**Important References:**

1. **SMOKE TESTING:** Type of testing done by tester once the build is given by the developers. This involves testing of critical and main functions of the build whether they are working fine for the testers to decide whether the application build can be further testable or no.
2. **SANITY TESTING:** It is a subset of regression testing. If there are any new minor functionalities included or any defect fixes done in the current build smoke testing is done to verify whether the above two situation have not caused any effect on the other functionalities of the application.
3. **DIFFERENCES:**

|  |  |
| --- | --- |
| SMOKE TESTING | SANITY TESTING |
| * Major objective is to verify if the build is stable and testable or no. | Objective is to verify whether new functionalities and defect fixes do not affect other parts of application. |
| * Done by developers and testers | Done only by testers |
| * Major test cases are referred to and tested | High level functionalities are tested without any test case doc. |
| * Application starting or no, main page appearing , we can login or no |  |
| * Entire testing end to end. | Only particular component of the entire system |

1. **DIFFERENCES:**

|  |  |
| --- | --- |
| REGRESSION | RE-TESTING |
| Repeated testing of an already tested program,after modification to discover any defects introduced or uncovered as a result of changes in software being tested or in a related or unrelated component. | Re-Testing involves testing of defects that are fixed against the build . It involves in depth testing of that big fixed area. |
| Done when defect is being fixed, CR, performance issue fixed, env change. | Only includes re-executing test case that are failed earlier not include related passed test cases |
| Checks unexpected side effects | Carried out before regression testing |
| Purpose- new code change does not affect existing functionality. | Purpose – that particular defect is working fine. |

1. **Architecture of WebDriver:**

WebDriver is an open source API collection used to automate web based applications.WebDriver allows you to work with multiple browsers on multiple platforms.

HTTP over Server

Selenium client library:

Java,C#,Ruby,Python.

Real Browser

Browser driver

Chrome,ie,firefox

Chrome,ie,firefox

Json Wire protocol over HTTP

HTTP over Server

1. **Entry and Exit criteria:**

Entry: Pre-Requisites needed to begin with the testing of the application. It includes approved requirements, Readiness of software, Test data availability, Test env readiness, Test cases readiness,

Exit: Post conditions after testing that need to be fulfilled before deploying into production. It includes test reports, all high priority bugs have to be closed, test logs, test case execution completion.

1. **Verification and Validation differences:**

Verification: is the process of checking whether the software meets the specifications. Are we building the product right?

Validation: Is the process of checking is the specification captures customer needs.

|  |  |
| --- | --- |
| VERIFICATION | VALIDATION |
| Static testing | Dynamic testing |
| Docs, design, code, program | Actual testing |
| Does not involve executing of code. | Involves executing of code. |
| Walkthrough, review, Inspection | White, black and grey box testing. |

Return employee with max salary:

*Select \* from Employee where sal=(Select max(sal) from emplyee);*

Return the max salary from employee table:

*Select max(sal) from employee*

Select second highest salary:

*Select max(sal) from Employee where sal not in(Select max(sal) from Employee);*

Return ,emp name, dept\_id, max salary, depant\_name wise:

*Select dept\_id,emp\_name,sal from Employee e inner join Department d ON (e.dept\_id=d.dept\_id where (Slect max(sal) from Employee group by Depat\_name));*

**Various Tabs in HP-ALM:**

DASHBOARD: analysis view, dashboard view.

MANAGEMENT: Releases.

REQUIREMENT:

TESTING: Test resources, Test plan, Test lab, Test runs.

DEFECTS:

**TestNG:**

1. **TestNG**: Testing next generation is an automation framework used to execute webdriver scripts written by developers. 3 types of annotations : precondition annotation, post condition, test annotation.

Eg:@BeforeSuite, @BeforeTest, @BeforeMetod, @BeforeClass, @Test, @after…..

Advantages: open source, easy to maintain the test scripts, lots of annotations, priorities can be assigned, grouping is poosible, generates reports,

* Assign priorities: @Test(priority=1)
* Grouping: @Test(groups=”name”)
* Dependency: @Test(dependsOnMethods=”loginTest”)
* Call a method several times: @Test(invocation=8)
* If any method in the script takes long time to execute then terminate using time out: @Test(timeout=2000)
* Annotation for exception: @Test(expectedException=numberformatException.class)
* Disable a test @Test(enabled=’false’)
* Parameters: @Test({“username,”“password”}) in .xml: <parameter name= “username” value=”para”>

Eg: Assert.assertFalse(ele.isSelected())

Hard assert: program stops after assert stamen fails.

Soft assert: program continues to execute even after assertion fails.

SoftAssert sassert= new SoftAssert();

Why testing.xml? to execute many classes and tests as a single test suite.

1. **Page Object Model:** Page object is a class that represents a webpage which has elements as objects and functionalities to interact with the webpage.

**Page factory:** is a way to initialize the web elements present inside the webpage

1. **Boundary value Analysis:**

It’s a test case design strategy for black box testing. This technique is used to identify errors at the extreme boundaries at the input domain.

Eg:Suppose our testing module accepts 1 to 1000as input range, we can have lower boundary inputs such as 0 and 99 and higher like 2 and 1001 as part of negative testing .

1. **Equivalence Partitioning:**

Its also a testing strategy for black box testing. Here huge range of input is divided into different equivalence data classes.

Eg : Divide input domain into 3 classes if we have to test a specific module which can accept inputs from 1 to 100.

Class 1 : input from 1 to 100 any one test data.

Class2: < 1 neg case

Class 3: > 100 neg case.

**10.Test Coverage:**

It is analysis which helps in knowing if all the requirements are tested at least once.It can have various things and different in different teams.

1. There should be at least on test case around one requirement.
2. All the testers should be assigned to test cases equally.
3. All the requirements must be covered with all the test cases designed.

Eg for test coverage: testing a notepad application. Check for font, colors format, tabs etc is fine but testing how it works when opened with different application, when it’s closed opened many times.

Also we need to test Risk coverage: eg- when testing web –based application.

How to adopt a proper test coverage method:

* Aware of how much work is involved and should they add any more test cases by knowing till where the implementation has progressed.
* Take help of the RTM
* Should be aware of each and every requirement.
* Focus on critical requirement. Get reminders from previous releases also.
* Have automation
* Have a test management tool.
* Keep track of bulid cycles and defect fixes.

1. **Testing without solution doc being give? Use** exploratory testing.
2. **Usability Testing:**  the ease with which an end user can easily access the application.
3. **Big Bang Approach:** Combining all the modules at once and testing the flow once to check if independent modules work fine as a whole application.
4. **Top- Down Approach;** Here all the high level
5. **How to initialize page factory:** by using*PageFactory.initElements(d, THIS);*

XPATH:

1. //tagname[@att name=att-val]
2. Functions in xpath: text(), contains().starts-with()
3. Contains: //tagname[conatains(@att-val,’part of the value’)]
4. Starts-with: //tagname[starts-with(@att-valu,’part of value’)]
5. Text(): //tagname[text(),’full text inner html’]
6. Contains and text: //tagname[conatins(text(),’full inner html’)]

Page Object Model:

EventFiringWebDriver class will give the detailed logs og each line execution of the test scripts.

*e\_driver* =**new** EventFiringWebDriver(*d*);

*eventListner*= **new** WebEventListener();

*d*= *e\_driver*.register(*eventListner*);

Extent Report:by implementing IReporter interface.

In testng.xml there should be a listener tag to specify the location of extent report class

JAVA QUESTION:

1. **Differences between overloading and overriding:**

|  |  |
| --- | --- |
| OVERLOADING | OVERRIDING |
| Same name but different argument types. | Changing the method implementation in the child class. |
| Argument types must be different ,at least the order | Argument types must be same including order |
| Method signature must be different | Method signature must be same. |
| Can have different return type | Co-variant return type allowed- can override with child class return type |
| Applicable for private , static , final | Not applicable for private static and final |
| No restrictions for overloading for access modifiers | Increasing the visibility of access modifiers is allowed |
| Known as compile time polymorphism | Runtime polymorphism |

1. **Examples of OOPS concepts in Selenium world:**
2. ABSTRACTION: *Page Object Model-* Here the locators are present in one class and utilized in test classes. Implementation relating the locators is hidden.
3. INTERFACE: WebDriver is an interface, whose reference is used to create an object of Firefox, chrome or any other browser classes.
4. INTERITANCE: WebDriver waits, property files, excel classes are created as base classes and are utilized in the test classes. Different banks with different ROIs.
5. METHOD OVERLOADING: assertEquals method is overloaded.
6. METHOD OVERIDING: get() and navigate() methods of different drivers in selenium are an example.
7. ENCAPSULATION: Every method in selenium is encapsulated. In POM classes we use @FindBy to declare the data members and initialize them using the constructor.
8. **Difference between char Array and a String:** String is terminated with a special character ‘\0’.
9. **How to print the address of the String:** getclass() returns the class name of the object. getName() returns the name of the entity(eg: class, interface, primitive type)

Address of the String instance: *String s = new String(“ABC”);*

*s.getClass().getName+’@’+Integer.toHexString(s.hashCode());*

1. **Methods in String class:a.** equals() || equalsIgnoreCase()

**b.** compareTo() : converts string to Unicode value and compares the String. S1> s2 positive values is returned.

S1= S2 gives 0

S2> S1 neg value

**c.**split(regex)|| spilt(regex,limit)

**d.** contains() e. length()

**e.** replace(char old , char new) || replace(charseq target, charseq repl)

replaceAll(String regex, String repl) || replaceFirst(String regex,String replace).

**f.**Concat() || +

**g.** substring(strt ind, end indix): gives the String starting from start index till end index-1.

1. **Methods in StringBuffer:** append(), insert(),delete()
2. All the wrapper classes have by default overridden hashcode() and equals() therefore duplicate values cannot be added in hashmap, even written it will override that value and make it as one
3. **Foreach loop**: is an advanced for loop usually used to traverse array and collections.

For(datatype variable: array/collection)

{

}

1. Map: Set keyset(), Set valueSet()
2. Differences between Iterator and Listiterator:

|  |  |
| --- | --- |
| Iterator | ListIterator |
| Can be used across all collection interfaces and the classes. | Can only be used in List interface and its implementing classes. |
| Can be used to traverse only in forward direction. | Can traverse both in forward and reverse direction. |
| Has basic next and hasNext(). Cannot get index | Has additional methods like add() and replace(), nextIndex(), previousIndex() |

1. **HashSet does not allow get():**HashSet is all about having unique values if element is not present in Set then we can add it. Here we can also check the presence of object not get that object. We can use contains() method instead to check if its present or if not present add it.
2. **Difference between remove() of collection and remove() of iterator:**remove(obj) of collection removes the current element in the collection classes as specified in the method. Remove() of iterator removes that element which is pointed by the next() of the iterator.
3. **Collection interfaces:**

|  |  |  |  |
| --- | --- | --- | --- |
| LIST | QUEUE | SET | MAP |
| \*Indexed type  \*duplicates are allowed  \*Insertion order preserved  \*add(),get()  \*collections.reverse(list) | \*not indexed type  \*insertion order not maintained.  \*Cannot insert null values.  \*Duplicates are allowed.  \*add(),peek(),poll() | \*Duplicate not allowed  \*null values allowed  \*does not maintain insertion order. | \*key value pairs |

1. **In detail:**

*ArrayList:* null values and duplicate values. Accessing faster, insertion and deletion slower.

*LinkedList:* Duplicates allowe , insertion order preserved and non -synchronized. Adding and removing data is easier and fetching is difficult.

*PriorityQueue:*Sorted in required way, Using comapreTo().

*HashMap:* Null allowed, not synchronized, no insertion order preserved.

*HashTable:* Insertion order preserved.

*LinkedHasMap:* Allows duplicate, insertion order preserved.

*TreeMap:* Does not allow null values, values sorted using comparator interface.

*HashSet*: duplicates not allowed, null values allowed, does not preserve insertion.

*LinkedHashSet:* Duplicates allowed, preserves insertion.

*TreeSet*: Elements are sorted, Also because it implemets navigable set has many extra methods like method to find value higher or lower than current element eg: higher(Object), lower(Object obj)

1. **System.out.println();**System is a class in java.lang package, out is a static variable in the System class of the type printStream and println() is a method present in the PrintStream class.

**Eg:** class Test

{

Static String s= “abc”;

}

Test.s.lenght();

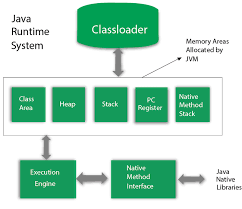
1. **Difference between Abstraction and Encapsulation:**

|  |  |
| --- | --- |
| ABSTRACTION | ENCAPSULATION |
| Hiding implementation and providing only utilization of the object | It is the way of binding the data and code acting on that data(Methods) together. |
| Abstract class-0 to 100% abstraction, interface – 100% abstraction | Private data, public getter setter eg: HashMap protecting inner working of object from outside world. |
| Hiding unwanted details by giving out most important details to the user. | Protecting inner working of objects from outside world. |
| Focuses on what object does instead of how it does | Hiding internal details of how an object works |
| Based on design level | Based on implementation level |
| Focuses on what abject does. | Hiding the mechanism of how the object does something. |
| Mobile phone’s display screen , buttons etc. | No need to understand how keypad is connected to display |
| Focuses on outer lookout | Focuses on inner lookout |

1. **Difference between Interfaces and Abstract class.**

|  |  |
| --- | --- |
| INTERFACE | ABSTRACT CLASS |
| When we don’t know what to implement at all | When we partially know what to implement |
| Methods are public and abstract | Methods can be concrete or abstract |
| 100% pure abstraction | Not pure abstraction. |
| Modifiers can’t be the following: public: so no protected, private. Abstract: private , static , final | Can be declared any how |
| Public static final data members | If a class has at least one abstract method then class should be declared as abstract |
| Always initialize at time of declaration | No such rule |
| Constructors , static and non-static block not allowed | All allowed |
| Cannot create and object, can have only a reference variable. | Can create and object |
|  | Public void test()  {  }  This is not an abstract method this is a concrete method with blank implementation |

1. **Architecture of JVM:**



*Class Loader*: loads the class files .Its the subcomponent of JVM.

*Memory*: has 5 parts:

1. *Method area*: Usually stores the method and the class data.
2. *Heap:* Object created data or instance data.
3. *Stack area*: all the currently running thread information is stored. It holds local variables and partial results in each stack called frame and frame is erased when execution is complete.
4. *Program counter Register*: Stores the address of the JVM instruction surrently being executed.
5. *Native Method Stack*: Contains all the native methods being used in the application.

Eg: getSystemTime() is native method for java.

*Execution Engine*: Executes the java code: contains 3 parts \*virtual processor \* Interpreter \*JIT compiler(just in time). JIT- Enhances execution speed by compiling byte code that have similar functionality. Interpreter:Readsbyte code stream then execute the instructions. Eg: if a method is being repeatedly called then JIT compiles it separately and executes the same compiled version to reduce time.

1. **Difference between final, finally, finalize():**

|  |  |  |
| --- | --- | --- |
| *final* | *finally* | *Finalize()* |
| Final is access modifier can be used with a variable method or a class.  Final variable one time declaration and initialization, methods can be overridden, final class can’t be extended. | Is a block of code used along with try-catch to write the clean up code. | Is a method used by garbage collector to clean up the unaddressed objects. |

1. **File Handling:**

FileInputStream: Read the data onto the file.

FileOutputStream: Write the data into the file.