***Object Oriented Programming in JAVA:***

***Introduction***

• Welcome to the course Advance Object Oriented Programming in JAVA. This course will cover a core set of computer science concepts needed to create a modern software application using Java.

Course Objectives On completion of this course we will be able to:

1. Identify the importance of Java . 2. Identify the additional features of Java compared to C++ . 3. Identify the difference between Compiler and Interpreter . 4. Identify the difference between applet and application. 5. Apply Advance Object Oriented Principles of Encapsulations, Data abstraction, Inheritance, Polymorphism. 6. Program using java API (Application Programming Interface). 7. Program using Exception Handling, Files and Threads . 8. Program Using applets and swings .

***JAVA Basics***

***Why Java is Important***

• Two reasons : – Trouble with C/C++ language is that they are not portable and are not platform independent languages. – Emergence of World Wide Web, which demanded portable programs • Portability and security necessitated the invention of Java

History • James Gosling -Sun Microsystems • Co founder –Vinod Khosla • Oak -Java, May 20, 1995, Sun World • JDK Evolutions – JDK 1.0 (January 23, 1996) – JDK 1.1 (February 19, 1997) – J2SE 1.2 (December 8, 1998) – J2SE 1.3 (May 8, 2000) – J2SE 1.4 (February 6, 2002) – J2SE 5.0 (September 30, 2004) – Java SE 6 (December 11, 2006) – Java SE 7 (July 28, 2011)

Cont..

• Java Editions. ➢J2SE(Java 2 Standard Edition) - to develop client-side standalone applications or applets. ➢J2ME(Java 2 Micro Edition ) - to develop applications for mobile devices such as cell phones. ➢J2EE(Java 2 Enterprise Edition ) - to develop server-side applications such as Java servlets and Java Server Pages.

***What is java?***

• A general-purpose object-oriented language. • Write Once Run Anywhere (WORA). • Designed for easy Web/Internet applications.

**How is Java different from C…**

• C Language: – Major difference is that C is a structure oriented language and Java is an object oriented language and has mechanism to define classes and objects. – Java does not support pointer type – Java does not have preprocessor, so we cant use #define, #include and #if def statements. – Java does not include structures, unions and enum data types. – Java does not include keywords like goto, sizeof and type def. – Java adds labeled break and continue statements. – Java adds many features required for object oriented programming.

**How is Java different from C++…**

• C++ language Features removed in java: ➢Java doesn’t support pointer sto avoid unauthorized access of memory locations. ➢Java does not include structures, unions and enum data types. ➢Java does not support operator overloading. ➢Preprocessor plays less important role in C++ and so eliminatedentirelyinjava. ➢Java does not perform automatic type conversions that resultin loss of precision.

Cont… ➢ Java does not support global variables. Every method and variable is declared within a class and forms part of that class. ➢ Java does not support inheritance of multiple super classes by a sub class (i.e., multiple inheritance). This is accomplished by using ‘interface’concept. ➢It is not possible to declare unsigned integers(Unsigned can hold a larger positive value, and no negative value. ➢In java objects are passed by reference only. In C++ objects may be passed by value or reference.

Cont …

**New features added in Java:**

➢Multithreading, that allows two or more pieces of the same program to execute concurrently. ➢C++ has a set of library functions that use a common header file. But java replaces it with its own set of API(Application Programming Interface)classes. ➢It add spackages and interfaces. ➢Java supports auto maticgarbage collection. ➢break and continue statements have been enhanced in java to accept labels astargets.

Cont …

***Features that differ***:

➢ Though C++ and java supports Boolean data type, C++ takes any nonzero value as true and zero as false. True and false in java are predefined literals that are values for a boolean expression. ➢ Java has replaced the destructor function with a finalize() function. ➢ C++ supports exception handling that is similar to java's. However, in C++ there is no requirement that a thrown exception be caught.

***Characteristics of Java***

• Java issimple • Java is object-oriented • Java is distributed • Java is interpreted • Java is robust

• Java is architecture-neutral • Java isportable • Java’s performance • Java is multithreaded • Java is dynamic • Java issecure

• Simple According to Sun, Java language is simple because syntax is based on C++ (so easier for programmers to learn it after C++). It removed many confusing and/or rarely-used features e.g., explicit pointers, operator overloading etc. No need to remove unreferenced objects because there is Automatic Garbage Collection In java.

• Object-oriented Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behavior . Object-oriented programming(OOPs) is a methodology that simplify software development and maintenance by providing some rules. • Platform Independent

Aplat form is the hardware or software environment in which a program runs. There are two types of platforms software-based and hardware-based. Java provides software-based platform. The Java platform differs from most other platform sin thesense that it is a software-based platform that run so nthetopofo there hardware-based platforms.

• Secured Java is secured because: No explicit pointer Java Programs run inside virtual machine sandbox • Robust Robust simply means strong. Java uses strong memory management .There are lack of pointers that avoids security problem. There is automatic garbage collection in java. There is exception handling and type checking mechanism in java .All these points makes java robust.

• Architecture-neutral There is no implementation dependent features e.g. size of primitivety pesis fixed. In C programming, in data type occupies 2 bytes of memory for 32bit architecture and 4 bytes of memory for 64-bit architecture. But in java, it occupies 4 bytes of memory for both 32 and 64 bit architectures.

• Portable We may carry the java bytecode to any platform. • High-performance Java is faster than traditional interpretation since byte code is "close" to native code still somewhat slower than a compiled language (e.g., C++) • Distributed We can create distributed applications in java. RMI and EJB are used for creating distributed applications. We may access files by calling the methods from any machine on the internet. •Interpreted The compiler takes your .java file and compiles it into a .class file (the .class file contains Java byte code). The interpreter comes in when your program is run. The JVM (or interpreter) takes your .class file and interprets it. • Multi-threaded A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications etc.

Java Environment

• Java includes many development tools, classes and methods – Development tools are part of Java Development Kit (JDK) and – The classes and methods are part of Java Standard Library (JSL), also known as Application Programming Interface(API). • JDK constitutes of tools like java compiler, java interpreter and many. • APIincludes hundreds of classes and methods grouped into several packages according to their functionality.

Example

• File: HelloWorldApp.java

public class HelloWorldApp{

public static void main(String[] args) {

System.out.println("Hello world");}}

Things to remember

• Name of file must match name of class – It is case sensitive

• Processing starts in main – public static void main(String[] args)

• Printing is done with System.out – System.out.println, System.out.print

An idiom explained

• You will see the following line of code often: – public static void main(String args[]) { …}

• About main() – “main” is the function from which your program starts

– Why public? • So that run time can call it from outside

– Why static ? • it is made static so that we can call it without creating an object

– What is String args[] ? • Way of specifying input at startup of application

Things to Remember

• “+” operator when used with Strings concatenates them – System.out.pritln(“Hello” + “World”) will produce Hello World on console

– String concatenated with any other data type such as int will also convert that datatype to String and the result will be a concatenated String displayed on console • For Example – int i = 4 – int j = 5 ; – System .out.println (“Hello” + i) // will print Hello 4 on screen • However – System,.out..println( i+j) ; // will print 9 on the console

– For comparing Strings never use == operator, use equals methos. • == compares addresses (shallow comparison) while equals compares values (deep comparison) • E.g string1.equals(string2)

public class StringTest {

public static void main(String[] args) { int i = 4; int j = 5;

System.out.println("Hello" + i); System.out.println(i + j);

String s1 = new String (“pakistan”); String s2 = “pakistan”;

if (s1 == s2) { System.out.println(“comparing string using == operator”); }

if (s1.equals( s2) ) { System.out.println(“comparing string using equal method”); } }

}

String Concatenation

Primitives Vs. Objects

Primitives Vs. Objects

• Everything in Java is an “Object”, as every class by default inherits from class “Object” , except a few primitive data types, which are there for efficiency reasons.

• Primitive Data Types – 8 Primitive Data types of java • boolean, byte  1 byte • char, short  2 bytes • int, float  4 bytes • long, double  8 bytes

• Primitive data types are generally used for local variables, parameters and instance variables (properties of an object)

• Primitive datatypes are located on the stack and we can only access their value, while objects are located on heap and we have a reference to these objects

• Also primitive data types are always passed by value while objects are always passed by reference in java. There is no C++ like methods – void some Method(int &a, int & b ) // not available in java

Primitives (cont)

• For all built-in primitive data types java uses lowercase. E.g int , float etc

• Primitives can be stored in arrays

• You cannot get a reference to a primitive

Input / Output

Java Inputting

2/13/2018 Object Oriented Programming using JAVA 29

▪ There are various ways to read input from the keyboard, the java.util.Scanner classis one of them.

▪ The Java Scanner class breaks the input into tokens using a delimiter that is whitespace by default. It provides many methods to read and parsevarious primitive values.

▪ Java Scanner class is widely used to parse text for string and primitive types using regular expression.

▪ Java Scanner class extends Object class and implements Iterator and Close able inter faces.

Console based Output System.out

• System class – Out represents the screen • System.out.println() – Prints the string followed by an end of line • System.out.print() – Does not print the end of line.

***Data types, expressions & assignment statements:***

Some Escape Sequences

**‘\n’ Newline (Line feed in ASCII)**

**‘\t’ Horizontal tab**

**‘\b’ Backspace**

***Choosing variable names***

* The name of each variable should describe the value to be stored

The goal is to make your code self-documenting – naming should make the purpose of both data declarations and subsequent instructions apparent

Examples:

|  |  |
| --- | --- |
| Good | Bad |
| int userAge; // age in years  double acctBalance;  // account balance, in dollars | int x;  double y;  boolean z; |

***Examples from Java API***

* Some Java standard classes:
  + String, Math, System, Random
  + InterruptedException, JTextField
* Some standard objects:
  + out, in
* Some standard methods:
  + println, readLine, parseInt

***Variable declaration & assignment***

* A variable must be declared before it is used in any other statement
* A variable must be ***initialized*** (assigned a value) before it is used in an expression
* Declaration and initialization can occur at the same time:

int age = 21;

* Or in separate lines of code:

char mInitial;

mInitial = ‘M’;

***Assignment statements:***

* Once a variable is declared, it can be assigned values multiple times in a program; for example:

int num; // num declared, uninitialized

num = 3; // num assigned 3

num = num + 1; // num assigned 3 + 1 (4)

num = 15 / num; // num assigned 15 / 4 (3)

***Assignment is not equality!***

* On the previous slide, we saw a couple of examples of perfectly legal Java statements that don’t make sense algebraically:

num = num + 1;

num = 15 / num;

* Remember, the operator “=” is pronounced “gets,” not “equals,” in Java
  + The expression to the right of the operator is evaluated first

***Assignment compatibility***

* When a variable is declared, the data type in the declaration indicates the nature of the values the variable is capable of storing
* For example, an int variable can store a whole number, a char variable can store a character, and a double variable can store a real number
* The value assigned to the variable must be compatible with its data type

***Assignment compatibility***

* Java is a ***strongly-typed*** language
* This means that, for the most part, you can only assign a value of a particular data type to a variable of the same type
* Some examples:

int x = -2; // this is fine; -2 is an integer

char c = ‘2’; // this is OK also

x = 2.5; // syntax error: 2.5 is a double value, not an int

c = 2; // also an error; 2 is an int value, not a char

* The last two lines of code on the previous slide were examples of errors the compiler would flag because they violate a rule of the Java language
* The rule is that a value can’t be “demoted” in an assignment; in particular:
  + A floating-point value can’t be assigned to an int variable
  + A numeric value can’t be assigned to a char

***Arithmetic operators in Java***

* The arithmetic operators in Java are:

+ Addition

* + Subtraction

\* Multiplication

/ Division

% Modulus (remainder)

* These operators can be used with simple expressions (e.g. variables, literal values) to form compound expressions
* As in algebra, multiplication and division (and modulus, which we’ll look at momentarily) take precedence over addition and subtraction
* We can form larger expressions by adding more operators and more operands
  + Parentheses are used to group expressions, using the same rule as in algebra: evaluate the innermost parenthesized expression first, and work your way out through the levels of nesting

Example: **int x = 4, y = 9, z;**

**z = x + y \* 2; // result is 22**

**z = (x + y) \* 2; // result is 26**

**y = y – 1; // result is 8**

***Operator precedence***

* The order in which operations are performed depends upon the order in which they are written and their relative precedence
* Unary negative takes precedence over the binary operators, while multiplication, division and modulus have precedence over addition and subtraction

***Associativity:***

* **left to right Associativity means that in an expression having 2 operators with the same priority, the left operator is applied first**
* **in Java the binary operators \*, /, %, +, - are all left associative**
* **expression 9 - 5 - 1 means ( 9 - 5 ) - 1**

**4 - 1**

**3**

***Parentheses:***

* Use of parentheses can change the order in which an expression is evaluated; for example, the expression:

4 + 2 \* 3 - 10 / 2

evaluates to 5; first 2 is multiplied by 3, then 10 is divided by 2; 4 is added to 6, and finally 5 is subtracted

with parentheses:

((4 + 2) \* (3 - 10)) / 2 produces -21

while (4 + 2) \* 3 - 10 / 2 produces 13

***Evaluate the Expression:***

**7 \* 10 - 5 % 3 \* 4 + 9**

**means (7 \* 10) - 5 % 3 \* 4 + 9**

**70 - 5 % 3 \* 4 + 9**

**70 - (5 % 3) \* 4 + 9**

**70 - 2 \* 4 + 9**

**70 - ( 2 \* 4 ) + 9**

**70 - 8 + 9**

**( 70 - 8 ) + 9**

**62 + 9**

**71**

Parentheses

* parentheses can be used to change the usual order
* parts in ( ) are evaluated first
* evaluate **(7 \* (10 - 5) % 3) \* 4 + 9**

**( 7 \* 5 % 3 ) \* 4 + 9**

**( 35 % 3 ) \* 4 + 9**

**2 \* 4 + 9**

**8 + 9**

**17**

***Importance of statement order:***

* As previously mentioned, it is important to initialize a variable before its use in an expression
* Failure to do so may result in a logic error, as in the example below:

int x, y, z;

x = y + z; // what value is stored in x?

y = 5;

z = 2;

***Integer division:***

* When one real number is divided by another, the result is a real number; for example:

**double x = 5.2, y = 2.0, z;**

**z = x / y; // result is 2.6**

* When dividing integers, we get an integer result
* For example:

**int x = 4, y = 9, z;**

**z = x / 2; // result is 2**

**z = y / x; // result is 2, again**

**z = x / y; // result is 0**

* There are two ways to divide integers
  + using **the / operator, produces the quotient of the two operands**
  + using **the % operator, produces the remainder** when the operands are divided. This is called **modular division**, or **modulus** (often abbreviated mod). For example:

**int x = 4, y = 9, z;**

**z = x % 2; // result is 0**

**z = y % x; // result is 1**

**z = x % y; // result is 4**

***Compound arithmetic/assignment operators:***

* Previous examples in the notes have included the following statements:

**y = y + 1;**

**y = y / 3;**

* In each case, the current value of the variable is used to evaluate the expression, and the resulting value is assigned to the variable (erasing the previously-stored value)
* This type of operation is extremely common; so much so, that Java (like C++ and C before it) provides a set of shorthand operators to perform this type of operation. The table on the next slide illustrates the use and meaning of these operators

***Compound arithmetic/assignment operators:;***

|  |  |  |
| --- | --- | --- |
| ***Operator*** | ***Use*** | ***Meaning*** |
| ***+=*** | ***X += 1;*** | ***X = X + 1;*** |
| ***-=*** | ***X -= 1;*** | ***X = X – 1;*** |
| ***\*=*** | ***X \*= 5;*** | ***X = X \* 5;*** |
| ***/=*** | ***X /= 2;*** | ***X = X / 2;*** |
| ***%=*** | ***X %= 10;*** | ***X = X % 10;*** |

***Increment and decrement:***

* Once an int variable has been initialized, a common operation performed on that variable is to add or subtract 1 from its value, then assign the result back to the variable, as in the example below:

int num = 0;

num = num + 1;

* Since the expression is evaluated first, the initial value of num (0) is added to 1, and the result (1) is assigned back to num
* Java provides a shorthand method for performing this common increment operation; the example below illustrates the shortcut:

int num = 0;

num++; /\* has same effect as num = num + 1 \*/

* A similar shortcut exists for subtracting 1, or decrementing, an int variable, as in the example below:

int countdown = 10;

countdown--;

* The increment and decrement operators (++ and --, respectively) have two forms: prefix and postfix
* All of the examples thus far have been the postfix form: num++; and countdown--;
* The prefix form places the operator before the variable: e.g. ++num and --countdown

***Prefix vs. postscript:***

* In the examples we’ve seen, it doesn’t make any difference which form is used
* When increment or decrement is used in a larger expression, however, the form used can change the outcome
* If the prefix form is used, the new value (plus or minus 1) is used in the larger expression; if the postfix form is used, the variable is not incremented or decremented until after it has been used to evaluate the larger expression
* In the example below, values assigned to variable f depend on when new values get assigned to e:

int e=0, f;

f = e++; /\* f gets 0; e gets 1 \*/

f = ++e; /\* f gets 2; e gets 2 \*/

***Named constants:***

* A variable is a named memory location that can hold a value of a specific data type; as we have seen, the value stored at this location can change throughout the execution of a program
* If we want to maintain a value in a named location, we use the Java keyword ***final*** in the declaration and immediately assign the desired value; with this mechanism, we declare a named constant. Some examples:

**final int LUCKY = 7;**

**final double PI = 3.14159;**

**final double LIGHTSPEED = 3.0e10.0** ;

* The name of the constant is used in expressions but cannot be assigned a new value. For example, to calculate the value of variable circleArea using the variable radius and the value π, we could write:

circleArea = 2 \* PI \* radius \* radius;

* The use of named constants is considered good programming practice, because it:
  + eliminates (or at least minimizes) the use of “magic” numbers in a program; it is easier to read code that contains meaningful names
  + allows a programmer to make global changes in calculations easily

**Basic Syntax**

* **Object -** Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behaviors -wagging, barking, eating. An object is an instance of a class.
* **Class -** A class can be defined as a template/ blue print that describes the behaviours/states that object of its type support.
* **Methods -** A method is basically a behaviour. A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.
* **Instance Variables -** Each object has its unique set of instance variables. An object's state is created by the values assigned to these instance variables.
* About Java programs, it is very important to keep in mind the following points.
* **Case Sensitivity -** Java is case sensitive, which means identifier **Hello** and **hello** would have different meaning in Java.
* **Class Names -** For all class names the first letter should be in Upper Case.   If several words are used to form a name of the class, each inner word's first letter should be in Upper Case.  Example *class MyFirstJavaClass*
* **Method Names -** All method names should start with a Lower Case letter.   If several words are used to form the name of the method, then each inner word's first letter should be in Upper Case.  Example *public void myMethodName()*
* **Program File Name -** Name of the program file should exactly match the class name.   When saving the file, you should save it using the class name (Remember Java is case sensitive) and append '.java' to the end of the name (if the file name and the class name do not match your program will not compile).  Example: Assume 'MyFirstJavaProgram' is the class name. Then the file should be saved as *'MyFirstJavaProgram.java'*
* **public static void main(String args[]) -** Java program processing starts from the main() method which is a mandatory part of every Java program.
* Java Identifiers:
* All Java components require names. Names used for classes, variables and methods are called identifiers.
* In Java, there are several points to remember about identifiers. They are as follows:
* All identifiers should begin with a letter (A to Z or a to z), currency character ($) or an underscore (\_).
* After the first character identifiers can have any combination of characters.
* A key word cannot be used as an identifier.
* Most importantly identifiers are case sensitive.
* Examples of legal identifiers: age, $salary, \_value, \_\_1\_value
* Examples of illegal identifiers: 123abc, -salary

***Java Keywords:***

The following list shows the reserved words in Java. These reserved words may not be used as constant or variable or any other identifier names.

***abstract assert boolean break***

***byte case catch char***

***class const continue default***

***do double else enum***

***extends final finally float***

***for goto if implement***

***import instanceof int Interface***

***long native new package***

***Private protected public return***

***short static strictfp super***

***switch synchronized this throw***

***Throws transient try void***

***Volatile while***

***Using Blank Lines:***

* A line containing only white space, possibly with a comment, is known as a blank line, and Java totally ignores it.

***ARRAYS IN JAVA:***

***Declaring an Array Variable:***

* Do not have to create an array while declaring array variable
  + *<datatype>* [] variable\_name;
  + *int* [] prime;
  + *int* prime[];
* Both syntaxes are equivalent
* No memory allocation at this point

***Defining an ArrayDefining an Array:***

* Define an array as follows:
  + variable\_name=new *<type>*[N];

int primes=new int[10];

* Declaring and defining in the same statement:
  + *int*[] primes=new *int*[10];
* In JAVA, int is of 4 bytes, total space=4\*10=40 bytes

***Default Initialization:***

* When array is created, array elements are initialized
  + Numeric values (int, double, etc.) to 0
  + Boolean values to false

***Accessing Array Elements:***

* Index of an array is defined as
  + Positive int, byte or short values
* Any other types used for index will give error
  + long, double, etc.
* Indexing starts from 0 and ends at N-1

primes[2]=0;

int k = primes[2];

***Validating Indexes:***

* JAVA checks whether the index values are valid at runtime
  + If index is negative or greater than the size of the array then an IndexOutOfBoundException will be thrown
  + Program will normally be terminated unless handled in the try {} catch {}

***What happens if …***

long[] primes = new long[20];

primes[25]=33;

….

*Runtime Error:*

Exception in thread “main” java. Lang .Array Index Out Of Bounds Exception: 25

at More Primes .main(MorePrimes.java:6)

***Reusing Array Variables:***

* Array variable is separate from array itself
  + Like a variable can refer to different values at different points in the program
  + Use array variables to access different arrays

int[] primes=new int[10];

……

primes=new int[50];

* Previous array will be discarded
* Cannot alter the type of array

***Initializing Arrays***

* Initialize and specify size of array while declaring an array variable

int[] primes={2,3,5,7,11,13,17}; //7 elements

* You can initialize array with an existing array

int[] even={2,4,6,8,10};

int[] value=even;

* + One array but two array variables!
  + Both array variables refer to the same array
  + Array can be accessed through either variable name

***Array Length***

* Refer to array length using *length*
  + A data member of array object
  + array\_variable\_name.length
  + for(int k=0; k<primes.length;k++)

**….**

* **Sample Code:**

long[] primes = new long[20];

System.out.println(primes.length);

* **Output: 20**

***Change in Array Length:***

* If number of elements in the array are changed, JAVA will automatically change the length attribute!

***Sample Program***

class MinAlgorithm

{

public static void main ( String[] args )

{

int[] array = { -20, 19, 1, 5, -1, 27, 19, 5 } ;

int min=array[0]; // initialize the current minimum

for ( int index=0; index < array.length; index++ )

if ( min > array[ index ] )

min = array[ index ] ;

System.out.println("The minimum of this array is: " + min );

}

System.out.println(“Testing”)

}

***Copying an Array***

public class A {

public static void main(String args[]) {

int a[] = { 1, 2, 3, 4, 5, 6 };

int b[] = new int[a.length]; //new int[6];

for (int i = 0; i < a.length; i++) {

b[i] = a[i];

System.out.println(b[i]);

}

}

}

***For sorting an Array***

import java.util.Arrays;

public class Program {

public static void main(String[] args) {

int[] array = { 100, 20, 0, 200 };

// Call Arrays.sort on the int array.

Arrays.sort(array);

for (int elem : array) {

System.out.println(elem);

}

}

}

***Filling an Array***

import java.util.Arrays;

public class Program {

public static void main(String[] args) {

int[] values = new int[10];

// Fill array with this number.

**Arrays.fill**(values, 5);

for (int value : values) {

System.out.print(value);

System.out.print(' ');

}

}

}

***Display Array in Reverse***

public class Program {

public static void main(String[] args) {

**boolean[]** values = { false, true, true, true };

// Loop over array elements in reverse order.

**for** (int i = values.length - 1; i >= 0; i--) {

System.out.println(values[i]);

}

}

}

***Array of Arrays***

* Two-Dimensional arrays
  + float[][] temperature=new float[10][365];
  + 10 arrays each having 365 elements
  + First index: specifies array (row)
  + Second Index: specifies element in that array (column)
  + In JAVA float is 4 bytes, total Size=4\*10\*365=14,600 bytes

***Initializing Array of Arrays***

int[][] uneven = { { 1, 9, 4 }, { 0, 2}, { 0, 1, 2, 3, 4 } };

//Three arrays

//First array has 3 elements

//Second array has 2 elements

//Third array has 5 elements

***Initializing Array of Arrays of Varying Length***

***??????????????????????????***

***Sample Program:***

class unevenExample3

{

public static void main( String[] arg )

{ // declare and construct a 2D array

int[][] uneven = { { 1, 9, 4 }, { 0, 2}, { 0, 1, 2, 3, 4 } };

// print out the array

for ( int row=0; row < uneven.length; row++ ) //changes row

{

System.out.print("Row " + row + ": ");

for ( int col=0; col < uneven[row].length; col++ ) //changes column

System.out.print( uneven[row][col] + " "); System.out.println();

}

}

}

***OUTPUT:***

Row 0: 1 9 4

Row 1: 0 2

Row 2: 0 1 2 3 4

**Dispalying array elements in 2 dim array.**

class TwoDArray {   
public static void main(String args[]) {   
int twoD[][]= new int[4][5];   
int i, j, k = 0;   
for(i=0; i<4; i++)   
for(j=0; j<5; j++) {   
twoD[i][j] = k;   
k++;   
}   
for(i=0; i<4; i++) {   
for(j=0; j<5; j++)   
System.out.print(twoD[i][j] + " ");   
System.out.println();   
}   
}   
}

***OUTPUT:***

1 2 3 4

5 6 7 8

9 10 11 12 13 14

15 16 17 18 19

***3 Dimensional Array***

* A 3-D array can be considered as an array of a 2-Dimensional array and that's very simple to understand. Consider an example ,say we have an array declaration int A[2][3][2],that's a 3-D array.
* Read it like this: int (A[2])[3][2] (Note the braces),we say that 'A' is made of 2 pages ,3 rows and 2 columns each, in all we have total=2\*3\*2=12 elements in our array but this is just a logical explanation ,physically an array be it any dimensional ,is always stored sequentially as memory is Linear.
* A 3 dimensional array program that will display numbers in different manner.
* class threeDMatrix {
* public static void main(String args[]) {
* int threeD[][][] = new int[3][4][5];
* int i, j, k;
* for(i=0; i<3; i++)
* for(j=0; j<4; j++)
* for(k=0; k<5; k++)
* threeD[i][j][k] = i \* j \* k;
* for(i=0; i<3; i++) {
* for(j=0; j<4; j++) {
* for(k=0; k<5; k++)
* System.out.print(threeD[i][j][k] + " ");
* System.out.println();
* }
* System.out.println();
* }
* }
* }
* Output:
* 0 0 0 0 0
* 0 0 0 0 0
* 0 0 0 0 0
* 0 0 0 0 0
* 0 0 0 0 0
* 0 1 2 3 4
* 0 2 4 6 8
* 0 3 6 9 12
* 0 0 0 0 0
* 0 2 4 6 8
* 0 4 8 12 16
* 0 6 12 18 24

***Loops In Arrays:***

for loop in java public class Test {

public static void main(String args[]) {

for(int x = 10; x < 20; x = x + 1) { System.out.print("value of x : " + x ); System.out.print("\n"); } } } ==================================================== for each loop in java public class Test {

public static void main(String args[]) { int [] numbers = {10, 20, 30, 40, 50};

for(int x : numbers ) { System.out.print( x ); System.out.print(","); } System.out.print("\n"); String [] names = {"James", "Larry", "Tom", "Lacy"};

for( String name : names ) { System.out.print( name ); System.out.print(","); } } } ======================================================= while Loop in java public class Test {

public static void main(String args[]) { int x = 10;

while( x < 20 ) { System.out.print("value of x : " + x ); x++; System.out.print("\n"); } } } ======================================================== do while loop in java public class Test {

public static void main(String args[]) { int x = 10;

do {

System.out.print("value of x : " + x ); x++; System.out.print("\n"); }while( x < 20 ); } } ========================================================= == scanner input in the Array

package javaapplication10; import java.util.Scanner;

public static void main(String[] args) { { Scanner input = new Scanner(System.in); int[] a = new int[5];

for (int i = 0; i < a.length; i++) { System.out.println("Please enter number"); a[i] = input.nextInt(); } System.out.println("Numbers are"); for (int i = 0; i <5; i++) { System.out.println(a[i]); } } } } ========================================================= ========= scanner input in 2-D Array public static void main(String[] args) { { Scanner input = new Scanner(System.in); int[][] a = new int[3][2];

for (int i = 0; i < 3; i++) for (int j = 0; j < 2; j++) { System.out.println("Please enter number"); a[i][j] = input.nextInt(); } System.out.println("Numbers are"); for (int i = 0; i < 3; i++) for (int j = 0; j < 2; j++) {

System.out.println(a[i][j]); } } } }

***Objects and Classes in Java:***

***Object and class***

* A class is a blueprint
* An object is an instance created from that blueprint

Classes are symbolic representations of objects; classes describe the properties, fields, methods, and events that form objects in the same way that blueprints describe the items that form buildings.

* All objects of the same class have the same set of attributes
  + Every Person object have name, weight, height
* But different value for those attributes
  + Name=A, name=B

Name: B

height: 0

weight: 0

Name: A

height: 0

weight: 0

B

A

***Object Creation:***

Person A; //only created the reference, not an object. It points to nothing now (null).

A = new Person(); //create the object (allocate storage in memory)

Person A= new Person();

**How instance method works?**

Person a=new Person(), b=new Persion();

a.setWeight(100); b.setWeight(120);

***Keyword this:***

* Can be used only inside method
* When call a method within the same class, don’t need to use this, compiler do it for you.
* When to use it?
  + method parameter or local variable in a method has the same name as one of the fields of the class

***Keyword this example :***

class A{

int **w**;

public void setValue (int **w**) {

**this**.**w** = **w**; //same name!

}

}

When a method parameter or local variable in a method has the same name as one of the fields of the class, you must use this to refer to the field.

***Identify local and Global variables using this keyword***

public class This\_Ex1 {

int a = 10;

void show() {

int a = 20;

System.out.println("The value of (Local Variable) 'a' is : " + a);

System.out.println("The value of (Global Variable) 'a' is : " + this.a);

}}

class MainClass {

public static void main(String args[]) {

This\_Ex1 obj = new This\_Ex1();

obj.show();

}}

***Area of Rectangle using this keyword***

public class This\_Ex3 {

int a,b;

void area(int length, int breadth) {

this.a = length;

this.b = breadth;

int area = a\*b;

System.out.println("The Area of the Rectangle is : " + area);

}}

class MainClass {

public static void main(String args[]) {

This\_Ex3 obj = new This\_Ex3();

obj.area(6, 4);

}

}

***Constructor:***

* A special method automatically called when an object is created by new()
* Java provide a default one that takes no arguments and perform no special initialization
* All fields set to default values: primitive types to 0
* Must have the same name as the class name
* Perform any necessary initialization
* No return type, even no void!
* It actually return current object

***Constructor example 1***

class Circle{

double r; }

public class MainClass{

public static void main(String[] args){

Circle c2 = new Circle(); // OK, default constructor

Circle c = new Circle(2.0); //error!!

}

}

Important: Both Classes have the same Package in all codes.

***Constructor example 2***

class Circle{

double r;

public Circle (double r) {

this.r = r; //same name!

}

public class MainClass{

public static void main(String[] args){

Circle c = new Circle(2.0); //OK

Circle c2 = new Circle(); //error!!, no more default

}

}

***Constructor example 3***

package test;

public class test {

public test(String name){

System.out.println("Passed Name is :" + name );

}

public static void main(String []args){

test t = new test( "Java" );

}

}

***Default and Parameterized Constructor***

class Circle{

double r;

public Circle(){

r = 1.0; //default radius value;

}

public Circle (double r) {

this.r = r; //same name!

} }

public class MainClass{

public static void main(String[] args){

Circle c = new Circle(2.0); //OK

Circle c2 = new Circle(); // OK now!

}

}

***Multiple constructor***

* Can invoke one constructor from another
* Use ***this(para)***
* Useful if constructors share a significant amount of initialization code, avoid repetition.

***Example:***

class Circle{

double r;

public Circle(){

// r = 1.0; //default radius value;

**this (1.0); //call another constructor**

}

public Circle (double r) {

this.r = r; //same name!

} }

public class MainClass{

public static void main(String[] args){

Circle c = new Circle(2.0); //OK

Circle c2 = new Circle(); // OK now!

}

}

***Methods OR Functions***

package javaapplication11;

import java.util.Scanner;

public class JavaApplication11 {

public static void main(String[] args) {

int a,b;

Scanner i = new Scanner(System.in);

System.out.println("Enter an integer");

a = i.nextInt();

System.out.println("Enter an integer");

b = i.nextInt();

int c String s;

public void setValue (int r, String s) {

this.r = r; this.s = s;

}

public void setValue (String s, int r) {

this.r =r; this.s =s;

}}

public class MainClass{

public static void main (String[] args){

Overload o = new Overload();

o.setValue(10, “ok”);

o.setValue(“ok?”, 20); //both are OK!

}

}

The compiler will decide which method to use based on the number and type of arguments you supply

* Members of a class are always accessible within the body of the class
* **public**: accessible from outside of the class
* **private**: only within this class, even not visible in subclass
  + Subclass inherit it, but cannot directly access it inside the subclass
* **protected**: accessible from itself and all the subclasses
* **Friendly:** accessible to all the other classes in the current package have access to the friendly member.

***Access Modifiers***

|  |  |
| --- | --- |
| ***Access Modifier*** | ***Class or member can be referenced by…*** |
| ***public*** | ***methods of the same class, and methods of other classes*** |
| ***private*** | ***methods of the same class only*** |
| ***protected*** | ***methods of the same class, methods of subclasses, and methods of classes in the same package*** |
| ***No access modifier (package access)*** | ***methods in the same package only*** |

***Methods and Constructors in Class Example 1***

package test;

public class test {

int Age;

public test(String name)

{System.out.println("Name chosen is :" + name ); }

public void setAge( int a )

{Age = a; }

public void getAge( )

{System.out.println("Person's age is :" + Age ); }

public static void main(String []args){

test m = new test( "C++" );

m.setAge( 2 );

m.getAge( ); }

}

***Methods and Constructors in Class Example 2***

**class** Computer {

Computer() {

System.out.println("Constructor of Computer class.");

}

**void** computer\_method() {

System.out.println("Power gone! Shut down your PC soon...");

}

**public class MainClass{**

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

**Computer my = new Computer();**

**Laptop your = new Laptop();**

**my.computer\_method();**

**your.laptop\_method();**

**}**

**}**

**class** Laptop {

Laptop() {

System.out.println("Constructor of Laptop class.");

}

**void** laptop\_method() {

System.out.println("99% Battery available.");}}

= minFunction(a, b);

System.out.println("Minimum Value = " + c);}

**public static int minFunction(int n1, int n2) {**

**int min;**

**if (n1 > n2)**

**min = n2;**

**else**

**min = n1;**

**return min;}}**

***Method overloading:***

* It’s legal for a class to define more than one method with the **same name**, as long as they have **different list of parameters**
  + Different number of parameter, or different type of parameter, or different order
  + Must be the same return type
* The compiler will decide which method to use based on the number and type of arguments you supply

Overload example

class Overload{

int r;

String s;

public void setValue (int r, String s) {

this.r = r; this.s = s;

}

public void setValue (String s, int r) {

this.r =r; this.s =s;

}}

public class MainClass{

public static void main (String[] args){

Overload o = new Overload();

o.setValue(10, “ok”);

o.setValue(“ok?”, 20); //both are OK!

}

}

The compiler will decide which method to use based on the number and type of arguments you supply

***Methods and Constructors in Class Example 1***

package test;

public class test {

int Age;

public test(String name)

{System.out.println("Name chosen is :" + name ); }

public void setAge( int a )

{Age = a; }

public void getAge( )

{System.out.println("Person's age is :" + Age ); }

public static void main(String []args){

test m = new test( "C++" );

m.setAge( 2 );

m.getAge( ); }

}

***Methods and Constructors in Class Example 2***

**class** Computer {

Computer() {

System.out.println("Constructor of Computer class.");

}

**void** computer\_method() {

System.out.println("Power gone! Shut down your PC soon...");

}

**public class MainClass{**

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

**Computer my = new Computer();**

**Laptop your = new Laptop();**

**my.computer\_method();**

**your.laptop\_method();**

**}**

**}**

**class** Laptop {

Laptop() {

System.out.println("Constructor of Laptop class.");

}

**void** laptop\_method() {

System.out.println("99% Battery available.");}}

Questions:

**Q.No.1:** Write a program that declares a class **“Distance”** that includes two data members feet and inches and member functions **setdist( )** that set distance as constants,**getdist( )** that get distance from user and **showdist( )** that display two distances.

**Q.NO.2:** write a Program that declares a class **“shapes”** that contains possible data members and member functions **area( )** that show area of circle and return this. **Triarea( )** that display area of triangle and return.

**Q.NO.3:** Write a class called rectangle. Your task is to store the length and width of the rectangle. Write a member function called increment that will add 1 to the value of length and width. Also write a function that will compute the area of the rectangle.

* Demonstrate the use of the object in the main function.

**Q.No.4:** Write a C++ program that creates a class called laptop. The data members of the class are brand (string), model(string), serial (int),colour (string), price (float), processor speed (float), RAM (int), screen size(float).Create member function that will set the individual values. Since the RAM canbe upgraded therefore create a function that allows you to upgrade the RAM only. In the end, create a function that will display all the data members.