue);
    tf.setBackground(Color.white);

    f.setLayout(new FlowLayout());
    f.add(b1);
    f.add(b2);
    f.add(b3);
    f.add(tf);
    f.addWindowListener(this);
    f.setSize(250,150);
    f.setBackground(Color.red);
    f.setVisible(true);
  }

  public void actionPerformed(ActionEvent e){
    String str;
    if(e.getActionCommand()=="yes button"){
      str = "You press YES button";
      tf.setText(str);
    }
    if(e.getActionCommand()=="no button"){
      str = "You press NO button";
      tf.setText(str);
    }
    if(e.getActionCommand()=="clear button"){
      str = " ";
      tf.setText(str);
    }
  }

  public void windowClosing(WindowEvent e){
    System.exit(0);
  }

  public void windowOpened(WindowEvent e){}
  public void windowIconified(WindowEvent e){}
  public void windowDeiconified(WindowEvent e){}
  public void windowClosed(WindowEvent e){}
  public void windowActivated(WindowEvent e){}
  public void windowDeactivated(WindowEvent e){}

  public static void main(String[] args){
    EventHandle eh = new EventHandle();
    eh.launchFrame();
  }
}
```

4.6.1 Mouse Example

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

public class MouseExample implements WindowListener, MouseMotionListen
  private Frame f;
  private TextField tf;

  public MouseExample(){
    f = new Frame("Mouse Example");
    tf = new TextField(30);
  }

  public void launchFrame(){
    Label label = new Label("Click and drag the mouse");
    f.add(label, BorderLayout.NORTH);
    f.add(tf, BorderLayout.SOUTH);
    f.addMouseMotionListener(this);
    f.addMouseListener(this);
    f.addWindowListener(this);
    f.setSize(300,200);
    f.setVisible(true);
  }

  public void mouseDragged(MouseEvent e){
    String s = "Mouse dragged: X= "+e.getX()+" Y="+e.getY();
    tf.setText(s);
  }

  public void mouseEntered(MouseEvent e){
    String s = "The mouse entered";
    tf.setText(s);
  }

  public void mouseExited(MouseEvent e){
     String s = "The mouse has left the building";
     tf.setText(s);
  }

  public void mousePressed(MouseEvent e){ }
  public void mouseReleased(MouseEvent e){ }
  public void mouseMoved(MouseEvent e){ }
  public void mouseClicked(MouseEvent e){ }

  public void windowClosing(WindowEvent e){
    System.exit(0);
```

Table 4.1: Table Color class static constants and RGB values.

Color Constant	**Color**	**RGB value**
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 black	Black	0, 0, 0
public final static Color white	White	255, 255, 255
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

```
  }

  public void windowOpened(WindowEvent e){}
  public void windowIconified(WindowEvent e){}
  public void windowDeiconified(WindowEvent e){}
  public void windowClosed(WindowEvent e){}
  public void windowActivated(WindowEvent e){}
  public void windowDeactivated(WindowEvent e){}

  public static void main(String[] args){
    MouseExample me = new MouseExample();
    me.launchFrame();
  }
}
```

4.7 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 arguments that is an instance of 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.

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

```

public class TestColors implements WindowListener, ActionListener{
  private Frame f;
  private Button b;

  public TestColors(){
     f = new Frame("Frame Title");
     b = new Button("Change Color");
     b.setActionCommand("button press");
  }

  public void launchFrame(){
     b.addActionListener(this);
     b.setForeground(Color.red);
     b.setBackground(Color.yellow);
     f.add(b);
     f.addWindowListener(this);
     f.setSize(300,300);
     f.setBackground(Color.green);
     f.setLayout(new FlowLayout());
     f.setVisible(true);
  }

  public void actionPerformed(ActionEvent e){
     int x, y, z;
     if(e.getActionCommand()=="button press"){
         x = (int) (Math.random()*100);
         y = (int) (Math.random()*100);
         z = (int) (Math.random()*100);
         Color c = new Color(x, y, z);
         f.setBackground(c);
     }
  }

  public void windowClosing(WindowEvent e){
     System.exit(0);
  }

  public void windowOpened(WindowEvent e){}
  public void windowIconified(WindowEvent e){}
  public void windowDeiconified(WindowEvent e){}
  public void windowClosed(WindowEvent e){}
  public void windowActivated(WindowEvent e){}
  public void windowDeactivated(WindowEvent e){}

  public static void main(String[] args){
     TestColors tc = new TestColors();
     tc.launchFrame();
  }
}
```

4.8 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 manage 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 The FlowLayout is the default layout manager of Panel and Applet.
2. BorderLayout The BorderLayout is the default layout manager of Window, Dialog, and Frame.
3. Layout The GridLayout provides flexibility for placing components.
4. CardLayout The CardLayout provides functionality comparable to a primitive tabbed panel.
5. GridBagLayout -

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.

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

public class FlowExample implements WindowListener{
  private Frame f;
  private Button b1, b2, b3;

  public FlowExample(){
    f = new Frame("FlowLayout");
    b1 = new Button("Button 1");
    b2 = new Button("Button 2");
    b3 = new Button("Button 3");
  }

  public void launchFrame(){
    f.setLayout(new FlowLayout());
    // f.setLayout(new FlowLayout(FlowLayout.LEFT));
    // f.setLayout(new FlowLayout(FlowLayout.RIGHT));
    // f.setLayout(new FlowLayout(FlowLayout.CENTER));
    // f.setLayout(new FlowLayout(FlowLayout.RIGHT, 20, 30));
    f.add(b1);
    f.add(b2);
    f.add(b3);
    f.addWindowListener(this);
    f.setSize(250,150);
    f.setBackground(Color.red);
    f.setVisible(true);
  }

  public void windowClosing(WindowEvent e){
    System.exit(0);
  }

  public void windowOpened(WindowEvent e){}
  public void windowIconified(WindowEvent e){}
  public void windowDeiconified(WindowEvent e){}
  public void windowClosed(WindowEvent e){}
  public void windowActivated(WindowEvent e){}
  public void windowDeactivated(WindowEvent e){}

  public static void main(String[] args){
    FlowExample fe = new FlowExample();
    fe.launchFrame();
  }
}
```

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

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

public class BorderExample implements WindowListener{
  private Frame f;
  private Button b1, b2, b3, b4, b5;

  public BorderExample(){
    f = new Frame("FlowLayout");
    b1 = new Button("Button 1");
    b2 = new Button("Button 2");
    b3 = new Button("Button 3");
    b4 = new Button("Button 4");
    b5 = new Button("Button 5");
  }

  public void launchFrame(){
    f.add(b1, BorderLayout.NORTH);
    f.add(b2, BorderLayout.SOUTH);
    f.add(b3, BorderLayout.WEST);
    f.add(b4, BorderLayout.EAST);
    f.add(b5, BorderLayout.CENTER);
    f.addWindowListener(this);
    f.setSize(250,250);
    f.setBackground(Color.red);
    f.setVisible(true);
  }

  public void windowClosing(WindowEvent e){
    System.exit(0);
  }

  public void windowOpened(WindowEvent e){}
  public void windowIconified(WindowEvent e){}
  public void windowDeiconified(WindowEvent e){}
  public void windowClosed(WindowEvent e){}
  public void windowActivated(WindowEvent e){}
  public void windowDeactivated(WindowEvent e){}

  public static void main(String[] args){
    BorderExample be = new BorderExample();
    be.launchFrame();
  }
}
```

The GridLayout manager placing components with a number of rows and columns. The following constructor creates a GridLayout with the specified equal size. new GridLayout(int rows, int cols);

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

public class GridExample implements WindowListener{
  private Frame f;
  private Button b1, b2, b3, b4, b5, b6;

  public GridExample(){
    f = new Frame(”FlowLayout”);
    b1 = new Button(”Button 1”);
    b2 = new Button(”Button 2”);
    b3 = new Button(”Button 3”);
    b4 = new Button(”Button 4”);
    b5 = new Button(”Button 5”);
    b6 = new Button(”Button 6”);
  }

  public void launchFrame(){
    f.setLayout(new GridLayout(3,2));
    f.add(b1);
    f.add(b2);
    f.add(b3);
    f.add(b4);
    f.add(b5);
    f.add(b6);
    f.addWindowListener(this);
    f.setSize(300,300);
    f.setBackground(Color.red);
    f.setVisible(true);
  }

  public void windowClosing(WindowEvent e){
    System.exit(0);
  }

  public void windowOpened(WindowEvent e){}
  public void windowIconified(WindowEvent e){}
  public void windowDeiconified(WindowEvent e){}
  public void windowClosed(WindowEvent e){}
  public void windowActivated(WindowEvent e){}
  public void windowDeactivated(WindowEvent e){}

  public static void main(String[] args){
    GridExample ge = new GridExample();
    ge.launchFrame();
  }
}
```

4.9 Java Foundation Classes/ Swing Technology

The Java Foundation Classes (J.F.C.)/ Swing technology is a second-generation GUI toolkit that is included in the Java 2 SDK as a standard extension. Swing technology has many improvement over AWT and adds many new and more-complex components including a table and tree component.

4.9.1 Fonts in Java

In Java, Font class manages the font of Strings. The constructor of Font class takes three 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.

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

public class TestFonts extends JFrame{
  private static final long serialVersionUID=0;

  public TestFonts(){
    super("Font Examples");
    setSize(300, 150);
    show();
  }

  public void paint(Graphics g){
    g.drawString("Welcome to Java.", 20, 50);
    g.setColor(Color.red);
    g.setFont(new Font("Monospaced", Font.BOLD, 14));
    g.drawString("Welcome to Java.", 20, 70);
    g.setColor(Color.blue);
    g.setFont(new Font("SansSerif", Font.BOLD + Font.ITALIC, 16));
    g.drawString("Welcome to Java.", 20, 90);
  }

  public static void main(String[] args){
    TestFonts tf = new TestFonts();
    tf.addWindowListener(
       new WindowAdapter(){
          public void windowClosing(WindowEvent e){
             System.exit(0);
```

```
            }
          }
       );
    }
}
```

4.9.2 Drawing Lines, Rectangles, and Ovals

Code 4-15 presents a variety if Graphics methods for drawing lines, rectangles, and ovals.

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

public class TestLineRectOval extends JFrame{
  private static final long serialVersionUID=0;

  public TestLineRectOval(){
    super("Lines Ractangles Ovals");
    setSize(360, 450);
    show();
  }

  public void paint(Graphics g){
    g.setColor(Color.red);
    g.drawLine(50,40, 300, 40);

    g.setColor(Color.blue);
    g.drawRect(50,60, 100, 60);
    g.fillRect(200,60, 100, 60);

    g.setColor(Color.cyan);
    g.drawRoundRect(50,150, 100, 60, 50, 50);
    g.fillRoundRect(200,150, 100, 60, 50, 50);

    g.setColor(Color.yellow);
    g.draw3DRect(50,250, 100, 60, true);
    g.fill3DRect(200,250, 100, 60, true);

    g.setColor(Color.magenta);
    g.drawOval(50,350, 100, 60);
    g.fillOval(200,350, 100, 60);
  }

  public static void main(String[] args){
    TestLineRectOval tlro = new TestLineRectOval();
    tlro.addWindowListener(
      new WindowAdapter(){
        public void windowClosing(WindowEvent e){
          System.exit(0);
        }
      }
    );
  }
}
```

Code 4-16 provides an password example using Jlabel, JtextField, JpasswordField, and Jbutton.

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

public class Password extends JFrame{
  private static final long serialVersionUID=0;

  private JLabel l1, l2;
  private JTextField t1;
  private JPasswordField pw;
  private JButton b1, b2;

  public Password(){
    super("Password Example");
    Container c = getContentPane();
    c.setLayout(new FlowLayout());

    l1 = new JLabel("Enter User Name   ");
    c.add(l1);

    t1 = new JTextField(15);
    c.add(t1);

    l2 = new JLabel("Enter Password      ");
    c.add(l2);

    pw = new JPasswordField(10);
    c.add(pw);

    b1 = new JButton("Enter");
    b1.addActionListener(
      new ActionListener(){
        public void actionPerformed(ActionEvent e){
          if(t1.getText().equals("farid") && pw.getText().equals("1234
             JOptionPane.showMessageDialog(null, "Welcome to Java");
          }else{
             JOptionPane.showMessageDialog(null, "Incorrect user name
          }
        }
      }
    );
    c.add(b1);

    b2 = new JButton("Cancel");
    b2.addActionListener(
      new ActionListener(){
        public void actionPerformed(ActionEvent e){
           String s = "";
           t1.setText(s);
           pw.setText(s);
        }
      }
    );
    c.add(b2);

    setSize(300, 125);
    show();
  }

  public static void main(String[] args){
    Password PW = new Password();
    PW.addWindowListener(
      new WindowAdapter(){
        public void windowClosing(WindowEvent e){
           System.exit(0);
        }
      }
    );
  }
}
```

Example of Jcomponents such as: JcheckBox, JradioButton, JcomboBox, Jlist, and JTextArea provided in code 4-17.

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

public class TestJComt extends JFrame{
  private static final long serialVersionUID=0;

  private JCheckBox bold, italic;
  private JRadioButton male, female;
  private ButtonGroup radioGroup;
  private JComboBox cBox;
  private String[] str1 = {"spring", "summer", "fall"};
  private JList colorList;
  private String[] str2 = {"Red", "Green", "Blue", "Black", "White", "
                          "Orange", "Pink", "Magenta", "Sky", "Cya
  private JTextArea ta1;

  public TestJComt(){
    super("Test JComponents");
    Container c = getContentPane();
    c.setLayout(new FlowLayout());

    bold = new JCheckBox("Bold");
    c.add(bold);
    italic = new JCheckBox("Italic");
    c.add(italic);

    male = new JRadioButton("Male");
    c.add(male);
    female = new JRadioButton("Female");
    c.add(female);
    radioGroup = new ButtonGroup();
    radioGroup.add(male);
    radioGroup.add(female);

    cBox = new JComboBox(str1);
    c.add(cBox);

    colorList = new JList(str2);
    colorList.setVisibleRowCount(5);
    c.add(new JScrollPane(colorList));

    String s = "Java is a object oriented programming language";
    ta1 = new JTextArea(s, 10, 15);
    c.add(new JScrollPane(ta1));

    setSize(200, 350);
    show();
  }

  public static void main(String[] args){
    TestJComt jc = new TestJComt();
    jc.addWindowListener(
      new WindowAdapter(){
        public void windowClosing(WindowEvent e){
          System.exit(0);
        }
      }
    );
  }
}
```

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

public class OpenFile extends JFrame{
  private static final long serialVersionUID=0;
  private JButton b1;
  private JFileChooser jfc;

  public OpenFile(){
    super("File Opener");
    b1 = new JButton("Open File Chooser");
    b1.addActionListener(
      new ActionListener(){
        public void actionPerformed(ActionEvent e){
          abc();
        }
      }
    );
    getContentPane().add(b1, BorderLayout.NORTH);
    setSize(300, 200);
    show();
  }

  private void abc(){
    jfc = new JFileChooser();
```

```
      jfc.showOpenDialog(this);
    }

    public static void main(String[] args){
      OpenFile of = new OpenFile();
      of.addWindowListener(
         new WindowAdapter(){
            public void windowClosing(WindowEvent e){
               System.exit(0);
            }
         }
      );
    }
}
```

Chapter 5

Threads, Sockets, and Collections API

5.1 Java Threads

Modern computer performs multiple jobs at the same time. Thread is the process of multi-tasking using CPU, which performs computations. A thread or execution context is composed of three main parts:

1. A virtual CPU.
2. The code that the CPU executes.
3. The data on which the code work.

The class java.lang.Thread enables us to create and control threads. A process is a program in execution. One or more threads a process. A thread is composed of CPU, code, and data. Code can share multiple threads, two threads can share the same code. We can create thread by implementing Runnable interface (Code 5-1) or by extending Thread class (Code 5-2). Thread class implements the Runnable interface itself. The Runnable interface provides the public void run() method. We override the run() method, which contain the code for CPU execution.

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

```
public class ThreadTest implements Runnable{
  public void run(){
    int i=1;
    while(i<=100){
       System.out.println("i: "+i);
       i++;
    }
  }

  public static void main(String[] args){
    ThreadTest t = new ThreadTest();
    Thread thread1 = new Thread(t);
    Thread thread2 = new Thread(t);
    Thread thread3 = new Thread(t);
    thread1.start();
    thread2.start();
    thread3.start();
  }
}
```

```
public class MyThread extends Thread{
  public void run(){
    int i=1;
    while(i<=100){
       System.out.println("i: "+i);
       i++;
    }
  }

  public static void main(String[] args){
    Thread thread1 = new MyThread();
    Thread thread2 = new MyThread();
    Thread thread3 = new MyThread();
    thread1.start();
    thread2.start();
    thread3.start();
  }
}
```

The sleep() is a static method in the Thread class, it operates on the current thread and is referred to as Thread.sleep(x); where x is the minimum number of millisecond.

```
public class ThreadSleep implements Runnable{
  public void run(){
    int i=1;
    while(i<=10){
      System.out.println("i: "+i);
      i++;
      try{
         Thread.sleep(300);
      }catch(InterruptedException e){
        System.out.println(e);
      }
    }
  }

  public static void main(String[] args){
    ThreadSleep ts = new ThreadSleep();
    Thread thread1 = new Thread(ts);
    Thread thread2 = new Thread(ts);
    Thread thread3 = new Thread(ts);
    thread1.start();
    thread2.start();
    thread3.start();
  }
}
```

Code: The StackTest.java application

```
class MyStack{
  private int idx=0;
  private char[] data = new char[6];

  public synchronized void push(char c){
    this.notify();
    if(idx!=5){
       data[idx]=c;
       idx++;
    }
  }

  public synchronized char pop(){
    if(idx==0){
       try{
          this.wait();
       }catch(InterruptedException e){
          System.out.println(e);
       }
    }
    idx--;
    return data[idx];
  }
}

class Producer implements Runnable{
  private MyStack stack;

  public Producer(MyStack s){
    stack = s;
  }

  public void run(){
    char c;
    for(int i=0; i<50; i++){
        c = (char) (Math.random()*26+'A');
        stack.push(c);
        System.out.println("Producer: "+c);
        try{
            Thread.sleep((int)(Math.random()*300));
        }catch(InterruptedException e){
            System.out.println(e);
        }
    }
  }
}

class Consumer implements Runnable{
  private MyStack stack;

  public Consumer(MyStack s){
    stack = s;
  }

  public void run(){
    char c;
    for(int i=0; i<50; i++){
        c = stack.pop();
        System.out.println("Consumer: "+c);
        try{
            Thread.sleep((int)(Math.random()*300));
        }catch(InterruptedException e){
            System.out.println(e);
        }
    }
  }
}

public class StackTest{
  public static void main(String[] args){
    MyStack s = new MyStack();
    Producer p = new Producer(s);
    Thread t1 = new Thread(p);
    t1.start();
    Consumer c = new Consumer(s);
    Thread t2 = new Thread(c);
    t2.start();
  }
}
```

5.2 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.

```
import java.util.*;

public class SetExample{
  public static void main(String[] agrs){
    Set set = new HashSet();
    set.add("One");
    set.add("2nd");
    set.add("3rd");
    set.add(new Integer(6));
    set.add(new Float(7.7F));
    set.add("2nd"); // duplicate are not added
    System.out.println(set);
  }
}
```

```
import java.util.*;

public class ListExample{
  public static void main(String[] agrs){
    List list = new ArrayList();
    list.add("One");
    list.add("2nd");
    list.add("3rd");
    list.add(new Integer(6));
```

```
      list.add(new Float(7.7F));
      list.add("2nd");  // duplicate is added
      System.out.println(list);
    }
}
```

5.3 Sockets or 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.

Code 5-7: The HostServer.java application

```
import java.net.*;
import java.io.*;

public class HostServer{
  public static void main(String[] args){
    ServerSocket s = null;
    try{
      s = new ServerSocket(5432);
      Socket s1 = s.accept();
      OutputStream s1out = s1.getOutputStream();
      BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(s1
      bw.write("Hellow Net World");
      bw.close();
      s1.close();
    }catch(IOException e){
      e.printStackTrace();
    }
  }
}
```

```
import java.net.*;
import java.io.*;

public class ClientServer{
  public static void main(String[] args){
    try{
       Socket s1 = new Socket("127.0.0.1", 5432);
       InputStream is = s1.getInputStream();
```

```
            DataInputStream dis = new DataInputStream(is);
            System.out.println(dis.readUTF());
            dis.close();
            s1.close();
        }catch(ConnectException e1){
            System.out.println(e1);
        }catch(IOException e2){
            e2.printStackTrace();
        }
    }
}
```