1.Overview of Java And Creation of Java

Structured programming cannot manage complexity of complex programs. Obeject Oriented Programming makes it possible to organise large and complex programs using classes, objects, inheritance, polymorphism and encapsulation.C++ for years was most widely used OOP language. Due to advent of internet, Java became widely used and gained tremendous popularity.

James Ghosling,Naughton at Sun Microsystems in 1991.Initial name was Oak but it was renamed to Java in 1995

2.Features of Java:

1.Object Oriented

2.Platform Independent

3.Robust

4.Secured

5.Multi threading

6.Portable

3.Setting Classpath and Path for Java

Setting the path variable is like telling command prompt that where it should search for javac and java commands

ClassPath:

>javac -d (Specify the path where to save generated .class files) FileName.java

I:\Sources> javac -d I:\Classes ProgramOne.java

This will save generated .class file in I:\Classes.

If you run java command from this location, you will get an error saying could not find or load main class. Because .class file is saved in another folder. To run .class file, you need to go to that location, again using CD command. This will be the time consuming.

The easy and best way to run .class files saved in another folder is use -classpath option of java command. Syntax of java command

with -classpath option is,

>java -classpath (path of generated .class files) ClassName

In our example it looks like,

I:\Sources> java -classpath I:\Classes ProgramOne

You can also avoid use of -classpath option each time you run java command by setting the classpath variable by using set classpath command.

I:\Sources> set classpath=I:\Classes

Command line argument in Java:Command line argument is a parameter that is passed to the application at the time of execution.

4.JVM---interpretor for ByteCode

5.JDK,JRE,JVM

JRE==JVM+ other components to run applets and applications written in Java Language

JRE does not contain tools such as compilers or debuggers

JDK==JRE + tools such as compiler and debugger

6.Program Explanation:

Static is again a keyword used to make a function static.To execute a static function

you do not have to create an object of the class.The main() method here is called by JVM without creating any object for class."System" is a final class from java.lang package. "out" is static member of System class of type PrintStream. "println" is a method of PrintStream class.

You can explore the source code of both System class and String class. Go to JDK installation directory and extract the 'src' zip file.

Then go to src –> java –> lang. In lang folder you will find both System and String Java files.

7.Data types in Java

8.Variables In Java:

1.Instance variables:Instance variables are variables that are declare inside a class but outside any method,constructor or block.

2.Static variables:Static are class variables declared with static keyword

3.Local variables

9.Java Operators:

Bitwise operators --

& Bitwise AND

| Bitwise OR

^ Bitwise exclusive OR

<< left shift

>> right shift

a b a & b a | b

0 0 0 0

0 1 0 1

1 0 0 1

1 1 1 1

instanceOf operator:

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

The operator checks whether the object is of particular type at runtime

public class Test

{

public static void main(String[] args)

{

Test t= new Test();

System.out.println(t instanceof Test);

}

}

10.Object and Classes:

Class is a blue print or prototype that defines the data member and method common to all objects of a particular type.

Object is an instance of a class.

Features of OOPS

11.Constructors

First statement in a constructor must be either super() or this(). If you put any other statements you will get compile time error.If you don't

include these statements, by default compiler will keep super() calling statement.super() – It is a calling statement to default constructor of super class. this()- it is a calling statement to constructor of the same class.

There are three types of Constructor

•Default Constructor

•Parameterized constructor

•Copy constructor

Constructor Overloading -One class can have more than one constructors. It is called Constructor Overloading

Constructor chaining --Constructor chaining is a phenomena of calling one constructor from another constructor of same class

this keyword:this keyword is used to refer to current object.

12.Garbage Collection--Garbage Collection is a process that only frees the memory of Objects that are no more in use.

Can the Garbage Collection be forced explicitly ?

No, the Garbage Collection can not be forced explicitly. We may request JVM for garbage collection by calling System.gc() method. But This

does not guarantee that JVM will perform the garbage collection.

Advantages of Garbage Collection:

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

a.Programmer doesn't need to worry about dereferencing an object.

b.It is done automatically by JVM.

c.Increases memory efficiency and decreases the chances for memory leak.

finalize() method:

Sometime an object will need to perform some specific task before it is destroyed such as closing an open connection or releasing any resources held.

To handle such situation finalize() method is used

Memory Leak in Java:

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

A memory leak in Java is a situation where some objects are not used by the application any more, but GC fails to recognize them as unused.

As a result, these objects remain in memory indefinitely, reducing the amount of memory available to the application.

13.Modifiers in Java

1.Access control modifier:

Default : Default has scope only inside the same package

Public : Public scope is visible everywhere

Protected : Protected has scope within the package and all sub classes

Private : Private has scope only within the classes

2.Non Access Modifier:

a.Final

b.Static

c.Transient

d.Synchronized

e.Volatile

Transient:

If the variable is declared as transient, then it will not be persisted. That is the main purpose of the transient keyword

Volatile:

By making a variable volatile using the volatile keyword in Java, application programmer ensures that its value should always be read from main memory and thread should not use cached value of that variable from their own stack. volatile is used to indicate that a variable's value will be modified by different threads

14.Inheritance (IS-A)

super keyword--super keyword is used to refer to immediate parent class of a class

Q. Can you use both this() and super() in a Constructor?

NO, because both super() and this() must be first statement inside a constructor. Hence we cannot use them together.

15.Aggregation (HAS-A)

Q.Can we Override static method ? Explain with reasons ?

No, we cannot override static method. Because static method is bound to class whereas method overriding is associated with object i.e at runtime.

16.Polymorphism

17.Package:

a.Built-in Package:java.lang, java.util etc.

b.User Defined Package

Static import:

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

Static import is used to import static member of a class.

import static package.class-name.static-member-name;

Example using static import:

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

import static java.lang.System.\*;

public class Test

{

public static void main(String[] args)

{

out.println("Welcome");

}

}

18.Abstract class and Interface

Interface is a contract for what a class can do.

19.Nested Class:

Example:

class Outer

{

public void display()

{

Inner in=new Inner();

in.show();

}

class Inner

{

public void show()

{

System.out.println("Inside inner");

}

}

}

class Test

{

public static void main(String[] args)

{

Outer ot=new Outer();

ot.display();

}

}

Annonymous class:

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

A class without any name is called Annonymous class.

20.String Handling,StringBuffer and StringBuilder:

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

Each time you create a String literal, the JVM checks the string pool first. If the string literal already exists in the pool, a reference to the pool instance is returned. If string does not exist in the pool, a new string object is created, and is placed in the pool. String objects are stored in a special memory area known as string constant pool inside the heap memory.

Example showing difference between String and StringBuffer:

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

class Test {

public static void main(String args[])

{

String str = "Test";

str.concat("Java");

System.out.println(str); // Output: Test

StringBuffer strB = new StringBuffer("Test");

strB.append("Java");

System.out.println(strB); // Output: TestJava

}

}

Example of StringBuilder:

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

class Test {

public static void main(String args[])

{

StringBuilder str = new StringBuilder("Test");

str.append( "Java" );

System.out.println(str);

str.replace( 6, 13, "today");

System.out.println(str);

str.reverse();

System.out.println(str);

str.replace( 6, 13, "today");

}

}

21.Exception Handling:

Exception is an event that occurs during the execution of a program which interupts the flow.

Exception Handling Mechanism

In java, exception handling is done using five keywords,

1.try

2.catch

3.throw

4.throws

5.finally

Exception handling is done by transferring the execution of a program to an appropriate exception handler when exception occurs.

Example for Unreachable Catch block:

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

While using multiple catch statements, it is important to remember that exception sub classes inside catch must come before any of their super classes

otherwise it will lead to compile time error.

class Excep

{

public static void main(String[] args)

{

try

{

int arr[]={1,2};

arr[2]=3/0;

}

catch(Exception e) //This block handles all Exception

{

System.out.println("Generic exception");

}

Throw and Throws Keyword:

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

By using throw we can throw an exception explicitly.

By using throws we can list the type of exceptions that a method might throw.

More about Overriden Methods and Exceptions:

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

If Super class method throws an exception, then Subclass overriden method can throw the same exception or no exception, but must not throw parent

exception of the exception thrown by Super class method.

It means, if Super class method throws object of NullPointerException class, then Subclass method can either throw same exception, or can throw no exception, but it can never throw object of Exception class (parent of NullPointerException class).

There are few things to remember when overriding a method with exception handling. If super class method does not declare any exception, then sub

class overriden method cannot declare checked exception but it can declare unchecked exceptions.

Example of Subclass overriden Method declaring Checked Exception

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

import java.io.\*;

class Super

{

void show() { System.out.println("parent class"); }

}

public class Sub extends Super

{

void show() throws IOException //Compile time error

{ System.out.println("parent class"); }

public static void main( String[] args )

{

Super s=new Sub();

s.show();

}

}

As the method show() doesn't throws any exception while in Super class, hence its overriden version can also not throw any checked exception.

Example of Subclass overriden Method declaring Unchecked Exception:

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

import java.io.\*;

class Super

{

void show(){ System.out.println("parent class"); }

}

public class Sub extends Super

{

void show() throws ArrayIndexOutOfBoundsException //Correct

{ System.out.println("child class"); }

public static void main(String[] args)

{

Super s=new Sub();

s.show();

}

}

Output : child class

Because ArrayIndexOutOfBoundsException is an unchecked exception hence, overrided show() method can throw it.

Example of Subclass overriden method with same Exception:

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

import java.io.\*;

class Super

{

void show() throws Exception

{ System.out.println("parent class"); }

}

public class Sub extends Super {

void show() throws Exception //Correct

{ System.out.println("child class"); }

public static void main(String[] args)

{

try {

Super s=new Sub();

s.show();

}

catch(Exception e){}

}

}

Example of Subclass overriden method with no Exception:

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

import java.io.\*;

class Super

{

void show() throws Exception

{ System.out.println("parent class"); }

}

public class Sub extends Super {

void show() //Correct

{ System.out.println("child class"); }

public static void main(String[] args)

{

try {

Super s=new Sub();

s.show();

}

catch(Exception e){}

}

}

Example of Subclass overriden method with parent Exception:

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

import java.io.\*;

class Super

{

void show() throws ArithmeticException

{ System.out.println("parent class"); }

}

public class Sub extends Super {

void show() throws Exception //Cmpile time Error

{ System.out.println("child class"); }

public static void main(String[] args)

{

try {

Super s=new Sub();

s.show();

}

catch(Exception e){}

}

}catch(ArrayIndexOutOfBoundsException e) //This block is unreachable

{

System.out.println("array index out of bound exception");

}

}

}

22. Multithreading

Synchronization:

Synchronization is a process that ensures only one thread can access one resource at a time.

The synchronization keyword in java creates a block of code referred to as critical section.

Every Java object with a critical section of code gets a lock associated with the object. To enter critical section a thread need to obtain the corresponding object's lock.

General Syntax :

synchronized (object)

{

//statement to be synchronized

}

Why we use Syncronization ?

If we do not use syncronization, and let two or more threads access a shared resource at the same time, it will lead to distorted results.

Consider an example, Suppose we have two different threads T1 and T2, T1 starts execution and save certain values in a file temporary.txt which will be used to calculate some result when T1 returns. Meanwhile, T2 starts and before T1 returns, T2 change the values saved by T1 in the file temporary.txt (temporary.txt is the shared resource). Now obviously T1 will return wrong result.

To prevent such problems, synchronization was introduced. With synchronization in above case, once T1 starts using temporary.txt file, this file will be locked(LOCK mode), and no other thread will be able to access or modify it until T1 returns.

Using Synchronized Methods

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

Using Synchronized methods is a way to accomplish synchronization. But lets first see what happens when we do not use synchronization in our program.

Example with no Synchronization:

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

class First

{

public void display(String msg)

{

System.out.print ("["+msg);

try

{

Thread.sleep(1000);

}catch(InterruptedException e)

{

e.printStackTrace();

}

System.out.println ("]");

}

}

class Second extends Thread

{

String msg;

First fobj;

Second (First fp,String str)

{

fobj = fp;

msg = str;

start();

}

public void run()

{

fobj.display(msg);

}

}

public class Syncro

{

public static void main (String[] args)

{

First fnew = new First();

Second ss = new second(fnew, "welcome");

Second ss1= new second (fnew,"new");

Second ss2 = new second(fnew, "programmer");

}

}

Output :

[welcome [ new [ programmer]

]

]

In the above program, object fnew of class First is shared by all the three running threads(ss, ss1 and ss2) to call the shared method(void display).

Hence the result is unsynchronized and such situation is called Race condition.

Synchronized Keyword

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

To synchronize above program, we must serialize access to the shared display() method, making it available to only one thread at a time. This is done by

using keyword synchronized with display() method.

synchronized void display (String msg)

Using Synchronized block

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

If you have to synchronize access to object of a class that has no synchronized methods, and you cannot modify the code. You can use synchronized block to use it.

class First

{

public void display(String msg)

{

System.out.print ("["+msg);

try

{

Thread.sleep(1000);

}

catch(InterruptedException e)

{

e.printStackTrace();

}

System.out.println ("]");

}

}

class Second extends Thread

{

String msg;

First fobj;

Second (First fp,String str)

{

fobj = fp;

msg = str;

start();

}

public void run()

{

synchronized(fobj) //Synchronized block

{

fobj.display(msg);

}

}

}

public class Syncro

{

public static void main (String[] args)

{

First fnew = new First();

Second ss = new second(fnew, "welcome");

Second ss1= new second (fnew,"new");

Second ss2 = new second(fnew, "programmer");

}

}

Output :

[welcome]

[new]

[programmer]

Because of synchronized block this program gives the expected output.

23.Autoboxing and Unboxing:

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

•Autoboxing and Unboxing features was added in Java5.

•Autoboxing is a process by which primitive type is automatically encapsulated(boxed) into its equivalent type wrapper

•Auto-Unboxing is a process by which the value of object is automatically extracted from a type wrapper.

Example of Autoboxing and Unboxing:

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

class Test

{

public static void main(String[] args)

{

Integer iob = 100; //Autoboxing of int

int i = iob; //unboxing of Integer

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

Character cob = 'a'; /Autoboxing of char

char ch = cob; //Auto-unboxing of Character

System.out.println(cob+" "+ch);

}

}

24.Serialization and Deserialization in Java

Serialization is the process for writing the state of an object to a byte stream.

25.Marker interface:

Marker Interface is an interface which does not have any method. Marker interface is used to inform compiler that the class implementing it has some special behaviour or meaning. Some example of Marker interface are,

•java.io.Serializable

•java.lang.Cloneable

•java.rmi.Remote

•java.util.RandomAccess

26.Generics:

Generics was added in Java 5 to provide compile-time type checking and removing risk of ClassCastException that was common while working with

collection classes. The whole collection framework was re-written to use generics for type-safety. Let’s see how generics help us using collection

classes safely.

List list = new ArrayList();

list.add("abc");

list.add(new Integer(5)); //OK

for(Object obj : list){

//type casting leading to ClassCastException at runtime

String str=(String) obj;

}

Above code compiles fine but throws ClassCastException at runtime because we are trying to cast Object in the list to String whereas one of the element

is of type Integer. After Java 5, we use collection classes like below.

List<String> list1 = new ArrayList<String>(); // java 7 ? List<String> list1 = new ArrayList<>();

list1.add("abc");

//list1.add(new Integer(5)); //compiler error

for(String str : list1){

//no type casting needed, avoids ClassCastException

}

27.Collection:

Collection

a.Set

b.List

c.Deque

Accessing a Collection:

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

To access, modify or remove any element from any collection we need to first find the element, for which we have to cycle throught the elements of the collection. There are three possible ways to cycle through the elements of any collection.

1.Using Iterator interface

2.Using ListIterator interface

3.Using for-each loop

Accessing elements using Iterator:

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

Iterator Interface is used to traverse a list in forward direction, enabling you to remove or modify the elements of the collection. Each collection

classes provide iterator() method to return an iterator.

import java.util.\*;

class Test\_Iterator

{

public static void main(String[] args)

{

ArrayList< String> ar = new ArrayList< String>();

ar.add("ab");

ar.add("bc");

ar.add("cd");

ar.add("de");

Iterator it = ar.iterator(); //Declaring Iterator

while(it.hasNext())

{

System.out.print(it.next()+" ");

}

}

}

Output :

ab bc cd de

Accessing element using ListIterator:

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

ListIterator Interface is used to traverse a list in both forward and backward direction. It is available to only those collections that implement the List Interface.

import java.util.\*;

class Test\_Iterator

{

public static void main(String[] args)

{

ArrayList< String> ar = new ArrayList< String>();

ar.add("ab");

ar.add("bc");

ar.add("cd");

ar.add("de");

ListIterator litr = ar.listIterator();

while(litr.hasNext()) //In forward direction

{

System.out.print(litr.next()+" ");

}

while(litr.hasPrevious()) //In backward direction

{

System.out.print(litr.previous()+" ");

}

}

}

Output :

ab bc cd de

de cd bc ab

Using for-each loop:

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

for-each version of for loop can also be used for traversing each element of a collection. But this can only be used if we don't want to modify the contents of a collection and we don't want any reverse access. for-each loop can cycle through any collection of object that implements Iterable interface.

import java.util.\*;

class ForEachDemo

{

public static void main(String[] args)

{

LinkedList< String> ls = new LinkedList< String>();

ls.add("a");

ls.add("b");

ls.add("c");

ls.add("d");

for(String str : ls)

{

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

}

}

}

Output :

a b c d

Why a non-static variable cannot be referenced from a static context ?

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

When you try to access a non-static variable from a static context like main method, java compiler throws a message like "a non-static variable cannot be referenced from a static context". This is because non-static variables are related with instance of class(object) and they get created when instance of a class is created by using new operator. So if you try to access a non-static variable

without any instance compiler will complain because those variables are not yet created and they don't have any existence until an instance is created and associated with it.

Example of accessing non-static variable from a static context

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

class Test

{

int x;

public static void main(String[] args)

{

x=10;

}

}

ClassLoader:

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

Java ClassLoader loads a java class file into java virtual machine.

1.Bootstrap Class Loader

2.Extensions Class Loader

3.System Class Loader

Polymorphism:

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

Polymorphism means one name multiple forms.It is a concept by which we can perform same action in different ways.

Method Overloading:

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

Method overloading is a programming concept when programmer declares two methods of the same name but with different method signature,

e.g. change in the argument list or change in the type of argument

One of the most popular examples of method overloading is System.out.println() method which is overloaded to accept all kinds of data types in Java.You have println() method which takes String, int, float,double or even char in output.All of those methods are collectively referred as an overloaded method in Java

Can we declare a class as Abstract without having any abstract method?

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

Ans: Yes we can create an abstract class by using abstract keyword before class name even if it doesn’t have any abstract method

abstract class Test

{

void m1()

{

System.out.println("ssss");

}

}

class Test2 extends Test

{

}

class A3

{

public static void main(String dd[])

{

Test2 ttt=new Test2();

ttt.m1();

}

}

Does Importing a package imports its sub-packages as well in Java?

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

Ans: In java, when a package is imported, its sub-packages aren’t imported and developer needs to import them separately if required.

When a lot of changes are required in data, which one should be a preference to be used? String or StringBuffer?

-------------------------------------------------------------------------Ans: Since StringBuffers are dynamic in nature and we can change the values of StringBuffer objects unlike String which is immutable,it’s always a good choice to use StringBuffer when data is being changed too much. If we use String in such a case, for every data change a new String object will be created which will be an extra overhead.

What’s meant by anonymous class?

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

Ans: An anonymous class is a class defined without any name in a single line of code using new keyword

Does java support multiple interitance? Why?

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

Ans) Java doesnt support multiple inheritance but it provide a way through which it can enact it. Consider the scenario is C++

Class A {

public void add() {

// some text

}

}

Class B {

public void add() {

// some text

}

}

Class C extends A,B {

public static void main(String arg[]){

C objC = new C();

objC.add(); // problem, compiler gets confused and cant

decide to call Class A or B method.

}

This problem is called Diamond problem.

This problem in java is taken care with the use of interfaces

In Java similar problem would look like:

interface A {

void add();

}

interface B {

void add();

}

class C implements A,B {

void add() {

// doesnt matter which interface it belong to

}

}

Java supports pass by value or pass by reference?

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

Ans) Java supports only pass by value. The arguments passed as a parameter to a method is mainly primitive data types or objects.

For the data type the actual value is passed.Java passes the references by value just like any other parameter. The pointer to the

object is passed as value.

Why is it preferred to declare: List<String> list = new ArrayList<String>(); instead of ArrayList<String> = new ArrayList<String>();

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

Ans) It is preferred because:

1.If later on code needs to be changed from ArrayList to Vector then only at the declaration place we can do that.

2.The most important one – If a function is declared such that it takes list. E.g void showDetails(List list);

When the parameter is declared as List to the function it can be called by passing any subclass of List like ArrayList,

Vector, LinkedList making the function more flexible.

What is difference between Enumeration and Iterator in Java?

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

Though both Iterator and Enumeration allows you to traverse over elements of Collections in Java,Iterator also allows you to remove elements from

collection during traversal but Enumeration doesn't allow that, it doesn't got the remove() method.

Iterator is more secure and safe as compared to Enumeration because it does not allow other thread to modify the collection object while some thread is iterating over it and throws ConcurrentModificationException.

SIB – Static Initialization Block, Static Variables And Static Methods

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

Static variables, Static Initialization Block and Static Methods – these all are static components or static members of a class.

These static members are stored inside the Class Memory. To access static members, you need not to create objects. Directly you can access

them with class name.

Static Initialization Block is used to initialize only static variables. It is a block without a name. It contains set of statements enclosed within { }.

The syntax of SIB looks like this,

static

{

//Set Of Statements

}

Consider the following program.

class StaticComponents

{

static int staticVariable;

static

{

System.out.println("StaticComponents SIB");

staticVariable = 10;

}

static void staticMethod()

{

System.out.println("From StaticMethod");

System.out.println(staticVariable);

}

}

public class MainClass

{

static

{

System.out.println("MainClass SIB");

}

public static void main(String[] args)

{

//Static Members directly accessed with Class Name

StaticComponents.staticVariable = 20;

StaticComponents.staticMethod();

}

}

Let us discuss execution of above program step by step.

Step 1:

When you trigger >java MainClass, java command divides allocated memory into two parts – Stack and Heap. First, java command enters stack memory for execution. First, it checks whether

MainClass is loaded into heap memory or not. If it is not loaded, loading operation of MainClass starts. Randomly some memory space is allocated to MainClass. It is called Class memory. All static members are loaded into this class memory. There is only one satic member in MainClass – main() method. It is loaded into class memory of MainClass.

Step 2:

After loading all static members, SIB – Static initialization Blocks are executed. Remember, SIBs are not stored in the heap memory. They just come to stack, execute their tasks and leaves the memory. So, after loading main() method, SIB of MainClass enters stack for execution. There is only one statement (Line 22) in SIB. it is executed. It prints "MainClass SIB" on console. After executing this statement, SIB leaves the stack memory.

Step 3:

Now, java command calls main() method for execution. main() method enters the stack. First statement (Line 28) is executed first. First, It checks whether class StaticComponents is loaded into memory. If it is not loaded, loading operation of StaticComponents takes place. Randomly, some memory is allocated to Class StaticComponents, then all static members of StaticComponents – 'staticVariable' and 'staticMethod()' are loaded into that class memory.'staticVariable' is a global variable. So, first it is initialized with default value i.e 0.

Step 4 :

After loading all static members of StaticComponents, SIB blocks are executed. So, SIB of class StaticComponents enters the stack for execution.

First Statement (Line 7) is executed. It prints "StaticComponents SIB" on the console. In the second statement, value 10 is assigned to'staticVariable'. There are no other statements left for execution, so it leaves stack memory.

Step 5 :

Now control comes back to main() method. The remaining part of first statement i.e value 20 is assigned to 'staticVariable' of class

StaticComponents,is executed. In the second statement (Line 29), it calls staticMethod() of class StaticComponents for execution.

Step 6:

staticMethod() of StaticComponents enters stack for execution. First statement (Line 13) is executed first. It prints "From staticMethod" on the console.In the second statement (Line 14), it prints the value of staticVariable i.e 20 on the console. There are no statements left. so, it leaves the stack.

Step 7:

Again, control comes back to main() method. There are no other statements left in main() method. so, it also leaves stack. java command also leaves

the stack.

Can We Overload main() method?

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

Yes, We can overload main() method. A Java class can have any number of main() methods. But to run the java class, class should have main()

method with signature as "public static void main(String[] args)". If you do any modification to this signature, compilation will be successful.

But, you can't run the java program. You will get run time error as main method not found.

public class MainMethod

{

public static void main(String[] args)

{

System.out.println("Execution starts from this method");

}

void main(int args)

{

System.out.println("Another main method");

}

double main(int i, double d)

{

System.out.println("Another main method");

return d;

}

}

Can we declare main() method as private or protected or with no access modifier?

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

No, main() method must be public. You can't define main() method as private or protected or with no access modifier.

This is because to make the main() method accessible to JVM. If you define main() method other than public, compilation will be successful but

you will get run time error as no main method found.

public class MainMethod

{

private static void main(String[] args)

{

//Run time error

}

}

Can We Declare main() Method As Non-Static?

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

No, main() method must be declared as static so that JVM can call main() method without instantiating it's class.

If you remove 'static' from main() method signature, compilation will be successful but program fails at run time.

public class MainMethod

{

public void main(String[] args)

{

System.out.println(1); //Run time error

}

}

Why main() method must be static?

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

Suppose, If main() is allowed to be non-static, then while calling the main method JVM has to instantiate it's class.

While instantiating it has to call constructor of that class. There will be an ambiguity if constructor of that class takes an argument.

For example, In the below program what argument JVM has to pass while instantiating class "MainMethod"

public class MainMethod

{

public MainMethod(int i)

{

//Constructor taking one argument

}

public void main(String[] args)

{

//main method as non-static

}

}

That’s why main() method must be static.

Can we change return type of main() method?

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

No, the return type of main() method must be void only. Any other type is not acceptable.

public class MainMethod

{

public static int main(String[] args)

{

return 1; //run time error : No main method found

}

}

Can main() method take an argument other than string array?

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

No, argument of main() method must be string array. But, from the introduction of var args you can pass var args of string type as an argument to main() method. Again, var args are nothing but the arrays.

public class MainMethod

{

public static void main(String... args)

{

//Var args as an argument

}

}

Can we run java class without main() method?

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

No, you can’t run java class without main method. But, there are some scenarios like if super class has main() method, then sub class can be run without defining main() method in it. For example, if you run class B in the below program, you will get 1 as output.

class A

{

public static void main(String[] args)

{

System.out.println(1);

}

}

public class B extends A

{

}

Note : Before Java 7, you can run java class by using static initializers. But, from Java 7 it is not possible.

Can you create an object without using new operator in Java?

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

Yes, We can create an object without using new operator. There are some other ways to create objects other than using new operator. But, 95% of object creation in java is done through new operator only.

a) Using newInstance() Method

Class c = Class.forName("packageName.MyClass");

MyClass object = (MyClass) c.newInstance();

b) Using clone() method.

MyClass object1 = new MyClass();

MyClass object2 = object1.clone();

c) Using object deserialization

ObjectInputStream inStream = new ObjectInputStream(anInputStream );

MyClass object = (MyClass) inStream.readObject();

d) Creating string and array objects :

String s = "string object";

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

What is constructor chaining?

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

Constructor Chaining is a technique of calling another constructor from one constructor. this() is used to call same class constructor where as super() is used to call super class constructor.

class SuperClass

{

public SuperClass(int i)

{

System.out.println("Super Class Constructor");

}

}

class SubClass extends SuperClass

{

public SubClass()

{

this(10); //Calling same class constructor

}

public SubClass(int i)

{

super(i); //Calling super class constructor

}

}

Can we call sub class constructor from super class constructor?

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

No. There is no way in java to call sub class constructor from a super class constructor.

Java Notes

Written By Sanjoy Dutta