1. WhatareWrapperclassesinJava?

Java has concept of Wrapper classes to allow primitive types to be accessed as objects. Primitive types like boolean, int, double, float etc. have corresponding Wrappers classes – Boolean, Integer, Double, Float etc.

Many of these Wrapper classes are in java.lang package.

Java 5.0 has launched the concept of Autoboxing and Unboxing in Java for Wrapper classes.

E.g.

```
public class WrapperTest{
public static void main(String args[]){
//Converting int into Integer
int count=50;
Integer i=Integer.valueOf(count);//converting int into Integer
Integer j=a;//autoboxing, now compiler will write Integer.valueOf(count) internally

System.out.println(count+" "+i+" "+j); }}
```

2. What is the purpose of native method in Java?

The native keyword is used for applying to a method to indicate that the method is implemented in native code using JNI(Java Native Interface).

Therefore, native methods allow Java Developer to directly access platform specific APIs.

Often, native methods are linked to native library.

3. What is System class?

System.class is a final class provided by java.lang package. It contains several useful class fields and methods.

The purpose of System class is to provide access to system resources.

4. What is System, out and println in System.out.println method call?

System is a final class provided by java.lang package.
out refers to PrintStream class and a static member of System class.

println is a method of PrintStream class.

5. What is the other name of Shallow Copy in Java?

Object Cloning. A Shallow Copy just copies the values of references in a Class.

6. What is the difference between Shallow Copy and Deep Copy in Java?

A Shallow copy just copies the values of the references in the class. A Deep copy copies the values of the objects as well.

7. What is a Singleton class?

A Singleton class in Java has maximum one instance of the class present in JVM, all the time. The constructor of this class is written in such a way that it never creates more than one object of same class.

8. What is the difference between Singleton class and Static class?

A static class in Java has only static methods. It is a container of functions. It is created based on procedural programming design.

Singleton class is a pattern in Object Oriented Design. A Singleton class has only one instance of an object in JVM. This pattern is implemented in such a way that there is always only one instance of that class present in JVM.

JSP

9. What are the implicit objects in JSP?

JSP has following implicit objects:

1. Request
2. Response
3. Application
4. Exception
5. Page
6. Config
7. Session

10. How will you extend JSP code?

We can extend JSP code by using Tag libraries and Custom actions.

11. How will you handle runtime exceptions in JSP?

We use Errorpage attribute in JSP to catch runtime exceptions. This attribute forwards user request to the error page automatically.

12.How will you prevent multiple submits of a page that come by clicking refresh button multiple times?

We can use Post Redirect Get (PRG) pattern to solve the issue of multiple submission of same data. It works as follows:

First time when a user submits a form to server by POST or GET method, then we update the state in application database.

Then we send a redirect response to send reply to client.

Then we load a view by using GET command. There is no data is sent in this. Since this a new JSP page, it is safe from multiple submits. The code that processes the request is idempotent. So it does not do same action twice for same request.

13.How will you implement a thread safe JSP page?

We can use SingleThreadModel Interface to implement a thread safe JSP page.

We can also add <%@page isThreadSafe=”false” %> directive in JSP page to make it thread safe.

14.How will you include a static file in a JSP page?

We can use include directive of JSP to include a Static page in JSP. In this approach, we use translation phase to include a static page. We have to specify the URL of the resource to be included as file attribute in this directive.

E.g. <%@ include file="footer.html" %>

15.What are the lifecycle methods of a JSP?

A JSP has following lifecycle methods:

1. **jspInit**(): This method is invoked when the JSP is called for the first time. We can do initial setup for servicing a request in this method.

2. **_jspService**(): This method is used to serve every request of the JSP.

3. **jspDestroy**(): Once we remove a JSP from the container, we call this method. It is used for cleanup of resources like Database connections etc.

16.What are the advantages of using JSP in web architecture?

We get following advantages by using JSP in web architecture:

1. **Performance**: JSP provides very good performance due to their design of using same code to service multiple requests.

2. **Fast**: Since JSP is pre-compiled, server can serve the pages very fast.

3. **Extendable**: JSP is based on Java Servlets. This helps in extending JSP architecture with other Java technologies like JDBC, JMS, JNDI etc.

4. **Design**: It is easier to design user interface with JSP, since it is very close to HTML. UI designers can create a JSP with mock data and developers can later provide implementation of dynamic data.

17. What is the advantage of JSP over Javascript?

In JSP we can write Java code seamlessly. It allows for writing code that can interact with the rest of the application.

Javascript code is mostly executed at client side. This limits the tasks that can be done in Javascript code. We cannot connect to database server from Javascript at the client side.

18.What is the Lifecycle of JSP?

JSP has following lifecycle stages:

1. **Compilation**: When a request is made for a JSP, the corresponding JSP is converted into Servlet and compiled. If there is already a compiled form of JSP and there is not change in JSP page since last compilation, this stage does not do anything.

2. **Initialization**: In this stage, jspInit() method is called to initialize any data or code that will be later used multiple times in _jspService() method.

3. **Se rvice** : In this stage, with each request to JSP, _jspService() method is called to service the request. This is the core logic of JSP that generates response for request.

4. **De stroy**: In this stage, JSP is removed from the container/server. Just before removal, this stage performs the cleanup of any resources held by JSP.

19. What is a JSP expression?

A JSP expression is an element of a JSP page that is used to evaluate a Java expression and convert into a String. This String is replaced into the locations wherever the expression occurs in JSP page.

E.g. <%= expression =%>

20. What are the different types of directive tags in JSP?

JSP has following directive tags:

1. **Page**: This directive is used for page related attributes. It can be put anywhere in the JSP page. But by convention we put it on the top of the page.
   E.g.
   <%@ page attribute="value" %>

2. **Taglib**: We can create custom tags in JSP and use these by taglib directive in a JSP page.
   E.g.
   <%@ taglib uri="abc.html" prefix="tag_prefix" >

3. **Include**: We use include directive to read a file and merge its content with the JSP page. This is done during compilation stage.
   <%@ include file="relative url" >

21.What is session attribute in JSP?

Session attribute in JSP is used for HTTP session mechanism. If we do not want to use HTTP session in JSP, then we set this attribute to false. If it is set to true, we can use built in session object in JSP.

22. What are the different scopes of a JSP object?

A JSP object, implicit or explicit, can have one of the following scopes:

1. **Page**: In this scope, the object is accessible from the page where it was created. Important point here is that when a user refreshes the page, the objects of this scope also get created again.

2. **Request**: In request scope, the object is accessible to the HTTP request that created this object.

3. **Session**: In this scope, the object is available throughout the same HTTP session.

4. **Application**: This is the widest scope. The object is available throughout the application in which JSP was created.

23. What is pageContext in JSP?

In JSP, pageContext is an implicit object. This is used for storing and accessing all the page scope objects of JSP.

It is an instance of the PageContext class from javax.servlet.jsp package.

24. What is the use of jsp:useBean in JSP?

We use jsp:useBean to invoke the methods of a Java Bean class. The Java Bean class has some data and setter/getters to access the data.

With this tag, container will try to locate the bean. If bean is not already loaded then it will create an instance of a bean and load it. Later this bean can be used in expressions or JSP code.

25. What is difference between include Directive and include Action of JSP?

Some of the main differences between include Directive and include Action are as follows:

1. Include directive is called at translation phase to include content in JSP. Include Action is executed during runtime of JSP.

2. It is not possible to pass parameters to include directive. Include action can accept parameters by jsp:param tag.

3. Include directive is just copying of content from another file to JSP code and then it goes through compilation. Include action will dynamically process the resource being called and

then include it in the JSP page.

26. How will you use other Java files of your application in JSP code?

We can use import tag to import a Java file in JSP code. Once a file is imported, it can be used by JSP code. It is a very convenient method to use Java classes in JSP code.

For better organization of Java code, we should create a package of classes that we are planning to use in JSP code.

27. How will you use an existing class and extend it to use in the JSP?

We can use extends attribute in include tag to use an existing class and extend it in the current JSP.

E.g.
<%@ include page extends="parent_class" %>

28. Why _jspService method starts with _ symbol in JSP?

All the code that we write in a JSP goes into _jspService method during translation phase. We cannot override this method. Where as other lifecycle methods jspInit() and jspDestroy() can be overridden.

It appears that container uses _ symbol to distinguish the method that cannot be overridden by client code.

29. Why do we use tag library in JSP?

At times we want to create a UI framework with custom tags. In such a scenario, taglib is a very good feature of JSP. With taglib we can create tags that can provide custom features.

Taglib is also a nice way to communicate with UI designers who can use custom tags in the html without going into the details of how the code is implemented.

Another benefit of taglib is reusability of the code. This promotes writing code only once and using is multiple times.

30. What is the different type of tag library groups in JSTL?

JSTL stands for JavaServer Pages Standard Tag Library. In JSTL, we have a collection of JSP tags that can be used in different scenarios. There are following main groups of tags in JSTL:

1. Core tags

2. SQL tags

3. Formatting tags

4. XML tags

5. JSTL Functions

31.How will you pass information from one JSP to another JSP?

We can pass information from one JSP to another by using implicit objects. If different JSP are called in same session, we can use session object to pass information from one JSP to another.

If we want to pass information from one JSP to another JSP included in the main JSP, then we can use jsp:param to pass this information.

32. How will you call a stored procedure from JSP?

JSP allows running Java code from a .jsp file. We can call a stored procedure by using JDBC code.

We can call a CallableStatement from JSP code to invoke a stored procedure.

If we are using Spring framework, then we can use JdbcTemplate class to invoke stored procedure from a JSP.

33. Can we override _jspService() method in JSP?

No, JSP specification does not allow overriding of _jspService method in JSP. We can override other methods like jspInit() and jspDestroy().

34. What is a directive in JSP?

JSP directive is a mechanism to pass message to JSP container. JSP directive does not produce an output to the page. But it communicates with JSP container.

E.g. <%@include ..%> directive is used for telling JSP container to include the content of another file during translation of JSP.

There can be zero or more attributes in a directive to pass additional information to JSP container.

Some of the important directives in JSP are: page, include and taglib.

35. How will you implement Session tracking in JSP?

We can use different mechanisms to implement Session tracking JSP. Some these mechanisms are as follows:

1. **Cookies**: We can use cookie to set session information and pass it to web client. In subsequent requests we can use the information in cookie to track session.

2. **Hidden Form Field**: We can send session id in a hidden field in HTML form. By using this we can track session.

3. **Session object**: We can use the built in session object to track session in JSP.

4. **URL Rewriting**: We can also add session id at the end of a URL.
   Like- www.abcserver.com?sessionid=1234

36. How do you debug code in JSP?

In simplest form we can write logger statements or System.out.println() statements to write messages to log files. When we call a JSP, the log messages get written to logs. With useful information getting logged we can easily debug the code.

Another option in debugging is to link JSP container with an IDE. Once we link IDE debugger to JSP Engine, we can use standard operations of debugging like breakpoint, step through etc.

37. How will you implement error page in JSP?

To implement an error-handling page in JSP, we first create a JSP with error page handling information. In most of the cases we gracefully handle error by giving a user-friendly message like "Sorry! There is system error. Please try again by refreshing page."

In this error page, we show user-friendly message to user, but we also log important information like stack trace to our application log file.

We have to add parameter isErrorPage=true in page directive of this page. This tells to JSP container that this is our error page.

<%@page isErrorPage="true" %>

Now we can use this error page in other JSP where we want to handle error. In case of an error or exception, these JSP will direct it to errorPage.

<% page errorPage="ErrorPage.jsp" %>

38. How will you send XML data from a JSP?

In general, JSP is used to pass HTML data to web browser. If we want to send data in XML format, we can easily do it by setting contentType=”text/xml” inpagedirective.

E.g. <%@page contentType=”text/xml” %>

39. What happens when we request for a JSP page from web browser?

When a user calls JSP page from web browser, the request first comes to web server. Web server checks for .jsp extension of page and passes the request to JSP container like Tomcat.

The JSP container checks whether it has precompiled JSP class or not. If this is the first time this JSP is called, then JSP container will translate JSP into a servlet and compiles it.

After compiling, JSP code if loaded in memory and JSP container will call jspInit() method and _jsp-Service() methods.

The _jspService() method will create the output that will be sent by JSP container to client browser.

40. How will you implement Auto Refresh of page in JSP?

We can use setIntHeader() method to set the refresh frequency with which we want to auto-refresh a JSP page.

We can send key “Refresh” with the time in seconds for auto refresh of the JSP page.

E.g. response.setIntHeader(“Refresh”,10)

41.What are the important status codes in HTTP?

Every HTTP request comes back with a status code from the server. The important status codes in HTTP are as follows:

1. 200: It means the request is successful.

2. 400: It means the request was bad.

3. 401: It means request was not authorized.

4. 404: It means the resource requested was not found.

5. 503: It means the service is not available.

42. What is the meaning of Accept attribute in HTTP header?

In HTTP header, Accept attribute is used to specify the MIME types that a HTTP client or browser can handle. MIME type is the identifier for specifying the type of file/data that we are planning to pass over the internet.

43. What is the difference between Expression and Scriptlet in JSP?

We use Expression in a JSP to return a value and display it at a specific location. It is generally used for dynamically print information like- time, counter etc in a HTML code.

Scriptlet is for writing Java code in a JSP. We can define variable, methods etc in a Scriptlet. A Scriptlet can handle much more complex code and can be also reused.

44. How will you delete a Cookie in JSP?

We can use following options to delete a Cookie in JSP:

1. **setMaxAge**(): we can set the maximum age of a cookie. After this time period, Cookie will expire and will be deleted.

2. **Header**: We can also set the expiry time in header of response. Respone.setHeader(). This will also expire the cookie after specified time period.

45. How will you use a Cookie in JSP?

We can use a Cookie in JSP by performing following steps:

First we create a Cookie object. We set the name and value of the cookie to be created.

We set the expiry time of the Cookie by setting the maximum age. We can use setMaxAge() method for this.

Finally, we can send the cookie in a HTTP Response by sending it in HTTP header. In this way cookie goes to client browser and gets stored there till the maximum age is not achieved.

Once a Cookie is set in the client browser, we can call getCookies() method to get the list of all the cookies set in Client. We iterate through the list of all the cookies and get the value of the cookie that was set in earlier request.

In this way we can use Cookie to set some information at client side and retrieve its value.

46. What is the main difference between a Session and Cookie in JSP?

A Session is always stored at the Server side. In JSP, session is a built-in object in JSP container.

A Cookie is always stored at the client side.

We can use both the methods for Session tracking. But Cookie method needs permission from user for storing cookie at the client location.

47. How will you prevent creation of session in JSP?

We can simply set the session attribute as false in page directive to prevent creation of session object.

E.g. <% @page session="false" %>

48. What is an output comment in JSP?

We can write output in JSP in such a way that it becomes a comment in HTML code. This comment will not be visible in the web browser. But when we view page source to see HTML, we can see output comment.

An HTML comment is of following format:
<!-- comment -->
If we output comment in above format, it will be visible to client.

49. How will you prevent caching of HTML output by web browser in JSP?

We can use set the header in response object for Cache-Control to specify no caching.

Sample code is as follows:

response.setHeader("Cache-Control", "no-store"); response.setDateHeader("Expires","0");

50. How will you redirect request to another page in browser in JSP code?

We can use sendRedirect() method in JSP to redirect the request to another location or page.

In this case the request will not come back to server. It will redirect in the browser itself.

Sample code is as follows:
<% response.sendRedirect(URL); %>

51.What is the difference between sendRedirect and forward in a JSP?

Both forward and sendRedirect are mechanisms of sending a client to another page. The main difference between these two are as follows:

1. In forward, the processing takes place at server side. In case of sendRedirect() the processing takes place the client side.

2. In forward, the request is transferred to another resource within same server. In case of send-Redirect the request can be transferred to resource on some other server.

3. In forward only one request call is consumed. In case of sendRedirect two request response calls are created and consumed.

4. The forward is declared in RequestDispatcher interface. Where as sendRedirect is declared in HttpServletResponse object.

52. What is the use of config implicit object in JSP?

In JSP, config object is of type ServletConfig. This object is created by Servlet Container for each JSP page. It is used for setting initialization parameters for a specific JSP page.

53. What is the difference between init-param and context-param?

We can specify both init-param and context-param in web.xml file.

We use init-param to specify the parameters that are specific to a servlet or jsp. This information is confined to the scope of that JSP.

We use context-param to specify the parameters for overall application scope. This information does not change easily. It can be used by all the JSP/Servlet in that Container.

54. What is the purpose of RequestDispatcher?

We use RequestDispatcher interface to forward requests to other resources like HTML, JSP etc.

It can also be used to include the content of another page in a JSP. It has two methods: forward and include.

We have to first get the RequestDispatcher object from the container and then we can call include or forward method on this object.

55. How can be read data from a Form in a JSP?

There is a built-in request object in a JSP that provides methods to read Form data. Some of the methods are as follows::

1. **getParameterNames():** This method returns the list of all the parameters in the Form.

2. **getParameter():** We call this method to get the value of parameter set in the Form. It returns null if the parameter is not found.

3. **ge tParame te rValue s():** If a Parameter is mentioned

multiple times in a Form, we use request.getParameterValues() method to get all the values. This method returns an array of String values.

4. **getParameterMap():** This method returns the map of all the Parameters in Form.

56. What is a filter in JSP?

We can define filters in JSP to intercept requests from a client or to change response from a server.

Filter is a Java class that is defined in the deployment descriptor of web.xml of an application. The JSP container reads filter from web.xml and applies a filter as per the URL pattern associated with the filter.

JSP Engine loads all the filters in when we start the server.

57. How can you upload a large file in JSP?

To upload a file by JSP we can use <input type="file"> in the Form data being passed from HTML.

If the file is very large in size, we can set enctype=multipart/form- data.

We have to use POST method in the Form to send a file.

Once the request is received, we can implement the logic to read mulitpart data in doPost() method of JSP. There are methods in JSP framework to read large files via this method.

58. In which scenario, Container initializes multiple JSP/Servlet objects?

To initialize multiple JSP objects, we have to specify same Servlet object multiple times in web.xml.

This indicates to JSP container to initialize separate JSP/Servlet object for each element. Each of the Servlet instance will have its own ServletConfig object and parameters.

Java Design Patterns

59. When will you use Strategy Design Pattern in Java?

Strategy pattern is very useful for implementing a family of algorithms. It is a behavioral design pattern.

With Strategy pattern we can select the algorithm at runtime. We can use it to select the sorting strategy for data. We can use it to save files in different formats like- .txt, .csv, .jpg etc.

In Strategy pattern we create an abstraction, which is an interface through which clients interact with our system. Behind the abstraction we create multiple implementation of same interface with different algorithms.

For a client, at runtime we can vary the algorithm based on the type of request we have received.

So we use Strategy pattern to hide the algorithm implementation details from client.

In Java Collections.sort() method uses strategy design pattern.

60. What is Observer design pattern?

In Observer design pattern, there is a Subject that maintains the list of Observers that are waiting for any update on the Subject. Once there is an update in Subject it notifies all the observers for the change.

E.g. In real life, students are waiting for the result of their test. Here students are the observers and test is the subject. Once the result of test is known, testing organization notifies all the students about their result.

The most popular use of Observer pattern is in Model View Controller (MVC) architectural pattern.

Main issue with Observer pattern is that it can cause memory leaks. The subject holds a strong reference to observers. If observers are not de-registered in time, it can lead to memory leak.

61.What are the examples of Observer design pattern in JDK?

In JDK there are many places where Observer design pattern is used. Some of these are as follows:

1. java.util.Observer, java.util.Observable

2. javax.servlet.http.HttpSessionAttributeListener

3. javax.servlet.http.HttpSessionBindingListener

4. All implementations of java.util.EventListener, and also in Swing packages

5. javax.faces.event.PhaseListener

62. How Strategy design pattern is different from State design pattern in Java?

State design pattern is a behavioral design pattern that is use for defining the state machine for an object. Each state of an object is defined in a child class of State class. When different actions are taken on an Object, it can change its state.

Strategy pattern is also a behavioral pattern, but it is mainly used for defining multiple algorithms. With same action of a client, the algorithm to be used can change.

Some people consider State pattern similar to Strategy pattern, since an Object changes its Strategy with different method invocations. But the main difference is that in State pattern internal state of an Object is one of the determining factors for selecting the Strategy for change of state.

Where as in Strategy pattern, client can pass some external parameter in input during method invocation that determines the strategy to be used at run time.

Therefore State pattern is based on the Object's internal state, where as Strategy pattern is based on Client's invocation.

State pattern is very useful in increasing the maintainability of the code in a large code-base.

63. Can you explain Decorator design pattern with an example in Java?

Some people call Decorator pattern as Wrapper pattern as well. It is used to add the behavior to an object, without changing the behavior of other objects of same class.

One of the very good uses of Decorator pattern is in java.io package. We can have a FileInputStream to handle a File. To add Buffering behavior we can decorate FileInputStream with BufferedInputStream. To add the gzip behavior BufferedInputStream we can decorate it with GzipInputStream. To add serialization behavior to GzipInputStream, we can decorate it with ObjectInputStream.

E.g.
Open a FileInputStream:

FileInputStream fis = new FileInputStream("/myfile.gz"); Add buffering:
BufferedInputStream bis = new BufferedInputStream(fis); Add Gzip:

GzipInputStream gis = new GzipInputStream(bis); Add Serialization:
ObjectInputStream ois = new ObjectInputStream(gis);

So with each step we have decorated the FileInputStream with additional behavior.

64. What is a good scenario for using Composite design Pattern in Java?

Some of the good scenarios where Composite design pattern can be used are as follows:

Tree Structure: The most common use of Composite design pattern is Tree structure. If you want to represent data in a Tree data structure, Composite pattern can be used.

E.g. In an Organization, to a Manager has Employees. But Manager is also an Employee. If we start from CEO level, there is one big tree for the whole organization structure. Under that big tree there are many sub-trees. This can be easily represented with Composite design pattern.

Recursion: Another use of Composite design pattern is Recursion. If we have a Recursion based algorithm, we need data to be passed to algorithm in a data structure that treats individual objects and compositions at each level of recursion uniformly.

E.g. To implement a recursive Polynomial Solving algorithm, we can use Composite design pattern to store the intermediate results.

Graphics: Another good use of Composite design pattern is in Graphics. We can group shapes inside a composite and make higher-level groups of smaller groups of shapes to complete the graphics to be displayed on screen.

65. Have you used Singleton design pattern in your Java project?

Yes. Singleton is one of the most popular design patterns in enterprise level Java applications. Almost in every project we see some implementation of Singleton.

With Singleton pattern we can be sure that there is only one instance of a class at any time in the application.

This helps in storing properties that have to be used in the application in a unique location.

66. What are the main uses of Singleton design pattern in Java project?

Some of the main uses of Singleton design pattern in Java are as follows:

1. Runtime: In JDK, java.lang.Runtime is a singleton-based class. There is only one instance of Runtime in an application. This is the only class that interfaces with the environment/machine in which Java process is running.

2. Enum: In Java, enum construct is also based on Singleton pattern. Enum values can be accessed globally in same way by all classes.

3. Properties: In an application it makes sense to keep only one copy of the properties that all classes can access. This can be achieved by making properties class Singleton so that every class gets same copy of properties.

4. Spring: In Spring framework, all the beans are by default Singleton per container. So there is only one instance of bean in a Spring IoC container. But Spring also provides options to make

the scope of a bean prototype in a container.

67. Why java.lang.Runtime is a Singleton in Java?

In Java, java.lang.Runtime is implemented on Singleton design pattern.

Runtime is the class that acts as an interface with the environment in which Java process is running. Runtime contains methods that can interact with the environment.

Like- totalmemory() method gives the total memory in JVM. maxMemory() method gives the maximum memory that JVM can use.

There is an exit() method to exit the Java process. We do not want multiple objects in JVM to have exit() method.

Similarly there is gc() method that can run the Garbage Collector. With only one copy of gc() method, we can ensure that no other object can run the Garbage Collector when one instance of GC is already running.

Due to all these reasons there is only one copy of Runtime in Java. To ensure single copy of Runtime, it is implemented as a Singleton in Java.

68. What is the way to implement a thread-safe Singleton design pattern in Java?

In Java there are many options to implement a thread-safe Singleton pattern. Some of these are as follows:

1. Double Checked Locking: This is the most popular method to implement Singleton in Java. It is based on Lazy Initialization. In this we first check the criteria for locking before acquiring a lock to create an object. In Java we use it with volatile keyword.

Sample code:

```
class DoubleCheckSingleton {
private volatile HelloSingleton helloSingleton; // Use Volatile

public HelloSingleton getHelloSingleton() { HelloSingleton result = helloSingleton;
if (result == null) {

synchronized(this) { // Synchronize for thread safety result = helloSingleton;
if (result == null) {

result = new HelloSingleton(); helloSingleton = result;

} }

}
return result; }

}
```

2. Bill Pugh Singleton: We can also use the method by Bill Pugh for implementing Singleton in Java. In this we use an Inner Static class to create the Singleton instance.

Sample code:

```
public class SingletonBillPugh {

// Inner class that holds instance private static class InnerSingleton{

private static final SingletonBillPugh INSTANCE = new SingletonBillPugh();

}

// Private constructor
private SingletonBillPugh(){}

public static SingletonBillPugh getInstance(){ return InnerSingleton.INSTANCE;

} }
```

When first time SingletonBillPugh is loaded in memory, InnerSingleton is not loaded. Only when getInstance() method is called, InnerSingleton class is loaded and an Instance is created.

3. Enum: We can also use Java enum to create thread-safe implementation. Java enum values are accessible globally so these can be used as a Singleton.

Sample Code:

```
public enum SingletonEnum {

INSTANCE;

public static void doImplementation(){ .....

} }
```

69. What are the examples of Singleton design pattern in JDK?

In JDK there are many places where Singleton design pattern is used. Some of these are as follows:

1. java.lang.Runtime.getRuntime(): This method gives Runtime class that has only one instance in a JVM.

java.lang.System.getSecurityManager(): This method returns a SecurityManager for the current platform.

java.awt.Desktop.getDesktop()

70. What is Template Method design pattern in Java?

It is a behavioral design pattern. We can use it to create an outline for an algorithm or a complex operation. We first create the skeleton of a program. Then we delegate the steps of the operation to subclasses. The subclasses can redefine the inner implementation of each step.

E.g. While designing a Game in Java, we can implement it as an algorithm with Template Method pattern. Each step in the game can be deferred to subclasses responsible for handling that step.

Let say we implement Monopoly game in Java. We can create methods like initializeGame(), makeMove(), endGame() etc. Each of these methods can be handled in subclasses in an independent manner.

We can use same algorithm for Chess game with same set of abstract methods. The subclass for Chess game can provide the concrete implementation of methods like initializeGame(), makeMove(), endGame() etc.

Template Method pattern is very useful in providing customizable class to users. We can create the core class with a high level implementation. And our users can customize our core class in their custom subclasses.

71.What are the examples of Template method design pattern in JDK?

In JDK there are many places where Template method design pattern is used. Some of these are as follows:

1. In Java Abstract Collection classes like java.util.AbstractList, java.util.AbstractSet and java.util.AbstractMap implement a template for their corresponding Collection.

2. javax.servlet.http.HttpServlet: In the HttpServlet class all the doGet(), doPost() etc. methods send a HTTP 405 "Method Not Allowed" error to the response. This error response is like a Template that can be further customized for each of these methods.

3. In java.io package there are Stream and Writer classes like java.io.InputStream, java.io.OutputStream, java.io.Reader and java.io.Writer that provide non-abstract methods. These methods are implementation of Template method design pattern.

72. Can you tell some examples of Factory Method design pattern implementation in Java?

Factory Method pattern is a creational design pattern. A Factory is an object that is used to create more objects.

In general, a Factory object has methods that can be used to create a type of objects. Some people call it Factory Method design pattern as well.

Some of the examples of Factory Method pattern in JDK are:

Java.lang.Class.forName() java.net.URLStreamHandlerFactory.createURLStreamHandler(String) java.util.Calendar.getInstance()
java.util.ResourceBundle.getBundle() java.text.NumberFormat.getInstance() java.nio.charset.Charset.-forName()
java.util.EnumSet.of() javax.xml.bind.JAXBContext.createMarshaller()

73. What is the benefit we get by using static factory method to create object?

By using Static Factory Method we encapsulate the creation process of an object. We can use new() to create an Object from its constructor. Instead we use static method of a Factory to create the object. One main advantage of using Factory is that Factory can choose the correct implementation at runtime and create the right object. The caller of method can specify the desired behavior.

E.g. If we have a ShapeFactory with createShape(String type) method. Client can call ShapeFactory.createShape("Circle") to get a circular shape. ShapeFactory.createShape("Square") will return square shape. In this way, ShapeFactory knows how to create different shapes based on the input by caller.

Another use of Factory is in providing access to limited resources to a large set of users.

E.g. In ConnectionPool, we can limit the total number of connections that can be created as well as we can hide the implementation details of creating connection. Here ConnectionPool is the factory. Clients call static method ConnectionPool.getConnection().

74. What are the examples of Builder design pattern in JDK?

In JDK there are many places where Builder design pattern is used.

Some of 1.

2.

3.

4. 5.

these are as follows:

java.lang.StringBuilder.append(): StringBuilder is based on Builder pattern.

java.nio.IntBuffer.put(): Invocation of put() method return IntBuffer. Also there are many variants of this method to build the IntBuffer.

javax.swing.GroupLayout.Group.addComponent(): We can use addComponent() method to build a UI that can contain multiple levels of components.

java.lang.Appendable

java.lang.StringBuffer.append(): StringBuffer is similar to StringBuilder and it is also based on Builder design pattern.

75. What are the examples of Abstract Factory design pattern in JDK?

In JDK there are many places where Abstract Factory design pattern is used. Some of these are as follows:

javax.xml.xpath.XPathFactory.newInstance()
javax.xml.parsers.DocumentBuilderFactory.newInstance() javax.xml.transform.TransformerFactory.newInstance()

76. What are the examples of Decorator design pattern in JDK?

In JDK there are many places where Decorator design pattern is used. Some of these are as follows:

1. In java.io package many classes use Decorator pattern. Subclasses of java.io.InputStream, OutputStream, Reader and Writer have a constructor that can take the instance of same type and decorate it with additional behavior.

2. In java.util.Collections, there are methods like checkedCollection(), checkedList(), checkedMap(), synchronizedList(), synchronizedMap(), synchronizedSet(), unmodifiableSet(), unmodifiableMap() and unmodifiableList() methods that can decorate an object and return the same type.

3. In javax.servlet package, there are classes like javax.servlet.http.HttpServletRequestWrapper and HttpServletResponseWrapper that are based on Decorator design pattern.

77. What are the examples of Proxy design pattern in JDK?

Proxy design pattern provides an extra level of indirection for providing access to another object. It can also protect a real object from any extra level of complexity.

In JDK there are many places where Proxy design pattern is used. Some of these are as follows:

java.lang.reflect.Proxy
java.rmi.*

javax.inject.Inject
javax.ejb.EJB javax.persistence.PersistenceContext

78. What are the examples of Chain of Responsibility design pattern in JDK?

In JDK there are many places where Chain of Responsibility design pattern is used. Some of these are as follows:

1. java.util.logging.Logger.log(): In this case Logger class provides multiple variations of log() method that can take the responsibility of logging from client in different scenarios. The client has to just call the appropriate log() method and Logger will take care of these commands.

2. javax.servlet.Filter.doFilter(): In the Filter class, the Container calls the doFilter method when a request/response pair is passed through the chain. With filter the request reaches to the appropriate resource at the end of the chain. We can pass FilterChain in doFilter() method to allow the Filter to pass on the request and response to the next level in the chain.

79. What are the main uses of Command design pattern?

Command design pattern is a behavioral design pattern. We use it to encapsulate all the information required to trigger an event. Some of

the main

uses of Command pattern are:

1. Graphic User Interface (GUI): In GUI and menu items, we use command pattern. By clicking a button we can read the current information of GUI and take an action.

2. Macro Recording: If each of user action is implemented as a separate Command, we can record all the user actions in a Macro as a series of Commands. We can use this series to implement the “Playback” feature. In this way, Macro can keep on doing same set of actions with each replay.

3. Multi-step Undo: When each step is recorded as a Command, we can use it to implement Undo feature in which each step can by undo. It is used in text editors like MS-Word.

4. Networking: We can also send a complete Command over the network to a remote machine where all the actions encapsulated within a Command are executed.

5. Progress Bar: We can implement an installation routine as a series of Commands. Each Command provides the estimate time. When we execute the installation routine, with each command we can display the progress bar.

6. Wizard: In a wizard flow we can implement steps as Commands. Each step may have complex task that is just implemented within one command.

7. Transactions: In a transactional behavior code there are multiple tasks/updates. When all the tasks are done then only transaction is committed. Else we have to rollback the transaction. In such a scenario each step is implemented as separate Command.

80. What are the examples of Command design pattern in JDK?

In JDK there are many places where Command design pattern is used. Some of these are as follows:

All implementations of java.lang.Runnable All implementations of javax.swing.Action

81.What are the examples of Interpreter design pattern in JDK?

Interpreter design pattern is used to evaluate sentences in a language. E.g. In SQL we can use it to evaluate a query by evaluating each keyword like SELECT, FROM, WHERE clause.

In an Interpreter implementation there is a class for each keyword/symbol. A sentence is just a composite of these keywords. But the sentence is represented by Syntax tree that can be interpreted.

In JDK there are many places where Interpreter design pattern is used. Some of these are as follows:

java.util.Pattern java.text.Normalizer

Subclasses of java.text.Format: DateFormat, MessageFormat, NumberFormat

Subclasses of javax.el.ELResolver: ArrayELResolver, MapELResolver, CompositeELResolver etc.

82. What are the examples of Mediator design pattern in JDK?

By using Mediator pattern we can decouple the multiple objects that interact with each other. With a Mediator object we can create many-to-many relationships in multiple objects.

In JDK there are many places where Mediator design pattern is used. Some of these are as follows:

java.util.Timer: schedule() methods in Timer class act as Mediator between the clients and the Timer-Task to be scheduled.

java.util.concurrent.Executor.execute(): The execute() method in an Executor class acts as a Mediator to execute the different tasks.

java.util.concurrent.ExecutorService

java.lang.reflect.Method.invoke(): In Method class of reflection package, invoke() method acts as a Mediator.

java.util.concurrent.ScheduledExecutorService: Here also schedule() method and its variants are Mediator pattern implementations.

83. What are the examples of Strategy design pattern in JDK?

In JDK there are many places where Strategy design pattern is used. Some of these are as follows:

1. java.util.Comparator: In a Comparator we can use compare() method to change the strategy used by Collections.sort() method.

2. javax.servlet.http.HttpServlet: In a HttpServlet class service() and doGet(), doPost() etc. methods take HttpServletRequest and HttpServletResponse and the implementor of Servlet processes it based on the strategy it selects.

84. What are the examples of Visitor design pattern in JDK?

By using Visitor design pattern we can add new virtual methods to existing classes without modifying their core structure.

In JDK there are many places where Visitor design pattern is used. Some of these are as follows:

javax.lang.model.element.AnnotationValue and AnnotationValueVisitor

java.nio.file.FileVisitor and SimpleFileVisitor javax.lang.model.type.TypeMirror and TypeVisitor javax.lang.model.element.Element and ElementVisitor

javax.faces.component.visit.VisitContext and VisitCallback

85. How Decorator design pattern is different from Proxy pattern?

Main differences between Decorator and Proxy design pattern are:

Decorator provides an enhanced interface after decorating it with additional features. Proxy provides same interface since it is just acting as a proxy to another object.

Decorator is a type of Composite pattern with only one component. But each decorator can add additional features. Since it is one component in Decorator, there is no object aggregation.

Proxy can also provide performance improvement by lazy loading. There is nothing like this available in Decorator.

Decorator follows recursive composition. Proxy is just one object to another object access.

Decorator is mostly used for building a variety of objects. Proxy is mainly used for access to another object.

86. What are the different scenarios to use Setter and Constructor based injection in Dependency Injection (DI) design pattern?

We use Setter injection to provide optional dependencies of an object. Constructor injection is used to provide mandatory dependency of an object.

In Spring IoC, Dependency Injection is heavily used. There we have to differentiate between the scenario suitable for Setter based and Constructor based dependency injection.

87. What are the different scenarios for using Proxy design pattern?

Proxy design pattern can be used in a wide variety of scenario in Java. Some of these are as follows:

1. Virtual Proxy: This is a virtual object that acts as a proxy for objects that are very expensive to create. It is used in Lazy Loading. When client makes the first request, the real object is created.

2. Remote Proxy: This is a local object that provides access to a remote object. It is generally used in Remote Method Invocation (RMI) and Remote Procedure Call (RPC). It is also known as a Stub.

3. Protective Proxy: This is an object that control the access to a Master object. It can authenticate and authorize the client for accessing the Master object. If client has right permissions, it allows client to access the main object.

4. Smart Proxy: It is an object that can add additional information to the main object. It can track the number of other objects accessing the main object. It can track the different clients from where request is coming. It can even deny access to an object if the number of requests is greater than a threshold.

88. What is the main difference between Adapter and Proxy design pattern?

Adapter pattern provides a different interface to an object. But the Proxy always provides same interface to the object.

Adapter is like providing an interface suitable to client's use. But Proxy is same interface that has additional feature or check.

E.g. In electrical appliances we use Adapter to convert from one type of socket to another type of socket. In case of proxy, we have a plug with built-in surge protector. The interface for plug and the original device remains same.

89. When will you use Adapter design pattern in Java?

If we have two classes with incompatible interfaces, we use Adapter pattern to make it work. We create an Adapter object that can adapt the interface of one class to another class.

It is generally used for working with third party libraries. We create an Adapter class between third party code and our class. In case of any change in third party code we have to just change the Adapter code. Rest of our code can remain same and just take to Adapter.

90. What are the examples of Adapter design pattern in JDK?

In JDK there are many places where Adapter design pattern is used. Some of these are as follows:

java.util.Arrays.asList(): This method can adapt an Array to work as a List.

java.util.Collections.list(): This method can adapt any collection to provide List behavior.

java.util.Collections.enumeration(): This method returns an enumeration over the collection.

java.io.InputStreamReader(InputStream): This method adapts a Stream to Reader class.

java.io.OutputStreamWriter(OutputStream): This method adapts an OutputStream to Writer class.

javax.xml.bind.annotation.adapters.XmlAdapter.marshal()

91.What is the difference between Factory and Abstract Factory design pattern?

With Factory design pattern we can create concrete products of a type that Factory can manufacture. E.g. If it is CarFactory, we can produce, Ford, Toyota, Honda, Maserati etc.

With Abstract Factory design pattern we create a concrete implementation of a Factory. E.g. DeviceFactory can be Abstract and it can give us GoogleDeviceFactory, AppleDeviceFactory etc. With AppleDeviceFactory we will get products like- iPhone, iPad, Mac etc. With GoogleDeviceFactory we will get products like- Nexus phone, Google Nexus tablet, Google ChromeBook etc.

So it is a subtle difference between Factory and Abstract Factory design pattern. One way to remember is that within Abstract Factory pattern, Factory pattern is already implemented.

92. What is Open/closed design principle in Software engineering?

Open/closed design principle states “software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification”.

Open/closed principle term was originated by Bertrand Meyer in his book Object Oriented Software Construction.

As per this principle, if a module is available for extension then it is considered open. If a module is available for use by other modules then it is considered closed.

Further Robert C. Martin has mentioned it as O in SOLID principles of Object Oriented design.

It is used in State and Strategy design patterns. Context class is closed for modification. But new functionality can be added by writing new strategy code.

93. What is SOLID design principle?

SOLID word in SOLID design principle is an acronym for:

1. S: Single responsibility. A Class should have a single responsibility.

2. O: Open-closed. Software entities should be open for extension but closed for modification.

3. L: Liskov substitution. Objects in a program should be replaceable by subclasses of same type without any adverse impact.

4. I: Interface segregation. Multiple client specific interfaces are preferable over single generic interface.

5. D: Dependency inversion. Program should depend on abstract entities. It should not depend on concrete implementation of an interface.

This principle was mentioned by Robert C. Martin. These are considered five basic principles of Object Oriented design.

If we follow these principles, then we can create a stable program that is easy to maintain and can be extended over time.

94. What is Builder design pattern?

Builder design pattern is a creational design pattern. We can use Builder pattern to create complex objects with multiple options.

E.g. when we have to create a Meal in a restaurant we can use Builder pattern. We can keep adding options like- Starter, Drink, Main Course, and Dessert etc. to create complete meal. When a user selects other options of Starter, Drink Main Course, Dessert another type of meal is created.

Main feature of Builder pattern is step-by-step building of a complex object with multiple options.

95. What are the different categories of Design Patterns used in Object Oriented Design?

In Object Oriented design mainly three categories of design patterns are used. These categories are:

Creational Design Patterns: Builder
Factory Method
Abstract Factory

Object Pool Singleton Prototype

Structural Design Patterns: Adapter
Bridge
Façade

Decorator Composite Flyweight Proxy

Behavioral Design Patterns: Command
Iterator
Chain of Responsibility Observer

State Strategy Mediator Interpreter

96. What is the design pattern suitable to access elements of a Collection?

We can use Iterator design pattern to access the individual elements of a Collection. In case of an ordered collection we can get Iterator that returns the elements in an order.

In Java there are many implementation of Iterator in Collections package. We have iterators like- Spliterator, ListIterator etc. that implement Iterator pattern.

97. How can we implement Producer Consumer design pattern in Java?

We can use BlockingQueue in Java to implement Producer Consumer design pattern.

It is a concurrent design pattern.

98. What design pattern is suitable to add new features to an existing object?

We can use Decorator design pattern to add new features to an existing object. With a Decorator we work on same object and return the same object with more features. But the structure of the object remains same since all the decorated versions of object implement same interface.

99. Which design pattern can be used when to decouple abstraction from the implementation?

We can use Bridge design pattern to detach the implementation from the abstraction.

Bridge is mainly used for separation of concern in design. We can create an implementation and store it in the interface, which is an abstraction. Where as specific implementation of other features can be done in concrete classes that implement the interface.

Often Bridge design pattern is implemented by using Adapter pattern.

E.g. we have Shape interface. We want to make Square and Circle shapes. But further we want to make RedSquare, BlackSquare shapes and GreenCircle, WhiteCircle shapes. In this case rather than creating one hierarchy of all the shapes, we separate the Color concern from Shape hierarchy.

So we create two hierarchies. One is Shape to Square and Shape to Circle hierarchy. Another one is Color to Red, Black, Green, White hierarchy. In this way we can create multiple types of shapes with multiple colors with Bridge design pattern.

100.Which is the design pattern used in Android applications?

Android applications predominantly use Model View Presenter design pattern.

1. Model: This is the domain model of the Android application. It contains the business logic and business rules.

2. View: These are the UI components in your application. These are part of the view. Also any events on UI components are part of view module.

3. Presenter: This is the bridge between Model and View to control the communication. Presenter can query the model and return data to view to update it.

4. E.g. If we have a Model with large news article data, and view needs only headline, then presenter can query the data from model and only give headline to view. In this way view remains very light in this design pattern.

101. How can we prevent users from creating more than one instance of singleton object by using clone() method?

First we should not implement the Cloneable interface by the object that is a Singleton.

Second, if we have to implement Cloneable interface then we can throw exception in clone() method.

This will ensure that no one can use clone() method or Cloneable interface to create more than one instance of Singleton object.

102.What is the use of Interceptor design pattern?

Interceptor design pattern is used for intercepting a request. Primary use of this pattern is in Security policy implementation.

We can use this pattern to intercept the requests by a client to a resource. At the interception we can check for authentication and authorization of client for the resource being accessed.

In Java it is used in javax.servlet.Filter interface.

This pattern is also used in Spring framework in HandlerInterceptor and MVC interceptor.

103.What are the Architectural patterns that you have used?

Architectural patterns are used to define the architecture of a Software system. Some of the patterns are as follows:

1. MVC: Model View Controller. This pattern is extensively used in the architecture of Spring framework.

2. Publish-subscribe: This pattern is the basis of messaging architecture. In this case messages are published to a Topic. And subscribers subscribe to the topic of their interests. Once the message is published to a topic in which a Subscriber has an interest, the message is consumed by the relevant subscriber.

3. Service Locator: This design pattern is used in a service like JNDI to locate the available services. It uses as central registry to maintain the list of services.

4. n-Tier: This is a generic design pattern to divide the architecture in multiple tiers. E.g. there is 3-tier architecture with Presentation layer, Application layer and Data access layer. It is also called multi-layer design pattern.

5. Data Access Object (DAO): This pattern is used in providing access to database objects. The underlying principle is that we can change the underlying database system, without changing

the business logic. Since business logic talks to DAO object, there is no impact of changing Database system on business logic.

6. Inversion of Control (IoC): This is the core of Dependency Injection in Spring framework. We use this design pattern

to increase the modularity of an application. We keep the objects loosely coupled with Dependency Injection.

104. What are the popular uses of Façade design pattern?

Some of 1.

2. 3.

4. 5.

the popular uses of Façade design pattern are as follows:

A Façade provides convenient methods for common tasks that are used more often.
A Façade can make the software library more readable.
A Façade can reduce the external dependencies on the working of inner code.

A Façade can act as a single well-designed API by wrapping a collection of poorly designed APIs.
A Façade pattern can be used when a System is very complex and difficult to use. It can simplify the usage of complex system.

105.What is the difference between Builder design pattern and Factory design pattern?

Both Factory and Builder patterns are creational design patterns. They are similar in nature but Factory pattern is a simplified generic version of Builder pattern.

We use Factory pattern to create different concrete subtypes of an Object. The client of a Factory may not know the exact subtype. E.g. If we call createDrink() of a Factory, we may get Tea or Coffee drinks.

We can also use Builder pattern to create different concrete subtypes of an object. But in the Builder pattern the composition of the object can be more complex. E.g. If we call createDrink() for Builder, we can getCappuccino Coffee with Vanilla Cream and Sugar, or we can get Latte Coffee with Splenda and milk cream.

So a Builder can support creation of a large number of variants of an object. But a Factory can create a broader range of known subtypes of an object.

106.What is Memento design pattern?

Memento design pattern is used to implement rollback feature in an object. In a Memento pattern there are three objects:

Originator: This is the object that has an internal state.

Caretaker: This is the object that can change the state of Originator. But it wants to have control over rolling back the change.

Memento: This is the object that Caretaker gets from Originator, before making and change. If Caretaker wants to Rollback the change it gives Memento back to Originator. Originator can use Memento to restore its own state to the original state.

E.g. One good use of memento is in online Forms. If we want to show to user a form pre-populated with some data, we keep this copy in memento. Now user can update the form. But at any time when user wants to reset the form, we use memento to make the form in its original pre-populated state. If user wants to just save the form we save the form and update the memento. Now onwards any new changes to the form can be rolled back to the last saved Memento object.

107.What is an AntiPattern?

An AntiPattern is opposite of a Design Pattern. It is a common practice in an organization that is used to deal with a recurring problem but it has more bad consequences than good ones.

AntiPattern can be found in an Organization, Architecture or Software Engineering.

Some of the AntiPatterns in Software Engineering are:

1.

2.

3. 4.

Gold Plating: Keep on adding extra things on a working solution even though these extra things do not add any additional value.

Spaghetti Code: Program that are written in a very complex way and are hard to understand due to misuse of data structures.

Coding By Exception: Adding new code just to handle exception cases and corner case scenarios.

Copy Paste Programming: Just copying the same code multiple times rather than writing generic code that can be parameterized.

108.What is a Data Access Object (DAO) design pattern?

DAO design pattern is used in the data persistent layer of a Java application. It mainly uses OOPS principle of Encapsulation.

By using DAO pattern it makes the application loosely coupled and less dependent on actual database.

We can even implement some in-memory database like H2 with DAO to handle the unit-testing.

In short, DAO hides the underlying database implementation from the class that accesses the data via DAO object.

Recently we can combine DAO with Spring framework to inject any DB implementation.