1. **What is difference between JDK, JRE and JVM?**

|  |  |  |
| --- | --- | --- |
| **JVM** | **JRE** | **JDK** |
| Java Virtual Machine | Java Runtime Environment | Java Development Kit |
| It is an abstract machine which provides the runtime environment in which java bytecode can be executed.  It is a specification. | JVM + Required Library to run Java Application.  It does not contain any development tools such as compiler, debugger, etc.  It is the implementation of JVM. | It contains JRE + development tools.  It physically exists. |
| It cannot run java code only bytecode | It can run java program | It can run java program |
| The JVM executes Java code, but is written in platform specific languages such as C/C++/ASM etc.  The JVM is not written in Java and hence cannot be platform independent. | It is platform dependent. | It is also platform dependent |

1. **If JRE, JDK, JVM are platform dependent then how java is platform independent?**

The bytecode generated after compilation of a Java source file (having the .java extension) can be executed on any platform (write once run anywhere WORA) that’s why java is called platform independent.

Java code can be compiled to bytecode by JRE/JDK. Bytecode is portable code, this can be moved to any machine (Windows / Linux) and executed by JVM.

Now JVM needs to convert the bytecode to machine specific code so that’s why JVM is machine specific.

So in this way the coder doesn’t need to bother with generating byte code. JVM takes care of portability. And this makes Java a platform independent language.

1. **If I don't provide any arguments on the command line, then the String array of Main method will be empty or null?**

It is empty. But not null.

1. **What if I write static public void instead of public static void?**

Program compiles and runs properly.

1. **What is the default value of the local variables?**

The local variables are not initialized to any default value, neither primitives nor object references.

1. **Does constructor return any value?**

Yes, that is current class instance (You cannot use return type yet it returns a value).

Constructor is not inherited. Constructor cannot be made final.

1. **Significance of Final keyword in Java?**

For primitive datatypes: When we use the *final* keyword with a variable declaration, the value stored inside that variable cannot be changed latter.

final int speedlimit=90;

Again if you try speedlimit=400;  Compile Time Error.

final int y; y = 0; y++; Compile Time Error

For reference Types:  In case of any object is made as final, like arraylist, list its value can change but not the reference.

A class declared as final cannot be extended or inherited. Methods inside final class cannot be overridden.

A function/ method declared as final cannot be overridden by any sub class.

1. **Significance of Static keyword in Java?**

The static variable can be used to refer the common property of all objects.

The static variable gets memory only once in class area at the time of class loading.

Example:

class Counter2 {

static int count=0;//will get memory only once and retain its value

Counter2() { count++; System.out.println(count); }

public static void main(String args[]) {

Counter2 c1=new Counter2();

Counter2 c2=new Counter2();

Counter2 c3=new Counter2();

}

} Output: 1 2 3

* A static method belongs to the class rather than object of a class.
* A static method can be invoked without the need for creating an instance of a class.
* Static method can access static data member and can change the value of it.
* The static method cannot use non static data member or call non-static method directly.
* this and super cannot be used in static context.

1. **Why main() is static?**

The main() method in C++, C# and Java are static because they can then be invoked by the runtime engine without having to instantiate an instance of the parent class.

The method is static because otherwise there would be ambiguity: which constructor should be called? Especially if your class looks like this:

public class JavaClass{

protected JavaClass(int x){}

public void main(String[] args){

}

}

Should the JVM call ***new JavaClass***(int)? What should it pass for x?

If not, should the JVM instantiate ***JavaClass*** without running any constructor method?

It shouldn't, because that will special-case entire class - sometimes you have an instance that hasn't been initialized, and you have to check for it in every method that could be called.

There are just too many edge cases and ambiguities for it to make sense for the JVM to have to instantiate a class before the entry point is called. That's why **main** is static.

1. **Why main is always public method?**

The main method is called by the JVM. JVM is outside the scope of the project. So if the main function is not public the JVM won't be able to call the function.

1. **Can we execute a program without main() method?**

Yes, one of the way is static block but in previous version of JDK not in JDK 1.7.

class A3{

static{

System.out.println("static block is invoked");

System.exit(0);

}

}

Output:static block is invoked (if not JDK7)

Output:Error: Main method not found in class A3, please define the main method as:

public static void main(String[] args)

1. **What if the static modifier is removed from the signature of the main method?**

Program compiles. But at runtime throws an error "NoSuchMethodError".

1. **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.

1. **Can you use this() and super() both in a constructor?**

No. Because super() or this() must be the first statement.

1. **Can we overload main() method?**

You can overload the main() method, but only public static void main(String[] args) will be used when your class is launched by the JVM.

class Simple{

public static void main(int a){

System.out.println(a);

}

public static void main(String args[]){

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

main(10);

}

}

Output: main() method invoked

10

1. **Can you declare the main method as final?**

Yes, such as, public static final void main(String[] args){}.

class A {

public static void main(String[] args) {

System.out.println("A");

}

}

class B extends A {

public static void main(String[] args) {

System.out.println("B");

}

}

class C extends B {

}

public class Test {

public static void main(String[] args) {

C.main(args); // Will invoke B.main

}

}

Adding final to A.main would prevent accidental hiding of A.main. In other words, adding final to A.main guarantees that B.main is not allowed, and that C.main therefore prints "A" as opposed to for instance "B".

1. **What is abstract class?**

A class that is declared as abstract is known as abstract class. It needs to be extended and its method implemented. It cannot be instantiated.

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

Abstract cannot be used with Final because abstract method needs to be overridden whereas you can't override final method.

1. **What is interface?**

Interface is a blueprint of a class that have static constants and abstract methods.It can be used to achieve fully abstraction and multiple inheritance.

Interface method cannot be declared static since interface is abstract by default and abstract and static cannot be used together.

Interface cannot be final as its implementation is provided by other class.

An interface that have no data member and method is known as a marker interface.For example Serializable, Cloneable etc.

Interface cannot have instance variables, constructor and static methods.

1. **What is difference between Checked Exception and Unchecked Exception?**

1)Checked Exception

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.

2)Unchecked Exception

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

Throwable is base class for Error and Exception

Each try block must be followed by either a catch or a finally block

Finally will always be executed unless System.exit() call is made or program terminated due to fatal error.

Subclass overriding method can declare an unchecked exception (not checked) if parent class method doesn't throw an exception

An exception can be rethrown.

|  |  |
| --- | --- |
| **throw keyword** | **throws keyword** |
| 1)throw is used to explicitly throw an exception. | throws is used to declare an exception. |
| 2)checked exceptions can not be propagated with throw only. | checked exception can be propagated with throws. |
| 3)throw is followed by an instance. | throws is followed by class. |
| 4)throw is used within the method. | throws is used with the method signature. |
| 5)You cannot throw multiple exception | You can declare multiple exception e.g. public void method()throws IOException,SQLException. |

1. **Why string objects are immutable in java?**

Meaning of immutable is unmodifiable or unchangeable. Java uses the concept of string literal. Suppose there are 5 reference variables, all refers to one object "caching". 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.

Two ways to create a String:

Using New Keyword:

char[] ch={'j','a','v','a','t','p','o','i','n','t'};

String s=new String(ch);

Using String Literal: String s="ABC";

For all three statements below only one object will be created:

String s1="Welcome";

String s2="Welcome";

String s3="Welcome";

Two objects will be for statement: String s = new String("Welcome");

1. **What is the purpose of toString() method in java ?**

The toString() method returns the string representation of any object. If you print any object, java compiler internally invokes the toString() method on the object. So overriding the toString() method, returns the desired output.

|  |  |
| --- | --- |
| 1. **class** Student{ 2. **int** rollno; 3. String name; 4. String city; 6. Student(**int** rollno, String **name**, String city){ 7. **this**.rollno=rollno; 8. **this**.name=name; 9. **this**.city=city;   }    **public** **static** **void** main(String args[]){     Student s1=**new** Student(101,"Raj","lucknow");     Student s2=**new** Student(102,"Vijay","ghazi");   System.out.println(s1);//compiler writes here s1.toString()   System.out.println(s2);   }  } | 1. **class** Student{ 2. **int** rollno; 3. String name; 4. String city; 6. Student(**int** rollno, String name, String city){ 7. **this**.rollno=rollno; 8. **this**.name=name; 9. **this**.city=city;    }    **public** String toString(){//overriding the toString() method  **return** rollno+" "+name+" "+city;   }  **public** **static** **void** main(String args[]){     Student s1=**new** Student(101,"Raj","lucknow");     Student s2=**new** Student(102,"Vijay","ghazi");       System.out.println(s1);//compiler writes here s1.toString()     System.out.println(s2);   }  } |
| Output: Student@1fee6fc  Student@1eed786 | Output:101 Raj lucknow  102 Vijay ghazi |

1. **What is Garbage Collection?**

Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.

Garbage Collector thread is a Daemon thread.

**finalize() method**

The finalize() method is invoked each time before the object is garbage collected. This method can be used to perform cleanup processing.

**gc() method**

The gc() method is used to invoke the garbage collector to perform cleanup processing.

1. **public** **class** TestGarbage1{
2. **public** **void** finalize(){System.out.println("object is garbage collected");}
3. **public** **static** **void** main(String args[]){
4. TestGarbage1 s1=**new** TestGarbage1();
5. TestGarbage1 s2=**new** TestGarbage1();
6. s1=**null**;
7. s2=**null**;
8. System.gc();
9. }
10. }

Unreferenced object can be referenced again

Ways to unreference an object:

|  |  |  |
| --- | --- | --- |
| **Nulling the reference** | **Assigning a reference to another** | **Anonymous object** |
| Employee e=new Employee();  e=null; | Employee e1=new Employee();  Employee e2=new Employee();  e1=e2; | new Employee(); |

1. **What is serialization and deserialization?**

Serialization is a process of writing the state of an object into a byte stream.It is mainly used to travel object's state on the network.

Serializable is a marker interface (has no data member and method). It is used to "mark" java classes so that objects of these classes may get certain capability. The Cloneable and Remote are also marker interfaces.

It must be implemented by the class whose object you want to persist.

The String class and all the wrapper classes implements java.io.Serializable interface by default.

Deserialization is the process of reconstructing the object from the serialized state.

|  |  |
| --- | --- |
| **Java Serialization** | **Java Deserialization** |
| import java.io.\*;  class Persist{   public static void main(String args[])throws Exception{    Student s1 =new Student(211,"ravi");      FileOutputStream fout=new FileOutputStream("f.txt");     ObjectOutputStream out=new ObjectOutputStream(fout);      out.writeObject(s1);    out.flush();    System.out.println("success");   }  } | import java.io.\*;  class Depersist{   public static void main(String args[])throws Exception{   ObjectInputStream in=new ObjectInputStream(new FileInputStream("f.txt"));    Student s=(Student)in.readObject();   System.out.println(s.id+" "+s.name);    in.close();   }  } |
| success | 211 ravi |

1. **Transient Keyword?**

If you define any data member as transient, it will not be serialized.

1. **import** java.io.Serializable;
2. **public** **class** Student **implements** Serializable{
3. **int** id;
4. String name;
5. **transient** **int** age;//Now it will not be serialized
6. **public** Student(**int** id, String name,**int** age) {
7. **this**.id = id;
8. **this**.name = name;
9. **this**.age=age;
10. }
11. }
12. **What are wrapper classes?**

Wrapper classes are classes that allow primitive types to be accessed as objects.

All the wrapper classes (Integer, Long, Byte, Double, Float, Short) are subclasses of the abstract class Number. Eight wrapper classes exist in java.lang package that represent 8 data types.

Converting primitive data types into object is called boxing, and this is taken care by the compiler.

public class Test {

public static void main(String args[]) {

Integer x = 5; // boxes int to an Integer object

x = x + 10; // unboxes the Integer to a int

System.out.println(x);

}

}

1. **What is the difference between ArrayList and Vector?**

|  |  |  |
| --- | --- | --- |
| **No.** | **ArrayList** | **Vector** |
| 1) | ArrayList is not synchronized. | Vector is synchronized. |
| 2) | ArrayList is not a legacy class. | Vector is a legacy class. |
| 3) | ArrayList increases its size by 50% of the array size. | Vector increases its size by doubling the array size. |

1. **What is the difference between HashSet and HashMap?**

|  |  |
| --- | --- |
| **HashSet** | **HashMap** |
| HashSet class implements the Set interface | HashMap class implements the Map interface |
| In HashSet, we store objects(elements or values) e.g. If we have a HashSet of string elements then it could depict a set of HashSet elements: {“Hello”, “Hi”, “Bye”, “Run”} | HashMap is used for storing key & value pairs. In short, it maintains the mapping of key & value.  This is how you could represent HashMap elements if it has integer key and value of String type:  e.g. {1->”Hello”, 2->”Hi”, 3->”Bye”, 4->”Run”} |
| HashSet does not allow duplicate elements that mean you can not store duplicate values in HashSet. | HashMap does not allow duplicate keys however it allows having duplicate values. |
| HashSet permits to have a single null value. | HashMap permits single null key and any number of null values. |

1. **What is the difference between HashTable and HashMap?**

|  |  |
| --- | --- |
| **HashTable** | **HashMap** |
| Hashtable is synchronized, slower, more memory | Not Synchronized, faster, less memory |
| Hashtable does not allow null keys or values. | HashMap allows one null key and any number of null values. |
| HashTable is the only class other than vector which uses enumerator to iterate the values of HashTable object. | Hashmap object values are iterated by using iterator |
| Legacy Class, Extends AbstractMap class | Not a legacy class, Extends Dictionary class |
| Preferred in multi-threaded applications, now not preferred because ConcurrentHashMap is a better option | HashMap is preferred in single threaded applications. If you want to use it in multi threaded wrap it using Collections.SynchronizedMap() method. |

1. **What is the difference between Comparable and Comparator?**

|  |  |  |
| --- | --- | --- |
| **No.** | **Comparable** | **Comparator** |
| 1) | Comparable provides only one sort of sequence. | Comparator provides multiple sort of sequences. |
| 2) | It provides one method named compareTo(). | It provides one method named compare(). |
| 3) | It is found in java.lang package. | It is found in java.util package. |
| 4) | If we implement Comparable interface, actual class is modified. | Actual class is not modified. |

1. **What does the hashCode() method?**

The hashCode() method returns a hash code value (an integer number).

The hashCode() method returns the same integer number, if two keys (by calling equals() method) are same. But, it is possible that two hash code numbers can have different or same keys.

The value received from hashCode() is used as the bucket number for storing elements of the set/map. This bucket number is the address of the element inside the set/map.

When you do contains() it will take the hash code of the element, then look for the bucket where hash code points to. If more than 1 element is found in the same bucket (multiple objects can have the same hash code), then it uses the equals() method to evaluate if the objects are equal, and then decide if contains() is true or false, or decide if element could be added in the set or not.

1. **Why we override equals() method?**

The equals method is used to check whether two objects are same or not. It needs to be overridden if we want to check the objects based on property.

For example, Employee is a class that has 3 data members: id, name and salary. But, we want to check the equality of employee object on the basis of salary. Then, we need to override the equals() method.

1. **Why to override both hashCode() and equals() method?**

Whenever*a.equals(b)*, then*a.hashCode()*must be same as*b.hashCode()*. So to check this condition both methods are implemented together.

Contract: **objects which are .equals() MUST have the same .hashCode()**.

1. **What is the default size of load factor in hashing based collection?**

The default size of load factor is 0.75. The default capacity is computed as initial capacity \* load factor. For example, 16 \* 0.75 = 12. So, 12 is the default capacity of Map.

1. **What is JDBC and why it is used?**

JDBC is a Java API that is used to connect and execute query to the database. JDBC API uses jdbc drivers to connect to the database. It is defined in java.sql package.

Before JDBC, ODBC API was the database API to connect and execute query with the database. But, ODBC API uses ODBC driver which is written in C language (i.e. platform dependent and unsecured). That is why Java has defined its own API (JDBC API) that uses JDBC drivers (written in Java language).

1. **What is JDBC Driver?**

JDBC Driver is a software component that enables java application to interact with the database.There are 4 types of JDBC drivers:

1. JDBC-ODBC bridge driver
2. Native-API driver (partially java driver)
3. Network Protocol driver (fully java driver)
4. Thin driver (fully java driver)
5. **What are the steps to connect to the database in java?**

There are 5 steps:

|  |  |
| --- | --- |
| Registering the driver class | Class.forName("oracle.jdbc.driver.OracleDriver"); |
| Creating connection | Connection con=DriverManager.getConnection(  "jdbc:oracle:thin:@localhost:1521:xe","system","password"); |
| Creating statement | Statement stmt=con.createStatement(); |
| Executing queries | ResultSet rs=stmt.executeQuery("select \* from emp");  while(rs.next()){  System.out.println(rs.getInt(1)+" "+rs.getString(2));  } |
| Closing connection | con.close(); |

1. **What are the JDBC statements?**

There are 3 JDBC statements.

1. Statement
2. PreparedStatement
3. CallableStatement.

The PreparedStatement interface is a subinterface of Statement. It is used to execute parameterized query.

In case of Statement, query is complied each time whereas in case of PreparedStatement, query is complied only once. So performance of PreparedStatement is better than Statement.

By using **Callable statement** interface, we can execute procedures and functions.

1. **What is the role of JDBC DriverManager class & Connection Interface?**

The **DriverManager class** manages the registered drivers. It can be used to register and unregister drivers. It provides factory method that returns the instance of Connection.

The **Connection interface** maintains a session with the database. It can be used for transaction management. It provides factory methods that returns the instance of Statement, PreparedStatement, CallableStatement and DatabaseMetaData.

1. **What is multithreading?**

Multithreading is a process of executing multiple threads simultaneously. Its main advantage is:

* Threads share the same address space.
* Thread is lightweight.
* Cost of communication between processes is low.

1. **What is a thread, give lifecycle of thread?**

A thread is a lightweight sub process. It is a separate path of execution. It is called separate path of execution because each thread runs in a separate stack frame.

According to Sun thread has 4 states: new, runnable, non-runnable and terminated. There is no running state.

|  |  |
| --- | --- |
| New | The thread is in new state if you create an instance of Thread class but before the invocation of start() method. |
| Runnable | The thread is in runnable state after invocation of start() method, but the thread scheduler has not selected it to be the running thread. |
| Running | The thread is in running state if the thread scheduler has selected it. |
| Non Runnable | This is the state when the thread is still alive, but is currently not eligible to run. |
| Terminated | A thread is in terminated or dead state when its run() method exits. |

Common Methods:

1. **public void run():**is used to perform action for a thread.
2. **public void start():**starts the execution of the thread.JVM calls the run() method on the thread.
3. **public void join():**waits for a thread to die.
4. **public void join(long miliseconds):**waits for a thread to die for the specified miliseconds.
5. **public void sleep(long miliseconds):**Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.
6. **What is the difference between preemptive scheduling and time slicing?**

Under preemptive scheduling, the highest priority task executes until it enters the waiting or dead states or a higher priority task comes into existence. Under time slicing, a task executes for a predefined slice of time and then reenters the pool of ready tasks. The scheduler then determines which task should execute next, based on priority and other factors.

1. **What is difference between wait() and sleep() method?**

|  |  |
| --- | --- |
| **wait()** | **sleep()** |
| 1) The wait() method is defined in Object class. | The sleep() method is defined in Thread class. |
| 2) wait() method releases the lock. | The sleep() method doesn't releases the lock. |

1. **Is it possible to start a thread twice?**

No. After starting a thread, it can never be started again. If you does so, an IllegalThreadStateException is thrown. In such case, thread will run once but for second time, it will throw exception.

public class TestThreadTwice1 extends Thread{

 public void run(){

   System.out.println("running...");

 }

 public static void main(String args[]){

  TestThreadTwice1 t1=new TestThreadTwice1();

  t1.start();

  t1.start();

 }

}

Output: running

Exception in thread "main" java.lang.IllegalThreadStateException

1. **What is the output?**

|  |  |
| --- | --- |
| 1. **class** TestCallRun2 **extends** Thread{ 2. **public** **void** run(){ 3. **for**(**int** i=1;i<5;i++){ 4. **try**{Thread.sleep(500);}**catch**(InterruptedException e){System.out.println(e);} 5. System.out.println(i); 6. } 7. } 8. **public** **static** **void** main(String args[]){ 9. TestCallRun2 t1=**new** TestCallRun2();     TestCallRun2 t2=**new** TestCallRun2();      t1.start();    t2.start();   }  } | 1  1  2  2  3  3  4  4 |

1. **How threads communicate between each other?**

Threads can communicate with each other by using wait(), notify() and notifyAll() methods.

1. **Can we call the run() method instead of start()?**

Yes, but it will not work as a thread rather it will work as a normal object so there will not be context-switching between the threads.

In the example, see the threads were executed as normal threads without any context switching.

class TestCallRun2 extends Thread{

public void run(){

for(int i=1;i<5;i++){

try{Thread.sleep(500);}catch(InterruptedException e){System.out.println(e);}

System.out.println(i);

  }

  }

public static void main(String args[]){

TestCallRun2 t1=new TestCallRun2();

TestCallRun2 t2=new TestCallRun2();

t1.run();

t2.run();

}

}

Output: 1 2 3 4 5 1 2 3 4 5

1. **What about the daemon threads?**

**Daemon thread in java** is a service provider thread that provides services to the user thread. Its life depend on the mercy of user threads i.e. when all the user threads dies, JVM terminates this thread automatically. A simple daemon thread example.

public class TestDaemonThread1 extends Thread{

 public void run(){

  if(Thread.currentThread().isDaemon()){//checking for daemon thread

   System.out.println("daemon thread work");

  }

  else{

  System.out.println("user thread work");

 }

 }

 public static void main(String[] args){

  TestDaemonThread1 t1=new TestDaemonThread1();//creating thread

  TestDaemonThread1 t2=new TestDaemonThread1();

  TestDaemonThread1 t3=new TestDaemonThread1();

  t1.setDaemon(true);//now t1 is daemon thread

  t1.start();//starting threads

  t2.start();

  t3.start();

 }

}

Output: daemon thread work

user thread work

user thread work

If you want to make a user thread as Daemon, it must not be started otherwise it will throw **IllegalThreadStateException**.

1. **What is shutdown hook?**

The shutdown hook is basically a thread i.e. invoked implicitely before JVM shuts down. So we can use it perform clean up resource. Performing clean resource means closing log file, sending some alerts or something else. So if you want to execute some code before JVM shuts down, use shutdown hook.

1. **What is the difference between notify() and notifyAll()?**

The **notify()** is used to unblock one waiting thread whereas **notifyAll()** method is used to unblock all the threads in waiting state.

1. **Explain Sleep() method for thread?**

The **sleep**() method of Thread class is used to sleep a thread for the specified amount of time.

class TestSleepMethod1 extends Thread{

 public void run(){

  for(int i=1;i<5;i++){

    try{Thread.sleep(500);}catch(InterruptedException e){System.out.println(e);}

    System.out.println(i);

  }

 }

 public static void main(String args[]){

  TestSleepMethod1 t1=new TestSleepMethod1();

  TestSleepMethod1 t2=new TestSleepMethod1();

  t1.start();

  t2.start();

 }

}

Output: 1 1 2 2 3 3 4 4

sd