Programming in Java Dr. M. T. Wanjari

**Unit – I**

**Java and Internet**

In recent years, Java has become popular language for the programs that are required to run on different systems. Java’s new innovation named “applet” has completely changed the Internet Programming. Applets are tiny programs that are designed in such a way that they can be transmitted over the Internet. They can be downloaded on demand and executed automatically by Java compatible web browser. They are used to handle user input, data supplied by the server and simple functions that executes locally on the client machine.

Applet is dynamic, self-executing program and is intelligent enough that can change it with the user inputs. The dynamic programs when downloaded and executed can cause serious harms to the computer as it may contain viruses like Trojan horse and malicious programs. These harmful programs may search the contents on local file system of the client computer and may gather private information like credit card numbers, passwords etc. Earlier viruses were scanned before executing the downloaded program, but Java has resolved the issue by confining the Java programs to Java execution environment only.

**Features of Java:**

Java has become a popular language for Internet applications because of various features which are as follows.

**Simple, Small and Familiar:** Java is a simple, small and familiar language. As java inherits the syntax of C/C++ and many of the OOPs features of C++ thus, one who is familiar with the concept of object oriented language can learn java with least effort. Moreover Java omits the complex and unreliable code of C and C++ like operator overloading, pointers, preprocessor header files. Java provides small and convenient way to accomplish a given task.

**Compiled and Interpreted:** Unlike other languages java uses two stage system as it uses both compiler and interpreter for its program execution. First, the compiler converts the program code to bytecode which in turn is converted to machine code on any machine using the interpreter. The machine code so generated can be executed irrespective of the system on which it is being executed.

**Platform-Independent and Portable:** This feature makes java language very special. Java programs can run on any platform, that is, they can run on different CPU and on different

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 1

Programming in Java Dr. M. T. Wanjari

operating system architectures. The bytecode produced by Java compiler can be run on any machine which has Java runtime environment.

**Object oriented language:** Java is an object oriented language as everything in Java is an object. The objects and classes contain the program code and data. The java object model is easily extensible and classes can be used anywhere in the program in the form of packages. **Robust and Secure:** Java is a robust language because of mainly two reasons. Firstly, it is strictly typed language that checks the code at the compile time. Secondly, Java does memory management in an effective way. In C++, the programmer has to manually deallocate the dynamic memory used by the objects. Java automatically deallocates free memory that is no longer referenced by the objects (with the help of garbage collector).

Since Java is a programming language that is used for programming on Internet, security becomes an important issue. Java systems ensure that no viruses are communicated with an applet.

**Distributed:** Since java is a platform independent, it suitable for developing applications for the networks. Java can handle TCP/IP protocols and hence applications developed in java can access remote objects on Internet like any object on a local system.

**Multithreaded and Interactive:** Java supports multithreaded programming which allows us to write a program that can perform more than one task simultaneously. Users need not wait for the program to finish one task, before starting another. For example, users can listen to an audio clip while downloading the applet. This feature helps to improve the performance of graphical applications. Java also supports multi-process synchronization and smoothly running interactive system.

**High Performance:** as stated earlier, the java program is converted to bytecode which is then converted to machine code using interpreter. Since bytecode is highly optimized, it enables the JVM to execute programs at much faster rate.

**Dynamic and Extensible:** It is dynamic in nature that is Java programs can link to new class libraries, objects, methods etc. at the runtime. Java language also provides the facility to include the functions of other languages like C and C++. They are referred to as native methods. These methods are also linked dynamically at run-time.

**Java Environment:**

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 2

Programming in Java Dr. M. T. Wanjari

In most of the programming languages, the program is converted to machine code either by using compiler or interpreter. The machine code so generated is machine dependent, that is, it may not run on the machine other than the one on which it is generated. Unlike in case of other programming languages, the Java compiler does not convert source code to machine code, it converts source code to a special intermediate code known as bytecode (fig. 1.1). The bytecode so generated is in the form of class files that can be interpreted. The command used for compilation in java is javac which converts the corresponding Java files into class file. The bytecode is machine independent that is it can be run on any machine with the help of Java virtual machine (JVM).

Compiler

Source Code Bytecode

.java file .class file

**Fig. 1.1 Compilation of a Java Program**

**i) JDK (Java Development Kit):**

JDK consists of various tools that are used to develop and execute Java programs. The tools included in JDK are listed in Table 1.1

**Table 1.1 Tools in Java Development Kit**

|  |  |
| --- | --- |
| **Tools** | **Functions** |
| Javac | Java compiler that converts source code to Java bytecode |
| Java | Java interpreter that interprets class files generated by java compiler and converts it to machine code |
| Javadoc | Document generator which automatically generates documentation from source code |
| Javah | Generates C headers and stub generator used for writing native methods |
| Javap | The class file disassembler which enables to convert bytecode files to a program description |
| Jdb | Java debugger which helps in tracking errors in the program |
| Appleteviewer | Used for running and debugging Java applets without a web browser |

**ii) Java Standard Library**

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 3

Programming in Java Dr. M. T. Wanjari

The Java standard library (or Application Programming Interface) is composed of various classes and methods which are grouped into several packages. some of the commonly used packages include the following.

**Language Support Package:** It contains classes and methods which are required to implement the basic features of Java.

**Input/Output Package:** It contains classes which are required for Input/Output manipulation. **Networking Package:** It contains classes that allows communication between computers through Internet.

**Utilities Package:** It contains classes that provide utility functions such as date and time functions.

**Applet Package:** It contains a set of classes for creating Java applets.

**AWT Package:** It contains classes for implementing graphical user interface. **Java Virtual Machine:**

Java uses both the compiler and Interpreter. Source code written in Java is compiled to generate bytecode and then this bytecode is interpreted to machine instructions for specific machine. It is generated for the virtual machine that exists only inside the computer memory known as Java Virtual Machine (JVM). This virtual machine is designed in such a way that it can be implemented on the top of any existing processor and itself acts as virtual processor clip. It hides the underlying operating system details from Java applications. The process of converting source code to machine code is shown in fig. 1.2

Compiler Interpreter

Source Bytecode Machine

JVM

**Fig. 1.2 Conversion of Source Code to Machine Code**

JVM obtains the bytecode stream from the .class file in the form of series of instruction. Each instruction consists of a one-byte opcode (a valid and understandable command which tells the JVM what to do) and zero or more operands, which are required to complete the opcode.

Some of the main components of JVM are stack, registers, garbage-collection heap and method area.

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 4

Programming in Java Dr. M. T. Wanjari

**Stack:** JVM stacks are similar to the stacks used in C or other conventional languages. It consists of frames, which holds the state of Java method invocation. It holds the local variables, parameters, its return value and the intermediate results. JVM stack cannot be manipulated directly and only pushing and popping of frames can take place. When a method is invoked, the JVM pushes a new frame into the stack and after the completion of execution of the method the frame is popped and discarded by the virtual machine. The size of the JVM stack can be fixed or may dynamically increase and decrease according to the requirement.

**Registers:** JVM uses different types of registers to control the stacks. Some of the important JVM registers are:

• **Optop:** It points to the top of operand stack.

• **Frame:** It points to the current execution environment.

• **Vars:** It points to the local variables.

• **Program counter:** It keeps record of the next instruction in a program to be executed. **Garbage Collection heap:** Heap is a free memory space used for allocations to the variables like arrays, class instances etc.,at runtime. Whenever memory is allocated with the help of new operator, that memory space is allocated from the heap. Heap provides an automatic storage management system known as garbage-collection heap. Whenever there are objects which are no longer referenced, then the runtime environment automatically reclaims the memory occupied by the objects. Size of the heap may be fixed or may increase or decrease according to the requirement.

**Method area:** Method area is a storage area for the compiled code. Although the method area is a part of heap, there is no specific area where it exists. This feature makes JVM more portable and secure. Size of the method area may be fixed or may increase or decrease according to the requirement. Method area is used to store:

• Runtime constant pool

• Field information

• Method information

• Code for methods and constructors

• Special methods that are used in class and instance initialization

• Interface type initialization

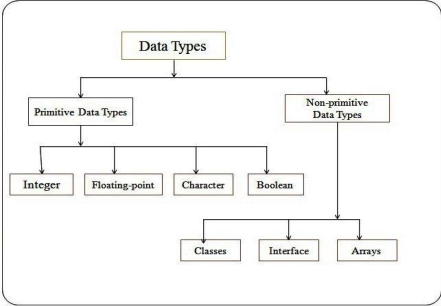
M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 5

Programming in Java Dr. M. T. Wanjari

The primitive data types that can be handled by the JVM are byte(8 bits), char(16 bits), short(16 bits), int (32 bits), float(32 bits), long(64 bits), double(64 bits).

**Data types:**

A data type determines the type and the operations that can be performed on the data. Java provides various data types and each data type is represented differently within the computer’s memory. The type of data selected by a programmer depends on the particular application. The various data types provided by Java are categorized into primitive data types and non-primitive data types shown in fig. 1.3.

**Fig. 1.3 Data Types**

**1. Primitive Data Types**

Primitive data types also known as built-in data types are the fundamental data types provided by a programming language. In Java, primitive data types include integer, floating point, character and Boolean.

**Integer Type**

The integer data type is used to store integers like 4, 42, 5233, -32, -745. Java supports four types of integers namely, byte, short, int and long. The default value of these integer types is

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 6

Programming in Java Dr. M. T. Wanjari

0. There is no concept of unsigned integer in Java. The various integer data types with their size and range are listed in Table 1.2.

**Table 1.2 Size and Range of Integer Data Types**

|  |  |  |
| --- | --- | --- |
| **Type** | **Size (bytes)** | **Range** |
| Byte | One | -128 to 127 |
| Short | Two | -32,768 to 32,767 |
| Int | Four | -2,147,483,648 to 2,147,483,647 |
| Long | Eight | -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 |

**Floating Point Type**

A floating point type is used to store real numbers such as 3.28,64.755765,-8.01,24.53. Java supports two floating point data types, namely, float and double.

**Float:** The float type represents a single-precision number. Single precision occupies lesser space than double precision but becomes inaccurate when the values are large. For example, it can be used to represent value of marks of the students. The default value of float data type is 0.0f.

**Double:** The double type specifies a double-precision number. It is the best choice when you need to store large-valued numbers. For example, it can be used when you want to use mathematical functions like sin(), cos(), sqr(). The default value of double data type is 0.0d.

The various floating point data types with their size and range are listed in Table 1.4 **Table 1.3 Size and Range of Floating Point Data Types**

|  |  |  |
| --- | --- | --- |
| **Type** | **Size(bytes)** | **Range** |
| Float | Four | 3.4e-038 to 3.4e+038 |
| double | Eight | 1.7e-308 to 1.7e+308 |

The character data type is used to store single character enclosed in single quotes. It is represented by using char keyword. It occupies 16-bit memory. The range of the character data type is 0 to 65,536. The default value of char data type is null character.

**Boolean Type:** The Boolean date type can hold Boolean values, that is, either true or false. The keyword Boolean is used to denote the Boolean data type. The default value of Boolean data type is false.

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 7

Programming in Java Dr. M. T. Wanjari

**Non-Primitive Data Types:**

Non-Primitive data types (user defined data types) also known a reference type are derived from the primitive data types. In Java, these include classes, interfaces and array. **Variables:**

A variable is an identifier that represents a memory location that is used to store data value. Data stored at a particular location can be accessed by using the variable name. The value of a variable can be changed anytime during the program execution. The variable name that you choose must be meaningful so as to understand what it represents in the program.

**Declaring Variables:**

Variables must be declared in a program before they are used. The declaration of a variable informs the compiler, the specific data type to which a variable is associated and allocates sufficient memory for it.

The syntax for declaring a variables is as follows:

data\_type variable\_name;

for example, a variable of a type int can be declared by using this statement. int a;

At the time of the variable declaration, more than one variable of the same data type can be declared in a single statement. for example, consider the following statement. int x, y, z;

**Initializing Variables:**

Declaration of variables allocates memory for variables but it does not store any data at the time of declaration. To store data in the variables, they need to be initialized. For example, consider the following statements:

int i;

i=10;

Here, a variable i of the integer type is declared and the value 10 is assigned to it. You can combine both the statements into a single statement as follows:

int i=10;

Besides initializing the variable with the constant values, variables can also be initialized at run-time using expressions. Initialization of variables at run-time is known as dynamic initialization.

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 8

Programming in Java Dr. M. T. Wanjari

**Example: A program to demonstrate initialization of variable**

public class dynamic\_initialization

{

public static void main(String[] args)

{

int x=40, y=40, z=10; //initialization with constant values

int result =(x\*y)+z; //dynamic initialization

System.out.println(“The value of z is:”+result);

}

}

**The output of the program is:**

The value of the z is: 1610

**Receiving input through keyboard**

Variables can also be given values interactively through the keyboard by using the readLine() method.

**Example: a program to demonstrate reading data from the keyboard** // importing package for using DataInputStream class

import java.io.\*;

public class ReadingData

{

public static void main(String[] args)

{

DataInputStream in=new DataInputStream(System.in);

int num1=0;

float num2=0;

try

{

System.out.println(“Enter integer value”);

num1=Integer.parseInt(in.readLine());

System.out.println(“Enter float value”);

num2=Float.valueOf(in.readLine()),floatValue();

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 9

Programming in Java Dr. M. T. Wanjari

}

catch(Exception e)

{

System.out.println(“The integer value is ”+num1);

System.out.println(“The float value is ”+num2);

}

}

**The output of the program is:**

Enter integer value

4

Enter float value

6.7

The integer value is 4

The float value is 6.7

The method readLine() of class DataInputStream is used to read string from the keyboard which is then converted to the corresponding data type, int and float. To handle the error which may occur while reading data from the keyboard, you have been provided with try and catch statements.

**Operators:**

Operators are the symbols, which perform operations on various data items known as operands. For example, in a+b, a and b are operands and + sign is an operator. Note that to perform an operation , operators and operands are combined together forming an expression. For example, to perform an addition operation on operands a and b, the addition (+) operator is combined with the operands a and b forming expression.

Depending on the function performed, the Java operators can be classified into various categories. These include arithmetic operators, increment and decrement operators, relational operators, logical operators, conditional operator, assignment operators, bitwise operators and special operators.

**1. Arithmetic Operators**

Arithmetic operators perform the basic arithmetic operations on operands. They can work on any built-in data type of Java except on boolean type.

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 10

Programming in Java Dr. M. T. Wanjari

Java provides various arithmetic operators that are. +(addition or unary plus), - (subtraction or unary minus), \*(multiplication), /(division) and %(modulus). For example, some of the expressions which involve arithmetic operators are x\_y, x-y, x\*y, x/y and x%y. when the unary minus operator is used with a single operand, the operand is multiplied by -1.

Expression formed by using arithmetic operators can be of following type: • **Integer expression:** The arithmetic expression where both the operands are integers is called an integer expression. The result of the integer arithmetic is always an integer. • **Real expression:** The arithmetic expression where both the operands are real is called real expression.

• **Mixed mode expression:** The expression is mixed mode if one operand is real and the other is integer. In this case, the integer operand is converted to real and the result is also of type real.

**Example: A program to demonstrate arithmetic operators.**

class ArithmeticOperators

{

public static void main(String[] args)

{

int a=10;

int b=3;

float c=5;

float d=2;

System.out.println(“Integer Arithmetic”);

System.out.println(“a+b=”+(a+b));

System.out.println(“a-b=”+(a-b));

System.out.println(“a\*b=”+(a\*b));

System.out.println(“a/b=”+(a/b));

System.out.println(“a%b=”+(a%b));

System.out.println(“Real Arithmetic”);

System.out.println(“c+d=”+(c+d));

System.out.println(“c-d=”+(c-d));

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 11

Programming in Java Dr. M. T. Wanjari

System.out.println(“c\*d=”+(c\*d));

System.out.println(“c/d=”+(c/d));

System.out.println(“c%d=”+(c%d));

System.out.println(“Mixed-mode Arithmetic”); System.out.println(“a+c=”+(a+c));

System.out.println(“a-c=”+(a-c));

System.out.println(“a\*c=”+(a\*c));

System.out.println(“a/c=”+(a/c));

System.out.println(“a%c=”+(a%c)); **The output of the program is**

Integer Arithmetic

a+b = 13

a-b = 7

a\*b = 30

a/b = 3

a%b = 1

Real Arithmetic

c+d = 7.0

c-d = 3.0

c\*d = 10.0

c/d = 2.5

c%d = 1.0

Mixed-mode Arithmetic

a+c = 15.0

a-c = 5.0

a\*c = 50.0

a/c = 2.0

a%c = 0.0

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 12

Programming in Java Dr. M. T. Wanjari

**2. Increment and Decrement Operators**

Java provides special unary arithmetic operators, namely the increment operator (represented by ++) and the decrement operator (represented by --). The operator ++ increases the value of operand by 1 and the operator – decreases the value of operand by 1. The increment and decrement operators can be used in two forms.

• **Prefix form:** in this form, the increment or the decrement operator precedes its operand. The prefix increment operator is represented as ++ operand and the prefix decrement operator is represented as \_\_ operand. The prefix increment or the prefix decrement increments or decrements the value of an operand respectively before its value is used in an expression.

Example: Evaluate the statements

x=10;

y=++x;

z=-x;

In this example, the statement y=++x first increments the value of x by 1 and then assigns the incremented value to y. Similarly, decrement operator is used to decrement the value of x by 1 and then assigns the decremented value to z. Thus, the value of y and z are 11 and 10, respectively.

• **Postfix form:** In this form, the increment or the decrement operator succeeds its operand. The postfix increment operator is represented as operand ++, and the postfix decrement operator is represented as operand --. The postfix increment or the decrement operator increments or decrements the value of an operand respectively after using it in the expression.

**Example: Evaluate these statements**

x=10;

y=x++;

z=x--;

In this example, the statement y=x++ first assign the value of x to y and then increments the value of x by 1. Similarly, the statement z=x—assigns the value of x to z and then decrements the value of x by 1. Thus, the value of y and z are 10 and 11, respectively.

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 13

Programming in Java Dr. M. T. Wanjari

**3. Relational Operators**

Relational operators are used for comparing two values or expressions. The various relational operators provided by Java are less that ‘<’, less than or equal to ‘<=’, greater than ‘>’, greater than or equal to ‘>=’, equal to ‘==’ and not equal to ‘!=’ operator. They return values of boolean type, that is, either true or false. For example, consider two variables a and b having values 20 and 30, respectively. In this case, the expression a<b returns true whereas the expression a>b return false.

The operators = = and ! = are also known as equality operators as they are used for checking the equality of operands.

**Example: A program to demonstrate relational operators**

class RelationalOperators

{

Public static void main(String[] args)

{

int x=20, y=40, z=20;

System.out.println(“The value of x is ”+x);

System.out.println(“The value of y is ”+y);

System.out.println(“The value of z is ”+z);

System.out.println(“ ”);

System.out.println(“The result of x<y is ”+(x<y));

System.out.println(“The result of x>y is ”+(x>y));

System.out.println(“The result of x==z is ”+(x==z));

System.out.println(“The result of x<=z is ”+(x<=z));

System.out.println(“The result of x>=y is ”+(x>=y));

System.out.println(“The result of y!=z is ”+(y!=z));

System.out.println(“The result of y==x+z is ”+(y==x+z));

}

}

**The output of the program is**

The value of x is 20

The value of y is 40

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 14

Programming in Java Dr. M. T. Wanjari

The value of z is 20

The result of x<y is true

The result of x>y is false

The result of x==z is true

The result of x<=z is true

The result of x>=y is false

The result of y!=z is true

The result of y= =x+z is true

**4. Logical Operators**

Logical operators combine expressions and then return true or false. The various logical operators provided by Java are listed as follows:

• **AND (&&) operator:** It returns true only if all the expressions evaluate to true, otherwise it return false.

• **OR (||) operator:** It returns true if any one or all the expressions evaluate to true and returns false only if all the expressions evaluate to false.

• **Negation (!) operator:** It returns true if the expression on which it is operating is false and vice versa.

For example, consider two expressions Exp1 and Exp2. Table1.4 shows the result after the logical operators are applied in these expressions.

**Table 1.4 Truth Table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Exp1 | Exp2 | Exp1 && Exp2 | Exp1 OR Exp2 | !Exp1 |
| True | True | true | true | false |
| True | False | false | true | false |
| False | True | false | true | true |
| False | False | false | false | true |

**Example: A program to demonstrate logical operators**

Class LogicalOperators

{

public static void main(String[] args)

{

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 15

Programming in Java Dr. M. T. Wanjari

Float a=10.0f, b=15.5f, c=30.0f;

System.out.println(“a= ”+a);

System.out.println(“b= ”+b);

System.out.println(“c= ”+c );

System.out.println(“ ”);

System.out.println(“a<b && a==c returns ”+(a<b&&a==c) );

System.out.println(“a<b && a!=c returns ”+(a<b&&a!=c) );

System.out.println(“a<b || a==c returns ”+(a<b||a==c) );

System.out.println(“a<b || a!=c returns ”+(a<b||a!=c) );

System.out.println(“!a<b && a==c) returns ”+!(a<b && a==c) );

}

}

**The output of the program is**

a= 10.0

b= 15.5

c= 30.0

a<b && a==c returns false

a<b && a!=c returns true

a<b || a==c returns true

a<b || a!=c returns true

!(a<b && a==c) returns true

**5. Conditional Operator**

The conditional operator selects a value based on specified condition. Note that the conditional operator is a ternary operator, that is, this operator involves three operands. The syntax of the conditional operator is

expression1 ? expression2:expression3

If expression1 is true, then Expression2 is evaluated, otherwise expression3 is evaluated. Example: Evaluate this statement

(x==5) ? 8 : 9;

In this example, if the value of x is equal to 5, then the expression returns 8, otherwise the expression returns 9.

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 16

Programming in Java Dr. M. T. Wanjari

**6. Assignment Operators**

Assignment operator assigns the value of an expression to a variable. Assignment operators are of two types, namely, the simple assignment operator and compound assignment operators.

**Simple Assignment Operator:**

The simple assignment operator assigns the value on its right hand side to the variable on its left hand side. Note that the left hand side of an assignment expression should be a variable. It cannot be a constant or an expression. However, the right hand side of an assignment expression can be a variable, constant or an expression.

Example: Evaluate the statement

x=8;

In this example, the value 8 is assigned to the variable x.

With the help of the assignment operator, several variables can be assigned a common value. This is accomplished by using multiple assignments in a single statement. for example, in the statement x=y=z=5, the value 5 is assigned to the three variable x, y and z. **Compound Assignment Operators**

Java provides compound assignment operators (also known as Java shorthands), which v op=exp;

Here, v is a variable, op is the binary operator and exp is an expression. This form is equivalent to the statement v= v op(exp);

where you need to access v only once.

For example, the expression x=x+6 can be written as x+=6. In this expression, x is incremented by 6 and then the result is assigned to x. The various compound assignment operators used in Java are ‘+=’,’-=’,’\*=’,’/=’ and ‘%=’.

**7. Other** O**perators**

In addition to the operator discussed, Java supports some other operators which include bitwise operators and special operators.

**Bitwise Operators:**

The bitwise operators are used to manipulate the data values at bit level. This operator can be applied to all the primitive data types such as long, int, short, char and byte. Various

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 17

Programming in Java Dr. M. T. Wanjari

bitwise operators are bitwise AND(&), bitwise OR(!), bitwise exclusive OR(^), one’s complement(~), shift left(<<), shift right(>>), shift right with zero fill(>>>). **Special Operators**

Java provides two special operators namely, instanceof and dot operator(.). **Instanceof Operator**

The instanceof operator is an operator which is used to check whether the object belongs to a particular class or not. For example, consider this statement.

**novel instanceof book**

This statement return true if the object novel belongs the class book, otherwise, false. **Dot Operator:**

The instance variable and the methods of a class are accessed through objects with the help of a dot operator. Dot operator links the name of the object with the name of the variable or method which needs to be accessed. For example, consider the following statements: obj.name; //accessing instance variable of the class

obj.marks(); //accessing method of the class

**Class:**

A class is a user-defined data type that can be used to create instances of its type called objects. Like any other user-defined data type, it also needs to be declared and defined in the program. A class definition specifies a new data type that can be treated as a built-in data type/ The syntax for defining a class is as follows:

class class\_name

{

//variable declaration

//methods declaration

}

The variables declared in the class are known as instance variables. The variables and methods declared within the curly braces are collectively known as members of the class. A class can also be empty. That is, the following class definition is also valid: class class\_name

{

}

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 18

Programming in Java Dr. M. T. Wanjari

Here, since the body of the class is empty, it does not contain any variable and methods so it cannot perform any useful action. However, this class can be successfully compiled and you can also create objects using it.

Note: In java, there is no semicolon after closing brace in class definition. A simple class definition without any method is as follows:

class Cubiod

{

int length;

int width; //variable declaration

int height;

}

In this example, a class named Cuboid with three instance variables of type int, namely, length, width and height is created.

**Defining Methods:**

As discuss earlier, a class consists of instance variables and methods. A class, which consists of only variables of only variables (and without methods, which manipulate them), cannot perform any useful operation. Therefore, to access the instance variables of a class and manipulate them, you must add methods in the class.

The syntax for defining method is as follows:

return\_type method\_name (parameter\_list)

{

body of the method

}

where,

return\_type is the type of data that is returned by the method.

method\_name specifies the name of the method. This can be any name other than the keyword in Java.

parameter\_list consists of a series of pairs of datatype and identifiers separated by commas.

Note: The parameter\_list can be empty and if a method does not return any value, its return type must be void.

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 19

Programming in Java Dr. M. T. Wanjari

For example, consider the following method definition:

int volume()

{

Body of the method

}

Here, the method volume() does not accept any parameter and return a value of type int. A class definition with method is as follows:

class Cuboid

{

int length;

int width; //variables declaretion

int height;

int volume() //method definition

{

return(length\*width\*height);

}

}

Note: Methods must be declared immediate after the declaration of the instance variables inside the body of the class.

**Instantiating the Object of a Class**

Once a class is defined, it can be used to create the variables of its type known as objects. The process of creating an object of a class is known as instantiation. The relationship between an object and a class is same as that of any variable and its data type. To create an object, you have to declare it first.

The syntax for declaring an object is as follows:

class\_name object\_name;

where,

class\_name is the name of the class

object\_name is the name of the object of class\_name type

for example, the statement to declare an object of Cuboid type is

Cuboi cobj;

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 20

Programming in Java Dr. M. T. Wanjari

This statement declares a variable cobj as a reference to an object of Cuboid type. After the execution of this statement, cobj contains the null value, which indicates that it does not point to an actual object. Once the object is declared, you have to create it by allocating the required memory space to it. In Java, this is done with the help of the new operator.

The syntax for creating an object is as follows:

Object\_name=new class\_name;

For example, the stamen to create an object of Cuboid type is as follows:

cobj=new Cuboid();

This statement creates an actual object by dynamically allocating memory space to it and returns a reference to cobj.

The preceding statements can be combined into one statement as follows:

Cuboid cobj=new Cuboid( );

**Accessing Members of a Class and Calling Methods:**

Each object of a class has its own set of variables. These variables should be assigned values before using them in the program. The instance variables and methods added in the program cannot be accessed directly outside the class using their names. To access the variables and methods outside the class, dot (.) operator is used as follows:

object\_name.variable\_name

object\_name.method\_name (parameter\_list)

where,

object\_name is the name of object

variable\_name is the name of the instance variable that is to be accessed.

method\_name is the name of the method which is to be called.

parameter\_list is the series of pairs of data types and their respective identifiers. For example, the instance variable length of Cuboid class can be accessed as follows. cobj.length;

Similarly, the method volume() of Cuboid class can be accessed as follows: cobj.volume( );

**Example: A program to demonstrate accessing the members of a class and calling the methods is as follows:**

Class Cuboid

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 21

Programming in Java Dr. M. T. Wanjari

{

int length;

int width;

int height;

int volume ( ); //method definition

{

return(length\*width\*height);

}

}

class ClassDemo

{

public static void main(String args[])

{

Cuboid cobj=new Cuboid( ); //object creation

cobj.length=60;

cobj.width=20; //accessing variables

cobj.height=40;

int vol=cobj.volume( ); //calling method

System.out.println(“The volume of the Cuboid is: ” +vol);

}

}

**The output of the program is as follows:**

The volume of the Cuboid is 48000

In the preceding program, the instance variables length, width and height of the object cobj are assigned values outside the class using the dot operator and the method volume( ) is also called using the dot operator. Alternatively, the instance variables can be assigned values by using a parameterized method.

**Example: A program to demonstrate the use of parameterized method is as follows:** class Cuboid

{

int length;

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 22

Programming in Java Dr. M. T. Wanjari

int width;

int height;

void initialize(int l, int w, int h) //parameterized method {

length=l;

width=w;

height=h;

}

int volume();

{

return(length\*width\*height);

}

}

class ParameterizedMethod

{

public static void main(String args[])

{

Cuboid c1=new Cuboid(); //creating object

Cuboid c2=new Cuboid();

c1.initialize(12,10,8); //calling initialized method int vol1=c1.volume(); //calling volume method c2.initialize(13,11,9);

int vol2=c2.volume();

System.out.println(“The volume of the Cuboid is:”+vol1); System.out.println(“The volume of the Cuboid is:”+vol2); }

}

**The output of the program is as follows:**

The volume of the Cuboid is: 960

The volume of the Cuboid is: 1287

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 23

Programming in Java Dr. M. T. Wanjari

Here, the initialize( ) method accepts three parameters (arguments) namely. l, w and h and assigns them to the instance variables, namely, length, width and height respectively of the appropriate object.

**Argument Passing**

The argument that appears in the method call statement are known as actual arguments and those which appear in the method definition are known as formal arguments. The number of actual arguments, their order and type in the method call must match with that of the formal arguments. The following two methods are used to pass an argument to the method:

**1. Call-by-value**

**2. Call-by-reference**

**Call-by-value:** In call-by-value method, the values of the actual arguments are copied into the formal arguments. Putting in simple words, the method creates a copy of argument values and then works with these copied values. As a result, any changes made to this set of copied values in the called method do not affect the values of the actual arguments in the calling method. In Java, when arguments of primitive type are passed to a method, they are passed by value.

**Example: A program to demonstrate the call-by-value mechanism is as follows:** class FirstCall

{

void vol (int I, int j, int k)

{

i+=5;

j+=5;

k+=5;

}

}

Class CallByValue

{

public static void main(String args[])

{

FirstCall fc = new FirstCall();

int l=10;

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 24

Programming in Java Dr. M. T. Wanjari

int w=20;

int h=15;

System.out.println(“Legnth, Width, Height before call is:” +l+ “ ” +w+ “ ” +h); fc.vol(l, w, h); //passing values as arguments

System.out.println(“Legnth, Bredth, Height after call is:” +l+ “ ” +w+ “ ” +h); }

}

**The output of the program is as follows:**

Length, Breadth, Height before call is: 10 20 15

Length, Breadth, Height after call is: 10 20 15

In this program, the value of l. w and h remain same before and after the call. This is because when the method vol( ) is invoked l ,w and h are passed by value and their values gets copied into formal arguments i, j and k. Hence, the method vol ( ) works with i, j and k and not with l,w and h. As a result, the values of l, w and h remain unaltered.

**Call-by-reference:** In call-by-reference method, argument’s reference is passed as method parameter. Unlike call-by-value method, call-by-reference does not create a copy of the actual argument and the called method works with the original values. This implies that any changes made to the variable in the method body are reflected in the calling method. In Java, when an object of a class is passed to a method as an argument, it is passed by reference.

**Example: A program to demonstrate the call-by-reference mechanism is as follows:** class SecondCall

{

int length, breadth, height; //variable declaration

void vlo(SecondCall s) //method definition

{

s.length+=5;

s.bredth+=5;

s.height+=5;

}

}

class CallByReference

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 25

Programming in Java Dr. M. T. Wanjari

{

public static void main(String args[])

{

SecondCall sc =new SecondCall( );

sc.length=10;

sc.breadth=20;

sc.height=30;

System.out.println(“The values of Length, Breadth and Height before method call: ” +sc.length + “ “ +sc.breadth + “ ” +sc.height);

sc.vol (sc); //passig object as method argument

System.out.println(“The values of Length, Breadth and Height after method call: ” +sc.length + “ “ +sc.breadth + “ ” +sc.height);

}

}

**The output of the program is as follows:**

The values of Length, Breadth and Height before method call: 10 20 30

The values of Length, Breadth and Height after method call: 15 25 35

In the preceding program, the value of length, breadth and height of object sc change after invoking the method. Because here when the method vol( ) is called, the object of a class is passed as parameter. As a result, changes made to the length, breadth and height of this passed object in the vol ( ) method are reflected in the original values.

**Constructors:**

A constructor is a special method, which is used to initialize the objects at the time of their creation. The name of the constructor is same as the name of the class in which it resides. Unlike other methods in Java, a constructor does not have any return type (not even void). This is because the implicit return type of constructor is the instance of the class to which it belongs.

The syntax to define a constructor is as follows:

class class\_name

{

//variable declaration

class\_name (parameter\_list) //constructor definition

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 26

Programming in Java Dr. M. T. Wanjari

{

//body of the constructor

}

}

It should be noted that if you do not explicitly define any constructor, then the Java compiler automatically provides a default constructor that initializes all the instance variables to 0.

**Example: A program to demonstrate the concept of a constructor is as follows:** class Cuboid

{

int length;

int width;

int height;

Cuboid () //constructor definition without parameters

{

length=20;

width=10;

height=15;

}

int volume()

{

}

}

class Constructor

{

public static void main(String args[])

{

Cuboid c1=new Cuboid ( );

Cuboid c2=new Cuboid ( );

int a=c1.volume( );

int b=c2.volume( );

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 27

Programming in Java Dr. M. T. Wanjari

System.out.println(“The volume of first cuboid ”+a);

System.out.println(“The volume of second cuboid ”+b);

}

}

**The output of the program is as follows:**

The volume of first cuboid is 3000

The volume of second cuboid is 3000

In this example, the constructor initializes both the objects c1 and c2 of Cuboid type with the same values as a result the volume of the cuboids is same. Clearly, a non-parameterized constructor provides same values to all the objects of a class.

**Parameterized Constructors:**

When different objects of a class need to be initialized with different values, a parameterized constructor can be defined. A parameterized constructor is a constructor that accepts one or more parameters at the time of creation of objects and initializes the instance variables of the objects with these parameters. It makes the program more flexible, because you can assign different values to the instance variables of the objects of a class by using it.

**Example: A program to demonstrate the concept of a parameterized constructor is as follows:**

class Cuboid

{

int length;

int width;

int height;

Cuboid(int l, int w, int h) //parameterized constructor

{

length=1;

width=w;

height=h;

}

int volume() //method definition

{

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 28

Programming in Java Dr. M. T. Wanjari

return(length\*width\*height);

}

}

class ParameterizedConstructor

{

public static void main(String args[])

{

Cuboid c1=new Cuboid(10,8,9);

Cuboid c2=new Cuboid(7,6,5);

int a=c1.voulume();

int b=c2.volume();

System.out.println(“The volume of the first Cuboid is:”+a);

System.out.println(“The volume of the second Cuboid is:”+b);

}

}

**The output of the program is as follows:**

The volume of the first cuboid is 720

The volume of the second cuboid is 210

**Types of classes:**

The classes define in Java 1.0 version have all been top-level classes. The top-level classes are the classes, which are direct members of packages and are not nested within any other classes. However, with the initialization of java 1.1 version, there are four other types of classes, broadly known as inner classes that can be defined in an Java program. Inner class is the biggest enhancement to the Java language. Classes can be defined as the members of other classes and also within a block of Java code. These four types of classes are explained as follows:

• **Static member classes:** A class which is defined as a static member of another class is called static member class. It is just like a top-level class with the difference that it can access the static members of the class of which it is a static member class.

• **Member classes:** It is also a class that is defined as a member of another class, but it is not declared as static. The code of a member class can access all the static and non-static

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 29

Programming in Java Dr. M. T. Wanjari

fields and methods of its enclosing class. Also, an instance of a member class is always linked with an instance of the enclosing class.

• **Local classes:** A class, which is defined within a block of code, is called local class. A local class is visible only within that block. It is analogous to a local variable in some ways. Though local classes are not member classes, they are still defined within an enclosing class and they can still use its fields and methods.

• **Anonymous classes:** A kind of local class that does not have any name is called anonymous class. An anonymous class combines the steps of class definition and object instantiation in a single Java expression. As the class is instantiated in the same expression that defines it, it can be instantiated only once. Otherwise, anonymous classes are quite similar to local classes in terms of behavior and use.

**Static Members:**

So far, you have studied that members of a class can be initialized or accessed by its objects only. However, it is possible to define the members of a class without reference to a specific object. This is done declaring the instance variables and methods of a class static. Unlike non-static members, the static members are associated with the class as a whole, rather than with individual objects. Therefore, outside the class, the static members of a class are called by using class names rather than objects. Since the static variables and static methods contain the properties of a class and not of the individual objects, they are also known as class variables and class methods, respectively.

The syntax to declare static variable is as follows:

static return\_type variable\_name;

The syntax to access static variable is as follows:

class\_name.variable\_name

The syntax to declare static method is as follows:

static return\_type method\_name (parameter\_list);

The syntax to access static method is as follows:

class\_name.method\_name

If a class member is declared as static, only one copy of that data member is created, regardless of the member of objects. All the objects of a class share this single copy of the static data member.

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 30

Programming in Java Dr. M. T. Wanjari

**Example: A program to demonstrate the use of static members of a class is as follows:**

class StaticMembers

{

static int i=10; //static variable

static void add(int x, int y) //static method

{

System.out.println(“Sum of two numbers is:”+(x+y));

}

}

class StaticExample

{

public static void main(String args[])

{

StaticMembers.add(30,40); //calling static method

int i =StaticMembers.i; //accessing static variable

System.out.println(“Value of i is:”+i);

}

}

**The output of the program is as follows:**

Sum of two numbers is: 70

Value of i is: 10

Hence, variable i and method add( ) are declared as static. Therefore, outside the class they are accessed using the name of the class in which they are defined.

Note: The methods declared as static can access only static data and call static methods. **Garbage collection and finalize() method:**

**Garbage Collection:**

We know that new operator is used to allocate object dynamically. In some language like C++, the delete operator is used to destroy the object i.e. to make memory free for further reallocation. Java uses a procedure called garbage collection to reclaim memory occupied by objects that are no longer accessible to a program. In java, it is the responsibility of the system, not the programmer, to keep track of which objects are “garbage”. The object doesn’t become garbage until all those references have been dropped. As Java uses garbage collection, errors like memory leak or dangling pointer are simply impossible. Because of Java’s automatic garbage collection, a lot of cleanup is done automatically.

**Finalize() method:**

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 31

Programming in Java Dr. M. T. Wanjari

When object is destroyed, sometimes, it needs to perform some operation. For example, when object holds some non-Java resources, then it is necessary to make sure that these resources are freed before destroying the object. In C++, we use destructor methods, but in Java, this situation is handled by a special mechanism called finalization. Using this mechanism we can define a specific action that will occur when an object is just about to be reclaimed by the garbage collector. General form of finalize method id

protected void finalize()

{

// code

}

**Method Overloading:**

Method overloading means same name to more than one function with different signature. In Java, it is possible to define two or more methods with same name within the same class, as long as, their parameter declarations are different. Through method overloading Java implements polymorphism i.e. “one interface, multiple methods”. Method overloading is used when objects are required to perform conceptually similar task using different input parameters. The return type of method is insufficient to distinguish two methods; hence, overloaded methods must differ in the type and/or number of their parameters. When an overloaded function is called, Java executes the version of the method whose parameters match the arguments. An example is in the java.util.Arrays class in which the method sort is overloaded.

sort(int[])

sort(double[])

sort(char[])

Above overloaded method can be used as

final int[] a int = {40, 55, 13};

final double[] a doub = {2.3, 5.8, 4.5};

final char[] a char = {‘x’, ‘a’, ‘r’};

Arrays.sort (adoub);

Arrays.sort (aint);

Arrays.sort (achar);

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 32

Programming in Java Dr. M. T. Wanjari

Overloaded methods don’t have to be static. We stay that the name of methods is overloaded because it has several different meanings. The Java doesn’t get the methods mixed up but it can tell which one you want to call by the number and types of the actual parameters that you provide in the method call statement. Another example of inbuilt overloaded methods is putln() method in TextIo class. This includes different methods named puln which are semantically related and having following different signatures

putln(int)

putln(int, int)

putln(double)

putln(string)

putln(string, int)

putln(char)

putln(Boolean)

putln(boolean, int)

putln()

Following example explains overloading user defined methods.

class method\_overload

{

void display()

{

System.out.println(“No argument”);

}

void display(int x)

{

System.out.println(“x=”+x);

}

void display(int x, int y)

{

System.out.println(“avg = ”+ (x+y)/2);

}

void display(char ch)

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 33

Programming in Java Dr. M. T. Wanjari

{

for(int I=1;I<=10;I++)

System.out.println(ch);

}

}

class OverLoading

{

public static void main(String args[])

{

method\_overload mo = new method\_overload();

mo.display();

mo.display(5);

mo.display(12, 13);

mo.display(“-”);

}

}

Like methods, constructors can be overloaded. At compiler time, the right implementation is chosen based on the signature of the method call. Although Java looks for a match between arguments, this match need not always be exact. In some cases Java’s automatic conversion play role in overload resolution. Above point can be made more clear with following example, consider the method display() overloaded as:

class method\_overload

{

void display(int x, int y)

{

-------

}

void display(double x)

{

------

}

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 34

Programming in Java Dr. M. T. Wanjari

class OverLoading

{

public static void main(String args[])

{

method\_overload mo = new method\_overload();

mo.display(5);

mo.display(12, 13);

}

}

In this program, as the method display(int) is not defined, no matching method found and hence, the method call mo.display(5) invokes display(double).

**Constructor Overloading**

Java allows to define two or more methods with same name within the same class, as long as, their parameters declaration are different. Constructors can be overloaded. Constructor Overloading means multiple constructor functions with different signature. Constructor overloading is allowed if the number of parameters or type of parameters is different. Constructor overloading is used when objects are required to perform conceptually similar task using different parameters. When an object is declared, Java executes the version of the constructor whose parameters match the arguments. Constructor overloading allows appropriate initialization of objects on creation, depending on the constructor invoked.

Eg. class Sample

{

int Roll;

boolean Feespaid;

String Result;

// default constructor

Sample()

{

Roll-10;

FeesPaid = true;

Result=new String(“FAIL”);

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 35

Programming in Java Dr. M. T. Wanjari

// parameterized constructor

Sample(int rn, Boolean fp, String rs)

{

Roll = rn;

FeesPaid = fp;

Result = rs;

}

// other function

}

**Command line argument:**

Command line arguments are parameters that are supplied to the application program at the time of execution. It may be recalled that program was invoked for execution of the command line as follows:

Java Test

Here, we have not supplied any command line arguments. Even if we supply arguments, the program does not know that what to do with them.

We can write java program that can receive and use the argument provided in the command line recall the signature of the main() method used in our earlier program. public static void main(String args[])

**args** is declared as an array of strings (known as string object). Any arguments provided in the command line(at the time of execution and use them in the program as we wish. Example, consider the command line

Java Test BASIC FORTRAN C++ Java

The command line arguments. These are assigned to the array as follows.

BASIC ----→ args[]

FORTRAN --→ args[1]

C++ --→ args[2]

Java ---→ args[3]

The individual elements of an array are accessed by using an index of subscript like args[i]. The value of I denotes the position of the elements inside the array. Eg. Program for use of command line arguments.

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 36

Programming in Java Dr. M. T. Wanjari

class ComLineTest

{

public static void main(String args[])

{

int count i=0;

String string;

count = args.length;

System.out.println(“Number of argyments = ” + count);

while(i<count)

{

string = args[i];

i = i + 1;

System.out.println(i+” ” + “Java is” + string+ “!”);

}

}

}

Above program use the command line arguments. Compile and run the program with the command line as follows”

Java ComLineTest Simple Object\_Oriented Distributed Robust Secure Portable Multithreaded Dynamic. Upon execution, the command line arguments Simple. Object\_Oriented, etc are passed to the program through the arrays args as discuss earlier. That is the element args[] contains Simple, args[1] contains Object-oriented, and so on. These elements are accessed using the loop variable i as an index like

name = args[i]

The index i is incremented using a while loop until al the argument are accessed. The number of argument is obtained by statement

Count = args.length;

The output of the program are as follows:

Number of arguments = 8

1: Java is Simple!

2: Java is Object\_Oriented!

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 37

Programming in Java Dr. M. T. Wanjari

3: Java is Distributed!

4: Java is Robust!

5: Java is Secure!

6: Java is Portable!

7: Java is Multithreaded!

8: Java is Dynamic!

**Arrays:**

Java uses the variables of different primitive data types to store data. However, these variables are incapable of holding more than one value at a time. For example, a single variable cannot be used for storing the marks of all the students in a class. For such purpose, Java provides a different kind of data type known as arrays.

Arrays are defined a fixed sequence of the same type of data elements. These data elements can be of any primitive or non-primitive data type. The elements of array are stored in contiguous memory locations and each individual element can be accessed using one or more indices or subscripts. A subscript or an index is a positive integer value, which indicates the position of an element in an array. Arrays are used when a programmer wants to store multiple data items of the type into a single list and also wants to access and manipulate individual elements of the list. Arrays can be subscripts used.

**1. Single Dimensional array**

A single dimensional array is the simplest form of an array that requires only one subscript to access an array element. Like an ordinary variable, an array must be declared before it is used in the program.

The syntax for declaring a single-dimensional array is as follows

data\_type array\_name[];

or

data\_type[] array\_name;

where, data\_type is an data type

array\_name is the name of the array.

For example, an array marks[] of type int can be declare using either of the following two statements:

int marks[];

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 38

Programming in Java Dr. M. T. Wanjari

or

int[] marks;

After array is an declared, you need to create it by allocating space to it in memory. Arrays are creating using new operator.

The syntax for creating an array is as follows

array\_name = new data\_type[size];

where, size is the size of the array

For example, an array marks[] of type int and size five can be created using this statemnent.

marks = new int[5];

The preceding steps of declaration and creation of an array can be combined into a single statement as follows:

data\_type array\_name=new data\_type[size];

Similarly, the statement to declare and create an array marks[] of type int and size five is as follows

int marks[] = new int[5];

Note: All the elements created using the new operator in the array will be automatically initialized to zero.

**Initialization of Single-Dimensional Array:**

Once an array is declared and memory is allocated to it, the next step is to initialize each array element with a valid and appropriate value. An array can be initialized at the time of its declaration.

The syntax for initializing an array at the time of its declaration is as follows data\_type array\_name[]={value\_1,value\_2,…..,value\_n};

The values are assigned to the array elements in the order in which they are listed. That is, value\_1,value\_2 and value\_n are assigned to the first, second and nth element of the array, respectively/

If an array is declared and initialized simultaneously, then specifying its size is optional. For example, the statement int marks[]={51,62,43,74,55} is also valid. The size of an array marks can be obtained by using marks.length() method.

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 39

Programming in Java Dr. M. T. Wanjari

**Note:** If you try to store or access values outside the range of array (index with negative value or value greater than the length of the array), run time error is generated. **Accessing Single-Dimensional Array Elements**

Once an array is declared and initialized, the values stored in the array can be accessed any time. Each individual array element can be access using the name of the array and the subscript value. Every element in an array is associated with a unique subscript value, starting from 0 to size-1(where, size refers to the maximum number of elements that can be stored in the array).

The syntax for accessing the values stored in a single-dimensional array is an follows: array\_name[subscript]

For example, the elements of the array marks can be referred to as marks[0], marks[1],marks[2],marks[3] and marks[4], respectively. Note that index of an array starts with 0. **Note:** the memory location, where the first element of an array is stored, is known as the base address, which is generally referred to by the name of the array.

Single-dimensional array are always allocated contiguous blocks of memory. This implies that every element in an array is always stored in a sequential manner next to each other. The memory representation of the array marks is shown in figure. As each element is of type int (that is 4 bytes long), the array marks occupies twenty contiguous bytes in the memory and these bytes are reserved in the memory at compile time.

marks marks[0] marks[1] marks[2] marks[3] marks[4]

51 62 43 74 55

2001 2005 2009 2013 2017

**Fig. 1.4 memory Representation of an Array marks**

**Manipulation of Single-Dimensional Array Elements**

An array can be manipulated with the help of various operations. These operations include finding the sum, average, maximum or minimum, sorting and searching of the array elements.

A program to sort the array elements is as follows:

class SortingArray

{

public static void main(String args[])

{

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 40

Programming in Java Dr. M. T. Wanjari

int a[]={67, 34, 12, 98, 26}; //array initialization at the time of declaration int n=a.length; //returns the length of the array

System.out.print(“The list of numbers:”);

for(int i=0;i<n;i++)

{

System.out.print(“ “+a[i]);

}

//sorting elements of an array

for(int i=0;i<n;i++)

{

for(int j=i+1;j<n;j++)

{

if(a[i]>a[j])

{

int temp=a[i];

a[i]=a[j];

a[j]=temp;

}

}

}

System.out.print(“\n”);

System.out.print(“The sorted list of given numbers:”);

for(int i=0;i<n;i++) //displaying sorted array

{

System.out.print(“ “ +a[i]);

}

}

}

The output of the program is as follows

The list of numbers: 67 34 12 98 26

The sorted list of given numbers: 12 26 34 67 98

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 41

Programming in Java Dr. M. T. Wanjari

**2. Multi-Dimensional Arrays**

Multi-dimensional arrays can be described as ‘an array of arrays’, that is, each element of the array is itself an array. A multi-dimensional array of dimension n is a collection of items that are accessed with the help of n subscript values.

**Two-Dimensional Array**

A two-dimensional array is the simple form of a multi-dimensional array that requires two subscript values to access an array element. Two-dimensional arrays are useful when the data being processed are to be arranged in the form of rows and columns (matrix form). The syntax for declaring a two-dimensional array is as follows:

data\_type array\_name[][];

or

data\_type[] [] array\_name;

The syntax for creating a two-dimensional array is as follows:

array\_name[] []=new data\_type[row\_size][column\_size];

The preceding two steps of declaration and creation can be combined into one using a single statement as follows:

data\_type array\_name[][]=new

data\_type[row\_size][column\_size];

For example, an array a[][] of type int having three row and two columns can be declared and created using this statement.

int a[][]=new int[3][2];

Here, 3 is the row and 2 is the column size.

**Initialization of Two-Dimensional Array**

Like a single-dimensional array, a two-dimensional array can also be declared and initialized at the same time. To understand how to initialize a two-dimensional array, consider this statement.

int a[3][2]={ {101, 51},

{102, 67},

{103, 76} };

In this statement, an array a[] [] of type int having three rows and two columns is declared and initialized. This type of initialization is generally used to increase the readability.

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 42

Programming in Java Dr. M. T. Wanjari

Now, another statement

int b[] [] ={ {2, 3, 4}, {1, 1, 1} };

In this statement, an array b[] [] of type int having two rows and three columns is initialized.

**Accessing Two-Dimensional Array Elements**

Once a two-dimensional array is declared and initialized, the value stored in the array elements can be accessed using two subscripts.

The syntax for accessing the two-dimensional array element is as follows: array\_name[row][column]

The first subscript value(row) specifies the row number and the second subscript value(column) specifies the column number. Both the subscript value specifies the position of the array element within the array.

For example, the element of array a(declared earlier) are referred to as a[0][0],a[0][1],a[1][0],a[1][1],a[2][0] and a[2][1], respectively.

Generally, two-dimensional arrays are represented with the help of a matrix. However, in actual implementation, two-dimensional arrays are always allocated contiguous blocks of memory. Figure shows matrix and memory representation of two-dimensional array a.

|  |  |
| --- | --- |
| 101 | 51 |
| 105 | 67 |
| 103 | 76 |

(a)

a[0][0] a[0][1] a[1][0] a[1][1] a[2][0] a[2][1] 101 51 105 67 103 76

2001 2005 2009 2013 2017 2021

| first row | second row | third row |

(b)

**Fig. 1.5 a) Matrix and b) Memory Representation of Array a[][]**

**Manipulation of Two-Dimensional Array Elements**

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 43

Programming in Java Dr. M. T. Wanjari

A two-dimensional array can be manipulated in many ways. Some of the common operations that can be performed on a two-dimensional array include finding the sum of row elements, column elements and diagonal elements, finding the maximum and minimum values, etc.

**Example: A program to calculate the sum of two matrices is as follows:** class MatricesSummation

{

public static void main(String args[])

{

int a[][]={{3,4,5},{3,2,7}}; //initializing matrix a

int b[][]={{2,4,7},{1,2,2}}; //initializing matrix b

int l=a.length;

System.out.println(“First matrix is:” + “ ” ”);

for(int i=0;i<l;i++) //displaying first matrix

{

for(int j=0;j<3;j++)

{

System.out.print(“ “ +a[i][j]);

}

System.out.println();

}

int m=b.length;

System.out.println(“Second matrix is:” + “ ” ”);

for(int i=0;i<m;i++) //displaying second matrix

{

for(int j=0;j<3;j++)

{

System.out.print(“ “ +b[i][j]);

}

System.out.println();

}

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 44

Programming in Java Dr. M. T. Wanjari

System.out.println(“Summation of the two matrices is:”);

//displaying sum of two matrices

for(int i=0;i<m;i++)

{

for(int j=0;j<=m;j++)

{

System.out.print(“ ”+(a[i][j]+b[i][j]));

}

System.out.println();

}

}

}

**The output of the program is as follows**

First matrix is:

3 4 5

3 2 7

Second matrix is:

2 4 7

1 2 2

Summation of the two matrices is:

5 8 12

4 4 9

**Variable Size Arrays**

Multi-dimensional arrays are arrays of arrays. In such multi-dimensional arrays, the size of each array can be varied. For example, consider the following statements: data\_type array\_name[][]=new data\_type[size][];

array\_name[0]=new data\_type[size\_1];

.

.

.

array\_name[size\_1]=new data\_type[size\_n];

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 45

Programming in Java Dr. M. T. Wanjari

where,

size is the number of rows in two-dimensional array.

size\_1,…,size\_n represents the number of column in each row of two-dimensional array. **Example: A program to demonstrate a variable size array is as follows:** class VariableArray

{

public static void main(String args[])

{

int a[][]=new int[4][];

a[0]=new int[2];

a[1]=new int[4];

a[2]=new int[3];

a[3]=new int[5];

int i,j;

System.out.println(“The variable sized array is:”);

for(i=0;i<4;i++)

{

for(j=0; j<a[i].length;j++)

{

a[i][j]=j;

System.out.print(“ ”+a[i][j]);

}

System.out.println();

}

}

}

**The output of the program is as follows:**

The variable sized array is:

0 1

0 1 2 3

0 1 2

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 46

Programming in Java Dr. M. T. Wanjari

0 1 2 3 4

**Strings**

In Java, a string is an object that created either using String or StringBuffer class. Each class has its own set of methods for creating and manipulating strings. The difference between these two classes is that a string created from the String class cannot be modified, that is, characters can be inserted to, replaced or removed from the string. However, the string created using StringBuffer class cannot only be modified can also be expanded or contracted dynamically whenever required.

**1. String class**

The String class is more commonly used for the following purposes:

• To Display messages

• Search or compares the strings

• Extract individual characters in a string as substrings

The syntax to declare a string as follows:

String string\_name;

The syntax for creating a string is as follows:

string\_name=new String(“Sequence\_of\_characters”);

These two steps of declaration and creation can be combined into a single statement as follows:

String string\_name=new String(“Sequence\_of\_characters”);

For example, the statement to declare and create a string strl using String class as follows: String str1=new String(“Java programming Language”);

The String class provides various methods for manipulating strings. Some of the most commonly used methods of String class along with their description are listed as tables **Table: 1.5 String Class Methods and their Description**

|  |  |
| --- | --- |
| **Methods** | **Description** |
| str1.length() | Returns the length of the string str1 |
| str1.equals(str2) | Returns ‘true’ if string str1 is equal to string str2 |
| str1.compareTo(str2) | Returns negative if str1< str2, positive if str1> str2, otherwise 0 |
| str1.concat(str2) | Concatenates string str1and string str2 |
| str1=str2.trim() | Removes all the value white spaces at the beginning and end of the |

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 47

Programming in Java Dr. M. T. Wanjari

|  |  |
| --- | --- |
|  | string str2 and assign it to str1 |
| str1=str2.replace(‘a’,’b’) | Replace all a appearing in the string str2 with b and assign it to str1 |
| str1=str2.toLowerCase() | Converts uppercase letters in the string str2 to lowercase and assign it to str1 |
| str1=str2.toUpperCase() | Converts lowercase letters in the string str2 to uppercase and assign it to str1 |
| str1.indexOf(‘a’) | Gives the position of the first occurrence of character ‘a’ in the string str1 |
| str1.indexOf(‘a’,n) | Gives the position of the first occurrence of character ‘a’ that occurs after nth position in the string str1 |

class StrinDemonstrate

{

public static void main(String args[])

{

String str1=new String(“New”); // creating string str1

String str2=new String(“Delhi”); // creating string str2

String str3=str1.concat(str2); // concatenating strings str1 and str2 String str4=str3.toUpperCase();

String str5=str3.toLowerCase();

System.out.println(“Combined String is: ” +str3);

System.out.println(“Combined String in UPPER CASE is: ” +str4);

System.out.println(“Combined String in LOWER CASE is: ” +str5);

}

}

**2. StringBuffer Class**

As stated earlier, string created using String class are of fixed length, whereas, string created using StrinBuffer class are of varying length.

The syntax to declare a string is as follows:

StringBuffer string\_name;

The syntax for creating a string is as follows:

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 48

Programming in Java Dr. M. T. Wanjari

string\_name=new StringBuffer(“Sequence\_of\_characters”);

These two steps of declaration and creation can be combined into a single statement as follows:

StringBuffer string\_name=new StringBuffer(“Sequence\_of\_characters”);

For example, the statement to declare and create a string str1 using StringBuffer class is as follows:

StringBuffer str1=new StringBuffer(“Java Programming Language”);

Some of commonly used methods of StringBuffer c lass along with their description are listed in table.

**Table: 1.6 StringBuffer Class Mehtods and their Description**

|  |  |
| --- | --- |
| Methods | Descriptions |
| str1.append(str2) | Appends the string str2 at the end of the string str1 |
| str1.inset(n, str2) | Insert the string str2 at the position n of the string str1 |
| str1.setCharAt(n,’x’) | Sets the nth character of the string str1 to x |
| str1.setLength(n) | Sets the length of the string str1 to n. If n<str1.length(), null characters are added at the end of str1 |
| str1.reverse() | Reverse the string str1 |
| str1.delete(m, n) | Deletes characters of the string str1 from mth index to (n-1)th index. |
| str1.deleteCharAt(m) | Remove characters of the string str1 at mth index |
| str1.replace(m, n, “size”) | Replaces the portion of the string str1 from mth index to (n-1)th with the string str2 |

**Example: A program to demonstrate the use of some of the methods of StrinBuffer class ia as follows:**

class StringBufferClass

{

public static void main(String args[])

{

StringBuffer str1=new StringBuffer(“Programming”);

System.out.println(“String is: ”+str1);

System.out,println(“Length of the Given string is: ”+str1.length());

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 49

Programming in Java Dr. M. T. Wanjari

str1.insert(0, “Java”); //inserting a string at oth/position

System.out.println(“String after inserting a string at oth position is: ”+str1); str1.setCharAt(1,’a’); // setting character ‘a’ at 1st position

System.out.println(“String after inserting a string at 1th position is: ”+str1); str1.append(“Language”); // appending a string

System.out.println(“String after appending another string is: ”+str1);

str1.replace(1, 5, “Hello”); // replacing a substring in str1

System.out.println(“String after replacement of a substring with another string is: ”+str1); }

}

The Disadvantage of StrinBuffer class is that it does not provide a set of methods for comparing strings, locating characters or substrings within a string. Thus, the String class is preferred for displaying or comparing strings whereas the StringBuffer class is used when addition and modification are required.

**Inheritance:**

Inheritance is one of the major strengths of object-oriented programming. It is the process of deriving a new class from the existing one in such a way that the new class inherits all the members (data members and methods) of the existing class. In other words, inheritance facilities a class to acquire the properties and functionality of another class. The new class depicts the acquired properties and behavior of the existing class as well as its own unique properties and behavior.

It also allows code reusability, that is, it facilitates classes to reuse the existing code. The new class acquires those members of the old class that are already tested and debugged. Hence, inheritance saves time as well as increases the reliability. Using inheritance, we can create a general class that defines common properties for a set of related classes.

**1. Superclass and Subclass**

The class that is inherited by other classes is called a base class, superclass or parent class. The class that inherits the properties of the superclass is called a subclass, derived class or child class. The subclass inherits all the instance variable and methods defined by the superclass and at the same time it also contains its own members. For example, in figure, Animal is the

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 50

Programming in Java Dr. M. T. Wanjari

superclass which is inherited by three subclasses Carnivore, Herbivore and Omnivore. Hence, Carnivore, and Omnivore inherits all the members of the superclass Animal.

superclass

Vehicles

subclasses

Automobiles Pulled vehicles**Fig. 1.6 Superclass and Subclass**

**Defining a Subclass**

Inheritance is implemented which defining the subclass. The name of the superclass is specified in the subclass definition. A subclass can be defined by using extends keyword. The syntax to define a subclass is

class sub\_class extends super\_class

{

//variable and methods declaration

}

where, sub\_class is the name of subclass that inherits the superclass

super\_class is the name of the superclass that is being inherited

extends is the keyword that indicates that the super\_class properties have been extended to the sub\_class

**Types of Inheritance**

Inheritance allows the creating of a logical relationship between two or more classes. Depending on the number of classes involved and the way the classes are inherited, inheritance may take different forms, namely, single inheritance, hierarchical inheritance, multilevel inheritance and multiple inheritance.

**1. Single Inheritance**

In a single inheritance, a class is derived from a superclass. Figure shows single inheritance in which subclass Employees is derived from the superclass Person.

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 51

Programming in Java Dr. M. T. Wanjari

Person

Employee

**Fig. 1.7 Example of Single Inheritance**

The syntax to define the subclass that implements single inheritance is

class

{

:

}

class Class2 extends Class1

{

:

}

**2. Hierarchical Inheritance**

The type of inheritance in which more than one class is derived from a single superclass is called hierarchical inheritance. In hierarchical inheritance, superclass includes all the properties that are common to all of its subclasses. Figure shows hierarchical inheritance with two classes Student and Employee deriving from a single superclass Person.

Person

Student Employee

**Fig. 1.8 Hierarchical Inheritance**

class Class1

{

:

}

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 52

Programming in Java Dr. M. T. Wanjari

class Class2 extends Class1

{

:

}

class Class3 extends Class1

{

:

}

**3. Multilevel Inheritance**

In multilevel inheritance, one class inherited from another class, which in turn is inherited from some other class. Multilevel inheritance comprises two or more levels. The subclass has all the properties of its direct superclass as well as its indirect superclass. That is known as the transitive nature of multilevel inheritance. Figure shows multilevel inheritance in which superclass Person is inherited by a subclass Employee, which is again inherited by another subclass Faculty. Hence, Person ia an indirect superclass to Faculty.

Superclass

Person

Employee

Intermediate superclass

Faculty

Subclass

**Fig. 1.9 Multilevel Inheritance**

class Class1

{

:

}

class Class2 extends Class1

{

:

}

class Class3 extends Class2

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 53

Programming in Java Dr. M. T. Wanjari

{

:

}

**4. Multiple Inheritance**

In multiple inheritance, a class inherits properties form more than one superclass as shown in figure. In java, it is not possible to implement multiple inheritance directly. However, there is a concept known as interface through which inheritance can be implemented.

Person Company

Owner

**Fig. 1.10 Multiple Inheritance**

**Access Modifier**

In the programs discussed, the class members are accessible everywhere in the program and the subclass can inherit all the variables and methods of a superclass by using the keyword extends. However, there might be certain situations when we want to restrict the accessibility of members of a class for various reasons, security being one of the main reasons. For this, Java provides three types of access modifier, namely, public, private and protected. They are also known as visibility controls.

• **Public:** When a member of a class is declared as public, it can be accessed everywhere in the program.

• **Private:** A member declared as private can be accessed only within a class. • **Protected:** A member declared as protected is accessible not only to all the classes and subclasses in the same package but also to subclass in other packages. If no access modifier is specified, by default, the data member of a class is visible only within the same package. We may use the word friendly in connection with the default access, however, friendly is not a Java keyword.

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 54

Programming in Java Dr. M. T. Wanjari

Apart from the access modifier discussed, there is another access modifier namely privateprotected which was used with the release of java 1.0. However it has been dropped in java 1.1 and further versions. If data members are declared as privateprotected, they can be accessed by all the subclasses irrespective of the package to which they belong. However, they are not visible in other classes of the same package.

**Interface:**

**Defining Interface:**

An interface is just like a class. The only difference is that it contains only final variables and the method declarations. Hence, we can think of an interface as a ‘fully abstract class’. There is no limitation to the number of interfaces that a class can implement. An interface is defined just like a class but rather than using the keyword class, the keyword interface is used.

The syntax to define an interface is

interface interface\_name

{

//variables and methods declaration

}

where,

interface is the Java keyword

interface\_name is the name of the interface

If there is no access specifier included in the interface definition, then the default access is used and the interface is visible only to the members of the same package. However, to make the interface accessible in any other code, it can be declared as public. The variable in an interface are by default static and final. Hence, they cannot be altered by the implementing class. The methods are by default abstract. All the methods must be implemented by the class which implements the interface. For example, consider the following code.

interface Area

{

double pi=3.142;

void compute ( );

}

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 55

Programming in Java Dr. M. T. Wanjari

Here, Area is the name of the interface. The variable pi is initialized with a constant value. Note that the method compute ( ) does not have the body part and its declaration ends with semicolon.

If the interface is declared as public then, all the variables and methods are implicitly public.

**Implementing Interface**

Once an interface defined, it can be used as a superclass whose members and properties can be inherited by other classes. One or more classes can implement in the interface by using the keyword implements in the class definition.

The syntax for implementing an interface is

class class\_name implements interface\_name

{

//variables and methods declaration

}

For example, consider the following code which implements the interface Area. class Circle implements Area

{

float r=4.3f;

public void compute( )

{

double carea=pi\*r\*r;

System.out.println(“The area of circle is: ”+carea);

}

}

When the methods in an interface are defined in the implementing class, the public keyword must be used. Also, the signature of the method implementing the interface must exactly match the signature of the method declaration in the interface.

A class can implement more than one interface as shown below.

class class\_name implements interface1, interface2

{

:

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 56

Programming in Java Dr. M. T. Wanjari

}

A class can extends another class while implementing interfaces as shown below. class class\_name extends superclass implements interface\_name

{

:

}

**Partial Implementations**

If a class that implements the interface does not provide complete implementation of the methods declared in the interface, then it is necessary for the class to be declared as abstract. For example, consider another class Square which implements the interface Area. abstract class Square implements Area

{

float side=2.4;

double sqarea=side\*side;

void display ( )

{

System.out.println(“The area of square is: ”+sqarea);

}

}

Here, the class Square is declared as abstract as it does not implement the method compute ( ) declared in Area. Any class that inherits Square must implement compute ( ) method or the class itself must be declared as abstract.

**Extending Interfaces:**

An interface can inherit another interface by using extends keyword in the same way as a class inherits from another class. Like a class, a subinterface will inherit all the properties of the superinterface and also adds its own data members. For example consider the following code segment.

interface Interface1

{

:

}

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 57

Programming in Java Dr. M. T. Wanjari

interface Interface2 extends Interface1

{

:

}

An interface can also inherit from more than one interface. To define an interface that extends several interfaces, the names of the superinterfaces are separated by comma (,) as shown here.

interface Interface3 extends Interface1, Interface2

{

:

}

The methods declared in the superinterfaces cannot be implemented by the subinterfaces. They must be implemented only by the class which implements the interface. When a class implements an interface which is inherited from another interface then the class must provide implementation for all the methods declared in both the interfaces.

An interface cannot extends classes. it can only extends another interface. Also, an interface cannot implement another interface.

**Extends and Implements Together**

By now you are familiar with the concept of interface. Now, you will learn how an interface can be used to implement multiple inheritances by taking a simple example, in this example, the class Faculty extends a class Employee and implements an interface Bonus. **Example: A program to demonstrate implementation multiple inheritance through interface.**

class Person

{

String name;

int age

String address;

void persondetails(String nm, int ag, String add)

{

name=nm;

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 58

Programming in Java Dr. M. T. Wanjari

age=ag;

address=add;

}

void displayperson()

{

System.out.println(“Name: ”+name );

System.out.println(“Age: ”+age );

System.out.println(“Adderess: ”+address ); }

}

class Employee extends Person

{

int empid;

int salary;

void empdetails(int id, int sal)

{

empid=id;

salary=sal;

}

void displayemployee()

{

System.out.println(“Empid: ”+empid ); System.out.println(“Salary: ”+salary ); }

}

interface Bonus

{

int bonus=1000;

void compute();

}

class Faculty extends Employee implements Bonus

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 59

Programming in Java Dr. M. T. Wanjari

{

int amount;

public void compute()

{

System.out.println(“The bonus is: ”+bonus );

amount=salary+bonus;

}

void facultydetails()

{

displayperson();

displayemployee();

compute();

System.out.println(“The total amount is: ”+amount );

}

}

public class MultipleInheritance

{

public static void main(String[] args)

{

Faculty obj=new Faculty();

obj.persondetails(“Shivaji”, 23, “101, Congress Nagar, Nagpur”); obj.empdetails(001, 20000);

obj.facultydetails();

System.out.println(“ ”);

obj.persondetails(“shahaji”, 27, “102, Sai nagar, Nagpur”);

obj.empdetails(002, 30000);

obj.facultydetails();

}

}

**The output of the program is**

Name: Shivaji

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 60

Programming in Java Dr. M. T. Wanjari

Age: 23

Address: 101, Congress Nagar, Nagpur

Empid: 1

Salary: 20000

The bonus is: 1000

The total amount is: 21000

Name: Shahaji

Age: 27

Address: 102, Sai nagar, Nagpur

Empid: 2

Salary: 30000

The bonus is: 1000

The total amount is: 31000

**Packages:**

A package is a named collection of classes. Any number of related classes can be grouped into a single package. By grouping the classes into a package, we can achieve the following.

• The classes which belong to a package of another program can be easily reused. • Two classes in two different packages can have the same name. for example, we can create a class called Student without being concerned about the collision of this name with some other class called Student which belong to some other package. Every class belongs to a package. The classes that we have created and used so far belong to the default package. Java packages are categorized into two types. namely, API packages and user-defined packages. Application Programming Interface (API) packages are the standard packages are the standard packages available in Java which contain all of the standard classes. Java also allows us to create our own package known as user-defined packages.

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 61

Programming in Java Dr. M. T. Wanjari

**Java API Packages:**

There is a huge list of API packages by Java. Some of the quite frequently used packages are shown in Table 1.7.

**Table: 1.7 Commonly Used API Packages**

|  |  |
| --- | --- |
| **Package** | **Description** |
| java.lang | It includes classes that are fundamental to Java such as String, Math, Exception etc. |
| java.io | This package provides classes to support input and output operations. |
| java.awt | This provides a set of classes to implement graphical user interface components such as windows, dialog boxes, menus, list, buttons, checkboxes, textfields, scrollbars, etc. |
| java.applet | This package includes classes to create applets that can be embedded in a webpage. |
| java.util | This package includes language utility classes such as time, date, random number, hash tables, vectors, enumeration, etc. |
| java.sql | This package contains classes for accessing database using standard SQL. |

Each Java API package is organized in a hierarchical form. For example, consider a package, say, java.lang. The java.lang package containing various classes is organized in hierarchical form as shown in Fig. 1.11.

lang

lang

String

Math

--------

Thread

**Fig. 1.11 Hierarchical Representation of java,lang Package**

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 62

Programming in Java Dr. M. T. Wanjari

Fig. 1.11 shows that the package java contains the package lang which in turn contains various classes such as Math, String, etc.

The classes stored in a package can be accessed in two ways depending on the following two situations:

• If we need to access the class only once or when we do not need to access any other class of the same package.

In this case, to access the class, the name of the package is followed by dot ( . ) operator which is followed by the name of the class. For example, the statement to access the Math class contained in lang package is as follows:

java.lang.Math

• If we want to use multiple classes contained in a package or use the same class in various places of the program.

This can be achieved with the help of import statement. The syntax to access the same class in various places is

import package\_name.class\_name;

where,

import is a Java keyword

package\_name is the name of the package

class\_name is the name of the class belonging to that package

For example, consider this statement.

import java.lang.Math;

In this statement, the class Math contained in the package java.lang is imported in the program and can now be used at various places in the program without using the package name. We can access multiple classes contained in a package using the statement given here, import package\_name.\*;

For example, the statement import java.lang.\* import all the classes contained in java.lang package.

**Naming Conventions:**

There are certain Java naming conventions using which packages can be named. The name of the packages and classes or interfaces should be such that it is easier to distinguish between the two. According to the conventions, the name of the packages starts with a lowercase

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 63

Programming in Java Dr. M. T. Wanjari

letter whereas the name of the classes begins with an uppercase letter. For example, consider this statement.

java.lang.Math

In this statement, the name of the package lang begins with lowercase letter and the name of the class Math begins with uppercase letter. However, as there are number of users working simultaneously on the internet, there may be a chance of duplicate packages being created which may lead to run-time error. Hence, package names must be unique. To ensure uniqueness of the package names, the name of the package can be preceded with the internet domain name. for example, consider this statement.

dpe.aet.package\_name

In this statement, the package package\_name is preceded by the domain name dpe.aet. **Defining Packages:**

To create a user-defined package, first a package must be declared using the keyword package followed by the name of the package.

The syntax to declare a package is

package package\_name;

where,

package is the Java keyword

package\_name is the name of the package

This statement must be the first statement in the Java source file. Once the package is declared, we can define any number of classes which will be the part of this package. For example, consider this code

package packagename;

public class Classname

{

//body of the class

}

Here, ClassName is the name of the class which will belong to the package packagename. In Java, packages are stored in the file system directories. The above code must be saved with the filename ClassName.java in a directory named packagename. When the Java compiler compiles the source file, the ClassName.class file is created and automatically stored in the same

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 64

Programming in Java Dr. M. T. Wanjari

directory packagename. Since Java is case sensitive, the name of the directory must exactly match with the package name.

Like API packages, user-define packages can be organized in a hierarchical structure. The package names forming part of this structure are separated by using dot ( .) operator. For example, consider this statement.

package mypackage1.mypackage2;

This package must be stored in mypackage1/ mypackage2 directory. By using multileveled package statement, the packages which are related to each other can be grouped into a single package.

If a Java package contains multiple classes, only one of them can be declared as public and the source file is saved with the name of public class having .java extension. When a source file having multiple class definitions is compiled, the compiler creates separate .class file for each class.

**Example: A program to demonstrate the creation of a package**

package mypackage;

class AreaRectangle

{

int length, breadth;

AreaRectangle(int l, int b)

{

length=l;

breadth=b;

int area=l\*b;

System.out.println(“The area of rectangle is: ” +area);

}

}

class SimplePackage

{

public static void main(String[] args)

{

AreaRectangle obj=new AreaRectangle(20, 30);

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 65

Programming in Java Dr. M. T. Wanjari

}

}

Save this file as SimplePackage.java under the directory called mypackage and compile this file. Now the file can be executed using command line as follows:

java mypackage.Simplepackage

We cannot execute it using the command line java SimplePackage because SimplePackage is now a part of the mypackage package.

**Accessing Packages:**

The user-defined packages are accessed in the same way as Java standard packages. for example, to access a class MyClassName of a package named mypackage1, following statements can be used.

import mypackage1.MyClassName; //satement1

or

import mypackage.\*; //statement2

Here, if statement 1 is used, all the members of MyClassName can be directly accessed without having to use the package name anywhere in the program. The asterisk ( \* ) in statement 2 specifies that that whenever the compiler has to find any class, it should search it in the package mypackage1. This allows all the classes of this package to be accessed directly. The main advantage of this approach is that the long package names need not be used repeatedly in the program. However, the disadvantage is that it becomes difficult to identify the package to which a particular member belongs.

**Using Packages:**

Once the required package is imported in the program, the classes belonging to that package can be used. To understand the concept of using the package, first let us create a package mypackagename1.

**Example: A program to create mypackagename1**

package mypackagename1;

public class MyClassName1

{

public void firstresult ( )

{

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 66

Programming in Java Dr. M. T. Wanjari

System.out.println(“This is first class’s result from the first package”);

}

}

Save the source file as MyClassName1.java in the subdirectory mypackagename1. After compiling this file, the corresponding compiled file MyClassName1.class will be stored in the same directory.

**Example: A program to demonstrate the use of package mypackagename1** import mypackagename1.MyClassName1;

class UsePackage

{

public static void main(String args[])

{

MyClassName1 obj=new MyClassName1 ( );

obj.firstresult ( );

}

}

**The output of the program is**

This is first class’s result from the first package

Now, consider another package mypackagename2.

**Example: A program to create mypackagename2**

package mypackagename2;

public class MyPackageName2

{

public void secondresult ( )

{

System.out.println(“This is second class’s result from the second package

”);

}

}

Example: A program to demonstrate importing multiple packages

import mypackagename1.MyClassName1; //importing MyClassName1

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 67

Programming in Java Dr. M. T. Wanjari

import mypackagename2.MyClassName2; //importing MyClassName2 class UseMultiPackage

{

public static void main(String[] args)

{

MyClassName1 obj1=new MyClassName1 ( );

MyClassName2 obj2=new MyClassName2 ( );

obj1.firstresult ( );

obj2.secondresult ( );

}

}

**The output of the program is**

This is first class’s result from the first package

This is second class’s result from the second package

When multiple packages are imported, there is a chance of having more than one package containing classes with the same name. For example, consider the definitions of two packages packagename1 and packagename2 having the same class name FirstClass as shown here. **Example: A program to create packagename1**

package packagename1;

public class FirstClass

{

int i=10;

int j=20;

public void dispaly1 ( )

{

System.out.println(“The value of i and j in first package is: “+i” and ”+j); }

}

**Example: A program to create packagename2**

package packagename2;

public class FirstClass

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 68

Programming in Java Dr. M. T. Wanjari

{

float p=3.4F;

float q=2.55F;

public void dispaly2 ( )

{

System.out.println(“The value of p and q in second package is: “+p+” and ”+q); }

}

Since both the packages contain the class FirstClass, the compiler will not be able to decide which package to use thereby causing an ambiguity. In such case, the class name must be preceded with the name of the package while creating the objects of the concerned class. **Example: A program to demonstrate importing multiple packages having classes with the same name**

import packagename1.\*;

import packagename2.\*;

class SameClassName

{

public static void main(String [] args)

{

packagename1.FirstClass obj1 =new packagname1.FirstClass ( );

obj1.display1 ( );

packagename2.FirstClass obj2 =new packagname2.FirstClass ( );

obj2.display2 ( );

}

}

**The output of the program is**

The value of i and j in first package is: 10 and 20

The value of p and q in second package is: 3.4 and 2.55

In this example, the class FirstClass in the package packagename1 has the same name as the class in the package packagename2. While creating the objects of the classes of each

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 69

Programming in Java Dr. M. T. Wanjari

package, the class name is preceded by their respective package names thereby causing no ambiguity.

**Adding a Class:**

A new class can be added to an already existing package. For example, consider a package called mypackage whose definition is as follows:

package mypackage;

public class MyClass1

{

//body of MyClass1

}

The package mypackage contains a public class MyClass1. Now, suppose we want to add another class MyClass2 to this package. As started earlier, a package cannot have more than one public class. If MyClass2 is non-public, then, simply we can add its definition to the same source file as follows:

package mypackage;

public class MyClass1 //existing class

{

// body of MyClass1

}

class MyClass2 //new class

{

// body of MyClass2

}

Then recompile this source file. The MyClass2.class file is created and stored automatically in the package mypackage. Now, mypackage contains two classes MyClass1 and MyClass2.

However, if the class MyClass2 is a public class, then we need to create this class in a separate source file and declare the package statement at the top of the source file as shown here. package mypackage;

public class MyClass2

{

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 70

Programming in Java Dr. M. T. Wanjari

// body of MyClass2

}

Compile this source file. The package mypackage will now contain MyClass2.class file also. Thus, if we want to create multiple public classes in a package, it is required to create a separate source file for each class and compile them. After compilation, the package will contain .class files of all the source files.

**Exception Handling:**

**/\* Exception is a condition caused by run-time error\*/**

➢ Exception handling provides a powerful mechanism for controlling complex programs that have many dynamic run time characteristics.

➢ It important to use try, catch, throw: to handle errors and unusual boundary condition in programs logic.

➢ When a method can fail to returning an error code, it is important to throw an exception. **Uncaught Exceptions**

class Expdemo1

{

public static void main(String args[])

{

int d=0;

int a=42/d;

}

}

➢ In this program includes an exception that intentionally causes divide-by-zero. ➢ When the java run time system detects the attempts to divide by zero, it constructs a new exception object and then throws this exception.

➢ This cause the execution of Expdemo1 to stop because once an exception has been thrown, it must be caught by an exception handler.

➢ But, in above program we have not supplied any handlers of our own, so that exception is caught by the default handler provides by java-runtime system.

➢ The default handler displays a string describing the exception prints a stack trace from the point at which exception is occurred and terminates the program.

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 71

Programming in Java Dr. M. T. Wanjari

➢ Here, is the output when above program is executed.

|  |
| --- |
| java.lang.ArithmeticException:/by zero at Expdemo1.main(Exception.java.4) |

➢ Classname : Expdemo1

mehtodname : main included in simple stack trace

filename : Expdemo1.java

Line no : 4

And ArithmeticException is a subclass of Exception which more specifically describes what type of error is happened.

**Using try and catch**

➢ To handle the exception manually rather than using default exception handler. ➢ There are two benefits to handle exception manually

i) It allows the programmer to fix the error

ii) It provides the program from automatically terminating

➢ To guard against and handle to run-time error, simply enclose the code inside a try block ➢ And immediately following the try block, include a catch clause that specifies the exception type which is to catch

➢ For example

class Expdemo2

{

public static void main(String args[])

{

int d,a;

try

{

d=0;

a=52/d;

System.out.println(“This is not executed”);

}

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 72

Programming in Java Dr. M. T. Wanjari

catch(ArithmeticException e1) //catch divide-by-zero error

{

System.out.println(“Divide by Zero error: catch block executed.”);

}

System.out.println(“Outside catch block”);

}

}

➢ println() inside try block is never executed

➢ Once an exception is thrown by try block then it is caught by catch block and never goes back to try block & after caught an exception i.e. execution of catch block it goes to next statement followed by catch block.

➢ So in above program after displaying statement “Division by zero error: catch block executed” it displays the statement “Outside catch block” and terminate the program. ➢ A try and its catch statement from a unit. There is a single try block for single catch or multiple catch block. There is no use of multiple try blocks. It means one try can have multiple catch but catch have not multiple try.

**Multiple catch clauses**

➢ If more than one exception could be raised by a single program to handle this situation, specify two or more catch blocks in single program for each one catching different types of exception.

➢ When an exception is thrown each catch statement is executed or followed by given order and the first one whose type matches that of the exception is executed. class Multicatch

{

public static void main(String args[])

{

try

{

int a=0; b=42;

System.out.println(“a= ”+a+”b=”+b);

int c=b/a;

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 73

Programming in Java Dr. M. T. Wanjari

int arr[] = {1, 2};

arr[30]=50;

}

catch(ArithmeticException AE1)

{

System.out.println(“Divide by zero error”);

}

catch(ArrayIndexOutOfBoundsException ArrE2)

{

System.out.println(“Array index is not available error”);

}

}

}

➢ When multiple catch statements are used then it is important to remember that exception subclasses must come before any of their superclasses.

**Nested try Statements**

➢ Multiple try statement are not allowed but try statement can be nested i.e. try statement can be inside the block of another try.

➢ Each time a try statement is entered, the context (area) of that exception is pushed on the stack.

➢ If an inner try statement does not have a catch handler for a particular exception the stack is unwound and the next try statements catch handlers are checked for a particular match.

➢ This process is continuous until one of the catch statements is successfully executed until all of the nested try statements are completed.

➢ If no catch statement matches then the java run time system will handle the exception. ➢ For eg.

class Nesttry

{

public static void main(String args[])

{

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 74

Programming in Java Dr. M. T. Wanjari

try

{

/\*if no command-line argument are present the following statement will generate a divide by zero exception\*/

int a=args.length;

int b=88/a;

System.out.println(“a=”+a);

try

{

/\*if one command-line argument is used then a divide-by-zero exception will be generated by following code\*/

if(a==1)

{

int arr[]={1};

arr[40]=50;

}

}

catch(ArrayIndexOutOfBoumdsException e1)

{

System.out.println(“ArrayIndexOutOfBoumd error”);

}

}

catch(ArithmeticException e2)

{

System.out.println(“Divide by zero error” +e2);

}

}

}

**throw**

➢ Throw keyword is used to throw an exception explicitly.

➢ The general form of throw is:

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 75

Programming in Java Dr. M. T. Wanjari

throw ThrowableInstance;

➢ ThrowableInstance is an object of class Throwable or a subclass of Throwable. ➢ There are two ways to obtain a Throwable object:

i) Using a parameter into a catch clause or

ii) Creating one with the new operator.

➢ The flow of execution stops immediately after the throw statement any subsequent statements are not executed.

➢ For eg.

class ThrowDemo

{

Static void fun1()

{

try

{

throw new NullPointerException(“obj1”);

}

catch(NullPointerException e1)

{

System.out.println(“caught an error inside fun1()”);

Throw e1; //rethrow an exception

}

}

public static void main(String args[])

{

try

{

fun1();

}

catch(NullPointerException e)

{

System.out.println(“Recaught:” +e1);

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 76

Programming in Java Dr. M. T. Wanjari

}

}

}

➢ This program gets two chances to deal with error

i) main() setup an exception context and then calls fun1().

ii) fun1() method then sets up another exception-handling context and immediately throws a new instance of NullPointerException, which is caught on the next line ➢ The exception is rethrown

throw new NullPointerException(“obj1”);

➢ In above program this above statement new is used to construct an instance of NullPointerException which is a subclass of Throwable.

➢ All of java’s built-in runtime exception have at least two constructor. One with no parameter and one that takes a string parameter.

➢ When the second form is used, the argument specifies a string that describes the exception.

**throws**

➢ If a method is capable of causing an exception that it dies not handle, it must specify this behavior so that called of the method can guard themselves against that exception for this throws clause is including in the method’s declaration.

class Expdemo

{

public static void main(String args[])

{

try

{

test();

}

catch(ArithmeticException e)

{

System.out.println(“Exception”);

}

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 77

Programming in Java Dr. M. T. Wanjari

}

Static void test() throws ArithmeticExeption

{

int a=2, b=0, c;

c=a/b;

}

}

**finally**

➢ finally is used to execute the something forcefully & it is optional.

➢ finally creates a block of code that will be executed after try/catch block has completed and before the code following the try/catch block

➢ the finally block will execute whether or not an exception is thrown.

➢ Each try requires at least one catch or a finally clause.

class FinallyDemo

{

static void test1()

{

try

{

System.out.println(“Inside test1: No Exception is thrown”);

}

finally

{

System.out.println(“test1: finally(): No Exception”);

}

}

static void test2()

{

int a,b,c;

try

{

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 78

Programming in Java Dr. M. T. Wanjari

A=20, b=0;

C=a/b;

System.out.println(“Divide by zero: Arithmetic Exception”);

}

catch(ArithmeticException e)

{

System.out.println(“Divide by zero error: Arithmetic Exception”);

}

}

public static void main(String args[])

{

try

{

test1();

}

catch(Exception e)

{

System.out.println(“Exception caught”);

}

test2();

}

}

**Creating your own Exception Subclasses**

➢ Java’s built-in exceptions handle most common errors.

➢ To create own exception types to handle situations specific to applications to do so justdefine a subclass of Exception.

class MyException extends Exception

{

private int detail;

MyException(int a)

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 79

Programming in Java Dr. M. T. Wanjari

{

detail =a;

}

public string tostring()

{

return “MyException[“+detail+”]”;

}

}

class Expdemo

{

static void compute(int a) throws MyException {

System.out.println(“called compute(“+a+”)”); int (a>10)

{

throw new MyException(a)l

}

System.out.println(“Normal exit”);

}

public static void main(String args[]) {

try

{

compute(1);

compute(20);

}

catch(MyException e)

{

System.out.println(“Caught” +e);

}

}

M.Sc. Semester – I (Computer Science)…..Shivaji Science College, Congress Nagar, Nagpur 80