**How garbage collector knows that the object is not in use and needs to be removed?**

Garbage collector reclaims objects that are no longer being used, clears their memory, and keeps the memory available for future allocations. This is done via bookkeeping the references to the objects. Any unreferenced object is a garbage and will be collected.

# Can Java thread object invoke start method twice?

public class MyExmpCode extends Thread{

public void run(){

System.out.println ("Run");

}

public static void main(String a[]){

Thread t1 = new Thread (new MyExmpCode ());

t1.start ();

t1.start ();

}

}

No, it throws IllegalThreadStateException

# Give the list of Java Object class methods.

**clone() -** Creates and returns a copy of this object.

**equals() -** Indicates whether some other object is "equal to" this one.

**finalize() -** Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.

**getClass() -** Returns the runtime class of an object.

**hashCode() -** Returns a hash code value for the object.

**notify() -** Wakes up a single thread that is waiting on this object's monitor.

**notifyAll() -** Wakes up all threads that are waiting on this object's monitor.

**toString() -** Returns a string representation of the object. wait() - Causes current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object.

# Can we call servlet destory() from service()?

destory() it is used to kill the servlet instance. In case, if you call destroy method from service(), it just execute the code written in the destory(), but it wont kill the servlet instance. destroy() will be called before killing the servlet instance by servlet engine.

# Can we override static method?

We cannot override static methods. Static methods are belogs to class, not belongs to object.

# Can you list serialization methods?

Serialization interface does not have any methods. It is a marker interface. It just tells that your class can be serializable.

# What is the difference between super() and this()?

super() is used to call super class constructor, whereas this() used to call constructors in the same class, means to call parameterized constructors.

# How to prevent a method from being overridden?

By specifying final keyword to the method you can avoid overriding in a subcalss. Similarlly one can use final at class level to prevent creating subclasses.

# How to destroy the session in servlets?

By calling invalidate() method on session object, we can destory the session.

# What is transient variable?

Transient variables cannot be serialized. During serialization process, transient variable states will not be serialized. State of the value will be always defaulted after deserialization.

# What is daemon thread?

Daemon thread is a low priority thread. It runs intermittently in the back ground, and takes care of the garbage collection operation for the java runtime system. By calling setDaemon() method is used to create a daemon thread.

# Does each thread in java uses seperate stack?

In Java every thread maintains its own separate stack. It is called Runtime Stack but they share the same memory.

# Find out below switch statement output.

public static void main(String a[]){

int price = 6;

switch (price) {

case 2: System.out.println("It is: 2");

default: System.out.println("It is: default");

case 5: System.out.println("It is: 5");

case 9: System.out.println("It is: 9");

}

}

|  |
| --- |
| It is: default It is: 5 It is: 9 |

# Does system.exit() in try block executes code in finally block?

try{

System.out.println("I am in try block");

System.exit(1);

} catch(Exception ex){

ex.printStackTrace();

} finally {

System.out.println("I am in finally block!!!");

}

|  |
| --- |
| **Answer:** |
| It will not execute finally block. The program will be terminated after System.exit() statement. |

# What is fail-fast in java?

A fail-fast system is nothing but immediately report any failure that is likely to lead to failure. When a problem occurs, a fail-fast system fails immediately. In Java, we can find this behavior with iterators. Incase, you have called iterator on a collection object, and another thread tries to modify the collection object, then concurrent modification exception will be thrown. This is called fail-fast.

# What is final, finally and finalize?

final:

final is a keyword. The variable decleared as final should be initialized only once and cannot be changed. Java classes declared as final cannot be extended. Methods declared as final

cannot be overridden.

finally:

finally is a block. The finally block always executes when the try block exits. This ensures that the finally block is executed even if an unexpected exception occurs. But finally is useful for more than just exception handling - it allows the programmer to avoid having cleanup code accidentally bypassed by a return, continue, or break. Putting cleanup code in a finally block is always a good practice, even when no exceptions are anticipated.

finalize:

finalize is a method. Before an object is garbage collected, the runtime system calls its finalize() method. You can write system resources release code in finalize() method before getting garbage collected.

# What are the different session tracking methods?

Cookies:

You can use HTTP cookies to store information. Cookies will be stored at browser side.

URL rewriting:

With this method, the information is carried through url as request parameters. In general added parameter will be sessionid, userid.

HttpSession:

Using HttpSession, we can store information at server side. Http Session provides methods to handle session related information.

Hidden form fields:

By using hidden form fields we can insert information in the webpages and these information will be sent to the server. These fields are not visible directly to the user, but can be viewed using view source option from the browsers. The hidden form fields are as given below:

<input type='hidden' name='siteName' value='java2novice'/>

# What is the purpose of garbage collection?

The garbage collection process is to identify the objects which are no longer referenced or needed by a program so that their resources can be reclaimed and reused. These identified objects will be discarded

# Difference between ODBC and JDBC

|  |  |
| --- | --- |
| **ODBC** | **JDBC** |
| ODBC Stands for Open Database Connectivity. | JDBC Stands for java database connectivity. |
| Introduced by Microsoft in 1992. | Introduced by SUN Micro Systems in 1997. |
| We can use ODBC for any language like C,C++,Java etc. | We can use JDBC only for Java languages. |
| We can choose ODBC only windows platform. | We can Use JDBC in any platform. |
| Mostly ODBC Driver developed in native languages like C,C++. | JDBC Stands for java database connectivity. |
| For Java applications it is not recommended to use ODBC because performance will be down due to internal conversion and applications will become platform Dependent. | For Java application it is highly recommended to use JDBC because there we no performance & platform dependent problem. |
| ODBC is procedural. | JDBC is object oriented. |

# Different name reusing techniques in Java

**Overriding** An instance method overrides all accessible instance methods with the same signature in superclasses, enabling dynamic dispatch; in other words, the VM chooses which overriding to invoke based on an instance’s run-time type. Overriding is fundamental to object-oriented programming and is the only form of name reuse that is not generally discouraged:

|  |
| --- |
| **// Base Class**  class Base {  public void f() { }  }  **// Derived Class**  class Derived extends Base {   // overrrides Base.f()  public void f() { }  } |

**Hiding** A field, static method, or member type hides all accessible fields, static methods, or member types, respectively, with the same name (or, for methods, signature) in supertypes. Hiding a member prevents it from being inherited :

|  |
| --- |
| **// Base class**  class Base {     public static void f() { }  }    **// Derived class**  class Derived extends Base {       // hides Base.f()     public static void f() { }  } |

**Overloading** Methods in a class overload one another if they have the same name and different signatures. The overloaded method designated by an invocation is selected at compile time:

|  |
| --- |
| // Base class  class GeeksForGeeks {       // int overloading     public void f(int i) { }       // String overloading     public void f(String s) { }  } |

**Shadowing** A variable, method, or type shadows all variables, methods, or types, respectively, with the same name in a textually enclosing scope. If an entity is shadowed, you cannot refer to it by its simple name; depending on the entity, you cannot refer to it at all:

|  |
| --- |
| // Base class  class GeeksForGeeks {    static String sentence = "I don’t know.";      public static void main(String[] args) {         // shadows static field       String sentence = "I know!";         // prints local variable       System.out.println(sentence);    }  } |

**Obscuring** A variable obscures a type with the same name if both are in scope: If the name is used where variables and types are permitted, it refers to the variable. Similarly, a variable or a type can obscure a package. Obscuring is the only kind of name reuse where the two names are in different namespaces: variables, packages, methods, or types. If a type or a package is obscured, you cannot refer to it by its simple name except in a context where the syntax allows only a name from its namespace. Adhering to the naming conventions largely eliminates obscuring

|  |
| --- |
| public class Obscure {      // Obscures type java.lang.System    static String System;      public static void main(String[] args) {         // Next line won’t compile:       // System refers to static field       System.out.println("hello, obscure world!");    }  } |

Checking Internet Connectivity using Java

Checking Internet connectivity using Java can be done using 2 methods:  
1) by using getRuntime() method of java Runtime class.  
2) by using methods of java URL and URLConnection classes.

**#Java Runtime Class:**This class is used to interact with java runtime environment (Java virtual machine) in which the application is running. It provides methods/functions to execute a process or a command, invoke garbage collector, get total and free memory in the JVM etc.

**#getRuntime():**This method of java runtime class returns the runtime object associated with the current java application.  
you can learn more about this class [here](https://www.geeksforgeeks.org/java-lang-runtime-class-in-java/)

**#Java URL Class:**This class provides methods that returns various information like protocol, hostname, file Name, port Number etc of the URL.

**#Java URLConnection class:**It represents a link between URL and application and can be used to read and write data to the specified resource referred by the URL.

**#openConnection():**This method of java URLConnection class opens the connection to the specified URL.  
you can learn more about these classes [here](https://www.geeksforgeeks.org/url-class-java-examples/) and [here](https://www.geeksforgeeks.org/reading-url-using-urlconnection-class/)

**NOTE:** Provided method should be run on a local machine and not on an online compiler.  
**METHOD 1:**  
Output by this method will be 0 if the internet is connected and it will be 1 if the internet is not connected.

|  |
| --- |
| // Java program for Checking Internet connectivity  import java.util.\*;  import java.io.\*;    class checking\_internet\_connectivity {      public static void main(String args[]) throws Exception      {          Process process = java.lang.Runtime.getRuntime().exec("ping www.geeksforgeeks.org");          int x = process.waitFor();          if (x == 0) {              System.out.println("Connection Successful, "                                 + "Output was " + x);          }          else {              System.out.println("Internet Not Connected, "                                 + "Output was " + x);          }      }  } |

**METHOD 2:**  
If Internet is not connected it will throw an exception and catch will execute printing respective message.

|  |
| --- |
| // Java program for checking Internet connectivity  import java.util.\*;  import java.io.\*;  import java.net.URL;  import java.net.URLConnection;    class checking\_internet\_connectivity {      public static void main(String args[])      {          try {              URL url = new URL("[https://](NULL) www.geeksforgeeks.org/");              URLConnection connection = url.openConnection();              connection.connect();                System.out.println("Connection Successful");          }          catch (Exception e) {              System.out.println("Internet Not Connected");          }      }  } |

# Object Pool Design Pattern

Object pool pattern is a software creational design pattern which is used in situations where the cost of initializing a class instance is very high.  
Basically, an Object pool is a container which contains some amount of objects. So, when an object is taken from the pool, it is not available in the pool until it is put back.  
Objects in the pool have a lifecycle:

* Creation
* Validation
* Destroy.
* **Client :** This is the class that uses an object of the PooledObject type.
* **ReuseablePool:** The PooledObject class is the type that is expensive or slow to instantiate, or that has limited availability, so is to be held in the object pool.
* **ObjectPool :**The Pool class is the most important class in the object pool design pattern. ObjectPool maintains a list of available objects and a collection of objects that have already been requested from the pool.

Let’s take the example of the database connections. It’s obviously that opening too many connections might affect the performance for several reasons:

* Creating a connection is an expensive operation.
* When there are too many connections opened it takes longer to create a new one and the database server will become overloaded.

Here the object pool manages the connections and provide a way to reuse and share them. It can also limit the maximum number of objects that can be created.

|  |
| --- |
| // Jave program to illustrate  // Object Pool Design Pattern  abstract class ObjectPool<T> {      long deadTime;        Hashtable<T, Long> lock, unlock;        ObjectPool()      {          deadTime = 50000; // 50 seconds          lock = new Hashtable<T, Long>();          unlock = new Hashtable<T, Long>();      }        abstract T create();        abstract boolean validate(T o);        abstract void dead(T o);        synchronized T takeOut()      {          long now = System.currentTimeMillis();          T t;          if (unlock.size() > 0) {              Enumeration<T> e = unlock.keys();              while (e.hasMoreElements()) {                  t = e.nextElement();                  if ((now - unlock.get(t)) > deadTime) {                      // object has deadd                      unlock.remove(t);                      dead(t);                      t = null;                  }                  else {                      if (validate(t)) {                          unlock.remove(t);                          lock.put(t, now);                          return (t);                      }                      else {                          // object failed validation                          unlock.remove(t);                          dead(t);                          t = null;                      }                  }              }          }          // no objects available, create a new one          t = create();          lock.put(t, now);          return (t);      }      synchronized void takeIn(T t)      {          lock.remove(t);          unlock.put(t, System.currentTimeMillis());      }  }    // Three methods are abstract  // and therefore must be implemented by the subclass    class JDBCConnectionPool extends ObjectPool<Connection> {      String dsn, usr, pwd;        JDBCConnectionPool(String driver, String dsn, String usr, String pwd)      {          super();          try {              Class.forName(driver).newInstance();          }          catch (Exception e) {              e.printStackTrace();          }          this.dsn = dsn;          this.usr = usr;          this.pwd = pwd;      }        Connection create()      {          try {              return (DriverManager.getConnection(dsn, usr, pwd));          }          catch (SQLException e) {              e.printStackTrace();              return (null);          }      }        void dead(Connection o)      {          try {              ((Connection)o).close();          }          catch (SQLException e) {              e.printStackTrace();          }      }        boolean validate(Connection o)      {          try {              return (!((Connection)o).isClosed());          }          catch (SQLException e) {              e.printStackTrace();              return (false);          }      }  }    class Main {      public static void main(String args[])      {          // Create the ConnectionPool:          JDBCConnectionPool pool = new JDBCConnectionPool(              "org.hsqldb.jdbcDriver", "jdbc:[hsqldb://](NULL) localhost/mydb",              "sa", "password");            // Get a connection:          Connection con = pool.takeOut();          // Return the connection:          pool.takeIn(con);      }  } |

**Advantages**

* It offer a significant performance boost.
* It manages the connections and provides a way to reuse and share them.
* Object pool pattern is used when the rate of initializing a instance of the class is high.

**When to use Object Pool Design Pattern**

* When we have a work to allocates or deallocates many objects
* Also, when we know that we have a limited number of objects that will be in memory at the same time.