***Index***

***1) Introduction to java…………………………………………………………..1***

***2) Data types…………………………………………………………………………6***

***3) Operators………………………………………………………………………….8***

***4)Instructions……………………………………………………………………….11***

***5)Functions/Methods…………………………………………………………..41***

***6) Type Casting…………………………………………………………………….50***

***7) Object Oriented Programming System……………………………..53***

***8) Encapsulation……………………………………………………………………61***

***9) Inheritance………………………………………………………………………..65***

***10) Abstraction………………………………………………………………………78***

***11) Polymorphism………………………………………………………………….85***

***12) Variables………………………………………………………………………….90***

***13) Constructors…………………………………………………………………….92***

***14) this and super keywords………………………………………………….101***

***15) Destructors……………………………………………………………………..110***

***16) Java Garbage Collection………………………………………………….110***

***17) To accept input from the user…………………………………………113***

***18) Packages………………………………………………….……………………127***

***19) Java Inner classes………………………………………………………..141***

***20) Interfaces…………………………………………….……………………..145***

***21) Arrays………………………………………………….……………………..156***

***22) Command Line Arguments…………………..……………………..167***

***23) Static blocks ………………………………………..……………………..170***

***24) Initializer blocks/Instance initializer blocks ………………..175***

***25) Exception Handling………………………………….………………….179***

***26) Multi threading…………………………………………………………..191***

***27) Collection framework………………………………………………….223***

***28)IO STREAMS ………………………………………….…………………….269***

***29)Strings…………………………………………………….……………………304***

***1)Introduction to java:***

Program:

Sequence of instructions that are needed to be executed in order to get the output/result.

Purpose of writing a program:

1)To get the results in a faster manner.

2)To get the results in an accurate manner.

That means,we are writing programs in order to accomplish some task.

What might be the tasks for which we are writing code:

1)Performing basic operations(Addtion,subtraction,etc)

2)Generating a calculator------>Desktop application

3)Creating web pages----------->Web applicaions

Desktop application:

Source code will be in one system and output will be seen in same system.

Web application:Source code will be in one system and output can be seen in another system.

Java is used to develop both desktop and web applications.

Modules:

Java was divided into three modules:

-------J2SE(Java Standared Edition)/Core java/Desktop applications

-------J2EE(Java Enterprise Edition)/Ad java/Web application

-------J2ME(Java Micro Edition)/Mobile applications

Java was introduced by "Sun Micro Systems" in 1991.It was initially called as "Oak" by James Gosling. Later it was renamed to Java.

Oak------->Name of the tree

Java------>Name of the coffee seed

In order to see the output of any program,we have to follow the two step process.They are-----

1)Compilation

2)Execution

1)Compilation:

Converting high level code(code written by us) into machine

understandable code or binary code.

2)Execution:

Resolving the binary code step by step.

Features:

1)Compiled and interpreted:

.java file on compilation produces .class file.(done by compiler)

.class file on Interpretation(Execution) produces .exe(ouput) file(done by Java Virtual machine).

2)Platform independent:

We can write the source code in a system having one OS and we can see the output in other system that have different OS.

3)Multi threaded and interactive:

To Perform more than one task simultaneously.

4)Extensible:To embed the code of native languages like c,cpp,etc into our java

program and to run the java program.This can be done using Java Native Interface(JNI).

5)Secure:

We can prevent our java programs from being interacted with the virus present in the internet.Java is more secure than C.More no. of. security checks will be there.Java is slower than C.Code for security will also be included in java.

Security in java:

Sandbox model.

1)Class loader---------->Loads the class into the memory.

2)Byte code verifier------>Check will be performed before compilation

3)Security checker-------->Check will be performed before execution.

Java API---------->JDK+JSL

Any language will be released in the form of API.(All the symbols,keywords,etc).

API-------->Application Programming Interface.

JDK------>Java Development Kit.Tools to develop java progs. are present here.

JSL------>Java Standard Library.---->Set of predefined classes and interfaces.

JRE------>Java Run Time Environment.Tools to execute or run a java program.Examples----JVM,UI tool kits will be in this.

Java utilities:

javac---->java compiler

java----->Launches JVM

javah--->header file support

javap---->java dis-assembler

Steps for executing a java program:

1)Install Java Development Kit(JDK).

2)Set the path.

3)Take a notepad and wite the program.

4)Open command prompt and give compilation and execution commands.

cd Desktop

Command for compilation:

javac Filename.java

Command for Execution:

java Filename

Path:To show the location of .exe files to the OS.

Class Path:To show the location of .class files to the JVM.

To set the path:

My Computer------------------->properties----->Advanced--->

Environmental variables------>User variables--->new--->

variable name:PATH

variable value:paste

ok----->ok----->ok.

C--->program files---->java---->jdk1.7------>bin------------->copy

Structure of a basic java program:

class ClassName

{

main()

{

//Logic to be written

}

}

First java program:

class Demo

{

public static void main(String a[])

{

System.out.println("welcome to java");

}

}

main()------->Predefined function/method.Starting point of program Execution

void---->main block is not going to return anything.

public----->To make main() accessible any where(through out the program).

static------->To load main() first(Top priority in loading main()).

System-------->Predefined class.

To gain access over the system resources.

out------>Predefined object.(An object of PrintStream class).

To gain access over the output devices of the system.

Variable:Name of a memory cell is called as a variable.

To store the information,we will use this.

***2)Data types:***

Provides information regarding:

1)No. of. bytes of memory to ba allocated to a variable.

2)What type of value should be allowed into it.

Data types available in java are as follows:

byte------>1 byte

short----->2 bytes

int------->4 bytes

long------>8 bytes

float----->4 bytes

double---->8 bytes

boolean--->1 bit

char------>2 bytes

String---->Depends on the string

Ex:Pinky--->5\*2=10 bytes

Example:

class Employee

{

public static void main(String[] args)

{

int a=5,b=4;

System.out.println(a+b);

float x=45.3f,y=89.4f;

System.out.println(x+y);

double d1=45.32,d2=908.4;

System.out.println(d1\*d2);

}

}

***3)Operators:***

These are used to perform operations.

In the following instruction,c=a+b...

a,b----->operands

+--->Operator

Operator Representations

Arithmetic + - \* / %

Relational < > <= >= == !=

incrementation and ++ --

Decrementation

Logical && || !

Assignment = += -= \*= /=

Conditional/ ?:

ternary

Examples:

class ternary

{

public static void main(String ar[])

{

System.out.println("The result is:"+((3==3)?4:5));

}

}

class postfix

{

public static void main(String ar[])

{

int a,z,x=10,y=20;

z=x\*y++;

a=x\*y;

System.out.println(z+" "+a);

}

}

output:

200,210

class prefix

{

public static void main(String ar[])

{

int a,z,x=10,y=20;

z=x\*++y;

a=x\*y;

System.out.println(z+" "+a);

}

}

output:

210,210

class LogicalOperators

{

public static void main(String a[])

{

System.out.println("5>3 and 5<10:"+(5>3 && 5<10)); 1 && 1

System.out.println("8>5 and 8<2:"+(8>5 ||8<2));1 || 0

}

}

output:

5>3 and 5<10:1

8>5 and 8<2:1

***4)Instructions:***

Types:

1)Arithmetic instructions---->

Those instructions that involves arithmetic operators are called as arithmetic instructions.

Example:

d=a\*b+c;

2)Type declaration instructions----> int a;

Control instructions

---->Sequence control

---->Decision control

---->if(pure if)

---->if-else

---->if-else ladder

---->Loop control

---->for

---->while

---->do-while

---->Case control(switch)

Sequence Control instructions:

In which order we do write the program,in that order it will be executed.

-----------

-----------

-----------

-----------

-----------

-----------

Decision control instructions:

Depending on the condition,the execution of instrctions will take place.

1)if (pure if):

Here condition is a boolean type (returns either true or false).If the condition is evaluated to true, statement(s) inside the body of if (statements inside parenthesis) are executed.If the condition is evaluated to false, statement(s) inside the body of if are skipped from execution.

Syntax:

------------

-------------

------------

-------------

if(condition)

{

-----------

-----------

}

Example 1:

class Demo

{

public static void main(String a[])

{

int a=3,b=3;

if(a==b)

{

System.out.println("They are equal");

}

System.out.println("I am the last");

}

}

output:

They are equal

I am the last

Example 2:

class IfStatement {

public static void main(String[] args) {

int number = 10;

if (number > 0) {

System.out.println("Number is positive.");

}

System.out.println("This statement is always executed.");

}

}

When you run the program, the output will be:

Number is positive.

This statement is always executed.

2)if-else:

If the condition under if is true then if block will be executed.If the condition is false then else block will be executed.

----

----

if(condition)

{

------

------

}

else

{

------

------

------

}

---

---

Example:

class Demo

{

public static void main(String a[])

{

int a=6,b=6;

if(a==b)

{

System.out.println("They are equal");

}

else

{

System.out.println("They are not equal");

}

System.out.println("I am the last");

}

}

Example 2:

class IfElse {

public static void main(String[] args) {

int number = 10;

if (number > 0) {

System.out.println("Number is positive.");

}

else {

System.out.println("Number is not positive.");

}

System.out.println("This statement is always executed.");

}

}

When you run the program, the output will be:

Number is positive.

This statement is always executed.

3)if-else-ladder

If we want to check for multiple conditions,we will use this.

Syntax:

if(condition)

{

}

else if(condition)

{

}

else if(condition)

{

}

else if(condition)

{

}

else

{

}

Example 1:

class Ladder {

public static void main(String[] args) {

int number = 0;

if (number > 0) {

System.out.println("Number is positive.");

}

else if (number < 0) {

System.out.println("Number is negative.");

}

else {

System.out.println("Number is 0.");

}

}

}

Output:

Number is 0.

Sample code:

int avg=16;

if(avg>=90)

{

System.out.println("Grade A");

}

else if(avg>=75)

{

System.out.println("Grade B");

}

else if(avg>=60)

{

System.out.println("Grade C");

}

else if(avg>=40)

{

System.out.println("Grade D");

}

else

{

System.out.println("Failed---Waste fellow");

}

}

Nested if:Nesting:One with in another.

syntax:

if(condition)

{

---------

---------

---------

if(condition)

{

---------------

--------------------

}

else

{

--------------

--------------

}

----------------------

----------------------

}

else

{

----------------

----------------

----------------

}

Example 1:

class Number{

public static void main(String[] args) {

double n1 = -1.0, n2 = 4.5, n3 = -5.3, largestNumber;

if (n1 >= n2) {

if (n1 >= n3) {

largestNumber = n1;

}

else {

largestNumber = n3;

}

}

else {

if (n2 >= n3) {

largestNumber = n2;

}

else {

largestNumber = n3;

}

}

System.out.println("Largest number is " + largestNumber);

}

}

When you run the program, the output will be:

Largest number is 4.5

***Loops:***

If you want to repeatedly execute a set of statements,you can use loops.

for loop:

----

----

for(initialization;condition;incrementation)

{

-----

-----

}

---

---

To print "Hello" for 5 times

class Demo

{

public static void main(String a[])

{

int i;

for(i=1;i<=5;i++)

{

System.out.println("Hello");

}

}

}

Program to print numbers:

public class ForExample {

public static void main(String[] args) {

for(int i=1;i<=10;i++){

System.out.println(i);

}

}

}

Output:

1

2

3

4

5

6

7

8

9

10

Decrementation:

class ForLoopExample {

public static void main(String args[]){

for(int i=10; i>1; i--){

System.out.println("The value of i is: "+i);

}

}

}

The output of this program is:

The value of i is: 10

The value of i is: 9

The value of i is: 8

The value of i is: 7

The value of i is: 6

The value of i is: 5

The value of i is: 4

The value of i is: 3

The value of i is: 2

In the above program:

int i=1 is initialization expression

i>1 is condition(Boolean expression)

i–- Decrementation operation

while loop:

initialization;

while(condition)

{

---

---

incremetation;

}

Example:

To print 1 to 5 nos. using while loop.

public class WhileExample {

public static void main(String[] args) {

int i=1;

while(i<=10){

System.out.println(i);

i++;

}

}

}

Output:

1

2

3

4

5

6

7

8

9

10

Example 2:

class WhileLoopExample {

public static void main(String args[]){

int i=10

while(i>1){

System.out.println(i);

i--;

}

}

}

The output of this program is:

10

9

8

7

6

5

4

3

2

do-while loop:

initialization;

do

{

-----

-----

incre/decre;

}while(condition);

Example:

class DoDemo

{

public static void main(String a[])

{

int i=1;

do

{

System.out.println(i);

i++;

}while(i<=5);

}

}

output:

1

2

3

4

5

break:

As soon as the cursor encounter the break

keyword,it will skip all the statements

after "break" and exits(comes out) of that

block in which break is written.

for(i;c;i)

{

-----

-----

----

-----

if(condition)

{

break;

}

----

-----

}

-----

-----

Example:

public class BreakDemo

{

public static void main(String[] args)

{

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

{

if (i == 5)

{

break; // terminate loop if i is 5

}

System.out.print(i + " ");

}

System.out.println("Loop is over.");

}

}

Output :

1 2 3 4 Loop is over.

continue:

As soon as the cursor encounter the continue keyword,it will skip all the statements

after continue and it will directly be

going to the 4th part(incrementation)

of the loop.

/\*\*

\* This program demonstrates continue

\* to skip remaining statements of iteration.

\*/

public class ContinueDemo

{

public static void main(String[] args)

{

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

{

if (i % 2 == 0)

{

continue; // skip next statement if i is even

}

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

}

}

}

Output :

1 3 5 7 9

Nested loops:

Loop inside another loop is called nested loop.

class Demo

{

public static void main()

{

int r,c,sum;

for(r=1;r<=3;r++)

{

for(c=1;c<=2;c++)

{

sum=r+c;

System.out.println("r=%d c=%d sum=%d",r,c,sum);

}

}

}

}

goto:

Used to skip some instructions.

------

------

------

goto label-name;

-------

-------

-------

label:

-------

-------

-------

------

------

------

goto pinky;

-------

-------

-------

pinky:

-------

-------

-------

Example:

class Demo

{

public static void main(String a[])

{

int n=5;

if(n%2==0)

{

goto even;

}

else

{

goto odd;

}

odd:

System.out.println("The number is odd");

even:

System.out.println("The number is even");

}

output:

The number is odd

switch:

switch(variablename)

{

case casename:

--------

--------

--------

break;

case casename:

--------

--------

--------

break;

case casename:

--------

--------

--------

break;

default:

--------

--------

--------

Explanation:

sop("enter 1 for a/c details");

sop("enter 2 for m w");

sop("enter 3 m d");

sop("enter 4 to exit");

int choice=3;

switch(choice)--->3

{

case 1:

//display logic

break;

case 2:

//withdrawl logic

case 3:

//deposition logic

case 4:

//To terminate

System.exit(0);

default:

System.out.println("Invalid choice");

}

Example 1:

class sw

{

public static void main(String aa[])

{

int a=2,b=3,c;

char ch='\*';

System.out.println("Arithmetic operations");

switch(ch)

{

case '+':

c=a+b;

System.out.println("The result is "+c);

break;

case '-':

c=a-b;

System.out.println("The result is "+c);

break;

case '\*':

c=a\*b;

System.out.println("The result is "+c);

break;

case '/':

c=a/b;

System.out.println("The result is "+c);

break;

}

}

}

output:

Arithmetic operations

The result is 6

Example 2:

class Day {

public static void main(String[] args) {

int week = 4;

String day;

switch (week) {

case 1:

day = "Sunday";

break;

case 2:

day = "Monday";

break;

case 3:

day = "Tuesday";

break;

case 4:

day = "Wednesday";

break;

case 5:

day = "Thursday";

break;

case 6:

day = "Friday";

break;

case 7:

day = "Saturday";

break;

default:

day = "Invalid day";

break;

}

System.out.println(day);

}

}

When you run the program, the output will be:

Wednesday

***4)Functions/Methods:***

Self contained block of statements that can be used any no. of. times any where in the program.

Parts:

1)Function declaration

ReturnType FunctionName();

2)Function call

FunctionName();

3)Function definition

ReturnType FunctionName()

{

}

modifier - It defines the access type of the method and it is optional to use.

returnType - Method may return a value.

nameOfMethod - This is the method name. The method signature consists of the method name and the parameter list.

Parameter List - The list of parameters, it is the type, order, and number of parameters of a method. These are optional, method may contain zero parameters.

method body - The method body defines what the method does with the statements.

Example:

Basic program:

class functions1

{

public static void main(String a[])

{

System.out.println("I am from main()");

italy();

System.out.println("I have executed italy");

brazil();

System.out.println("I have executed brazil");

argentina();

System.out.println("I have executed argentina");

}

static void italy()

{

System.out.println("I am from italy");

}

static void brazil()

{

System.out.println("I am from brazil");

}

static void argentina()

{

System.out.println("I am from argentina");

}

}

Function call in the function definition:

class functions1

{

public static void main(String a[])

{

System.out.println("I am from main()");

italy();

System.out.println("I have executed italy");

}

static void italy()

{

System.out.println("I am from italy");

brazil();

System.out.println("I have executed brazil");

}

static void brazil()

{

System.out.println("I am from brazil");

}

}

Parameter:

Extra information passed through a function call to a function definition is called as parameter.

Parameter passing techniques to a function:

1)Call by value:

We will call the function(definition) by passing values.

class functions3

{

public static void main(String args[])

{

int a=2,b=3,sum;

sum=calsum(a,b);//5

System.out.println("The result is:"+sum);

}

static int calsum(int x,int y)

{

int w;

w=x+y;

return w;

}

}

Example 2:

public class ExampleMinNumber {

public static void main(String[] args) {

int a = 11;

int b = 6;

int c = minFunction(a, b);

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

}

/\*\* returns the minimum of two numbers \*/

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

int min;

if (n1 > n2)

min = n2;

else

min = n1;

return min;

}

}

Example 3:

public class ExampleVoid {

public static void main(String[] args) {

methodRankPoints(255.7);

}

public static void methodRankPoints(double points) {

if (points >= 202.5)

{

System.out.println("Rank:A1");

}else if (points >= 122.4) {

System.out.println("Rank:A2");

}else {

System.out.println("Rank:A3");

}

}

}

This will produce the following result -

Output:

Rank:A1

Example 4:

public class swappingExample {

public static void main(String[] args) {

int a = 30;

int b = 45;

System.out.println("Before swapping, a = " + a + " and b = " + b);

// Invoke the swap method

swapFunction(a, b);

System.out.println("\n\*\*Now, Before and After swapping values will be same here\*\*:");

System.out.println("After swapping, a = " + a + " and b is " + b);

}

public static void swapFunction(int a, int b) {

System.out.println("Before swapping(Inside), a = " + a + " b = " + b);

// Swap n1 with n2

int c = a;

a = b;

b = c;

System.out.println("After swapping(Inside), a = " + a + " b = " + b);

}

}

Function recursion:

A function calling itself is called as recursion.

Example 1:

public class Recursion {

static int factorial(int n){

if (n == 1)

return 1;

else

return(n \* factorial(n-1));

}

public static void main(String[] args) {

System.out.println("Factorial of 5 is: "+factorial(5));

}

Example 2:

public class RecursionExample2 {

static int count=0;

static void p(){

count++;

if(count<=5){

System.out.println("hello "+count);

p();

}

}

public static void main(String[] args) {

p();

}

}

Output:

hello 1

hello 2

hello 3

hello 4

hello 5

***6)Type Casting:***

Converting one data type into another data type.

or

Assigning a value of one type to a variable of another type is known as Type Casting.

int x = 10;

byte y = (byte)x;

Types:

1)Narrowing

2)Widening

1)Narrowing:

Converting larger datatype to smaller datatype is called as narrowing.

2)Widening:

Converting smaller datatype to larger datatype is called as widening.

Example 1:

class DatatypeCasting

{

public static void main(String arg[])

{

byte b;

int i = 81;

double d = 323.142;

float f = 72.38f;

char c = 'A';

c = (char) i;

System.out.println("i = " + i + " c = " + c);

i = (int) d; // LINE A

System.out.println("d = " + d + " i = " + i); // LINE B

i = (int) f; // LINE C

System.out.println("f = " + f + " i = " + i); // LINE D

}

}

OUTPUT

i = 81 c = Q

d = 323.142 i = 323

f = 72.38 i = 72

Example2:

class ImplicitTypeCasting

{

public static void main(String aa[])

{

long a=2;/\*long to int\*/

System.out.println(a);

}

}output:2

/\*Widining\*/

class ExplicitTypeCasting

{

public static void main(String aa[])

{

System.out.println("The result is:"+(5/2));//2

System.out.println("The result is:"+(float)5/2);/\*int to float \*/

System.out.println("The result is:"+(char)65);

}

}

output:

The result is:2

The result is:2.5

The result is:A

***7)Object Oriented Programming System:***

Object Oriented Programming system is a programming approach where a given task/problem is divided into no.of. objects and we establish communication among the objects to fulfil the task.

or

Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects", which may contain data, in the form of fields, often known as attributes; and code, in the form of procedures, often known as methods. A feature of objects isthat an object's procedures can access and often modify the data fields of the object with which they are associated (objects have a notion of "this" or "self"). In OOP, computer programs are designed by making them out of objects that interact with one another.

There is significant diversity of OOP languages, but the most popular ones are class-based, meaning that objects are instances of classes, which typically also determine their type.Many of the most widely used programming languages (such as C++, Object Pascal, Java, Python etc.) are multi-paradigm programming languages that support object-oriented programming to a greater or lesser degree, typically in combination with imperative,procedural programming. Significant object-oriented languages include Java, C++, C#, Python, PHP, Ruby, Perl, Object Pascal,Objective-C, Dart, Swift, Scala, Common Lisp, and Smalltalk.Pure object oriented programming language is "small talk".

Java oops concepts:

Object means a real word entity such as pen, chair, table etc. Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects. It simplifies the software development and maintenance by providing some concepts:

Object

Class

Inheritance

Polymorphism

Abstraction

Encapsulation

Description:

Object:

Any entity that has state and behavior is known as an object. For example: chair, pen, table, keyboard, bike etc. It can be physical and logical.

Class:

Collection of properties/variables and/or behaviours/methods.

or

Collection of objects is called class. It is a logical entity.

Inheritance:

When one object acquires all the properties and behaviours of parent object i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

Polymorphism:

When one task is performed by different ways i.e. known as polymorphism.

For example: to convince the customer differently, to draw something e.g. shape or rectangle etc.

In java, we use method overloading and method overriding to achieve polymorphism.

Another example can be to speak something e.g. cat speaks meaw, dog barks woof etc.

Abstraction:

Hiding internal details and showing functionality is known as abstraction. For example: phone call, we don't know the internal processing.

In java, we use abstract class and interface to achieve abstraction.

7)Encapsulation:

Binding (or wrapping) code and data together into a single unit is known as encapsulation. For example: capsule, it is wrapped with different medicines.

A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private

here.

Advantage of OOPs over Procedure-oriented programming language:

1)OOPs makes development and maintenance easier where as in Procedure-oriented programming language it is not easy to manage if

code grows as project size grows.

2)OOPs provides data hiding whereas in Procedure-oriented programming language a global data can be accessed from anywhere.

3)OOPs provides ability to simulate real-world event much more effectively. We can provide the solution of real word problem

if we are using the Object-Oriented Programming language.

What is an object-based programming language?

Object based programming language follows all the features of OOPs except Inheritance. JavaScript and VBScript are examples of object based programming languages.

A language is said to be object oriented if everything(the entire code) is implemented in the form of classes and objects.

java--more oops bcoz main() should be written inside the class.

cpp--less oop bcoz main() should be written outside the class.

Java is only 99% pure object oriented because it supports primitive data types(int,float,char,etc)and it supports the concept of

static members(that can be called without objects).

Java is slower than C because of code for security is also existed in java.

Program using OOPS:

Approach1:

class Student

{

int rollno,age;

void join()

{

System.out.println("Going to join Qshore");

}

public static void main(String a[])

{

Student d=new Student();//d---->object

d.rollno=101;

d.age=20;

System.out.println(d.rollno+" "+d.age);//10 20

d.join();

}

}

output:

10 20

Going to join Qshore

Approach2:

class Student

{

int rollno,age;

void join()

{

System.out.println("Going to join Qshore");

}

}

class FinalDemo

{

public static void main(String a[])

{

Student d=new Student();//d---->object

d.rollno=101;

d.age=20;

System.out.println(d.rollno+" "+d.age);//10 20

d.join();

}

}

Structure of a java program:

Package packagename;

Import instructions;

class Demo

{

variables+methods

main()

{

object creation

+

Accessing part

}

}

classes and interfeces----->Captial

Examples:

Scanner

BufferedReader

methods------>small

Example:

nextInt()------>camel case

keywords-------------->small

String------------->class and a data type--->S should always be a capital letter in a string.

***Pillars of java:***

There are four pillars for java namely....

Encapsulation

Abstraction

Polymorphysm

Inheritence

***8)Encapsulation:***

Encapsulation in java is a process of wrapping code and data together into a single unit, for example capsule i.e. mixed of severalmedicines.

We can create a fully encapsulated class in java by making all the data members of the class private. Now we can use setter and getter methods to set and get the data in it.

The Java Bean class is the example of fully encapsulated class.

Real Time example of Encapsulation:

The best example of car driving, car driver knows how to start car by pressing start button. The driver doesn't know what happened inside when pressing start button. Here the starting process is hidden from driver. So this process can be called as “starting process is encapsulated from driver”

Advantages of Encapsulation:

- The main advantage of Encapsulation is, when using objects, the object need not reveal all its attributes and behaviors.

- Code changes can be made independently

- Increases usability

Disadvantages of Encapsulation:

- The main drawback or disadvantage of encapsulation is that it relies heavily on standards to maintain readability

It provides you the control over the data. Suppose you want to set the value of id i.e. greater than 100 only, you can write the

logic inside the setter method.

class Encap-->oe

{

private int a=2,b=3;

void display()

{

System.out.println("Hello from Encap class");

}

}

class owncap

{

public static void main(String a[])

{

Encap oe=new Encap();

//oe.a=10;//a is the private member of Encap

//oe.b=20;

oe.display();//diplay is the private member of Encap

}

}

Example 2:

//save as Student.java

package com.Qshore;

public class Student{

private String name;

public String getName(){

return name;

}

public void setName(String name){

this.name=name

}

}

//save as Test.java

package com.Qshore;

class Test{

public static void main(String[] args){

Student s=new Student();

s.setName("vijay");

System.out.println(s.getName());

}

}

***9)Inheritance:***

Deriving a new class from the existing class so that

the new class have its own qualities and also the

qualities acquired from the parent class.That means.....

Inheritance allows a class to use the properties and methods of another class.

In other words, the derived class inherits the states and behaviors from the base class.

The derived class is also called subclass and the base class is also known as

super-class. The derived class can add its own additional variables and methods.

These additional variable and methods differentiates the derived class from the base

class.Inheritance is a compile-time mechanism. A super-class can have any number of

subclasses. But a subclass can have only one superclass. This is because Java does

not support multiple inheritance.

The superclass and subclass have “is-a” relationship between them. Let’s have a look

at the example below.

Example:

Let’s consider a superclass Vehicle. Different vehicles have different features and properties however there few of them are common to all. Speed, color, fuel used, size are few which are common to all. Hence we can create a class ‘Vehicle’ with states and actions that are common to all vehicles. The subclass of this super class can be any type of vehicle.

Example:

Class Car A has all the features of a vehicle. But it has its own attributes ,which makes it different from other subclasses. By using inheritance we need not rewrite the code that we’ve already used with the Vehicle. The subclass can also be extended. We can make a class ‘Sports Car’ which extends ‘Car’. It inherits the features of both ‘Vehicle’ and ‘Car’.

The keyword used for inheritance is extends.

Syntax:

public class ChildClass extends BaseClass {

// derived class methods extend and possibly override

Types:

Single----->One child for one parent(Base,Super).

Hierarchical----->More than one children for one parent

Multilevel--------->child--parent--grand father relationship

Multiple-------->Not supported in java directly.(Only through interfaces).

hybrid----->Combination of two inheritences(Not supported).

Single Inheritence:

Example 1:

class Animal{

void eat(){System.out.println("eating...");}

}

class Dog extends Animal{

void bark(){System.out.println("barking...");}

}

class TestInheritance{

public static void main(String args[]){

Dog d=new Dog();

d.bark();

d.eat();

}}

Output:

barking...

eating...

Example 2:

class A

{

void displayA()

{

System.out.println("Hello from A");

}

}

class B extends A

{

void displayB()

{

System.out.println("I am from B");

}

public static void main(String[] args) {

B si=new B();

si.displayA();

si.displayB();

}

}

Multilevel Inheritance Example:

File: TestInheritance2.java

class Animal{

void eat(){System.out.println("eating...");}

}

class Dog extends Animal{

void bark(){System.out.println("barking...");}

}

class BabyDog extends Dog{

void weep(){System.out.println("weeping...");}

}

class TestInheritance2{

public static void main(String args[]){

BabyDog d=new BabyDog();

d.weep();

d.bark();

d.eat();

}}

Output:

weeping...

barking...

eating...

Hierarchical Inheritance Example 1:

File: TestInheritance3.java

class Animal{

void eat(){System.out.println("eating...");}

}

class Dog extends Animal{

void bark(){System.out.println("barking...");}

}

class Cat extends Animal{

void meow(){System.out.println("meowing...");}

}

class TestInheritance3{

public static void main(String args[]){

Cat c=new Cat();

c.meow();

c.eat();

//c.bark();//C.T.Error

}}

Output:

meowing...

eating...

Example 2:

class Ah

{

public void displayA()

{

System.out.println("from A");

}

}

class Bh extends Ah

{

public void displayB()

{

System.out.println("from B");

}

}

class Ch extends Ah

{

public void displayc()

{

System.out.println("from c");

}

}

class hierar

{

public static void main(String[] args)

{

Ch c=new Ch();

c.displayA();

c.displayc();

//c.displayB();

Bh b=new Bh();

b.displayA();

b.displayB();

//b.displayc();

Ah a=new Ah();

a.displayA();

//a.displayB();

//a.displayc();

}

}

Multiple Inheritence(Not supported):

Why multiple inheritance is not supported in java?

To reduce the complexity and simplify the language, multiple inheritance is not supported in java.Consider a scenario where A, B and C are three classes. The C class inherits A and B classes. If A and B classes have same method and you call it from child class object, there will be ambiguity to call method of A or B class.Since compile time errors are better than runtime errors, java renders compile timeerror if you inherit 2 classes. So whether you have same method or different, there will be compile time error now.

Program:

class A{

void msg(){System.out.println("Hello");}

}

class B{

void msg(){System.out.println("Welcome");}

}

class C extends A,B{//suppose if it were

public Static void main(String args[]){

C obj=new C();

obj.msg();//Now which msg() method would be invoked?

}

}

Compile Time Error

Hybrid inheritence:

public class A

{

public void methodA()

{

System.out.println("Class A methodA");

}

}

public class B extends A

{

public void methodA()

{

System.out.println("Child class B is overriding inherited method A");

}

public void methodB()

{

System.out.println("Class B methodB");

}

}

public class C extends A

{

public void methodA()

{

System.out.println("Child class C is overriding the methodA");

}

public void methodC()

{

System.out.println("Class C methodC");

}

}

public class D extends B, C

{

public void methodD()

{

System.out.println("Class D methodD");

}

public static void main(String args[])

{

D obj1= new D();

obj1.methodD();

obj1.methodA();

}

}

Output:

Error!!

Why? Most of the times you will find the following explanation of above error – Multiple inheritance is not allowed in java so class D cannot extend two classes(B and C). But do you know why it’s not allowed? Let’s look at the above code once again, In the above program class B and C both are extending class A and they both have overridden the methodA(), which they can do as they have extended the class A. But since both have different version of methodA(), compiler is confused which one to call when there has been a call made to methodA() in child class D (child of both B and C, it’s object is allowed to call their methods), this is a ambiguous situation and to avoid it, such kind of scenarios are not allowed in java. In C++ it’s allowed.

What’s the solution? Hybrid inheritance implementation using interfaces.

interface A

{

public void methodA();

}

interface B extends A

{

public void methodB();

}

interface C extends A

{

public void methodC();

}

class D implements B, C

{

public void methodA()

{

System.out.println("MethodA");

}

public void methodB()

{

System.out.println("MethodB");

}

public void methodC()

{

System.out.println("MethodC");

}

public static void main(String args[])

{

D obj1= new D();

obj1.methodA();

obj1.methodB();

obj1.methodC();

}

}

Output:

MethodA

MethodB

MethodC

Note: Even though class D didn’t implement interface “A” still we have to define the

methodA() in it. It is because interface B and C extends the interface A.The above code would work without any issues and that’s how we implemented hybrid inheritance in java using interfaces.

Real Time Example:

The real time example for Inheritance is “parent child relationship”. Because child receives all the property from parent.

Advantages of Inheritance:

- Inheritance is the process of object reusability

- Inheritance reduce code redundancy

- Inheritance reduce the size of program code

- Inheritance helps in adding additional features to an existing class without modifying it

Disadvantages of Inheritance:

- In the concept of Inheritance , Both child class and parents class are tightly coupled. If you modify the program code in parent

class then all child classes are effected, because those child classes are inheriting from parent class.

***Abstraction:***

The term abstraction represents incompleteness.Representing essential features by hiding the background details is known as abstraction.That means,abstraction is a process of hiding the implementation details and showing only functionality to the user.Another way, it shows only important things to the user and hides the internal details for example sending sms, you just type the text and send the message. You don't know the internal processing about the message delivery.Abstraction lets you focus on what the object does instead of how it does it.

Abstract class:

A class that is declared with abstract keyword, is known as abstract class in java. It can have abstract and non-abstract methods (method with body).

Abstract method:

Method that is declared as abstract and does not have implementation is known as abstract method.

abstract public void printStatus();//no body and abstract .

Example 1:

abstract class demoabs

{

abstract public void abst(); /\*demoabs a=new demoabs();

a.sum();\*/

void sum()

{

int x=2,y=3,result;

result=x+y;

System.out.println(result);

}

}

class abstraction extends demoabs

{

public void abst()

{

System.out.println("I am from class demoabs");

}

public static void main(String[] args)

{

abstraction ab=new abstraction();

ab.sum();

ab.abst();

}

}

Example 2:

abstract class Shape{

abstract void draw();

}

//In real scenario, implementation is provided by others i.e. unknown by end user

class Rectangle extends Shape{

void draw(){System.out.println("drawing rectangle");}

}

class Circle1 extends Shape{

void draw(){System.out.println("drawing circle");}

}

//In real scenario, method is called by programmer or user

class TestAbstraction1{

public static void main(String args[]){

Shape s=new Circle1();//In real scenario, object is provided through method e.g. getShape() method

s.draw();

}

}

output:

drawing circle

Example 3:

File: TestBank.java

abstract class Bank{

abstract int getRateOfInterest();

}

class SBI extends Bank{

int getRateOfInterest(){return 7;}

}

class PNB extends Bank{

int getRateOfInterest(){return 8;}

}

class TestBank{

public static void main(String args[]){

Bank b;

b=new SBI();

System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");

b=new PNB();

System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");

}}

Output:

Rate of Interest is: 7 %

Rate of Interest is: 8 %

Example 4:

abstract class Bike{

Bike(){System.out.println("bike is created");}

abstract void run();

void changeGear(){System.out.println("gear changed");}

}

class Honda extends Bike{

void run()

{

System.out.println("running safely..");}

}

class TestAbstraction2{

public static void main(String args[]){

Bike obj = new Honda();

obj.run();

obj.changeGear();

}

}

Test it Now

bike is created

running safely..

gear changed

Rule:

If there is any abstract method in a class, that class must be abstract.

Rule: If you are extending any abstract class that have abstract method, you must either provide the implementation of the method or make this class abstract.

Real Time Example for Abstraction:

The Best real time example for abstraction is ATM (Automated Teller Machine). We doesnt know how ATM internally works when we are using ATM, we know only select options like withdraw, Balance Inquiry, Mini Statement etc.Here abstraction hides all unnecessary things, it shows only necessary things.

Advantages of Abstraction:

- Abstraction makes code as readable and more simpler. Because complexity of code will be hidden.

- Abstraction makes code from longer to small , because abstraction doesn't show unnecessary things.

- Without Abstraction maintaining code is very complex in Java and .Net. Abstraction gives one structure to program code.

Disadvantage of Abstraction:

- A disadvantage is that abstract classes cannot be instantiated, but most of the time it is logical not to create a object of an abstract class.

***11)Polymorphism:***

Polymorphism name itself tells “many forms”. That is Poly means “Many” morphism means “forms”. So Polymorphism meaning in Oops. “one name many forms”. Or Polymorphism can also define as “same operation may behave differently on different classes”.

Types:

Method Overloading and Method Overriding.

In overloading,signature is different.

In overriding,signature is same.

Signature:

It has four parts namely----

method-name

Return type of parameters

No. of. parameters

Type of Exception thrown

Method overloading:

Signature is different.

Should be implemented with in one class.

Example 1:

class over

{

public void sum(int x,int y)

{

System.out.println("The sum with two parameters is:"+(x+y));

}

public void sum(int x,int y,int z)

{

System.out.println("The sum with three parameters is:"+(x+y+z));

}

public void sum(float x,int y,float z)

{

System.out.println("The sum with three parameters and different types is:"+(x+y+z));

}

}

class overdemo

{

public static void main(String a[])

{

over ov=new over();

ov.sum(2,3,4);

ov.sum(5,6);

ov.sum(2.3f,4,1.1f);

}

}

Overriding/Runtime polymorphysm:

Signature is same.

Should be implemented with in two classes.

class fruits

{

public void apple()

{

System.out.println("I am from apple()");

}

}

class juciyfruits extends fruits

{

public void apple()

{

System.out.println("I am from overriden-apple()");

System.out.println("Yes this is an overriden method");

}

}

class fruitsdemo

{

public static void main(String a[])

{

juciyfruits jf=new juciyfruits();

jf.apple();

}

}

Overloading and Overriding differences

Overloading Overriding

Signature is different Signature is same

Will be implemented Will be implemented

with in the same class. with in two classes

Real time example:

Real time example for polymorphism is “Door”, why mean we can use these doors for home, car, lift etc.

Dynamic Method dispatch/Runtime polymorphysm:

Assigning sub class to the reference of the super class.

class Shape{

void draw(){System.out.println("drawing...");}

}

class Rectangle extends Shape{

void draw(){System.out.println("drawing rectangle...");}

}

class Circle extends Shape{

void draw(){System.out.println("drawing circle...");}

}

class Triangle extends Shape{

void draw(){System.out.println("drawing triangle...");}

}

class TestPolymorphism2{

public static void main(String args[]){

Shape s;

s=new Rectangle();

s.draw();

s=new Circle();

s.draw();

s=new Triangle();

s.draw();

}

}

Test it Now

Output:

drawing rectangle...

drawing circle...

drawing triangle...

***Variables:***

instance(default)

static

final

instance varaibles:

The variables are by default instance.For each and every instance(object) of a class,a separate copy of instance variables is available.To access an instance variable or an instance method,an object must be created for the class in which this instance variables and methods are present.

Example:

int a;//a is an instance variable by default.

static varaibles:

Only one copy of static variables have to be shared by all the instances(objects) of a class because only one copy of static variables will be created in the memory.(Loaded only for once)To access a static variable or a static method,no need for us to create an object for the class in which this variable or method is present.We can directly access them using the class name.

Ways to access the static members:

object.variable/method name

classname.variable/method name

variable/method name(direct without preceeding by anything)

final keyword:

final variables:

A variable which is preceeded by the final keyword is called a final variable.

A final variable's value can never be changed i.e., it becomes a constant.

Example:

final int a=10;//Value of "a" cannot be changed furthur.

A final method can never be overriden.

A final class can never be inherited.

local variables:

A variable defined inside a block is known as a local variable.

***Constructors:***

Constructor in java is a special type of method that is used to initialize the object.

Java constructor is invoked at the time of object creation. It constructs the values i.e. provides data for the object that is why it is known as constructor.

Rules for creating java constructor:

There are basically two rules defined for the constructor.

Constructor name must be same as its class name

Constructor must have no explicit return type

Types of java constructors

There are two types of constructors:

Parameterless constructor (no-arg constructor)

Parameterized constructor

Syntax:

class\_name()

{

}

Rule: If there is no constructor in a class, compiler automatically creates a default constructor.

Parameterless constructor Example 1:

class Bike1{

Bike1(){System.out.println("Bike is created");}

public static void main(String args[]){

Bike1 b=new Bike1();

}

}

Test it Now

Output:

Bike is created

Default constructor provides the default values to the object like 0, null etc. depending on the type.

class Student3{

int id;

String name;

void display(){System.out.println(id+" "+name);}

public static void main(String args[]){

Student3 s1=new Student3();

Student3 s2=new Student3();

s1.display();

s2.display();

}

}

Test it Now

Output:

0 null

0 null

Explanation:In the above class,you are not creating any constructor so compiler provides you a default constructor.Here 0 and null

values are provided by default constructor.

Example 2:

class cons

{

int width;

int height;

int depth;

cons()

{

width=10;

height=10;

depth=10;

}

public void display()

{

System.out.println("The area is:"+width\*height\*depth);

}

}

class consdemo

{

public static void main(String a[])

{

cons cs=new cons();

cs.display();

}

}

stack heap

cs--->w=10

h=10

d=10

Parameterized constructor:

Java parameterized constructor:

A constructor that have parameters is known as parameterized constructor.

Why use parameterized constructor?

Parameterized constructor is used to provide different values to the distinct objects.

Example of parameterized constructor:

In this example, we have created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor.

class Student4{

int id;

String name;

Student4(int i,String n){

id = i;

name = n;

}

void display(){System.out.println(id+" "+name);}

public static void main(String args[]){

Student4 s1 = new Student4(111,"Karan");

Student4 s2 = new Student4(222,"Aryan");

s1.display();

s2.display();

}

}

Test it Now

Output:

111 Karan

222 Aryan

Example 2:

class cons

{

int width;

int height;

int depth;

cons(int w,int h,int d)//class type is the implicit return type of the constructor

{

width=w;

height=h;

depth=d;

}

public void display()

{

System.out.println("The area is:"+width\*height\*depth);

}

}

class consdemo

{

public static void main(String a[])

{

cons cs1=new cons(5,6,7);

cons cs2=new cons(3,2,1);

cs1.display();

cs2.display();

}

}

Realtime example:

Have you used mould anytime which we generally use to make some small shapes with POP or some times with clay? Most of us must have used it during art and craft work done in our school. Now you can consider a mould as a class which makes objects for you.

Now it has some predefined properties for your object like height of your object width of your object , height or width of a particular part of your object . This is what exactly your Java constructor does in Java for you it assign some predefined values to your objects . Now you can change a height or length of a particular part of your object by covering that very part of your mould with some thing like we use sand some time to make some change in object shape along with POP, This can be considered as Constructor with parameters or constructor overloading is in Java.

Another Good real world example of Java. constructor is your Ice tray which you keep in refrigerator you must have seen the ice trays of different shape and once you put water into it . It produces ice cubes exactly of same height and length every time.

You can consider your ice tray as class and your ice cubes as your objects of that particular class . So your objects are getting some predefined values for properties like Height , length, breadth etc. Now if you want to make a parameterised constructor you can add some colored syrup to water to make ice cubes of different colors. This is what we do in Constructors with parameters in Java .

Advantages of Constructors:

A constructor eliminates placing the default values.

A constructor eliminates calling the normal method implicitly.

OVERLOADED CONSTRUCTORS:

class over

{

over()

{

System.out.println("I am a default constructor");

}

over(float a,int b)

{

System.out.println("The area is:"+a\*b);

}

over(int a)

{

System.out.println("The value is:+a);

}

over(int a,int b,int c)

{

System.out.println("The area is:+ a\*b\*c);

}

}

class overdemo

{

public static void main(String a[])

{

over o1=new over();

over o2=new over(4);

over o3=new over(2.3f,5);

over o4=new over(1,2,3);

}

}

Advantage:

If we want to construct an object in different way then we need constructor overloading.

***this and super keywords:***

this keyword:

1)Compiler passes reference to the current

object being used,which is known as "this".

2)If the instance variables and formal parameters have the same name,then we have to preceed the instance variables using "this" keyword.

3)If the instance variables and formal parameters have different names,then the compiler will implicitly preceeds the instance variables with "this" keyword.

4)This is used to call the current class members

(methods,constructors,variables...)

this before instance variables:

class cons

{

int w;.

int h;

int d;

cons(int w,int h,int d)//class type is the implicit return type of the constructor

{

this.w=w;

this.h=h;

this.d=d;

}

public void display()

{

System.out.println("The area is:"+w\*h\*d);

}

}

class consdemo

{

public static void main(String a[])

{

cons cs1=new cons(5,6,7);

cons cs2=new cons(3,2,1);

cs1.display();

cs2.display();

}

}

"this" to call a method of current class:

class A

{

public static void main(String a[])

{

A a=new A();

a.methodTwo();

}

void methodOne()

{

System.out.println("Inside the method one");

}

void methodTwo()

{

System.out.println("Inside the method two");

this.methodOne();

}

}

this to call the constructor of current class:

class A

{

A()

{

this("Hai");//this();

System.out.println("Inside the parameterless constructor");

}

A(String str)//Hai

{

System.out.println("Inside the parameterized constructor:"+str);

}

public static void main(String a[])

{

A a=new A();

//A a1=new A("Hai");

}

}

Restrictions:

1)"this" and "super" keywords...when used in a constructor

must be the first statements.

2)"this" and "super" keywords cannot be used with static

members.

super keyword:

Used to call the super class members(variables

,methods,constructors) from with in the sub class.

Normal program:

class one

{

int i=10;

void show()

{

System.out.println("super class method:"+i);

}

}

class two extends one

{

int i=20;//Field shadowing--Parent and child classes will have the variable

//with same name

void show()

{

System.out.println("sub class method:"+i);

}

}

class sdemo

{

public static void main(String a[])

{

two t=new two();

t.show();

/\*One o=new One();

o.show();\*/

}

}

calling variables and methods of super class:

class one

{

int i=10;

void show()

{

System.out.println("super class method:"+i);

}

}

class Two extends one

{

int i=20;

void show()

{

System.out.println("sub class method:"+i);//20

super.show();

System.out.println("super class i:"+super.i);//10

}

}

class super2

{

public static void main(String a[])

{

Two t=new Two();

t.show();

}

}

Calling super class constructor:

class one

{

int i;//11

one(int i)//11

{

this.i=i;//11

}

}

class two extends one

{

int i;//22

two(int a,int b)//11,22

{

super(a); //super(11);

i=b;//22

}

}

void show()

{

System.out.println("sub class i:"+i);//22

System.out.println("super class i:"+super.i);//11

}

}

class super3

{

public static void main(String a[])

{

two t=new two(11,22);

t.show();

}

}

Limitations:

1)this and super keywords cannot be used with static.

2)this and super, if used in a constructor,should

be the first statement(s).

***15)Destructor:***

This is used to deallocate the resources acquired by the object during its life cycle.

stack heap

cs1----->w=10

h=10

d=10

finalize()--------->To call the destructor

***16)Java Garbage Collection:***

In java, garbage means unreferenced objects.

Garbage Collection is process of reclaiming the runtime unused memory automatically.

In other words, it is a way to destroy the unused objects.

To do so, we were using free() function in C language and delete() in C++. But, in java it is performed automatically. So, java provides better memory management.

Advantage of Garbage Collection:

It makes java memory efficient because garbage collector removes the unreferenced objects from heap memory.

It is automatically done by the garbage collector(a part of JVM) so we don't need to make extra efforts.

How can an object be unreferenced?

There are many ways:

By nulling the reference

By assigning a reference to another

By annonymous object etc.

1) By nulling a reference:

Employee e=new Employee();

e=null;

2) By assigning a reference to another:

Employee e1=new Employee();

Employee e2=new Employee();

e1=e2;//now the first object referred by e1 is available for garbage collection

3) By annonymous object:

new Employee();

finalize() method

The finalize() method is invoked each time before the object is garbage collected. This method can be used to perform cleanup processing. This method is defined in Object class as:

protected void finalize()

{

}

Note: The Garbage collector of JVM collects only those objects that are created by new keyword.

So if you have created any object without new, you can use finalize method to perform cleanup processing (destroying remaining objects).

gc() method:

The gc() method is used to invoke the garbage collector to perform cleanup processing. The gc() is found in System and Runtime classes.

public static void gc()

{

}

Note: Garbage collection is performed by a daemon thread called Garbage Collector(GC). This thread calls the finalize() method before object is garbage collected.

Simple Example of garbage collection in java

public class TestGarbage1{

public void finalize(){System.out.println("object is garbage collected");}

public static void main(String args[]){

TestGarbage1 s1=new TestGarbage1();

TestGarbage1 s2=new TestGarbage1();

s1=null;

s2=null;

System.gc();

}

}

Test it Now

object is garbage collected

object is garbage collected

*Garbage collector:*

This will remove the object from the memory.

System.gc();========>To call the garbage collector thread

***17)To accept input from the user:***

1)Scanner class

2)BufferedReader class

3)DataInputStream class

Using Scanner class:

/\*

package java.util;

class Scanner-->sc

{

public void nextInt()

{

}

public void nextFloat()

{

}

public void nextDouble()

{

}

public void nextLine()

{

}

\*/

Program:

import java.util.Scanner;

class inp1

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

Scanner sc1=new Scanner(System.in);//Create an extra object for

//accepting a string

System.out.println("Enter your age");//28

int age=sc.nextInt();

System.out.println("Your age is:"+age);

System.out.println("Enter your gender:");//F

char gender=sc.next().charAt(0);

System.out.println("Your gender is:"+gender);

System.out.println("Enter your salary:");//6.6

float salary=sc.nextFloat();

System.out.println("Your salary is:"+salary);

System.out.println("Enter your name:");//pinky

String name=sc.nextLine();

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

}

}

}

Using BufferedReader class:

/\*

package io;

class BufferedReader--br

{

public void readLine()

{

}

}

\*/

import java.io.\*;

class buf

{

public static void main(String a[])

{

BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

System.out.println("Enter your name:");//Pinky

String name=br.readLine();

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

System.out.println("Enter your age:");//"28"

//String to integer

int age=Integer.parseInt(br.readLine());

System.out.println("Your age is:"+age);

System.out.println("Enter your gender");//"f"

char gender=(char)br.read();

System.out.println("Your gender is:"+gender);

System.out.println("Enter your salary:");//"6.6"

float salary=Float.parseFloat(br.readLine());

System.out.println("Your salary is:"+salary);

}

}

Using DataInputStream class:

import java.io.\*;

class buf

{

public static void main(String a[])

{

DataInputStream ds=new DataInputStream(System.in);

System.out.println("Enter your name:");//Pinky

String name=br.readLine();

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

System.out.println("Enter your age:");//"28"

//String to integer

int age=Integer.parseInt(br.readLine());

System.out.println("Your age is:"+age);

System.out.println("Enter your gender");//"f"

char gender=(char)br.read();

System.out.println("Your gender is:"+gender);

System.out.println("Enter your salary:");//"6.6"

float salary=Float.parseFloat(br.readLine());

System.out.println("Your salary is:"+salary);

}

}

***Other examples:***

Railway Reservation System:

import java.util.Scanner;

class Railways

{

public static void main(String a[])

{

Scanner sc=new Scanner(System.in);

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*Wecome to Indian railways\*\*\*\*\*\*\*\*\*\*\*");

String username="Apoorva";

String password="HelloWorld@123";

System.out.println("Enter the user name:");

String uname=sc.nextLine();

System.out.println("Enter the password:");

String pwd=sc.nextLine();

if(uname.equals(username) && password.equals(password))//pssvali

{

System.out.println("Enter the source:");

String source=sc.nextLine();

System.out.println("Enter the destination:");

String destination=sc.nextLine();

System.out.println("Enter the date of travel");

String date=sc.nextLine();

System.out.println("\*\*\*\*\*\*\*\*\*\*Available trains on "+date+"\*\*\*\*\*\*\*\*\*\*\*\*");

System.out.println("TrainNo"+"\t"+"TrainName"+"\t"+"ArraivalTime"+"\t"+"DepartureTime");

System.out.println("123456\tAmaravathiExpress\t16:50\t17:05");

System.out.println("123488\tGodavariExpress\t17:50\t18:05");

System.out.println("123489\tGareebradthExpress\t18:50\t19:00");

System.out.println("123487\tPhalaknomaExpress\t22:50\23:05");

System.out.println("Provide number of the train in which you want to travel");

Long trainNo=sc.nextLong();

if(trainNo==123456 || trainNo==123488||trainNo==123489 || trainNo==123487 )

{

System.out.println("Enter the no. of. seats required");

int seats=sc.nextInt();

int fare=seats\*500;

System.out.println("Total fare:"+fare);

System.out.println("Want to confirm your request[y/n]");

char confirm=sc.next().charAt(0);

if(confirm=='y')

{

System.out.println("Your tickets have been confirmed");

System.out.println("Your train number "+trainNo);

System.out.println("No of seats "+seats);

System.out.println("Date of journey "+date);

System.out.println("Amount to be paid "+fare);

System.out.println("Thank you......Visit again");

}

else

{

System.out.println("Thank you......Visit again");

}

}

else

{

System.out.println("No train available");

}

}//if

else

{

System.out.println("Username or password is not valid");

}//password

}//main

}//class

***ATM application:***

import java.util.\*;

class ATM

{

public static void main(String a[])

{

int Balance=10000;

double newAccBal=0.0;

long AccountNo=1029384756658L;

String AccountHolder="Durga";

Scanner sc=new Scanner(System.in);

System.out.println("..........Welcome to ICICI Bank.............");

System.out.println("===============Select an option============");

System.out.println("Press 1 to deposit");

System.out.println("Press 2 for withdrawl");

System.out.println("Press 3 to display");

System.out.println("Press 4 to Exit");

System.out.println("===============================");

System.out.println("Enter your choice:");//2

int choice=sc.nextInt();

switch(choice)

{

case 1:

System.out.println("Enter the amount to be deposited");

int depositAmount=sc.nextInt();

if(depositAmount<0)

{

System.out.println("Negative amount");

}

else

{

Balance=Balance+depositAmount;

newAccBal=Balance;

System.out.println("The current balance in your account is:"+newAccBal);

}

break;

case 2:

System.out.println("Enter the amount to be withdrawn:");

int WithAmount=sc.nextInt();

if(WithAmount>Balance)

{

System.out.println("No sufficient funds");

}

else

{

System.out.println("With");

Balance=Balance-WithAmount;

newAccBal=Balance;

System.out.println("The current balance in your account is:"+newAccBal);

}

break;

case 3:

System.out.println("Your account details are:");

System.out.println("Account No:"+AccountNo);

System.out.println("Account Holder:"+AccountHolder);

System.out.println("Balance:"+Balance);

case 4:

System.exit(0);

default:

System.out.println("Invalid choice");

}

}

}

Names application:

import java.util.Scanner;

class Names

{

public static void main (String a[])

{

String gender,name;

Scanner sc=new Scanner(System.in);

System.out.println("select the gender of the baby");

gender=sc.nextLine();

if(gender.equals("male"))//Male

{

System.out.println(" Do you want to know American, Indian or Chinese names?");

name=sc.nextLine();//Indian

if (name.equals("American"))

{

System.out.println("bob");

System.out.println("Sam");

System.out.println("Bill");

}

if(name.equals("Indian"))

{

System.out.println("Surya");

System.out.println("Madhu");

}

if (name.equals("Chinese"))

{

System.out.println("lee");

System.out.println("kong");

}

}

else//Female

{

System.out.println(" Do you want to know American, Indian or Chinese names?");

name=sc.nextLine();//

if (name.equals("American"))

{

System.out.println("katy");

System.out.println("Sarah");

System.out.println("sofia");

}

if (name.equals("Indian"))

{

System.out.println("apoorva");

System.out.println("anita");

}

if (name.equals("Chinese"))

{

System.out.println("pink");

System.out.println("iss");

}

}//else

}//main

}//class

***18)Packages:***

Grouping of related classes and interfaces.

It is a mechanism which avoids naming collisions,there by allowing a programmer to use the same class name with different implementation.

Syntax to create a package:

package packagename;

Example:

package p1;

Syntax to put a class under a package:

package p1;

class A

{

}

Packages are used for reusing ,multiple classes of one file into any other file.

Syntax to access a class of a particular package:

import packagename.classname;

import packagename.\*;//Here \* indicates all the classes of a particular package.

Example:

package p1;

class A

{

body 1

}

package p2;

class A

{

body 2

}

If p1 is a package for jvm,then p1 is a folder for os.os will not allow two folders to have the same name.All the .class files are placed in the current working directory.

Basic program:

package mypack;

public class Simple1{

public static void main(String args[]){

System.out.println("Welcome to package");

}

}

Compilation and execution:

C:\Users\Welcome\Desktop>javac -d . Simple1.java

(A folder with the packname will be created in the desktop and .class file will be placed inside it.)

C:\Users\Welcome\Desktop>java mypack.Simple1

Welcome to package

Explanation:

Here -d indicates the destination,where the dot class files to be placed.If we give Dot(.), then the dot class files will be kept under current working directory.

For example....

C:\Users\Welcome\Desktop>javac -d d:/classfiles Simple1.java

In the above case,.class files will be kept under d--->classfiles folder.

Example 2:

PacksDemo========>Project name

===packdemo.java:

package p1;

public class packdemo

{

public void display()

{

System.out.println("Hello from the package p1");

}

}

===packagecall.java

import p1.packdemo;

public class packagecall

{

public static void main(String[] args)

{

packdemo pd=new packdemo();

pd.display();

}

}

Without importing the package...how to use a class?

By using fully qualified name...

packs=======>Project name

===A.java

package pack;

public class A{

public void msg()

{

System.out.println("Hello");

}

}

===B.java

package mypack;

class B{

public static void main(String args[]){

pack.A obj = new pack.A();//using fully qualified name

obj.msg();

}

}

This will also be used when two classes have same name...

===A.java

package pack;

public class A{

public void msg(){System.out.println("Hello");}

}

===A.java

package mypack;

class A{

public static void main(String args[]){

/\*A ob=new A();

ob.msg();\*/

pack.A obj = new pack.A();//using fully qualified name

obj.msg();

}

}

Access specifiers:

Access specifier tells the visibility of a member.

public,pivate,protected,default.

public---------------->accessible any where.

private-------------->accessible only within the class.

protected----------->accessible by the classes and

subclasses of the same package and the subclasses of other

packages.

default-------------->accessible only by the classes and

subclasses of the same package.

/\*

======one.java==========

package p1;

class A

{

int a;

}

class B extends A

{

}

======two.java==========

package p2;

class C

{

}

class D extends C

{

}

\*/

Example:

Access====>Project name

===Aa.java

package p2;

public class Aa

{

public void m1()

{

System.out.println("public member");

}

protected void m2()

{

System.out.println("protected member");

}

void m3()

{

System.out.println("Default member");

}

}

===Bb.java

package com;

import p2.\*;

public class Bb extends Aa

{

public static void main(String args[])

{

Aa a1=new Aa();

a1.m1();

//a1.m2();

//a1.m3();

Bb b1=new Bb();

b1.m1();

b1.m2();

//b1.m3();

//default classes(only defaultlevel scope)

}

}

Points:

1)There should only one package statement per a file.

2)\* indicates all classes of a particular package.

3)If there exist a package statement in a file,it should be the

first statement.

There can be any no. of. packages inside a package.

A package inside another package is called a subpackage.

If we want to use the classes of a sub package...

the syntax is.....

import java.packagename.subpackegename.classname;

P1-------->p2-c3,p3-c4,p4-c5

-C1

-C2

C1 c4

import P1.\*;

import P1.p3.c4;

=one.java:

import P1.\*;

//import p3.c4;

import P1.p3.c4;

Example:

import javax.servlet.\*;

import javax.servlet.http.\*;

servlet------->packagename

http---------->subpackagename

\*------------>all classes of subpackage

/\*

P1-------->p2-c3,p3-c4,p4-c5

-C1

-C2

import P1.C2;

import P1.p3.c4;

\*/

The standard of defining package is domain.company.package e.g. com.qshore.bean or org.sssit.dao.

Example on sub packages

subpack========>project name

===Simple.java

package com.qshore.core;

public class Simple{

public void display()

{

System.out.println("Hello subpackage");

}

}

===other.java

import com.qshore.core.\*;

public class other

{

public static void main(String args[])

{

Simple s=new Simple();

s.display();

}

}

How to put two public classes in a package?

If you want to put two public classes in a package, have two java source files containing one public class, but keep the package name same.

For example:

//save as A.java

package Qshore;

public class A

{

}

//save as B.java

package Qshore;

public class B

{

}

***19)Java Inner classes:***

Java inner class or nested class is a class i.e. declared inside the class or interface.

We use inner classes to logically group classes and interfaces in one place so that it can be more readable and maintainable.

Additionally, it can access all the members of outer class including private data members and methods.

Syntax of Inner class:

class Java\_Outer\_class{

//code

class Java\_Inner\_class{

//code

}

}

Advantage of java inner classes

There are basically three advantages of inner classes in java. They are as follows:

1) Nested classes represent a special type of relationship that is it can access all the members (data members and methods) of outer class including private.

2) Nested classes are used to develop more readable and maintainable code because it logically group classes and interfaces in one place only.

3) Code Optimization: It requires less code to write.Java Member inner class

A non-static class that is created inside a class but outside a method is called member inner class.

Syntax:

class Outer{

//code

class Inner{

//code

}

}

inner class==

class TestMemberOuter1{

private int data=30;

class Inner{

void msg(){System.out.println("data is "+data);}

}

public static void main(String args[]){

TestMemberOuter1 obj=new TestMemberOuter1();

TestMemberOuter1.Inner in=obj.new Inner();

in.msg();

}

}

Anonymous inner class:

A class that have no name is known as anonymous inner class in java. It should be used if you have to override method of class or interface. Java Anonymous inner class can be created by two ways:

Class (may be abstract or concrete).

Interface

Java anonymous inner class example using class

Example:

abstract class Person{

abstract void eat();

}

class TestAnonymousInner{

public static void main(String args[]){

Person p=new Person()

{

void eat()

{

System.out.println("nice fruits");

}

};

p.eat();

}

}

static nested classes:

A static class i.e. created inside a class is called static nested class in java. It cannot access non-static data members and methods. It can be accessed by outer class name.

It can access static data members of outer class including private.

Static nested class cannot access non-static (instance) data member or method.

Java static nested class example with instance method

Example:

class TestOuter1{

static int data=30;

static class Inner{

void msg()

{

System.out.println("data is "+data);

}

}

public static void main(String args[]){

TestOuter1.Inner obj=new TestOuter1.Inner();

obj.msg();

}

}

Test it Now

Output:

data is 30

In this example, you need to create the instance of static nested class because it has instance method msg(). But you don't need to create the object of Outer class because nested class is static and static properties, methods or classes can be accessed without object.

Innerclasses.txtOpen

Displaying innerclasses.txt.

***20)Interfaces:***

An interface in java is a blueprint of a class. It has static constants and abstract methods.

The interface in java is a mechanism to achieve abstraction. There can be only abstract methods in the java interface not method body. It is used to achieve abstraction and multiple inheritance in Java.

Advantages of using an interface:

It is used to achieve abstraction.

By interface, we can support the functionality of multiple inheritance.It can be used to achieve loose coupling.

As shown in the figure given below, a class extends another class, an interface extends another interface but a class implements an interface.

Program to show the problem with multiple inheritence:

class A

{

void display()

{

System.out.println("A()");

}

}

class B

{

void display()

{

System.out.println("B()");

}

}

class C extends A,B//Not supported

{

public static void main(String a[])

{

C ob=new C();

ob.display();

}

}

Java indirectly supports multiple inheritence through

interfaces.

Syntax:

interface InterfaceName

{

}

Writing an interface is similar to writing a class. But a class describes the attributes and behaviors of an object. And an interface contains behaviors that a class implements.Unless the class that implements the interface is abstract, all the methods of the interface need to be defined in the class.

An interface is similar to a class in the following ways -

An interface can contain any number of methods.An interface is written in a file with a .java extension, with the name of the interface matching the name of the file.The byte code of an interface appears in a .class file.

Interfaces appear in packages, and their corresponding bytecode file must be in a directory structure that matches the package name.However, an interface is different from a class in several ways, including -

You cannot instantiate an interface.

An interface does not contain any constructors.

All of the methods in an interface are abstract.

An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.

An interface is not extended by a class; it is implemented by a class.

An interface can extend multiple interfaces.

Differences between a class and an interface:

Class Interface

1)Collection of both 1)Collection of only abstract

abstract and non abstract methods.

methods.

2)Variables are instance 2)By default,variables are

by default. static and final.

3)Methods are instance 3)Methods are public and

by default. abstract by default.

Example 1:

interface A

{

void display();

}

class B implements A

{

void display()

{

System.out.println("Interface Example");

}

public static void main(String a[])

{

B b=new B();

b.display();

}

}

There should not be any object for the interface.An interface can have references.

c--------->c ----- extends

I----------->I ------ extends

I------------>c ------ implements

c------------->I ------- implements(Not there)

Example 2:

interface printable

{

void print();

}

interface showable

{

void show();

}

class Hai implements printable,showable

{

public void print()

{

System.out.println("Print it");

}

public void show()

{

System.out.println("Show it");

}

public static void main(String a[])

{

Hai h=new Hai();

h.print();

h.show();

}

}

(Same method names) Multiple inheritance is not supported through

class in java but it is possible by interface:

interface Printable{

void print();

}

interface Showable{

void print();

}

class TestTnterface1 implements Printable,Showable

{

public void print()

{

System.out.println("Hello");

}

public static void main(String args[])

{

TestTnterface1 obj = new TestTnterface1();

obj.print(); //No ambiguity

}

}

//class A extends B implements C

Extending interfaces

An interface can extend another interface

interface Printable{

public abstract void print();

}

interface Showable extends Printable

{

public abstract void show();

}

class Testinterface2 implements Showable{

public void print()

{

System.out.println("Hello");

}

public void show()

{

System.out.println("Welcome");

}

public static void main(String args[])

{

Testinterface2 obj = new Testinterface2();

obj.print();

obj.show();

}

}

\*\*\*\*\*\*\*\*\*\*\*We can never create an object for an abstract

class or for an interface\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

classes-------------->Objects

interfaces----------->references

Interfaces with dynamic method dispatch:

Example 1:

interface Area

{

static final float pi=3.14f;

public abstract float compute(float x,float y);

}

class Circle implements Area

{

public float compute(float x,float y)

{

return (x\*y);

}

}

class Rectangle implements Area

{

public float compute(float x,float y)

{

return (pi\*x\*y);

}

}

class InterfaceTest

{

public static void main(String aa[])

{

Area a;///Resolving ambiguity

Circle c=new Circle();

Rectangle r=new Rectangle();

a=c;

System.out.println(a.compute(14,14)+"is the area of circle");

a=r;

System.out.println(a.compute(15,15)+"is the area of rectangle");

}

}

Example 2:

interface Bank{

float rateOfInterest();

}

class SBI implements Bank{

public float rateOfInterest(){return 9.15f;}

}

class PNB implements Bank{

public float rateOfInterest(){return 9.7f;}

}

class TestInterface2{

public static void main(String[] args){

Bank b=new SBI();

System.out.println("ROI: "+b.rateOfInterest());

}

}

***21)Arrays:***

Array is a collection of homogeneous/similar datatype elements sharing a common name.Array in java is index based, first element of the array is stored at 0 index.

Advantage of Java Array:

Code Optimization: It makes the code optimized, we can retrieve or sort the data easily.Random access: We can get any data located at any index position.

Disadvantages:

We can store homogeneous values only.

Size Limit: We can store only fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in java.

Types:

There are two types of array.

1)Single Dimensional Array

2)Multidimensional Array

Single Dimensional Array in java:

Syntax to Declare an Array in java

dataType[] arr; (or)

dataType []arr; (or)

dataType arr[];

Instantiation of an Array in java:

arrayRefVar=new datatype[size];

Example of single dimensional java array:

Let's see the simple example of java array, where we are going to declare, instantiate, initialize and traverse an array.

class Testarray{

public static void main(String args[]){

int a[]=new int[5];//declaration and instantiation

a[0]=10;//initialization

a[1]=20;

a[2]=70;

a[3]=40;

a[4]=50;

//printing array

for(int i=0;i<a.length;i++)//length is the property of array

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

}

}

Test it Now

Output: 10

20

70

40

50

Declaration, Instantiation and Initialization of Java Array

We can declare, instantiate and initialize the java array together by:

int a[]={33,3,4,5};//declaration, instantiation and initialization

Let's see the simple example to print this array.

class Testarray1{

public static void main(String args[]){

int a[]={33,3,4,5};//declaration, instantiation and initialization

//printing array

for(int i=0;i<a.length;i++)//length is the property of array

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

}

}

Test it Now

Output:33

3

4

5

Passing Array to method in java:

We can pass the java array to method so that we can reuse the same logic on any array.

Let's see the simple example to get minimum number of an array using method.

class Testarray2{

static void min(int arr[]){

int min=arr[0];

for(int i=1;i<arr.length;i++)

if(min>arr[i])

min=arr[i];

System.out.println(min);

}

public static void main(String args[]){

int a[]={33,3,4,5};

min(a);//passing array to method

}

}

Test it Now

Output:3

Multidimensional array in java:

In such case, data is stored in row and column based index (also known as matrix form).

Syntax to Declare Multidimensional Array in java

dataType[][] arrayRefVar; (or)

dataType [][]arrayRefVar; (or)

dataType arrayRefVar[][]; (or)

dataType []arrayRefVar[];

Example to instantiate Multidimensional Array in java

int[][] arr=new int[3][3];//3 row and 3 column

Example to initialize Multidimensional Array in java

arr[0][0]=1;

arr[0][1]=2;

arr[0][2]=3;

arr[1][0]=4;

arr[1][1]=5;

arr[1][2]=6;

arr[2][0]=7;

arr[2][1]=8;

arr[2][2]=9;

Example of Multidimensional java array

Let's see the simple example to declare, instantiate, initialize and print the 2Dimensional array.

class Testarray3{

public static void main(String args[]){

//declaring and initializing 2D array

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

//printing 2D array

for(int i=0;i<3;i++){

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

System.out.print(arr[i][j]+" ");

}

System.out.println();

}

}

}

Test it Now

Output:1 2 3

2 4 5

4 4 5

What is the class name of java array?

In java, array is an object. For array object, an proxy class is created whose name can be obtained by getClass().getName() method on the object.

class Testarray4{

public static void main(String args[]){

int arr[]={4,4,5};

Class c=arr.getClass();

String name=c.getName();

System.out.println(name);

}

}

Test it Now

Output:I

Copying a java array:

We can copy an array to another by the arraycopy method of System class.

Syntax of arraycopy method

public static void arraycopy(

Object src, int srcPos,Object dest, int destPos, int length )

Example of arraycopy method

class TestArrayCopyDemo {

public static void main(String[] args) {

char[] copyFrom = { 'd', 'e', 'c', 'a', 'f', 'f', 'e', 'i', 'n', 'a', 't', 'e', 'd' };

char[] copyTo = new char[7];

System.arraycopy(copyFrom, 2, copyTo, 0, 7); System.out.println(new String(copyTo));

}

}

Test it Now

Output:caffein

Addition of 2 matrices in java

Let's see a simple example that adds two matrices.

class Testarray5{

public static void main(String args[]){

//creating two matrices

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

int b[][]={{1,3,4},{3,4,5}};

//creating another matrix to store the sum of two matrices

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

//adding and printing addition of 2 matrices

for(int i=0;i<2;i++){

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

c[i][j]=a[i][j]+b[i][j];

System.out.print(c[i][j]+" ");

}

System.out.println();//new line

}

}

}

Test it Now

Output:2 6 8

6 8 10

Program to accept elements from the user and storing them in an array and also to display the elements out of an array:

import java.util.Scanner;

class Array1

{

public static void main(String args[])

{

int a[]=new int[10];

int i;

Scanner sc=new Scanner(System.in);

System.out.println("Enter the number of elements you want to store into the array");

int n=sc.nextInt();

System.out.println("Enter the " +n+ " values");

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

{

//System.out.println("Enter the values:");

a[i]=sc.nextInt();

}

System.out.println("The array elements are:");

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

{

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

}

}

}

Output:

Enter the number of elements you want to store into the array

6

Enter the 6 values

1

2

3

4

5

6

The array elements are:

1

2

3

4

5

6

To print the elements out of an array using enhanced for loop/for each loop:

class Demo

{

public static void main(String aa[])

{

int a[]=new int[5];

a[0]=10;

a[2]=30;

a[4]=40;

for(int result:a)

{

System.out.println(result);

}

}

}

output:

10

0

30

0

40

***22)Command Line Arguments:***

Argumets that are passed in the same line where the execution

command is given are called as Command Line Arguments.

(or)

The java command-line argument is an argument i.e. passed at the time of running the java program.

The arguments passed from the console can be received in the java program and it can be used as an input.So, it provides a convenient way to check the behavior of the program for the different values. You can pass N (1,2,3 and so on) numbers of arguments from the command prompt.

Purpose:

This is generally used for testing our applications with different test cases.

Example 1:

class cla

{

public static void main(String ar[])

{

int a=Integer.parseInt(ar[0]);

int b=Integer.parseInt(ar[1]);

int c=a+b;

System.out.println("The sum is:"+c);

}

}

output:

javac cla.java

java cla 2 3

The sum is:5

ar

2

3

Example 2:

class cml

{

{

public static void main(String a[])

{

System.out.println("Your first arg is:"+a[0]);

}

}

javac cml.java

java cml 23 hello

Example 3:

class cml

{

public static void main(String a[])

{

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

{

System.out.println("The args are:"+a[i]);

}

}

}

javac cml.java

java cml hello hai 22 42 ok bye

a

0-hello

1-hai

2-22

3-42

4-ok

5-bye

***23)Static blocks:***

A static initialization block is a normal block of code enclosed in braces, { }, and preceded by

the static keyword. Here is an example:

static {

// whatever code is needed for initialization goes here

}

A class can have any number of static initialization blocks, and they can appear anywhere in the

class body. The runtime system guarantees that static initialization blocks are called in the order

that they appear in the source code. And dont forget, this code will be executed when JVM loads the

class. JVM combines all these blocks into one single static block and then executes. Here are a couple of points I like to mention:

If you have executable statements in the static block, JVM will automatically execute these statements when the class is loaded into JVM.

If you’re referring some static variables/methods from the static blocks, these statements will be executed after the class is loaded into JVM same as above i.e., now the static variables/methods referred and the static block both will be executed.

So,Static blocks have higher priority than the static methods.

Example:

class StaticDemo

{

static

{

System.out.println("From block 1");

}

static

{

System.out.println("From block 2");

}

void display()

{

System.out.println("I am from display");

}

static

{

System.out.println("From block 3");

}

public static void main(String a[])

{

StaticDemo sd=new StaticDemo();

sd.display();

}

}

output:

From block 1

From block 2

From block 3

I am from display

We can initialize static variables in static blocks.

class Test {

static int i;

int j;

// start of static block

static {

i = 10;

System.out.println("static block called ");

}

// end of static block

}

class Main {

public static void main(String args[]) {

// Although we don't have an object of Test, static block is

// called because i is being accessed in following statement.

System.out.println(Test.i);

}

}

Also, static blocks are executed before constructors. For example, check output of following Java program.

class Test {

static int i;

int j;

static {

i = 10;

System.out.println("static block called ");

}

Test(){

System.out.println("Constructor called");

}

}

class Main {

public static void main(String args[]) {

// Although we have two objects, static block is executed only once.

Test t1 = new Test();

Test t2 = new Test();

}

}

Output:

static block called

Constructor called

Constructor called

Advantages of static blocks:

1)If you’re loading drivers and other items into the namespace. For example, Class class has a static block where it registers the natives.

2)If you need to do computation in order to initialize your static variables,you can declare a static block which gets executed exactly once,when the class is first loaded.

3)Security related issues or logging related tasks

Disadvantages:

1)There is a limitation of JVM that a static initializer block should not exceed 64K.

2)You cannot throw Checked Exceptions.

3)You cannot use this keyword since there is no instance.

4)You shouldn’t try to access super since there is no such a thing for static blocks.

5)You should not return anything from this block.

6)Static blocks make testing a nightmare.

***24)Initializer blocks/Instance initializer blocks:***

Syntax:

{

//Common code to be executed by the constructors will be written here.

}

Before the constructor is executed,first all the initializer blocks will be executed in the top down manner and then only the original constructor will be executed.

For how many times there is a call for constructor for that many times all the initializer blocks will be executed.

Example:

class Demo

{

{

System.out.println("From initializer block 1");

}

{

System.out.println("From initializer block 2");

}

Demo()

{

System.out.println("I am from constructor");

}

{

System.out.println("From initializer block 3");

}

main()

{

Demo d1=new Demo();

Demo d2=new Demo();

}

output:

From initializer block 1

From initializer block 2

From initializer block 3

I am from constructor

From initializer block 1

From initializer block 2

From initializer block 3

I am from constructor

Instance Initializer block is used to initialize the instance data member. It run each time when object of the class is created.

The initialization of the instance variable can be directly but there can be performed extra operations while initializing the instance variable in the instance initializer block.

What is the use of instance initializer block while we can directly assign a value in instance data member? For example:

class Bike{

int speed=100;

}

Why use instance initializer block?

Suppose I have to perform some operations while assigning value to instance data member e.g. a for loop to fill a complex array or error handling etc.

Example of instance initializer block:

Let's see the simple example of instance initializer block the performs initialization.

class Bike7{

int speed;

Bike7(){System.out.println("speed is "+speed);}

{speed=100;}

public static void main(String args[]){

Bike7 b1=new Bike7();

Bike7 b2=new Bike7();

}

}

Test it Now

Output:speed is 100

speed is 100

There are three places in java where you can perform operations:

Method

constructor

block

What is invoked first, instance initializer block or constructor?

class Bike8{

int speed;

Bike8(){System.out.println("constructor is invoked");}

{System.out.println("instance initializer block invoked");}

public static void main(String args[]){

Bike8 b1=new Bike8();

Bike8 b2=new Bike8();

}

}

Test it Now

Output:instance initializer block invoked

constructor is invoked

instance initializer block invoked

constructor is invoked

In the above example, it seems that instance initializer block is firstly invoked but NO. Instance intializer block is invoked at the time of object creation. The java compiler copies the instance initializer block in the constructor after the first statement super(). So firstly, constructor is invoked. Let's understand it by the figure given below:

Note: The java compiler copies the code of instance initializer block in every constructor.

***25)Exception Handling:***

What is exception:Its a situation that deviates the program from normal flow of execution.If an exception occurs,the program will be terminated abnormally.Even the server may crash.So we can treat an exception is an abnormal condition.In java, exception is an event that disrupts the normal flow of the program. It is an object which isthrown at runtime.

statement 1;

statement 2;

statement 3;

statement 4;

statement 5;//exception occurs

statement 6;

statement 7;

statement 8;

statement 9;

statement 10;

Suppose there is 10 statements in your program and there occurs an exception at statement 5,rest of the code will not be executed i.e. statement 6 to 10 will not run. The exception handling in java is one of the powerful mechanism to handle the runtime errors.

Java Exception Handling mechanism involves 5 keywords.

try

catch

throw

throws

finally

In java,Exceptions are properly represented using a class.If an exception occurs in a try block,JVM will implicitly creates an object and embeds the information about the exception.

The corresponding catch block throws a reference to that object.

try:

Those statements that probably raise an exception are under this try block.

try

{

}

catch:

This block is capable of catching the exception.

catch

{

}

throws:

It is used to declare/specify an exception.

throw:

To create custom or userdefined exceptions.

finally:

That block of statements that have to be executed for sure are kept under this block.Important instructions will be kept in this block.For example,closing of resources.,etc.

con.close()//To close the connection

Types of exceptions:

Checked

Unchecked

-Errors

-RunTimeException

Checked Exceptions:

The classes that extend Throwable class except RuntimeException and Error are known as checked

exceptions e.g.IOException, SQLException etc. Checked exceptions are checked at compile-time.

Unchecked Exceptions:

The classes that extend RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time rather they are checked at runtime.

Error:

It is external to the application.For example,insufficient memory.

RunTimeException:These can be avoided if a little bit of care is taken by the programmer.Common scenarios where exceptions may occur There are given some scenarios where unchecked exceptions can occur. They are as follows:

1) Scenario where ArithmeticException occurs

If we divide any number by zero, there occurs an ArithmeticException.

int a=50/0;//ArithmeticException

2) Scenario where NullPointerException occurs

If we have null value in any variable, performing any operation by the variable occurs an

NullPointerException.

String s=null;

System.out.println(s.length());//NullPointerException

3) Scenario where NumberFormatException occurs

The wrong formatting of any value, may occur NumberFormatException. Suppose I have a string variable that have characters, converting this variable into digit will occur NumberFormatException.

String s="abc";

int i=Integer.parseInt(s);//NumberFormatException

4) Scenario where ArrayIndexOutOfBoundsException occurs

If you are inserting any value in the wrong index, it would result ArrayIndexOutOfBoundsException as shown below:

int a[]=new int[5];

a[10]=50; //ArrayIndexOutOfBoundsException

Example on try-catch:

class trycatch

{

public static void main(String aRGS[])

{

try

{

int a=10,b=0,c;

c=a/b;//Object

int d[]=new int[15];

d[55]=79;

}

catch(ArrayIndexOutOfBoundsException abe)

{

System.out.println("Sorry");

}

catch(ArithmeticException ae)

{

//ae.printStackTrace();

System.out.println("Sorry...Pls try after some time");

}

finally

{

System.out.println("I will be executed for sure");

}

}

}

output:

Sorry...pls try after some time

I will be executed for sure

printStackTrace() is used to print the predefined messages.

If we apply printStackTrace() instead of SOP,then the output is:

java.lang.ArithmeticException: / by zero

at trycatch.main(trycatch.java:8)

I will be executed for sure

throws Keyword:

If an exception occurs in a method and we dont want to catch it using try catch blocks,then we can leave that burden to the compiler using "throws" keyword after the method.

Example:

void display() throws ArithmeticException

{

}

throw keyword:

To throw custom exceptions/our own exceptions:

Example:

class throwdemo

{

static void display(int age)//21

{

if(age<23) throw new ArithmeticException("Not eligible for marriage");

else

System.out.println("Eligible for marriage");

}

public static void main(String a[])

{

throwdemo.display(21);

}

}

java.lang.ArithmeticException:Not eligible for marriage at trycatch.main(trycatch.java:8)

You can also throw an already defined exception like ArithmeticException, IOException etc.

Syntax of throw statement

throw AnyThrowableInstance;

Example:

//A void method

public void sample()

{

//Statements

//if (somethingWrong) then

IOException e = new IOException();

throw e;

//More Statements

}

Rethrowing an Exception:

We all know that exceptions occurred in the try block are caught in catch block. Thus caught exceptions can be re-thrown using throw keyword. Re-thrown exception must be handled some where in the program, otherwise program will terminate abruptly. If we want to throw an exception again,then we will write as below

catch(IOException e)

{

...................

throw e;

}

java.lang.Throwable------------->super class of all the exceptions.

Example:

public class ExceptionHandling

{

public static void main(String[] args)

{

try

{

methodWithThrow();

}

catch(NullPointerException ex)

{

System.out.println("NullPointerException Re-thrown in methodWithThrow() method will be handled here");

}

}

static void methodWithThrow()

{

try

{

String s = null;

System.out.println(s.length()); //This statement throws NullPointerException

}

catch(NullPointerException ex)

{

System.out.println("NullPointerException is caught here");

throw ex; //Re-throwing NullPointerException

}

}

}

Multi catch blocks:

In Java SE 7 and later, a single catch block can handle more than one type of exception. This feature can reduce code duplication and lessen the temptation to catch an overly broad exception.

Consider the following example, which contains duplicate code in each of the catch blocks:

catch (IOException ex) {

System.out.println(ex);

throw ex;

catch (SQLException ex) {

System.out.println(ex);

throw ex;

}

In releases prior to Java SE 7, it is difficult to create a common method to eliminate the duplicated code because the variable ex has different types.

The following example, which is valid in Java SE 7 and later, eliminates the duplicated code:

catch (IOException|SQLException ex) {

logger.log(ex);

throw ex;

}

The catch clause specifies the types of exceptions that the block can handle, and each exception type is separated with a vertical bar (|).

Classes Hierarchy:

Throwable(parent)

-Exception(child)

-ArithmeticException

-NullPointerException =======>chils classes of Exception class

-NumberFormatException,etc

Throwable is a class of java.lang package.

***26)MULTI THREADING:***

Thread:

1. A flow of control is known as thread.

2. If a program contains multiple flow of controls for achieving concurrent execution then that program is known as multi threaded program.

3. A program is said to be a multi threaded program if and only if in which there exist ‘n’ number of sub-programs there exist a separate flow of control. All such flow of controls are executing concurrently such flow of controls are known as threads and such type of applications or programs is called multi threaded programs.

The languages like C, C++ comes under single threaded modeling languages, since there exist single flow of controls where as the languages like JAVA, DOT NET are treated as multi threaded modeling languages, since there is a possibility of creating multiple flow of controls.When we write any JAVA program there exist two threads they are fore ground thread and back ground thread.

Fore ground threads are those which are executing user defined sub-programs where as back ground threads are those which are monitoring the status of fore ground thread. There is a possibility of creating ‘n’ number of fore ground threads and always there exist single back ground thread.Multi threading is the specialized form of multi tasking of operating system.In information technology we can develop two types of applications. They are process base dapplications and thread based applications.

Process Based Applications:

1. It is the one in which there exist single flow of control.

2. All C, C++ applications comes under it.

3. Context switch is more (context switch is the concept of operating system and it says switching the control from one address page to another address page).

4. For each and every sub-program there exist separate address pages.

5. These are treated as heavy weight components.

6. In this we can achieve only sequential execution and they are notrecommending for developing internet applications.

Thread Based Applications:

1. It is the one in which there exist multiple flow of controls.

2. All JAVA, DOT NET applications comes under it.

3. Context switch is very less.

4. Irrespective of ‘n’ number of subprograms there exist single address page.

5. These are treated as light weight components.

6. In thread based applications we can achieve both sequential and concurrent execution and they are always recommended for developing interact applications.

States of a thread:

When we write any multi threading applications, there exist ‘n’ numbers of threads. All the threads will under go different types of states. In JAVA for a thread we have five states. They are new, ready, running, waiting and halted or dead state.New state: It is one in which the thread about to enter into main memory.

Ready state: It is one in which the thread is entered into memory space allocated and it is waiting for CPU for executing.

Running state: A state is said to be a running state if and only if the thread is under the control of CPU.

Waiting state:

It is one in which the thread is waiting because of the following factors:

a) For the repeating CPU burst time of the thread (CPU burst time is an amount of the required by the thread by the CPU).

b) Make the thread to sleep for sleep for some specified amount of time.c) Make the thread to suspend.

d) Make the thread to wait for a period of time.

e) Make the thread to wait without specifying waiting time.

Halted state:

It is one in which the thread has completed its total execution. As long as the thread is in new and halted states whose execution status is false where as when the thread is in ready, running and waiting states that execution states is true.

Creating a thread:

In order to create a thread in JAVA we have two ways. They are by using java.lang.Thread

class and by using java.lang.Runnable interface.

In multi threading we get only one exception known as java.lang.InterruptedException.

Using java.lang.Thread:

Creating a flow of control in JAVA is nothing but creating an object of java.lang.Thread class.

An object of Thread class can be created in three ways. They are:

i) Directly Thread t=new Thread ();

ii) Using factory method Thread t1=Thread.currentThread ();

iii) Using sub-class of Thread class

class C1 extends Thread

{

……………………;

……………………;

};

C1 o1=new C1 ();

Thread t1=new C1 ();

Here, C1 is the sub-class of Thread class.

Thread API:

public static final int MAX\_PRIORITY (10);

public static final int MIN\_PRIORITY (1);

public static final int NORM\_PRIORITY (5);

The above data members are used for setting the priority to threads are created. By default,whenever a thread is created whose default priority NORM\_PRIORITY.

Constructors:

i) Thread (): With this Constructor we can create an object of the Thread class whose default thread name is Thread-0.

For example:

Thread t=new Thread ();

System.out.println (t.getName ());// Thread-0

ii) Thread (String): This Constructor is used for creating a thread and we can give the user specified thread name.

For example:

Thread t=new Thread (“JAVA”);

t.setName (“JAVA”);

t.setPriority (Thread.MAX\_PRIORITY);

iii) Thread (Runnable):

This Constructor is used for converting Runnable object into Thread object for entering into run method of Runnable interface by making use of start method of Thread class without giving thread name.

iv) Thread (Runnable, String):

This Constructor is similar to above Constructor but we give thread

name through this Constructor.

Instance methods:

public final void setName (String);

public final String getName ();

The above two methods are used for setting the name of the thread and getting the name from the thread respectively.

For example:

Thread t1=new Thread ();

T1.setName (“JAVA”);

String tp=t1.getName ();

System.out.println (tp);// JAVA

public final void setPriority (int);

public final int getPriority ();

The above two methods are used for setting the priority to the thread and getting the priority of the thread respectively.

For example:

Thread t1=new Thread ();

Int pri=t1.getPriority ();

System.out.println (pri);// 5 by default

t1.setPriority (Thread.MAX\_PRIORITY);

pri=t1.getPriority ();

System.out.println (pri); // 10

public void run ():

Any JAVA programmer want to define a logic for the thread that logic must be defined only

run () method. When the thread is started, the JVM looks for the appropriate run () method for

executing the logic of the thread. Thread class is a concrete class and it contains all defined methods and all these methods are being to final except run () method. run () method is by default contains a definition with null body. Since we are providing the logic for the thread in run () method. Hence it must be overridden by extending Thread class into our own class.

For example:

class C1 extends Thread

{

public void run ()

{

………………………;

………………………;

}

};

public final void start ():

This is the method which is used for making the Thread to start to execute the thread logic.

The method start is internally calling the method run ().

For example:

Thread t1=new Thread ();

t1.start ();

Thread t2=Thread.currentThread ();

t2.start ();

public final void suspend ():

This method is used for suspending the thread from current execution of thread. When the thread is suspended, it sends to waiting state by keeping the temporary results in process controlblock (PCB) or job control block (JCB).

public final void resume ():

This method is used for bringing the suspended thread from waiting state to ready state.

When the thread is resumed to start executing from where it left out previously by retrieving the previous result from PCB.

public final void stop ():

This method is used to stop the execution of the current thread and the thread goes to halted state from running state. When the thread is restarted it starts executing from the beginning only.

public final void wait (long msec):

This method is used for making the currently executing thread into waiting state for a period of time. Once this period of time is over, automatically the waiting thread will enter into ready state from waiting state.

Static methods:

i) public static void sleep (long msec) throws InterruptedException method is used (waiting state). If the sleep time is over automatically thread will come from waiting state to ready state.

For example:

Thread.sleep (1000);

ii) public static Thread currentThread () is used for obtaining the threads which are running in the main memory of the computer.

For example:

Thread t=Thread.currentThread ();

System.out.println (t);// Thread [main (fat), 5, main (bat)]

Write a JAVA program to find the threads which are running internally and print priority values.

Answer:

class ThDemo

{

public static void main (String [] args)

{

Thread t=Thread.currentThread ();

System.out.println (t);

t.setName ("ABC");

System.out.println (t);

System.out.println ("IS IT ALIVE..?"+t.isAlive ());// true

Thread t1=new Thread ();// new state

System.out.println ("IS IT ALIVE..?"+t.isAlive ());// false

System.out.println ("DEFAULT NAME OF THREAD = "+t1.getName ());// Thread-0

System.out.println ("MAXIMUM PRIORITY VALUE = "+Thread.MAX\_PRIORITY);// 10

System.out.println ("MINIMUM PRIORITY VALUE = "+Thread.MIN\_PRIORITY);// 1

System.out.println ("NORMAL PRIORITY VALUE = "+Thread.NORM\_PRIORITY);// 5

}

};

Output:

Thread[main,5,main]

Thread[ABC,5,main]

IS IT ALIVE..?true

IS IT ALIVE..?true

DEFAULT NAME OF THREAD = Thread-0

MAXIMUM PRIORITY VALUE = 10

MINIMUM PRIORITY VALUE = 1

NORMAL PRIORITY VALUE = 5

Instance methods:

public boolean isAlive () method is used for checking whether the thread is executing or not. It returns ‘true’ as long as the thread is in ready running and waiting states. It returns ‘false’ as long as the thread is in new and halted state.

Write a thread program which displays 1 to 10 numbers after each and every 1 second.

Answer:

class Th1 extends Thread

{

public void run ()

{

try

{

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

{

System.out.println ("VALUE OF I = "+i);

Thread.sleep (1000);

}

}

catch (InterruptedException ie)

{

System.out.println (ie);

}

}

};

class ThDemo1

{

public static void main (String [] args)

{

Th1 t1=new Th1 ();

System.out.println ("IS T1 ALIVE BEFORE START = "+t1.isAlive ());

t1.start ();

System.out.println ("IS T1 ALIVE AFTER START = "+t1.isAlive ());

}

};

Output:

IS T1 ALIVE BEFORE START = false

IS T1 ALIVE AFTER START = true

VALUE OF I = 1

VALUE OF I = 2

VALUE OF I = 3

VALUE OF I = 4

VALUE OF I = 5

VALUE OF I = 6

VALUE OF I = 7

VALUE OF I = 8

VALUE OF I = 9

VALUE OF I = 10

Re-write the above program using runnable interface.

Answer:

class Th1 implements Runnable

{

public void run ()

{

try

{

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

{

System.out.println ("VALUE OF I = "+i);

Thread.sleep (1000);

}

}

catch (InterruptedException ie)

{

System.out.println (ie);

}

}

};

class ThDemo2

{

public static void main (String [] args)

{

Runnable t=new Th1 ();

Thread t1=new Thread (t, "ABC");

System.out.println ("THREAD NAME = "+t1.getName ());

System.out.println ("IS T1 ALIVE BEFORE START = "+t1.isAlive ());

t1.start ();

System.out.println ("IS T1 ALIVE AFTER START = "+t1.isAlive ());

}

};

Output:

THREAD NAME = ABC

IS T1 ALIVE BEFORE START = false

IS T1 ALIVE AFTER START = true

VALUE OF I = 1

VALUE OF I = 2

VALUE OF I = 3

VALUE OF I = 4

VALUE OF I = 5

VALUE OF I = 6

VALUE OF I = 7

VALUE OF I = 8

VALUE OF I = 9

VALUE OF I = 10

Write a JAVA program which produces 1 to 10 numbers in which even numbers are produced by one thread and odd numbers are produced by another thread.

Answer:

class Th1 extends Thread

{

public void run ()

{

try

{

for (int i=1; i<=10; i+=2)

{

System.out.println ("VALUE OF ODD : "+i);

Thread.sleep (1000);

}

}

catch (InterruptedException ie)

{

System.out.println (ie);

}

}

};

class Th2 implements Runnable

{

public void run ()

{

try

{

for (int j=2; j<=10; j+=2)

{

System.out.println ("VALUE OF EVEN : "+j);

Thread.sleep (1000);

}

}

catch (InterruptedException ie)

{

System.out.println (ie);

}

}

};

class ThDemo6

{

public static void main (String [] args)

{

Th1 t1=new Th1 ();// object of Thread class

Th2 t2=new Th2 ();// object of Runnable class

Thread t=new Thread (t2);// Runnable is converted into Thread object

System.out.println ("BEFORE START T1 IS : "+t1.isAlive ());

System.out.println ("BEFORE START T2 IS : "+t.isAlive ());

t1.start ();

t.start ();

System.out.println ("AFTER START T1 IS : "+t1.isAlive ());

System.out.println ("AFTER START T2 IS : "+t.isAlive ());

try

{

t1.join ();// to make thread to join together for getting performance

t.join ();

}

catch (InterruptedException ie)

{

System.out.println (ie);

}

System.out.println ("AFTER JOINING T1 IS : "+t1.isAlive ());

System.out.println ("AFTER JOINING T2 IS : "+t.isAlive ());

}

};

Output:

BEFORE START T1 IS : false

BEFORE START T2 IS : false

AFTER START T1 IS : true

AFTER START T2 IS : true

VALUE OF ODD : 1

VALUE OF EVEN : 2

VALUE OF ODD : 3

VALUE OF EVEN : 4

VALUE OF ODD : 5

VALUE OF EVEN : 6

VALUE OF ODD : 7

VALUE OF EVEN : 8

VALUE OF ODD : 9

VALUE OF EVEN : 10

AFTER JOINING T1 IS : false

AFTER JOINING T2 IS : false

public void join () Throws InterruptedException

This method is used for making the fore ground threads to join together so that JVM can call the garbage collector only one time for collecting all of them instead of collecting individually.

CRITICAL SECTION:

That piece of code that has to be executed by multiple threads is called critical section.

SYNCHRONIZATION:

It is the process of allowing only one thread at a time among ‘n’ number of threads to access the critical section.

Let us assume that there is a sharable variable called

There are two threads t1 and t2 respectively. t1 and t2 want to update the balance variable with

their respective values i.e., 10 and 20 at the same time. After completion of these two threads the

final value in the balance is either 10 or 20 but not 30 which is the inconsistent result. To achieve

the consistent result we must apply the concept of synchronization.

When we apply synchronization concept on the above scenario, when two threads are

started at same time, the first thread which is started is given a chance to update balance variable

with its respective value. When second thread is trying to access the balance variable value JVM

makes the second thread to wait until first thread completes its execution by locking balance

variable value. After completion of first thread the value in the balance is 10 and second thread will

be allowed to update balance variable value. After completion of second thread the value of the

balance is 30, which is the consistent result. Hence, in synchronization locking and unlocking is

taking place until all the threads are completed their execution.

Synchronization Techniques:

In JAVA we have two types of synchronization techniques. They are synchronized methods and synchronized blocks.

1. Synchronized methods:

If any method is sharable for ‘n’ number of threads then make the method as synchronized by using a keyword synchronized. In JAVA we have two types of synchronized methods. They are synchronized Instance methods and synchronized static methods.

Synchronized Instance methods: If the ordinary instance method is made it as synchronized then the object of the corresponding class will be locked.(Method level lock)

Syntax:

synchronized <return type> method name (method parameters if any)

{

Block of statements;

}

For example:

class Account

{

int bal=0;

synchronized void deposit (int amt)

{

bal=bal+amt;

System.out.println (“CURRENT BALANCE = ”+bal);

}

};

Synchronized static method: If an ordinary static method is made it as synchronized then the corresponding class will be locked.(class level lock)

Syntax:

synchronized static <return type> method name (method parameters if any)

{

Block of statements;

}

For example:

class Account

{

static int bal=0;

synchronized static void deposit (static int amt)

{

bal=bal+amt;

System.out.println (“CURRENT BALANCE = ”+bal);

}

};

2. Synchronized block:

This is an alternative technique for obtaining the concept of synchronization instead of synchronized methods.

When we inherit non-synchronized methods from either base class or interface into the derived class, we cannot make the inherited method as synchronized. Hence, we must use synchronized blocks.

Syntax:

synchronized (object of current class)

{

Block of statement(s);

}

For example:

class BankOp

{

void deposit (int amt);// p a instance

};

class Account implements BankOp

{

int bal=0;

public void deposit (int amt)

{

synchronized (this);

{

bal=bal+amt;

System.out.println (“current value=”+bal);

}

}

};

Write a synchronization program in which there exist an account, there exist ‘n’ number of customers and all the customers want to deposit 10 rupees to the existing balance of account.

Answer:

class Account

{

private int bal=0;

synchronized void deposit (int amt)

{

bal=bal+amt;

System.out.println ("CURRENT BALANCE="+bal);

}

int getBal ()

{

return (bal);

}

};

class cust extends Thread

{

Account ac;// has-a relationship

cust (Account ac)

{

this.ac=ac;

}

public void run ()

{

ac.deposit (10);

}

};

class SyncDemo

{

public static final int noc=5;

public static void main (String [] args)

{

Account ac=new Account ();

cust cu []=new cust [noc];

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

{

cu [i]=new cust (ac);// giving account object 'ac' to each & every customer object

}

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

{

cu [i].start ();

}

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

{

System.out.println ("IS ALIVE..? "+cu [i].isAlive ());

}

try

{

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

{

cu [i].join ();

}

}

catch (InterruptedException ie)

{

System.out.println (ie);

}

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

{

System.out.println ("IS ALIVE..? "+cu [i].isAlive ());

}

System.out.println ("TOTAL BALANCE="+ac.getBal ());

}

};

Output:

IS ALIVE..? true

IS ALIVE..? true

IS ALIVE..? true

IS ALIVE..? true

CURRENT BALANCE=10

CURRENT BALANCE=20

CURRENT BALANCE=30

CURRENT BALANCE=40

CURRENT BALANCE=50

IS ALIVE..? false

IS ALIVE..? false

IS ALIVE..? false

IS ALIVE..? false

IS ALIVE..? false

IS ALIVE..? false

TOTAL BALANCE=50

NOTE:

If we execute the above program on single user operating systems it is mandatory for the

JAVA programmer to write synchronized keyword since user operating system cannot take care about synchronization by default. Whereas if we run the above program in multi user or threaded operating systems we need not to use a keyword synchronized (optional).It is always recommended to write synchronized keyword irrespective which ever operating system we use.

Inter thread communication:

If two or more threads are exchanging the dat thread communication. In inter thread communication; an output of one thread is given as an input to another thread. In order to develop inter thread communication applications we must make use of a class called java.lang.Object.

Methods in java.lang.Object:

1. public void wait (long msec)

2. public void wait (long msec, int nanosec)

3. public void wait ()

4. public void notify ()

5. public void notifyAll ()

Methods 1 and 2 are used for making the thread to wait for some per waiting time is over automatically the thread will come from waiting state to ready state. Method 3 is used for making the thread to wait without specifying waiting time. Method 4 is used for bringing a single waiting thread into ready state. Method 5 is used for bringing all the threads from waiting state to ready state.

Develop producer consumer program by using inter thread communication?

Answer:

class Q

{

int n;

boolean valset;

synchronized void put (int i)

{

try

{

if (valset)

{

wait ();

}

}

catch (InterruptedException ie)

{

System.out.println (ie);

}

n=i;

System.out.println ("PUT="+i);

valset=true;

notify ();

}// put

synchronized int get ()

{

try

{

if (!valset)

{

wait ();

}

}

catch (InterruptedException ie)

{

System.out.println (ie);

}

System.out.println ("GET="+n);

valset=false;

notify ();

return (n);

}// get

};// Q

class Producer implements Runnable

{

Q q;

Thread t;

Producer (Q q)

{

this.q=q;

t=new Thread (this, "Producer");

t.start ();

}

public void run ()

{

int i=0;

System.out.println ("NAME OF THE THREAD = "+t.getName ());

while (true)

{

q.put (++i);

}

}

};// Producer

class Consumer implements Runnable

{

Q q;

Thread t;

Consumer (Q q)

{

this.q=q;

t=new Thread (this, "Consumer");

t.start ();

}

public void run ()

{

System.out.println ("NAME OF THE THREAD = "+t.getName ());

while (true)

{

int i=q.get ();

}

}

};// Consumer

class PCDemo

{

public static void main (String [] args)

{

Q q=new Q ();

Producer p=new Producer (q);

Consumer c=new Consumer (q);

}

};// PCDemo

***27)COLLECTION FRAMEWORK:***

Collection framework is the standardized mechanism of grouping of similar or dissimilar

objects into a single object.

Collection:

Collection is an object that is capable of storing similar or dissimilar objects.

So we can store only objects into a collection object.

For convering primitive types to Wrapper class objects,we use Wrapper classes.

DataType Wrapper class

int-------->Integer

float------>Float

char------->Character

String----->String

double----->Double

Goals of collection frameworks:

1. Collection framework improves the transfer the bulk amount of data framework we can transfer that entire data at a time).

2. Collection framework is dynamic in nature i.e., they are is fixed in nature and they allows similar type of data).

4. Collection framework contains

collection object at the end of another collection object).

5. Collection framework is algorithmic oriented

searching techniques of data structures as a predefined concepts).

6. In order to deal with collection framework we must import a package called Collection framework

To traverse along the elements of the collection we use the following interfeces:

---Iterator(forward only)

---ListIterator(Bidirectional)

---Enumerator

Collection framework is divided into two categories. They are legacy (old) collection framework and

new collection framework.

NEW COLLECTION FRAMEWORK:

New collection framework is again broadly divided into two categories. They are one and two

dimensional collection framework.

One dimensional collection framework:

A one dimensional collection framework is one in which the data is organized either in the

form of row or in the form of column by containing similar or dissimilar categories into

single object. This single object is known as one dimensional collection framework object.

As a part of one dimensional collection framework interfaces in JAVA we have five

interfaces. They are collection, list, set, sorted set,Queue.

java.util.Collection:

Collection is an interface whose object allows us to organize similar or different type of

objects into single object. The Collection interface is having the following features:

i) It is available at the top of the hierarchy of all the interfaces.

ii) An object of Collection allows us to add duplicate elements.

iii) Collection object always displays the data in forward direction only.

iv) Collection object will print the data on the console in random order.

v) Collection object always allows us to insert an element at the end only.

java.util.List:

i) List is the sub-interface of java.util.Collection

ii) List object also allows us to add duplicates.

iii) List object automatically displays the data in the sorted order.

iv) List object allows us to add an element either at the ending position or at specific position.

v) List object allows us to retrieve the data in forward direction, backward direction and random

retrieval.

java.util.Set:

i) Set is the sub-interface of java.util.Collection interface.

ii) An object of Set does not allows duplicates i.e., all the elements in the set must be distinct

(unique).

iii) Set object always displays the data in random order.

iv) Set object allows us to add the elements only at ending position.

v) Set object allows us to retrieve the data only in forward direction.

java.util.SortedSet:

i) SortedSet is the sub-interface of java.util.Set interface.

ii) SortedSet object does not allows duplicates.

iii) SortedSet object will displays the data automatically in sorted order.

iv) SortedSet object allows us to add the elements only at ending position.

v) SortedSet object allows us to retrieve only in forward direction.

java.util.Queue

Methods in Collection interface:

1. public int size (): This method is used for determining the number of elements in Collection

interface object.

2. public boolean add (java.lang.Object): This method is used for adding an object to Collection

object. When we use this method to add the object to Collection objects and List object, this

method always returns true. Since, Collection object and List object allows duplicates. When we

apply the same method with Set and SortedSet methods and when we are trying to add

duplicates, this method returns false.

3. public boolean addAll (Collection):

This method is used for adding the entire collection object at the end of another Collection

object. As long as we apply this method with List and Collection interfaces, it returns true. Since,

they allow duplicates. When we apply this method with Set and SortedSet, this method may return

false. Since, duplicates are not allowed.

4. public boolean isEmpty ():

Returns true when there are no object found in Collection interface object otherwise return

false.

5. public Object [] toArray ():

This method is used for extracting the data from Collection objects in the form of an array of

objects of java.lang.Object class.

For example:

int s=0;

Object obj []=s.toArray ();

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

{

Interger io= (Integer) obj [i];

int x=io.intValue ();

s=s+x;

}

System.out.println (“Sum = ”+s);

6. public Iterator iterator ():

This method is used for extracting the data from Collection framework object.

For example:

Iterator itr=co.iterator ();

int s=0;

While (itr.hasNext ())

{

Object obj=itr.next ();

Integer io= (Integer) obj;

int x=io.intValue ();

s=s+x;

}

Iterator interface: Iterator is an interface which always uses the extract the data from any Collection

object.

Methods in Iterator interface:

1. public boolean hasNext ()

2. public Object next ()

3. public Object remove ()

Method 1 is used for checking whether we have next element or not. If next element

available in Collection object it returns true otherwise it returns false. Method 2 is used for obtaining

the next element in the Collection object. Method 3 is used for removing the element from

Collection object. Methods 2 and 3 can be used as long as method 1 returns true.

List: List is an interface which extends Collection.

Methods in List interface:

1. public Object get (int): This method is used for obtaining that element from the specified

position. If the get method is not returning any value because of invalid position the value of

object of object is NULL.

2. public Object remove (int): This method is used for removing the objects from List object based

on position.

3. public Object remove (Object): This method is used for removing the objects from List object

based on content.

4. Public void add (int pos, Object): This method is used for adding an object at the specified

position.

5. public void addAll (int pos, Collection): This method is used for adding one Collection object to

another Collection object at the specified position.

6. public List headList (Object obj): This method is used for obtaining those objects Xi’s which are

less than or equal to target object (Xi=obj).

7. public List tailList (Object obj): This method is used for obtaining those objects Xi’s which are

greater than target object (Xi>obj or Xi=(obj-1)).

8. public List subList (Object obj1, Object obj2): This method is used for obtaining those values

which are a part of List i.e., range of values [or] In subList method is select those values Xi’s

which are less than or equal to object 1 and greater than object 2 (obj1=Xi<obj2).

9. public ListIterator listIterator (): This method is used for extracting the data from List object

either in forward or in backward or in both the directions. ListIterator is an interface which

extends Iterator interface. This interface contains the following methods:

public boolean hasPrevious (); Method-1

public Object previous (); Method-2

public void set (Object); Method-3

All the methods of iterator are also coming. Method-1 is used for checking weather we have

previous element or not, this method returns true as long as we have previous elements otherwise

false. Method-2 is used for obtaining previous element.Method 3 is used for modifying the existing

element.

java.util.LinkedList:

LinkedList is the concrete sub class of collection classes. LinkedList object allows us to group

similar or dissimilar type of objects. Creating a LinkedList is nothing but creating an object of

java.util.LinkedList class.

The data is organized in LinkedList in the form of nodes. The node contains two parts, they

are data part and data part.

Data part always gives the actual data which we want to represents the address of the next node.

For the last node in the LinkedList the address part must be NULL which indicates end of the

LinkedList.

LinkedList API:

Constructors:

LinkedList ();

LinkedList (int size);

Instance methods:

Object getFirst (); Method-1

Object getLast (); Method-2

public void addFirst (Object obj);Method-3

public void addLast (Object obj);Method-4

public void removeFirst ();Method-5

public void removeLast (); Method-6

Methods 1 and 2 are used for obtaining first and last objects of LinkedList respectively.

Methods 3 and 4 are used for adding similar or different objects to the LinkedList object respectively.

Methods 5 and 6 are used for removing first and last objects from LinkedList respectively.

Set: Set does not contain any special method except Collection interface methods.

SortedSet:

SortedSet extends Set. The following are the methods which are specially available in

SortedSet interface.

public Object first ();

public Object last ();

public SortedSet headSet (Object obj);

public SortedSet tailSet (Object obj1, Object obj2);

Write a java program which implements the concept of LinkedList?

Answer:

import java.util.\*;

class Linkedlist

{

public static void main (String [] args)

{

LinkedList ll=new LinkedList ();

System.out.println ("CONTENTS OF l1 = "+ll);

System.out.printl

ll.add (new Integer (10));

ll.add (new Integer (20));

ll.add (new Integer (30));

ll.add (new Integer (40));

System.out.println ("CONTENTS OF ll = "+ll);

System.out.println ("SIZE = "+ll.size ());

// retrieving data of ll using toArray ()

Object obj []=ll.toArray ();

int s=0;

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

{

Integer io= (Integer) obj [i];

int x=io.intValue ();

s=s+x;

}

System.out.println ("SUM USING toArray () = "+s);

ll.addFirst (new Integer (5));

ll.addFirst (new Integer (6));

System.out.println ("CONTENTS OF ll = "+ll);

System.out.println ("SIZE = "+ll.size ());

// retrieving data of ll using iterator ()

Iterator itr=ll.iterator ();

int s1=0;

while (itr.hasNext ())

{

Object obj1=itr.next ();

Integer io1= (Integer) obj1;

int x1=io1.intValue ();

s1=s1+x1;

}

System.out.println ("SUM USING iterator () = "+s1);

// retrieving data of ll using ListIterator ()

ListIterator litr=ll.listIterator ();

while (litr.hasNext ())

{

Object obj2=litr.next ();

System.out.print (obj2+",");

}

System.out.println ("\n");

while (litr.hasPrevious ())

{

Object obj3=litr.next ();

System.out.print (obj3+",");

}

System.out.println ("\n");

Object obj4=ll.get (2);// random retrieval

System.out.println (obj4);

}

};

Disadvantages of LinkedList:

1. Additional memory space is created for address part of the node in heap memory.

2. Retrieval time is more.

3. Since, we are wasting most of the memory space for addresses, performance will be reduced.

java.util.ArrayList:

ArrayList is also concrete sub-class of collection framework classes whose object allows us to

organize the data either in similar type or in different type.

Creating ArrayList is nothing but creating an object of ArrayList class.

ArrayList API:

ArrayList ();

ArrayList (int size);

Advantages of ArrayList over LinkedList:

1. No additional memory space is required for data of ArrayList.

2. Retrieval time is quite faster.

3. Performance is high. Since, there is no memory space is required for maintaining address of

data of ArrayList.

Write a JAVA program which illustrates the concept of TreeSet?

Answer:

import java.util.\*;

class tshs

{

public static void main (String [] args)

{

TreeSet ts=new TreeSet ();

System.out.println ("CONTENTS OF ts = "+ts);

System.out.println ("SIZE OF ts = "+ts.size ());

ts.add (new Integer (17));

ts.add (new Integer (188));

ts.add (new Integer (-17));

ts.add (new Integer (20));

ts.add (new Integer (200));

ts.add (new Integer (177));

System.out.println ("CONTENTS OF ts = "+ts);

System.out.println ("SIZE OF ts = "+ts.size ());

Iterator itr=ts.iterator ();

while (itr.hasNext ())

{

Object obj=itr.next ();

System.out.println (obj);

}

}

};

Write a JAVA program which illustrates the concept of HashSet?

Answer:

import java.util.\*;

class hsts

{

public static void main (String [] args)

{

HashSet hs=new HashSet ();

System.out.println ("CONTENTS OF hs = "+hs);

System.out.println ("SIZE OF hs = "+hs.size ());

hs.add (new Integer (17));

hs.add (new Integer (188));

hs.add (new Integer (-17));

hs.add (new Integer (20));

hs.add (new Integer (200));

hs.add (new Integer (177));

System.out.println ("CONTENTS OF hs = "+hs);

System.out.println ("SIZE OF hs = "+hs.size ());

Iterator itr=hs.iterator ();

while (itr.hasNext ())

{

Object obj=itr.next ();

System.out.println (obj);

}

}

};

Two dimensional framework or maps:

Two dimensional framework organize the data in the form of (key,value) pair. The value of

key is an object and they must be unique. The value of value is also an object which may or may not

be unique. Two dimensional framework contains collection of interfaces and collection of classes

which are also known as map interfaces and map classes.

Map interfaces:

In maps we have three essential interfaces; they are java.util.Map, java.util.Map.Entry and

java.util.SortedMap

java.util.Map:

java.util.Map extends Collection. An object of Map allows to organize the data in the form of

(key, value) pair. Here key and value must be objects. An object of Map allows displaying the data in

that order in which order we have added the data.

Methods:

public boolean put (Object kobj, Object vobj):

This method is used for adding the data in the form of (key, value). This method returns false

when we are trying to add duplicate key and values. This method returns true as long as we enter

unique key objects.

public boolean putAll (Map): This method is used for adding one Map object at the end of another

Map object.

public Set entrySet (): This method is used for obt

object.

public Object get (Object vobj): This method is used for obtaining value of value by passing value of

key object.

public void remove (Object kobj): This method is used for removing the entire map entry

the value of key object.

NOTE:

The following diagram gives an idea about how to organize the data in the form of (key,

value) and how to retrieve the data from Map object.

java.util.Map.Entry:

Here Map is an interface and Entry is a class in the Map interface.This is used for retrieving key

and value objects separately.

Methods:

public Object getKey (); 1

public Object getValue (); 2

Method 1 and 2 is used for obtaining key object and value object separately.

java.util.SortedSet:

SortedMap extends Map. An object of SortedMap displays the data by default in sorted

order.

Methods:

public Object first ();

public Object last ();

public SortedSet headMap (Object kobj); xi <= kobj

public SortedSet tailMap (Object kobj); xi > kobj

public SortedSet subMap (Object kobj, Object vobj); kobj1 <= xi < kobj2

Map classes:

Map classes contains all the definitions for the abstract methods of Map interface. In

java.util.\* package we have the following Map classes and whose hierarchy is given below:

1. AbstractMap implements Map

2. AbstractSortedMap extends AbstractMap implements SortedMap

3. HashMap extends AbstractMap

4. TreeMap extends AbstractSortedMap

Write a java program which illustrates the concept of HashMap?

Answer:

import java.util.\*;

class hmtm

{

public static void main (String [] args)

{

HashMap hm=new HashMap ();

System.out.println ("CONTENTS OF hm = "+hm);

System.out.println ("SIZE OF hm = "+hm.size ());

hm.put (new Integer (10), new Float(129.97f));

hm.put (new Integer (1), new Float(143.93f));

hm.put (new Integer (100), new Float(99.8f));

System.out.println ("CONTENTS OF hm = "+hm);

System.out.println ("SIZE OF hm = "+hm.size ());

Set s=hm.entrySet ();

Iterator itr=s.iterator ();

while (itr.hasNext ())

{

Map.Entry me= (Map.Entry) itr.next ();

Object kobj=me.getKey ();

Object vobj=me.getValue ();

System.out.println (vobj+"-->"+kobj);

}

}

};

Write a java program which illustrates the concept of TreeMap?

Answer:

import java.util.\*;

class tmhm

{

public static void main (String [] args)

{

TreeMap tm=new TreeMap ();

System.out.println ("CONTENTS OF tm = "+tm);

System.out.println ("SIZE OF tm = "+tm.size ());

tm.put (new Integer (10), new Float(129.97f));

tm.put (new Integer (1), new Float(143.93f));

tm.put (new Integer (100), new Float(99.8f));

System.out.println ("CONTENTS OF tm = "+tm);

System.out.println ("SIZE OF tm = "+tm.size ());

Set s=tm.entrySet ();

Iterator itr=s.iterator ();

while (itr.hasNext ())

{

Map.Entry me= (Map.Entry) itr.next ();

Object kobj=me.getKey ();

Object vobj=me.getValue ();

System.out.println (vobj+"-->"+kobj);

}

}

};

Program to remove the duplicates from a string and print:

import java.util.\*;

class duplicateremoval

{

public static void main(String a[])

{

String s = "Tomatto";

char[] chars = s.toCharArray();

Set<Character> charSet = new LinkedHashSet<Character>();

for (char c : chars)

{

charSet.add(c);

}

StringBuilder sb = new StringBuilder();

for (Character ch : charSet)

{

sb.append(ch);

}

System.out.println(sb.toString());

}

}

Program to count the number of characters in a string:

import java.util.\*;

class duplicate

{

public static void main(String a[])

{

String s="Harika";

Map<Character,Integer> map = new HashMap<Character,Integer>();

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

char c = s.charAt(i);

if (map.containsKey(c))

{

int cnt = map.get(c);

map.put(c, ++cnt);

}P

else

{

map.put(c, 1);

}

}

System.out.println(map);

}

}

output:

{r=1, a=2, H=1, k=1, i=1}

Old Collection framework/Legacy collection framework:

Interface:

We have only one interface, namely java.util.Enumeration. This interface is used for

extracting the data from legacy collection framework classes.

Classes:

As a part of legacy collection framework we have the following essential classes: Vector,

Stack, Dictionary, Hashtable and properties. Here, Vector and Stack belongs to one dimensional

classes whereas Dictionary, Hashtable and Properties belongs to two dimensional classes.

What is the difference between normal collection framework and legacy collection framework?

Answer: All the classes in the normal collection framework are by default belongs to non-

synchronized classes whereas all classes in legacy collection framework are by default belongs to

synchronized classes.

Example on Vector:

import java.util.\*;

class vector

{

public static void main (String [] args)

{

Vector v=new Vector ();

v.addElement (new Integer (10));

v.addElement (new Float (100.37f));

v.addElement (new Boolean (true));

v.addElement ("K.V.R");

System.out.println ("SIZE = "+v.size ());

System.out.println ("CONTENTS = "+v);

Enumeration en=v.elements();

while (en.hasMoreElements ())

{

Object val=en.nextElement ();

System.out.println (val);

}

}

};

Example for Stack:

Write a java program which illustrates the concept of Stack?

Answer:

import java.util.\*;

class stack

{

public static void main (String [] args)

{

Stack st=new Stack ();

System.out.println ("IS STACK EMPTY ? "+st.empty ());

System.out.println (st);

st.push (new Integer (10));

st.push (new Integer (20));

st.push (new Integer (30));

st.push (new Integer (40));

System.out.println (st);

System.out.println ("TOP MOST ELEMENT = "+st.peek ());

System.out.println (st);

System.out.println ("DELETED ELEMENT = "+st.pop ());

System.out.println ("MODIFIED STACK = "+st.peek ());

System.out.println ("IS 10 FOUND ? "+st.search (new Integer (10)));

Enumeration en=st.elements ();

while (en.hasMoreElements ())

{

Object obj=en.nextElement ();

System.out.println (obj);

}

}

};

Example for HashTable(2D):

Write a java program which illustrates the concept of Hashtable?

Answer:

import java.util.\*;

class hashtable

{

public static void main (String [] args)

{

Hashtable ht=new Hashtable ();

ht.put ("AP","Hyd");

ht.put ("Orissa","Bhuvaneshwar");

ht.put ("Karnatake","Bng");

ht.put ("TN","Chennai");

ht.put ("Bihar","Patna");

System.out.println (ht);

Enumeration en=ht.keys ();

while (en.hasMoreElements ())

{

Object k=en.nextElement ();

Object v=ht.get (k);

System.out.println (k+" "+v);

}

Example for Properties:

Write a java program which illustrates the concept of Properties class?

Answer:

import java.util.\*;

import java.io.\*;

class demo

{

public static void main (String [] args)

{

try

{

Properties p=new Properties ();

FileInputStream fis=new FileInputStream ("x.prop");

p.load (fis);

Object vobj1=p.get ("dno");

Object vobj2=p.get ("dname");

Object vobj3=p.get ("pwd");

System.out.println ("USER NAME : "+vobj2);

System.out.println ("DEPT NUMBER : "+vobj1);

System.out.println ("PASSWORD : "+vobj3);

}

catch (Exception e)

{

System.out.println (e);

}

}

};

Generics:

The Java Generics programming is introduced in J2SE 5 to deal with type-safe objects.

This is used for putting restriction over the (data type of objects )items that are to be stored in a

collection class.

Before generics, we can store any type of objects in collection i.e. non-generic. Now generics,

forces the java programmer to store specific type of objects.

Advantage of Java Generics

There are mainly 3 advantages of generics. They are as follows:

1) Type-safety : We can hold only a single type of objects in generics. It doesn’t allow to

store other objects.

2) Type casting is not required: There is no need to typecast the object.

Before Generics, we need to type cast.

List list = new ArrayList();

list.add("hello");

String s = (String) list.get(0);//typecasting

After Generics, we don't need to typecast the object.

List<String> list = new ArrayList<String>();

list.add("hello");

list.add(32);//Compile Time Error

ArrayList<String>

import java.util.\*;

class TestGenerics1{

public static void main(String args[]){

ArrayList<String> list=new ArrayList<String>();

list.add("rahul");

list.add("jai");

//list.add(32);//compile time error

String s=list.get(1);//type casting is not required

System.out.println("element is: "+s);

Iterator<String> itr=list.iterator();

while(itr.hasNext()){

System.out.println(itr.next());

}

}

}

Output:element is: jai

rahul

jai

Type Diamond/Diamond operator:

ArrayList<Integer> al=new ArrayList()<>;

Generics in Map:

import java.util.\*;

class TestGenerics2{

public static void main(String args[]){

Map<Integer,String> map=new HashMap<Integer,String>();

map.put(1,"vijay");

map.put(4,"umesh");

map.put(2,"ankit");

//Now use Map.Entry for Set and Iterator

Set<Map.Entry<Integer,String>> set=map.entrySet();

Iterator<Map.Entry<Integer,String>> itr=set.iterator();

while(itr.hasNext()){

Map.Entry e=itr.next();//no need to typecast

System.out.println(e.getKey()+" "+e.getValue());

}

}}

Output:1 vijay

2 ankit

4 umesh

Differences in Collections:

Set List

1.No duplicate 1.Allows duplicates

elements

2.This is an

unordered collection 2.This is an ordered collection

3.Allows only null 3.It accepts the insertion of

null values as many as we

want

--------------------------------------------------------------

List Map

1. Collection of elements 1. Collection of key-value pairs

--------------------------------------------------------------

HashMap HashTable

1. Not synchronized 1. Synchronized

2. Allows one key and 2.Does not allow null

keys or values

any number of null values

3. Not thread safe 3.Thread safe

4. Less secure 4.More secure

5. Fast 5.Slow

--------------------------------------------------------------

Vector ArrayList

1. Synchronized 1.NOt synchronized

2. Thread safe 2.Not thread safe

3. More secure 3.Less secure

4. Slow 4.Fast

5.It is a legacy class added 5.It is a part of jdk 1.2,

in the first release of jdk when the collection framework was introduced in java

6.Vector doubles the size of 6.ArrayList will be increased

its array when it is by half of its size,when

re-sized internally resized

--------------------------------------------------------------

Iterator Enumerator

1.Available in the new 1.Available in legacy

collection framework collection framework

2. hasNext(),next() 2.hasMoreElements(),nextElement()

--------------------------------------------------------------

Iterator ListIterator

1.Forward direction only 1.Bi-directional

hasNext(),next() hasPrevious(),previous()

hasNext(),next()

2.Elements cannot be added 2.Elements can be added

or removed in the middle or removed in the middle

--------------------------------------------------------------

HashMap HashSet

1.Collection of key value 1.It is an Unordered

pairs collection of unique elements

--------------------------------------------------------------

TreeSet Sorted Set

\* Sorted set is an interface that implements Tree Set

--------------------------------------------------------------

ArrayList LinkedList

1.Stores elements within a 1.Stores elements within a

dynamically resized array doubly linked list

2.Less memory overhead because, 2.More memory overhead

data (or) element or object each node should the data

is directly available at the along with the address of

index next Node

--------------------------------------------------------------

HashMap TreeMap

1.Returns unordered values 1.Returns elements in the

sorted order(ascending by

2.Allows one null key and 2.Does not allow to store

multiple null values even one null key but allows

multiple null values to be

stored

3. Time complexity is O(1) 3.Time complexity is O(log(n))

4. Performance is high 4.Performance is slow

--------------------------------------------------------------

Comparator Comparable

1.Uses compare(n) to sort 1.Uses compareTo() to sort

the elements the elements

2. java.util package 2. java.lang package

***28)IO STREAMS:***

Generally, in JAVA programming we write two types of applications or programs. They are

volatile or non-persistent program and non-volatile or persistent program.

A volatile program is one whose result is always stored in main memory of the computer i.e.,

RAM. Whatever the data which is stored in the main memory of the computer that data will be

temporary i.e., the data which is available in main memory is volatile.

A non-volatile program is one whose result is stored in secondary storage devices i.e., hard disk,

magnetic tapes, etc. the data which is stored in secondary storage devices is permanent. To

store the data we have two approaches. They are using files and using database.

If we store the data permanently in the form of a file, then the data of that file can be

modified by any unauthorized user. Hence, industry always recommends not storing the data in the

files. Since, files do not provide any security. Hence, industry always recommends storing the data in

the form of database. Since, most of the popular databases provide security in the form of user

names and passwords.

In order to store the data permanently in the form of files we must use or import a package

called java.io.\*

Collection of records is known as file. A record is a collection of field values.

A stream is a flow of data between primary memory and secondary memory either locally

(within the system) or globally (across the network) or [A stream is a flow of bytes between

primary memory and secondary memory either locally (within the system) or globally (across the

network)] or [A stream is a flow of bits between primary memory and secondary memory either

locally (within the system) or globally (across the network)

Types of operations or modes on files:

On files we perform two types of operations they are read and write.

Types of streams in JAVA:

Based on transferring the data between primary memory to secondary memory and

secondary memory to primary memory. In JAVA we have two types of streams they are byte streams

and char streams.

Byte streams are those in which the data will be transferred one byte at a time from primary memory

to secondary memory and secondary memory to primary memory either locally or globally.

java.io.\* package contains some set of classes and interfaces which will transfer one byte at

a time.

Hierarchy chart for byte streams:

Object

-InputStream

-BufferedInputStream

-FilteredInputStream

-FileInputStream

-DataInputStream

-OutputStream

-BufferedOutputStream

-FilteredOutputStream

-FileOutputStream

-DataOutputStream

InputStream class:

This is an abstract class; hence we cannot create an object of this class directly. This class is

basically used to open the file in read

1. public int read ();

2. public int length (); // total size of the file

3. public int available (); // available number of bytes only

4. public void close ();

In JAVA end of file (EOF) is indicated by -1.

OutputStream class:

This is also an abstract class; hence we cann

class is used for opening the file in write

1. public void write (int);

2. public int length ();

3. public void available ();

4. public void close ();

FileInputStream class:

This is the concrete (which we can create an object or it contains all defined methods) subclass

of all InputStream class. This class is always used to open the file in read mode. Opening the file

in read mode is nothing but creating an object of FileInputStream class.

Constructor:

FileInputStream (String fname) throws FileNotFoundException

For example:

FileInputStream fis=new FileInputStream (“abc.txt”);

If the file name abc.txt does not exist then an object of FileInputStream fis is null and hence

we get FileNotFoundException. If the file name abc.txt is existing then the file abc.txt will be

opened successfully in read mode and an object fis will point to beginning of the file.

FileOutputStream class:

This is the concrete sub-class of all OutputStream classes. This class is always used for

opening the file in write mode is nothing but creating an object of FileOutputStream class.

Constructors:

1. FileOutputStream (String fname);

2. FileOutputStream (String fname, boolean flag);

If the flag is true the data will be appended to the existing file else if flag is false the data will

be overlapped with the existing file data.

If the file is opened in write mode then the object of FileOutputStream will point to that file

which is opened in write mode. If the file is unable to open in write mode an object of

FileOutputStream contains null.

Write the following JAVA programs: 1) create a file that should contain 1 to 10 numbers. 2) read the

data from the file which is created above.

Answer:

1)

import java.io.\*;

class Fos

{

public static void main (String [] args)

{

try

{

String fname=args [0];

FileOutputStream fos=new FileOutputStream (fname, true);// mean append mode

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

{

fos.write (i);

}

fos.close ();

}

catch (IOException ioe)

{

System.out.println (ioe);

catch (ArrayIndexOutOfBoundsException aiobe)

{

System.out.println ("PLEASE PASS THE FILE NAME..!");

}

System.out.println ("DATA WRITTEN..!");

}

};

2)

import java.io.\*;

class Fis

{

public static void main(String[] args)

{

try

{

String fname=args [0];

FileInputStream fis=new FileInputStream (fname);

int i;

while ((i=fis.read ())!=-1)

{

System.out.println (i);

}

fis.close ();

}

catch (FileNotFoundException fnfe)

{

System.out.println ("FILE DOES NOT EXIST..!");

}

catch (IOException ioe)

{

System.out.println (ioe);

}

catch (ArrayIndexOutOfBoundsException aiobe)

{

System.out.println ("PLEASE PASS THE FILE NAME..!");

}

}

};

DataInputStream class:

This is used for two purposes. They are reading the data from input device like keyboard

and reading the data from remote machine like server.

In order to perform the above operations we must create an object of DataInputStream

class.

Constructors:

DataInputStream (InputStream);

For example:

DataInputStream dis=new DataInputStream (System.in);

An object of InputStream called ‘in’ is created as a static data member in System class.

Instance methods:

1. public byte readByte ();

2. public char readChar ();

3. public short readShort ();

4. public int readInt ();

5. public long readLong ();

6. public float readFloat ();

7. public double readDouble ();

8. public boolean readBoolean ();

9. public String readLine ();

DataOutputStream class:

This is used for displaying the data onto the console and also used for writing the data to

remote machine.

Constructor:

DataOutputStream (OutputStream);

For example:

DataOutputStream dos=new DataOutputStream (System.out);

Instance methods:

Old methods:

1. public void writeByte (byte);

2. public void writeChar (char);

3. public void writeShort (short);

4. public void writeInt (int);

5. public void writeLong (long);

6. public void writeFloat (float);

7. public void writeDouble (double);

8. public void writeBoolean (boolean);

9. public void writeLine (String);

New methods:

1. public void write (byte);

2. public void write (char);

3. public void write (short);

4. public void write (int);

5. public void write (long);

6. public void write (float);

7. public void write (double);

8. public void write (boolean);

9. public void write (String);

Write a JAVA program to read two numbers from keyboard and display their product (using older

methods)?

Answer:

import java.io.\*;

class DataRead

{

public static void main (String [] args)

{

try

{

DataInputStream dis=new DataInputStream (System.in);

System.out.println ("Enter first number : ");

String s1=dis.readLine ();

System.out.println ("Enter second number : ");

String s2=dis.readLine ();

int n1=Integer.parseInt (s1);

int n2=Integer.parseInt (s2);

int n3=n1\*n2;

System.out.println ("Product = "+n3);

}

catch (FileNotFoundException fnfe)

{

System.out.println ("FILE DOES NOT EXISTS");

}

catch (IOException ioe)

{

System.out.println (ioe);

}

catch (NumberFormatException nfe)

{

System.out.println ("PASS INTEGER VALUES ONLY");

}

}

};

Serialization:

Converting an object into sequence of bytes and storing them into a file or making the file to travel

along a network.That means,it is the process of saving the state of the object permanently in the

form of a file.

Deserialization:

Converting a sequence of bytes into an object.

Example on serialization and Deserialization:

Serialization:

Filename:trans.java

import java.io.Serializable;

public class trans implements Serializable

{

int id;

String name;

int age;//Now it will not be serialized

public trans(int id, String name,int age)

{

this.id = id; //211

this.name = name; //ravi

this.age=age;//22

}

}

Filename:tkey.java

import java.io.\*;

class tkey

{

public static void main(String args[])throws Exception{

trans s1 =new trans(211,"ravi",22);//creating object

//writing object into file

FileOutputStream f=new FileOutputStream("f.txt");

ObjectOutputStream out=new ObjectOutputStream(f);

out.writeObject(s1);

out.flush();

out.close();

f.close();

System.out.println("success");

}

}

Deserialization:

Filename:tds.java

import java.io.\*;

class tds{

public static void main(String args[])throws Exception{

FileInputStream fis=new FileInputStream("f.txt");

ObjectInputStream in=new ObjectInputStream(fis);

trans s=(trans)in.readObject();

System.out.println(s.id+" "+s.name+" "+s.age);

in.close();

}

}

output:

211 ravi 22

Example 2:

Write a JAVA program for student serializable sub-class?

Answer:

package sp;

import java.io.\*;

public class Student implements Serializable

{

int stno;

String sname;

float marks;

public void setSno (int stno)

{

this.stno=stno;

}

public void setSno (String sname)

{

this.sname=sname;

}

public void setSno (float marks)

{

this.marks=marks;

}

public int getStno ()

{

return (stno);

}

public String getSname ()

{

return (sname);

}

public float getMarks ()

{

return (marks);

}

};

SERIALIZATION PROCESS:

1. Create an object of serializable sub-class.

For example:

Student so=new Student ();

2. Accept the data either from keyboard or from command prompt.

3. Call set of set methods to set the values for the serializable sub-class (Student) object.

For example:

so.setSno (sno);

4. Choose the file name and open it into write mode with the help of FileOutputStream class.

5. Since an object of FileOutputStream class cannot write the entire object at a line to the file. In

order to write the entire object at a time to the file we must create an object of

ObjectOutputStream class and it contains the following Constructor:

ObjectOutputStream (FileOutputStream);

For example:

ObjectOutputStream oos=new ObjectOutputStream (fos);

The object oos is pointing to object fos, hence such type of streams are called chained or

sequenced stream.

6. ObjectOutputStream class contains the following instance method, which will write the entire

object at a time to the file.

For example:

ObjectOutpurStream.writeObject (so);

Write a JAVA program to save or serialize student data?

Answer:

import java.io.\*;

class StudentData

{

public static void main (String [] args)

{

try

{

sp.Student so=new sp.Student ();

DataInputStream dis=new DataInputStream (System.in);

System.out.println ("ENTER STUDENT NUMBER : ");

int stno=Integer.parseInt (dis.readLine ());

System.out.println ("ENTER STUDENT NAME : ");

String sname=dis.readLine ();

System.out.println ("ENTER STUDENT MARKS : ");

float marks=Float.parseFloat (dis.readLine ());

so.setStno (stno);

so.setSname (sname);

so.setMarks (marks);

System.out.println ("ENTER THE FILE NAME TO WRITE THE DATA");

String fname=dis.readLine ();

FileOutputStream fos=new FileOutputStream (fname);

ObjectOutputStream oos=new ObjectOutputStream (fos);

oos.writeObject (so);

System.out.println ("STUDENT DATA IS SERIALIZED");

fos.close ();

oos.close ();

}

catch (IOException ioe)

{

System.out.println ("PROBLEM IN CREATING THE FILE");

}

catch (Exception e)

{

System.out.println (e);

}

}

};

Write a JAVA program to retrieve or de-serialize student data?

Answer:

Steps for developing deserialization process:

1. Create an object of that class which was serialized.

For example:

Student so=new Student ();

2. Choose the file name and open it into read mode with the help of FileInputStream class.

For example:

FileInputStream fis=new FileInputStream (“Student”);

3. Create an object of ObjectInputStream class. The Constructor of ObjectInputStream class is

taking an object of FileInputStream class.

For example:

ObjectInputStream ois=new ObjectInputStream (fis);

4. ObjectInputStream class contains the following method which will read the entire object or

record.

public Object readObject ();

For example:

Object obj=ois.readObject ();

5. Typecast an object of java.lang.Object class into appropriate Serializable sub-class object for

calling get methods which are specially defined in Serializable sub-class.

For example:

So= (Student) obj;

6. Apply set of get methods for printing the data of Serializable sub-class object.

For example:

int stno=so.getStno ();

String sname=so.getSname ();

flaot marks=so.getMarks ();

7. Close the chained stream.

For example:

fis.close ();

ois.close ();

import java.io.\*;

import sp.Student;

class DeSerial

{

public static void main (String [] args)

{

try

{

Student so=new Student ();

DataInputStream dis=new DataInputStream (System.in);

System.out.println ("ENTER FILE NAME TO READ");

String fname=dis.readLine ();

FileInputStream fis=new FileInputStream (fname);

ObjectInputStream ois=new ObjectInputStream (fis);

Object obj=dis.readObject ();

so=(Object) obj;

System.out.println ("STUDENT NUMBER "+so.getStno ());

System.out.println ("STUDENT NAME "+so.getSname ());

System.out.println ("STUDENT MARKS "+so.getMarks ());

fis.close ();

ois.close ();

}

catch (FileNotFoundException fnfe)

{

System.out.println ("FILE DOES NOT EXISTS");

}

catch (Exception e)

{

System.out.println (e);

}

}

};

Types of serialization:

In java we have four types of serialization; they are complete serialization, selective

serialization, manual serialization and automatic serialization.

Complete serialization is one in which all the data members of the class participates in

serialization process.

Selective serialization is one in which few data members of the class or selected members of the

class are participating in serialization process.

In order to avoid the variable from the serialization process, make that variable

declaration as transient i.e., transient variables never participate in serialization process.

Manual serialization is one in which the user defined classes always implements

java.io.Serializable interface.

Automatic serialization is one in which object of sub-class of Serializable sub-class participates in

serialization process.

class RegStudent extends Student

{

………………..

………………..

};

Buffered Streams:

Buffered streams are basically used to reduce physical number of read and write operation

i.e., buffered streams always increases the performance of ordinary streams.

In byte streams we have two buffered streams, they are BufferedInputStream and

BufferedOutputStream

BufferedInputStream:

BufferedInputStream class is used for reducing number of physical read operation. When we

create an object of BufferedInputStream, we get a temporary peace of memory space whose default

size is 1024 bytes and it can be increased by multiples of 2.

Constructor:

public BufferedInputStream (FileInputStream);

For example:

FileInputStream fis=new FileInputStream (“abc.dat”);

BufferedInputStream bis=new BufferedInputStream (fis);

BufferedOutputStream:

BufferedOutputStream class is used for reducing number of physical write operation when

we create an object of BufferedOutputStream, we get a temporary peace of memory space whose

default size is 1024 bytes and it can be increased by multiple of 2.

Constructor:

public BufferedOutputStream (FileOutputStream);

For example:

FileOutputStream fos=new FileOutputStream (“abc.dat”);

BufferedOutputStream bos=new BufferedOutputStream (fos);

Byte streams will transfer one byte of data and they will be implemented in system level

applications such as flow of data in electronic circuits, development of ftp protocol etc.

Character streams are those in which 2 bytes of data will be transferred and it can be

implemented in higher level applications such as internet applications, development of protocols

etc., like http etc.

transient keyword:

If we want to prevent any data member(variable) from being serialized,then

we will preceed that data member using "transient" keyword.

Example:

Filename:trans.java

import java.io.Serializable;

public class trans implements Serializable

{

int id;

transient String name;//Name cannot not be serialized

int age;//Now it will not be serialized

public trans(int id, String name,int age)

{

this.id = id; //211

this.name = name; //ravi

this.age=age;//22

}

}

Filename:tkey.java

import java.io.\*;

class tkey

{

public static void main(String args[])throws Exception{

trans s1 =new trans(211,"ravi",22);//creating object

//writing object into file

FileOutputStream f=new FileOutputStream("f.txt");

ObjectOutputStream out=new ObjectOutputStream(f);

out.writeObject(s1);

out.flush();

out.close();

f.close();

System.out.println("success");

}

}

Filename:tds.java

import java.io.\*;

class tds{

public static void main(String args[])throws Exception{

FileInputStream fis=new FileInputStream("f.txt");

ObjectInputStream in=new ObjectInputStream(fis);

trans s=(trans)in.readObject();

System.out.println(s.id+" "+s.name+" "+s.age);

in.close();

}

}

output:

211 null 22

Name is not serialized

Marker interface:

An interface without body is called marker interface.

Example is Serializable interface

Serializable interface is present in java.io package. It is used to make an

object eligible for saving its state into a file. This is called Serialization.

Classes that do not implement this interface will not have any of their state serialized or

deserialized.

This is also used to indicate the JVM that the class thatimplements the marker interface is having a

special purpose i.e., the members(variables) of the class will involve in the concept of serialization.

interface A

{

}

Example:

Filename:Payment.java

interface Cheque

{

}

interface BankDraft

{

}

public class Payment implements BankDraft

{

public void paymentByCheck

{

System.out.println("Payment By Cheque");

}

public void paymentByDraft

{

System.out.println("Payment By Draft");

}

}

Filename:MainClass.java

public MainClass

{

public static void main(String a[])

{

Payment p=new Payment();

if(p instanceof Cheque)

{

p.paymentByCheque();

}

if(p instanceof BankDraft)

{

p.paymentByBankDraft();

}

}

}

Console class:

import java.io.Console;

public class CC {

public static void main(String ma[]){

Console c=System.console();

String name,pwd;

name=c.readLine("%s","user name :");

pwd=c.readLine("%s","password :");

/\*

To make password invisible

pwd=new String(c.readPassword("%s","password :"));

\*/

if(name.equals("123") && pwd.equals("123")){

//System.out.println("hai "+name);

c.printf("Hai %s",name);

}

else System.out.println("Sorry! Wrong UserName or Password");

}

}

Reading a String file:

import java.io.\*;

class FileReading{

public static void main(String[] args)throws IOException {

FileReader fis=new FileReader("read.txt");

BufferedReader br=new BufferedReader(fis);

String str=br.readLine();

try{

while(!str.equals(null)){

System.out.println(str);

str=br.readLine();

}

} catch (FileNotFoundException f) {

System.out.println ("File not found: " + f);

} catch (IOException e) {

System.out.println ("IOException: " + e);

}

finally{

fis.close();

}

}

}

***29)Strings:***

Sequence of characters is called as a String.

Strings are stored in a special memory called as "String constant pool".

In java,String is both a class and a datatype.

//We are creating a String literal(if we use new keyword,then we are creating new

String object).

String s1="Hello";//Hello is a literal

String s2="Hello";//Hello is a literal

Each time you create a string literal, the JVM checks the string constant pool first.

If the string already exists in the pool, a reference to the pooled instance is

returned. If string doesn't exist in the pool, a new string instance is created and

placed in the pool. For example:

String s1="Welcome";

String s2="Welcome";//will not create new instance

//The String object(s1) is created in pool and the literal "Hello" is also placed in pool.

stack String Constant pool

s1------>Hello

/

s2 /

Both s1 and s2 points to the same Content

So memory was efficiently used.

Using "new" keyword,we are creating string objects.

String s1=new String("Hello");

String s2=new String("Hello");

//The String object(s1) is created in Heap and the literal "Hello" is placed in pool.

In such case, JVM will create a new string object in normal(non pool) heap memory

and the literal "Welcome" will be placed in the string constant pool. The variable s

will refer to the object in heap(non pool).

============ "==" and .equals()=========

String a = "Java";

String b = "Java";

System.out.println(a == b); // True

As both refers to the same content,we will get true in this case

String c = new String("Java");

String d = new String("Java");

System.out.println(c == d); // False

Here the references/addresses are checked because here there are separate objects(c,d)

and separate references.So we will get false.

Similarly when you compare a String literal with an String object created using new()

operator using == operator, it will return false, as shown below :

String e = "JDK";

String f = new String("JDK");

System.out.println(e == f); // False

Example to show the working of methods in String:

class strings

{

public static void main(String[] args) {

String s1="software solutions";

String s2="34";

String s3="SOFTWARE SOLUTIONS";

String s4="mani@gmail.com";

String s5="baby";

String s6="This,is,Pinky";

String s7="Pinky";

System.out.println(s1.length());//18

System.out.println(s1.charAt(0));//s

System.out.println(s1);//software solutions

System.out.println(s1.concat(s2));//software solutions34

System.out.println(s1+s2);//software solutions34

System.out.println(s1.equals(s3));//false

System.out.println(s1.equalsIgnoreCase(s3));//true

System.out.println(s4.trim());//mani@gmail.com

System.out.println(s1.substring(7));//e solutions(From 0)

System.out.println(s1.substring(0,7));//softwar(0to 7-1)

System.out.println(s5.replace('y','i'));//babi

System.out.println(s5);//baby

System.out.println(s3.toLowerCase());//software solutions

System.out.println(s1.toUpperCase());//SOFTWARE SOLUTIONS

System.out.println(s1.indexOf('t'));//3(From 0)

System.out.println(s1.lastIndexOf('t'));//13(From 0)

String s66[]=s6.split(",");

for(String r:s66)

System.out.println(r);

/\*

This

is

Pinky

\*/

String s7="Pinky";

char s77[]=s7.toCharArray();

for(char r:s77)

System.out.println(r);

/\*

P

i

n

k

y

\*/

}

}

Example 2:

public class StringExample{

public static void main(String args[]){

String s1="java";//creating string by java string literal

char ch[]={'s','t','r','i','n','g','s'};

String s2=new String(ch);//converting char array to string

String s3=new String("example");//creating java string by new keyword

System.out.println(s1);

System.out.println(s2);

System.out.println(s3);

}

}

Output:

java

strings

example

String Immutability:

Strings are immutable(Unchangable).We cannot perform

modifications on strings.

Hence StringBuffer and StringBuilder classes are used to

make strings mutable.

class Demo

{

public static void main(String args[]){

String s="Sachin";

s.concat(" Tendulkar");//concat() method appends the string at the end

System.out.println(s);//will print Sachin because strings are immutable objects

}

}

Test it Now

Output:Sachin

Now it can be understood by the diagram given below. Here Sachin is not changed but a

new object is created with sachintendulkar. That is why string is known as immutable.

s(stack)

----Sachin

==========>String constant pool

----Sachin Tendulkar

Fig:Heap diagram

As you can see in the above figure that two objects are created but s reference

variable still refers to "Sachin" not to "Sachin Tendulkar".

But if we explicitely assign it to the reference variable, it will refer to "Sachin Tendulkar" object.

For example:

class Testimmutablestring1{

public static void main(String args[]){

String s="Sachin";

s=s.concat(" Tendulkar");

System.out.println(s);

}

}

Output:

Sachin Tendulkar

In such case, s points to the "Sachin Tendulkar". Please notice that still sachin object

is not modified.

Why string objects are immutable in java?

Because java uses the concept of string literal.Suppose there are 5 reference variables,

all referes to one object "sachin".If one reference variable changes the value of the

object, it will be affected to all the reference variables. That is why string objects

are immutable in java.

Another example:

class stingmodi

{

public static void main(String[] args) {

String s1="Hello";

StringBuffer sb=new StringBuffer(s1);

sb.reverse();//olleH

String s=sb.toString();

System.out.println(s);//olleH

}

}

Differences between StringBuffer and StringBuilder classes.

StringBuffer StringBuilder

1)Synchronous 1)Asynchronous

2)More security 2)Less security

3)Thread safe 3)Not thread safe

4)Slow 4)Fast

StringTokenizer class:

import java.util.StringTokenizer;

public class StringTokenizerExample

{

public static void main(String[] args)

{

String shirts = "Blue Shirt, Red Shirt, Bla#ck Shirt, Mar#oon Shirt";

StringTokenizer st = new StringTokenizer(shirts, ", #");

while(st.hasMoreTokens())

{

System.out.println(st.nextToken());

}

}

}

The output of program will be:

Blue

Shirt

Red

Shirt

Bla

ck

Shirt

Mar

oon

Shirt

package com.EncryptDecrypt;

public class Modulo26Crypto {

public static void main(String[] args) {

String plainText = "DROPIT";

String secretKey = "SECRETKEY";

System.out.println("Plain Text Before Encryption: " + plainText);

String encryptedText = encrypt(plainText, secretKey);

System.out.println("Encrypted Text After Encryption: " + encryptedText);

String decryptedText = decrypt(encryptedText, secretKey);

System.out.println("Decrypted Text After Decryption: " + decryptedText);

}

private static String encrypt(String plainText, String secretKey) {

StringBuffer encryptedString = new StringBuffer();

int encryptedInt;

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

int plainTextInt = (int) (plainText.charAt(i) - 'A');

int secretKeyInt = (int) (secretKey.charAt(i) - 'A');

encryptedInt = (plainTextInt + secretKeyInt) % 26;

encryptedString.append((char) ((encryptedInt) + (int) 'A'));

}

return encryptedString.toString();

}

private static String decrypt(String decryptedText, String secretKey) {

StringBuffer decryptedString = new StringBuffer();

int decryptedInt;

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

int decryptedTextInt = (int) (decryptedText.charAt(i) - 'A');

int secretKeyInt = (int) (secretKey.charAt(i) - 'A');

decryptedInt = decryptedTextInt - secretKeyInt;

if (decryptedInt < 1)

decryptedInt += 26;

decryptedString.append((char) ((decryptedInt) + (int) 'A'));

}

return decryptedString.toString();

}

}

Output:

Plain Text Before Encryption: DROPIT

Encrypted Text After Encryption: VVQGMM

Decrypted Text After Decryption: DROPIT

***Tasks:***

***Encapsulation:***

Create a class like student....Take four properties and some behaviours inside the student class.

Two properties and one behaviour must be not be accessed by the outside world.

***Inheritence:***

Write a program that includes both single and hierarchical inheritence.

Like:

A is Parent.B is child of A.C and D are children of B.

***Polymorphysm:***

Write a program that involves both the concepts like overloading and overriding.(Reflect both the

concepts in the same program).

***Abstraction:***

Create a class like "Employee".Create sub classes like FullTime and PartTime under it.Under PartTime

Create two subclasses like HourlyBased and Contract based.The parent classes "Employee" and "PartTime"

should have an abstract method like "salary()".The body for these abstract methods should be provided

by the child classes(FullTime,Hourly based,Contract based).

***Constructors:***

Create a class that have overloaded constructors that is going calculate the area of circle with

different parameters.Every constructor should involve "this" keyword.

***Packages:***

Create three packages.Reuse the classes of one package into another package.

***Collections:***

Create a stack and perform different operations like push(),pop(),peek(),search(),etc.

***Threads:***

Create three threads.Two by extending "Thread" class and one that implements Runnable interface.

First thread should print odd nos. from 0 to 10.

Second thread should print even nos. from 0 to 10.

Third thread should print Hello.

All the three threads should run simultaneously and should do their respective tasks by maintaining

3 seconds pause.

***Annotations:***

Write a program that includes the following annotations...

@Override

@SuppressWarnings

@Deprecated

***Frequently asked interview questions:***

***Introduction to Java***

1)What is a class?

2)What is an object?

3)What is the difference between a class and an object in terms of memory?

Ans:A class is a logical entity but an object is a physical entity because an object

occupies space in the memory.

4)JVM is platform dependent or independent?

Ans:Platform depedent.

5)Java is platform dependent or independent language?

Ans:Platform indepedent.

6)What are some features of java?

7)No.of dot class files created per each class-one.

8)Default value for int-0,char-space,string-null,boolean-false.

9)By default,a class is having which scope?

Ans:default scope

10)No. of private classes in a program-0

11)No. of public classes in a program-1

12)No. of protected classes in a program-0

13)No. of default classes in a program-any no.

14)Java API consists of:

Ans:JDK+JSL

15)Process of java program execution?

Ans: .java to .class and .class to .exe file.

16)What is Typecasting?Types?

Ans:promoting one datatype to other.Types are widening and narrowing

17)Give some examples of java utilities?

Ans:javac,java,javah,javap,etc.

18)What do you mean by command line arguments?

Ans:arguments or parameters given in the command of program execution(java classname 34 45 67)

19)Why java is more oop than C++?

Ans:Because,in java main is also written inside the class.

20)Differences between C and Java?

21)Differences between C++ and java?

22)why java is slower than C?

Ans:Because security checks are more in java.

23)No.of Bytes-

Ans:

short-2

byte-1

char-2

float-4

int-4

long-8

double-8

24)What are the regularly used class to accept the input from the user?

Ans:Scanner,BufferedReader,DataInputStream,etc

25)What is the super class of all the classes?

Ans:object

26)What is the default package in java?

Ans:java.lang

27)What are the to create an object?

Ans:using new keyword,using deserialization,using object cloning,using reflection

***Pillars of Java***

28)What is Encapsulation?

29)Best example of encapsulation?

Ans:class

30)What is polymorphysm?

31)What is inheritence?

32)What is Abstraction?

***Variables***

33)What is the difference between an instance variable,a static variable,a local variable.

Ans:Only one copy of static variables is avaible to all the objects.For each object,separate

copy of static variables is available.If a variable is recognized only in that block in

which it is defined,then it is called as a local variable.

***final Keyword***

34)A final variable can never be?

Ans:changed

35)A final method can never be?

Ans:Overriden

36)A final class can never be?

Ans:inherited

***Constructors***

37)What is the purpose of a consrtuctor?

Ans:To initialize an object

38)The name of the constuctor is equal to?

Ans:classname

39)What are the differences between a method and a constructor?

Ans:Constructor-no return type,classname=constructor name

40)What are the types of constructor?

Ans:Parameterless and Parameterized.

41)If you dont write a constructor in your program,what JVM will do?

Ans:It will create a default constructor and uses it.

42)When a constructor be called?

Ans:AS soon as the object is created.

***Destructors***

43)What are Destructors in java?

Ans:Used to deallocate the resources of an object.

44)What is the method used which acts like destructors in java?

Ans:finalize()

45)When a destructor be executed on an object?

Ans:When there are no references for an object.

46)When a destructor be executed on an object?

Ans:Just before the object is garbage collected,destructors are executed.

47)Why destructors are not there in java?

Ans:Because the Garbage collection is done by JVM itself.

***Garbage collection***

48)What is Garbage collection?

49)Deallocating the memory of an object.

50)Performance of a garbage collector will depend on?

Ans:JVM performance.

51)What is garbage collection pause?

Ans:When garbage collection thread begins its execution,it stops all the apllication threads.After

performing garbage collection,again it starts all the application threads.This delay is called as

Garbage Collection Pause.

52)Types of garbage collectors

Ans:Serial,parallel and concurrent

53)Example of a serial and parallel gc?

Serial-Hotspot,parallel-JRocket.

54)To start gc process explicitly-

Ans:System.gc()

Dynamic method dispatch

55)What is Dynamic method dispatch?

Ans:Assigning a subclass object to a super class reference.

56)What did the term Dynamic in java?

Ans:At the time of execution.

this and super keywords

57)What is the purpose of this and super keywords?

58)What are the restrictions on this and super keywords---

Ans:super or this,if used, they should be the first statements in the constructor.

They cannot be used in methods only in constructors.

***static***

59)What is the priority of the static block compared with main?

Ans:static blocks will be executed before main.

60)Limitation on static members.

Ans:A static member can access only other static members

Also they cannot use this and super

61)From where the memory is allocated for the static variables?

Ans:From the static-pool.

62)What is the advantage of declaring a method as static?

Ans:We can directly access the members using the class name.

***Abstract class***

63)What is an abstract class?

64)What is an abstract method?

65)Can we create an object for an abstract class?

Ans:No

66)Can an abstract class contain abstract and non abstract methods?

67)What are the differences between an abstract classes and interfaces?

***Interfaces***

68)What is an interface?

69)Can an interface extend other interface?

Ans:Yes

70)Can an interface implement other interface?

Ans:No

71)Can you create an object directly for an interface?

Ans:No,only references can be created.

72)What is the access specifier for the variables in an interface?

Ans:static and final

73)What is the access specifier for the methods in an interface?

Ans:public and abstract

***Polymorphysm***

74)What the types of polymorphysm?

Ans:Method Overloading and Method Overriding.

75)Differences between overloading and overriding

***Inheritence***

76)What are the types of inheritences in java?

77)Does java directly supports multiple inheritence?

78)Java supports multiple inheritence through?

Ans:Interfaces

***Exception handling***

79)What is an exception?

80)What are the five keywords involved in exception handling?

81)If you want some set of statements to be executed for sure,you have to place them in?

Ans:finally block

82)What are the types of exceptions?

Ans:Checked and Unchecked.

83)What is the super class of all exceptions

Ans:Throwable

84)You can through your own exception using?

Ans:throw keyword

***IO streams***

85)What is serialization?

86)What is deserialization?

87)What is a marker interface?

88)What is the purpose of a transient keyword in java?

Multi threading

89)What is a thread?

90)If you put t.run() instead of t.start(),what will hppen?

Ans:Context-swithing will not be performed.

91)If you start a thread for two times,what will happen?

Ans:IllegalThreadStateException occurs.

92)We can implement multithreading using?

Ans:Thread class and Runnable interace

93)What is the lifecycle of a tread?

Ans:Newly born,runnable,running,dead.

94)What is the difference between sleep() and wait()?

Ans:In sleep(),we can specify the time upto which the thread should be inactive.But in wait()

time is not specified.So,we dont know how much time the thread is inactive.

95)Sleep() causes which exception?

Ans:Interrupted Exception

96)What are the methods present in Runnable interface?

Ans : Only run()

97)what is the normal priority of a thread?

Ans : 5

98)What is the method to determine whether a thread is alive or not ?

Ans: t.isAlive()

99)What is an orphan thread?

Ans : A thread that doesn't have parent

100)What is the use of join() ?

101)What is the deamon thread ?

102)What is the method used to make a thread as deamon ?

Ans : t.setDeamon()

103)What is synchronization ?

104)What is critical section ?

Ans:That piece of code which is required to be executed by more than one thread.

105)What is starvation ?

Ans:A low priority thread is waiting for longer time to acquire the lock on the critical section.

106)What is the difference between a synchronized block and synchronized method ?

Ans : If we want to make whole method to be synchronized,then synchronized keyword is given for a

method.If we want only some statements of a method to be synchronized,then we use synchronized block.

107)What are the types of locks?

Ans : Class level , Object level

108)If we want to acquire lock on whole class then for which method we have to give synchronized

keyword?

Ans:static method.

109)What is inter-thread communication?

110)What are the methods used for interthread communication ?

Ans : wait(), notify(),notifyAll()

111)In which class the methods for performing interthread communication present?

Ans : Object

***Strings***

112)What is a string ?

113)What is the use of StringTokenizer?

Ans:It is like scanf statement in C language.Used to break up a string into tokens.

114)Can you perform modifications on strings?

115)What are the uses of StringBuilder and StringBuffer class ?

116)What is the difference between StringBuilder and StringBuffer classes ?

117)What is the purpose of split() method?

118)Why to compare the strings using .equals()

***Collections***

117)What is a collection framework?

118)Waht are the types of iterators?Ans:iterator,Listiterator,enumeration

119)What are the differences between enumeration and iterator?

120)What are the differences between Listiterator and iterator?

121)Differences between all the classes in collection framework?

122)How to retrieve an element randomly from a collection?

123)How to remove an element randomly from a collection?