**SDLC (Software Development Lifecycle):**

The process of developing a software through successive phases in an orderly way.

**Who are the people involved?**

* **Developers:** write the programming code.
* **Business Analysts:** gather and create requirements.
* **Testers:** verify that the software meets requirements.
* **Project Manager:** manages the overall project to insure it is within the budget and on time.

**Why is software testing necessary:**

* To prove to customer that the application is problem free
* To ensure that the product works as the user requested
* Testing helps increase the reliability of the software as well as improving the quality
* To discover defects or bugs and determine the root cause of the issue

**SDLC Phases:**

1. Planning
2. Requirements Analysis
3. Designing
4. Developing
5. QA/Testing
6. Production/Maintenance

**Planning**:

* We gather information and based on that information we plan basic project approach and conduct feasibility study.
* In this phase, we plan the entire project, determine the deliverables and schedules and set budgets.
* People involved: senior members of development team (PMs, Sr. BAs, Sr. Dev, Sr. Testers and Test Lead/Test Manager)

**Requirements analysis:**

* BAs gather business requirements from the client
* BAs convert the business requirements into Technical Requirements which will be used by the development and testing teams
* BAs are the subject matter experts (SMEs) for the requirements

What exactly happens in this phase:

* BAs meet the client and gather Business Requirements
* BAs define and document Business Requirements and get final approval from the client
* Once the client approves BAs work, the Business Analyst creates a document with the formalized business requirements (BRD)
* Project Manager approves BRD
* Business Analyst converts Business Requirements (BRD) to Technical Requirements (SRS)

Documentation:

* BRD – Business Requirement Document: consists of business requirements
* SRS – Software Requirement Specification: consists of technical requirements

**Designing:**

* Main contributors: developers, BAs, PMs
* Developers plan their work (how they will develop the software application)
* Based on the requirements in SRS, multiple design approaches for the product architecture are proposed and documented in DDS (Design Document Specifications)
* The best design approach is selected
* A blueprint is created that shows how the various servers are going to be working together – App Servers, DB Servers, Web Servers, Load Balancers, etc.

**Developing:**

* The actual development of the software product begins and the product is built.
* Developers use Technical Requirements found in the SRS and the Design Documents (DDS) as a guide to know what and how to develop.

**Testing:**

* Application is tested against the requirements.
* Ensures that the product is actually solving the needs addressed and gathered during the requirements analysis phase.

**Production:**

* Also known as implementation or deployment phase.
* After the application is successfully tested and all issues have been resolved, the product is delivered to the client.

**Maintenance:**

* Finished application needs to be updated from time to time.

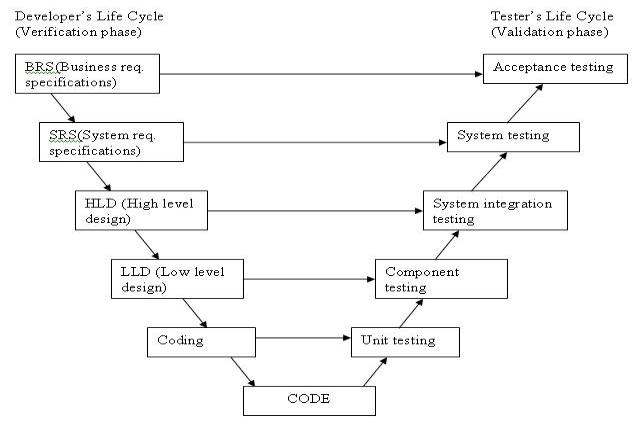
**DEVELOPMENT METHODOLOGIES:**

**Waterfall Methodology:**

* Each phase of SDLC must be completed before the next phase can begin.
* Waterfall is good for small projects and projects where all requirements are known and are not likely to change.
* Testing starts after development has been completed.

**V-Model:**

* Means Verification and Validation Model. Testing of the product is planned parallelly with a corresponding phase of development.



**Incremental Methodology:**

* All of the requirements are split up into several builds (versions of the application).
* Multiple development cycles take place: Multi-Waterfall cycle.
* A working version of the software is produced during the first module.
* Each subsequent release of the module adds functionality to the entire application.

**Agile Methodology:**

* A type of incremental methodology.
* Software is developed in incremental, rapid cycles. Each increment is called a Sprint.
* SCRUM and Extreme Programming (XP) are the two most commonly used agile methodologies. Others being Lean & Kanban Development, Crystal Methods, Feature Driven Development.

**Components of Agile Scrum Development:**

* The Scrum Team
* Scrum Events (Ceremonies)
* Scrum Artifacts
* Scrum Rules

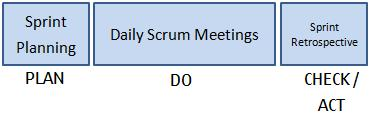
**The Scrum Team:**

* Composed of 7 +/-2 members (only 5-9 members)
* The scrum team as a whole unit decides how to address issues and solve problems

**Scrum Team Roles:**

* The Product Owner
  + Understands the users’/customers’ needs
  + Collaborates with development team
  + Sets Sprint goals
  + Manages the product backlog
  + Accepts the completed increments of work by the development team
* The Scrum Master
  + Servant Leader, has no hierarchical authority over the Scrum Team
  + Facilitator: ensures that the Scrum Team adheres to the Scrum Theory, practices and rules
  + Protects the team by doing anything possible to help the team perform at the highest level:
    - Removes impediments
    - Facilitates meetings
    - Helps product owner in Backlog Grooming
* The Development Team
  + Anyone who participates in the creation of the delivered increment is part of this team.

**Scrum Ceremonies:**

* The Sprint
  + A time boxed period. Within which a specific work is completed and ready for review.
  + 2-4 weeks long. Can be as short as 1 week.
  + No changes to requirements are made during a Sprint.
  + Only the Product Owner has the authority to cancel a sprint.
* Sprint Planning
* Daily Stand-up:
  + Meetings held daily to organize the day.
  + 3 questions answered by all Scrum Team members: what did you complete yesterday, what will you work on today, what impediments you faced that blocks your progress.
  + No more than 15-20 minutes long.
* The Sprint Review:
  + The Scrum Team presents the work they completed during the sprint.
  + Product Owner checks the work against predefined acceptance criteria:
    - accepts or rejects the work.
  + Stakeholders/Clients provide feedback to ensure that the delivered increment met the business need.
* The Retrospective:
  + Final team meeting in the Sprint, facilitated by the Scrum Master.
  + Determines what went/didn’t go well and how the team can improve in the next Sprint.

**Scrum Artifacts:**

* **Product Backlog:** 
  + Owned by the Product Owner
  + Single most important document that outlines every requirement for a system or product
  + To do list
  + Completion of these tasks produces the deliverable product with business value.
* **Sprint Backlog:**
  + Specific list of items taken from the product backlog
  + Items on here are what must be completed during that sprint
  + Scrum Team decides what to take from the product backlog during Sprint Planning
  + Once Sprint Backlog is created, it doesn’t change for the duration of the Sprint
    - Requirements can change, but only outside of the Sprint
* **Increment**
  + Sum of all product backlog items that have been completed since the last software release.
  + Also referred to as Potentially Shippable Increment (PSI).
  + Potential because product owner decides if he/she wants to accepts or reject it.
  + Up to the product owner to decide when an Increment is released.

**Epic:** in Agile Scrum Methodology, an Epic can be defined as a big chunk of work that has one common objective. It could be feature, customer request or business requirement. An Epic usually takes more than one sprint to complete.

**TDD (Test Driven Development):** is the process of using examples to guide the development of each function of your software. In TDD you create your program in a loop like: write an example as a simple test -> watch the test fail -> write code to make the test pass -> refactor and clean your code

**BDD (Behavior Driven Development):** is somewhat similar and complementary to TDD, but TDD focuses on driving the development of individual functions in the code, while BDD applies the similar at the level of features. Since features are driven by user needs and users express their needs in natural language rather than language of codes, BDD starts with a natural language to create examples. To make the examples executable BDD requires a little bit of structure and that structure allows a bridge between the world of programming and the world of natural language requirements. A BDD feature file consists of one or more scenarios, these scenarios are just examples of how the application should behave from the standpoint of a user. For example, you might have a scenario that says:

Given I am on the login page

When I attempt to login with valid credentials

Then I am shown the application dashboard

And another scenario that says:

Given I am on the login page

When I attempt to login with invalid credentials

Then I am shown an error message

The examples above are very natural language, but still very structured:

Given – tells the starting condition

When – tells what you actually do

Then – tells what results are expected

Behind the scene, your BDD Framework will match the natural language to the code that the developers write to actually do these steps, but it is always driven from the natural language.

**STLC (Software Testing Lifecycle):**

The process of testing a software in a well-planned and systematic way.

**Generic phases in STLC:**

1. Requirements Analysis
2. Test Planning
3. Test Analysis
4. Test Design
5. Test construction and verification
6. Test Execution and Bug Reporting
7. Final Testing and Implementation
8. Post Implementation

**1. Requirements Analysis:**

* Testers analyze requirements and work with BAs and Developers during the design phase to see which requirements are testable and how they are going to test those requirements.
* Testers ask Business Analysts questions about the requirements to make sure they are clear.
* Vague or unclear requirements are the leading causes for bugs to slip past the testing team and into production.

**2. Test Planning:**

Plans are made to determine:

* What needs to be tested
* How the testing will be done
* Test strategy to be followed
* Test environment
* Testing methodologies
* Hardware/software requirements
* Resources (human, technologies, etc.)
* Risks

Test planning is typically allocated 1/3rd of the time of the entire QA engagement. Other 1/3rd is for Test Designing and the rest for Test Execution.

Test Plan is documented by QA Manager/Lead based on inputs from QA team members.

Test Plan is not static, it is updated on an on-demand basis.

**Test Plan Components:**

Test Plan ID, References, Introduction, Test Items, Risk, Items to be tested, Features excluded from testing, Testing Approach, test pass/fail criteria, resumption/suspension criteria (entry/exit criteria), test deliverables (RTM, Test Reports, Bug Reports, etc.), test environment setup, training and staffing, team member responsibilities, testing schedule, planning for risk and contingency plans, approvals.

**3. Test Analysis:**

* Determine what testing needs to be completed in each SDLC phase
* Automation activities can be decided in this phase: how will the automation be done, how much time will it take to automate, what features will be automated.

**4. Test Design:**

* Testers design their test cases for both black and white box testing.
* Every test case must have at least the followings:
  + Test Case ID
  + Test Case Name
  + Test Case Description
  + Step Name
  + Step Description
  + Expected Result
  + RTM Information
* If automation testing is planned to be done, those scripts are also written in this phase.

**5. Test Construction and Verification:**

* Test cases are reviewed by leads, managers and other stakeholders.
* Test Case Review meetings are held to ensure that the test cases are properly testing the requirements and that appropriate data is being tested.

**6. Test Execution and Bug Reporting:**

* Test cases are executed after unit testing is completed and the build has been released to the QA environment.
* Defects are reported using a defect tracking tool (ALM, JIRA, etc.).
* Test Execution Reports are generated and distributed to all necessary parties (Development Team, BAs, PMs, Test Lead/Managers).
* Once developers have fixed the bugs raised by testers, the testers do retesting and then regression testing to ensure that the defects have been fixed and that the fix has not affected any other area of the software.

**7. Final Testing and Implementation:**

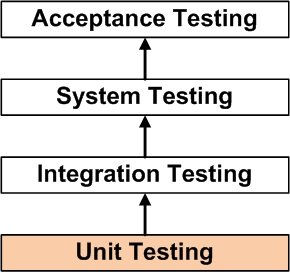
* Final testing is conducted: all kinds of performance testing
* Software is verified in an environment similar to production

**8. Post Implementation:**

* Process review meetings are conducted
* Lessons Learned Document is created. Retrospective meeting – led by Scrum Master

**SOFTWARE TESTING LEVELS**

Software testing levels are the different stages of the software development lifecycle where testing is conducted. There are four levels of software testing: Unit >> Integration >> System >> Acceptance.



|  |  |
| --- | --- |
| **Level** | **Summary** |
| Unit/Component Testing | A level of the software testing process where individual units of a software are tested. The purpose is to validate that each unit of the software performs as designed. It is performed by using the [White Box Testing](http://softwaretestingfundamentals.com/white-box-testing/) method. |
| Integration Testing | A level of the software testing process where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units. Any of [Black Box Testing](http://softwaretestingfundamentals.com/black-box-testing/), [White Box Testing](http://softwaretestingfundamentals.com/white-box-testing/) and [Gray Box Testing](http://softwaretestingfundamentals.com/gray-box-testing/) methods can be used. |
| System Testing | A level of the software testing process where a complete, integrated system is tested. The purpose of this test is to evaluate the system’s compliance with the specified requirements. Usually, Black Box Testing method is used. |
| Acceptance Testing | A level of the software testing process where a system is tested for acceptability. The purpose of this test is to evaluate the system’s compliance with the business requirements and assess whether it is acceptable for delivery. Usually, [Black Box Testing](http://softwaretestingfundamentals.com/black-box-testing/) method is used in Acceptance Testing.   * Internal Acceptance Testing (Also known as Alpha Testing) is performed by members of the organization that developed the software but who are not directly involved in the project (Development or Testing). Usually, it is the members of Product Management, Sales and/or Customer Support. * External Acceptance Testing is performed by people who are not employees of the organization that developed the software. * Customer Acceptance Testing is performed by the customers of the organization that developed the software. They are the ones who asked the organization to develop the software. * User Acceptance Testing (Also known as Beta Testing) is performed by the end users of the software. They can be the customers themselves or the customers’ customers. |

**SOFTWARE TESTING METHODS**

Software testing methods listed here are the major methods used while conducting various [Software Testing Types](http://softwaretestingfundamentals.com/software-testing-types/) during various [Software Testing Levels:](http://softwaretestingfundamentals.com/software-testing-levels/)

|  |  |
| --- | --- |
| **Method** | **Summary** |
| [Black Box Testing](http://softwaretestingfundamentals.com/black-box-testing/)  (Behavioral Testing) | A software testing method in which the internal structure/design/implementation of the item being tested is not known to the tester. These tests can be functional or non-functional, though usually functional.   * A tester, without knowledge of the internal structures of a website, tests the web pages by using a browser; providing inputs (clicks, keystrokes) and verifying the outputs against the expected outcome. * Applicable to integration testing, system testing and acceptance testing.   Techniques:   * Equivalence Partitioning: It is a software test design technique that involves dividing input values into valid and invalid partitions and selecting representative values from each partition as test data. * Boundary Value Analysis: It is a software test design technique that involves the determination of boundaries for input values and selecting values that are at the boundaries and just inside/ outside of the boundaries as test data. * Cause-Effect Graphing: It is a software test design technique that involves identifying the cases (input conditions) and effects (output conditions), producing a Cause-Effect Graph, and generating test cases accordingly. |
| [White Box Testing](http://softwaretestingfundamentals.com/white-box-testing/) (Clear Box, Open Box, Glass Box, Transparent Box, Code Based or Structural Testing) | A software testing method in which the internal structure/design/implementation of the item being tested is known to the tester.  Applicable to Unit, Integration and System Testing, but mainly to Unit Testing.  A tester, usually a developer as well, studies the implementation code of a certain field on a webpage, determines all legal (valid and invalid) AND illegal inputs and verifies the outputs against the expected outcomes, which is also determined by studying the implementation code. White Box Testing is like the work of a mechanic who examines the engine to see why the car is not moving. |
| [Gray Box Testing](http://softwaretestingfundamentals.com/gray-box-testing/) | A software testing method which is a combination of Black Box Testing method and White Box Testing method. Here the internal structure is partially known.  Primarily used in integration testing. |
| [Agile Testing](http://softwaretestingfundamentals.com/agile-testing/) | A method of software testing that follows the principles of agile software development. |
| [Ad Hoc Testing](http://softwaretestingfundamentals.com/ad-hoc-testing/)  (Random/ Monkey Testing) | A method of software testing without any planning and documentation.  Normally used during acceptance testing. |

**SOFTWARE TESTING TYPES**

Software Testing Types listed here are a few out of the hundreds of software testing types. The different types of testing you can perform on a software is limited only by the degree of your imagination. Here, we provide you summary of some of the major ones.

|  |  |
| --- | --- |
| **Type** | **Summary** |
| [Smoke Testing](http://softwaretestingfundamentals.com/smoke-testing/)  (Build Verification Testing) | A type of software testing that comprises of a selective set of tests that aim at ensuring that the most important functions work.  It covers most of the major functions of the software but none of them in depth. If an application is badly broken, detailed testing might be a waste of time.  Normally used in integration, system and acceptance testing levels. |
| [Functional Testing](http://softwaretestingfundamentals.com/functional-testing/) | A type of software testing whereby the system is tested against the functional requirements/specifications. Black box testing technique is used.  Normally performed during the levels of system testing and acceptance testing. |
| [Usability Testing](http://softwaretestingfundamentals.com/usability-testing/) | Usability Testing is a type of software testing done from an end-user’s perspective to determine if the system is easily usable.  Normally performed during system testing and acceptance testing levels. |
| [Security Testing](http://softwaretestingfundamentals.com/security-testing/) | A type of software testing that intends to uncover vulnerabilities of the system and determine that its data and resources are protected from possible intruders. Ex: sign in, sign out, click back button, see if it takes you to logged in application. |
| [Performance Testing](http://softwaretestingfundamentals.com/performance-testing/) | A type of software testing that intends to determine how a system performs in terms of responsiveness and stability under a certain load. Types/kinds:   * **Load Testing** checks how system functions under heavy number of concurrent users. Using tools like JMeter. * **Stress Testing** evaluates the behavior of a system at or beyond the limits of its anticipated workload. * **Endurance Testing** evaluates the behavior of a system when a significant workload is given continuously. * **Spike Testing** evaluates the behavior of a system when the load is suddenly and substantially increased. |
| [Regression Testing](http://softwaretestingfundamentals.com/regression-testing/) | A type of software testing that intends to ensure that changes (enhancements or defect fixes) to the software have not adversely affected it.  Can be performed during any level of testing, but mostly system testing. |
| [Compliance Testing](http://softwaretestingfundamentals.com/compliance-testing/)  (Conformance, regulating, standards testing) | Compliance Testing [also known as conformance testing, regulation testing, standards testing] is a type of testing to determine the compliance of a system with internal or external standards. |

|  |  |  |
| --- | --- | --- |
| **Sanity Testing** | **Smoke Testing** | **Regression Testing** |
| It is performed when a new functionality, change or bug fix is implemented. | It is performed in the initial phases when the release is unstable or at the final phase when the release is ready for deployment. | It is performed when a new functionality, change or bug fix is implemented. |
| It has a narrow scope. | It has only critical functionalities in the scope. | It has a broad scope. |
| The aim is to quickly verify if the new functionality, change or fix is working and has not broken down existing functionality. | The aim is to check if the critical functionalities are working as expected. | The aim is to check if the older functionalities are still working fine, after the change. |
| It doesn’t catch all the bugs of the functional areas which are impacted by the code change. | It catches the bugs in critical functionalities only. | It catches all the bugs of the functional areas which are impacted by the code change. |
| It is non-scripted. | It is scripted. | It is scripted. |
| It takes very less time to be performed. | It takes not more than 30 minutes. | It takes more time and testing effort. |
| It determines wheter the application should be tested for regression. | It determines whether the application is stable or not. | It determines whether the old and new functionalities working together correctly or not. |

**TEST ARTIFACTS**

Test artifacts are the deliverables or documents which are prepared during the testing process. Below are the different test artifacts:

**1. Test Strategy:**

A Test Strategy document is a high level document and normally developed by project manager. This document defines “Software Testing Approach” to achieve testing objectives. The Test Strategy is normally derived from the Business Requirement Specification document.

This document enlists how to achieve the desired result using available resources. It provides clarity of processes, tools, techniques, approaches, infrastructure etc. to complete all of the testing process.

The Test Strategy document is a static document meaning that it is not updated too often. It sets the standards for testing processes and activities and other documents such as the Test Plan draws its contents from those standards set in the Test Strategy Document.

Some companies include the “Test Approach” or “Strategy” inside the Test Plan, which is fine and it is usually the case for small projects. However, for larger projects, there is one Test Strategy document and different number of Test Plans for each phase or level of testing.

Components of the Test Strategy document:

* Scope and Objectives
* Business issues
* Roles and responsibilities
* Communication and status reporting
* Test deliverables
* Industry standards to follow
* Test automation and tools
* Testing measurements and metrices
* Risks and mitigation
* Defect reporting and tracking
* Change and configuration management
* Training plan

**2. Test Plan:**

The Test Plan document on the other hand, is derived from the Product Description, Software Requirement Specification SRS, or Use Case Documents.

The Test Plan is usually prepared by the Test Lead or Test Manager and the focus of the document is to describe what to test, how to test, when to test, who will do what test, test environment, entry and exit criteria, etc.

It is not uncommon to have one Master Test Plan which is a common document for the test phases and each test phase have their own Test Plan documents.

Components of the Test Plan document:

* Test Plan id
* Introduction
* Test items
* Features to be tested
* Features not to be tested
* Test techniques
* Testing tasks
* Suspension criteria
* Features pass or fail criteria
* Test environment (Entry criteria, Exit criteria)
* Test deliverables
* Staff and training needs
* Responsibilities
* Schedule

**Agile Test Plan – Do we really need one?**

It is clearly evident that test plan doesn’t reveal defects but test scenarios will. The effort needs to be shifted on creating better scenarios than creating a test plan. We may not need to have an extensive test plan in agile projects for each sprint due to time constraints, but we do require a high-level agile test strategy as a guideline for the agile teams. The purpose of the agile test strategy document is to list best practices and some form of structure that the teams can follow.

**Difference between Test Plan and Test Strategy:**

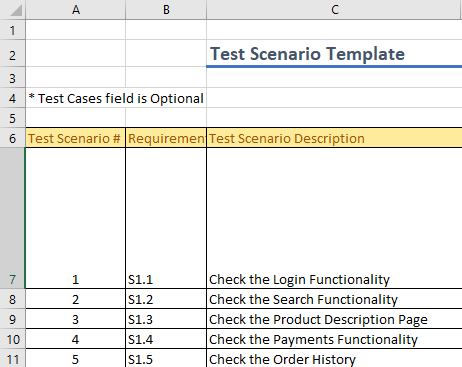
Test strategy is a high level document which defines the approach for software testing. It is basically derived from the Business Requirement document. Test strategy is developed by project manager or business analyst. It is kind of static document which sets the standards for testing so not updated often.

Test plan is derived from SRS (Software Requirement Specification) which is prepared by test lead or manager. The main goal of test plan is to include all the details related to testing such as what to test, when to test, how to test and who will be the tester. Test plan is often not updated but if there is some new feature or change is introduced then it has to be updated accordingly.

**3. Test Scenario:**

It explains what to test in the application. For example, “login” test scenario for the login page of the application. Many test cases can be prepared from this scenario.

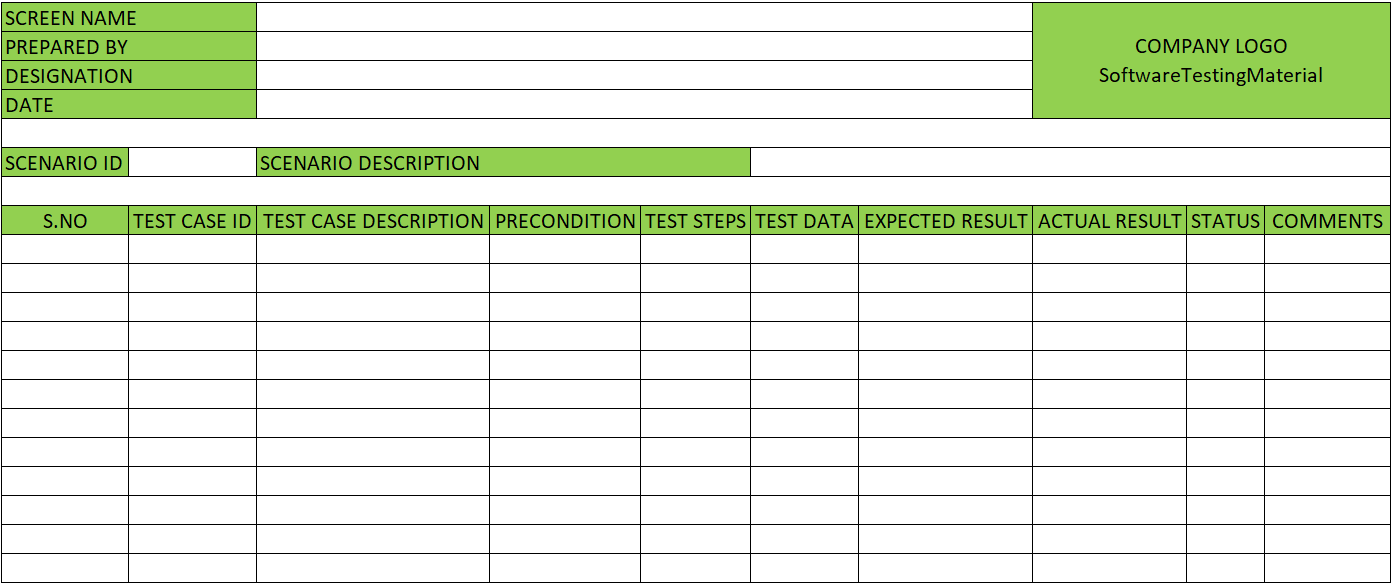
**Test Scenario Template:**



**4. Test Case Document:**

A test case is a set of actions with detailed steps and the expected results.

**Test Case Template:**



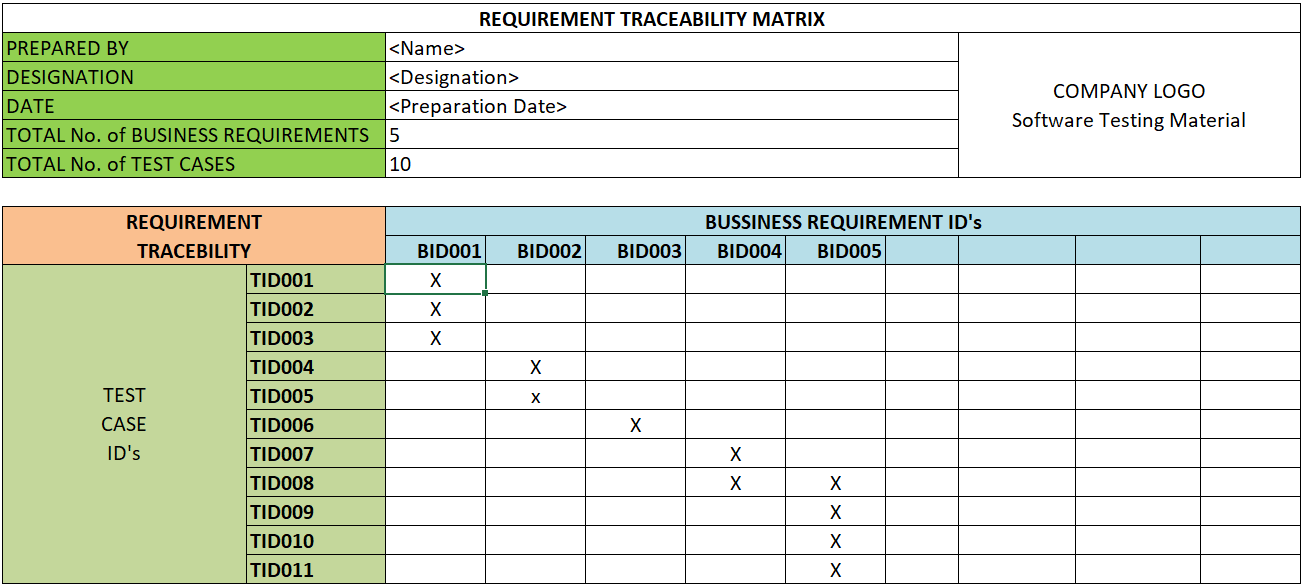
**Test Case vs. Test Scenario:**

* A Test Case is a set of actions executed to verify a particular feature or functionality of your software application
* A Test Scenario is defined as any functionality that can be tested
* Test scenarios are derived from test artifacts like BRS, SRS, etc.
* A test case is mostly derived from test scenarios. Multiple Test case can be derived from a single Test Scenario
* Both Test case and Test Scenario are indispensable in STLC process

Example of Test Cases: Check system behavior when valid email id and password is entered.

Example of Test Scenario: Check the Login Functionality

**5. Requirement Traceability Matrix:**



It contains tables that shows the relationship between the requirements and the test cases.

It shows which requirement is implemented by which test cases and which test cases are mapped to which specific requirements.

**6. Test Data:**

**Test data is the data that is used by the testers to run the test cases. Whilst running the test cases, testers need to enter some input data. To do so, testers prepare test data. It can be prepared manually and also by using tools.**

**For example, to test a basic login functionality having user id, password fields. We need to enter some data in the user id and password fields. So we need to collect some test data.**

**7. Test Reports:**

**Test Summary Report (TSR) or Test Closure Report: As the name suggests it is summary of your testing phase. The document provides the snapshot of testing efforts so far. TSR is prepared by Test Leader/ Manager when testing is completed and sent to Client or higher management team to review the testing efforts by the testing team against the Test Plan.**

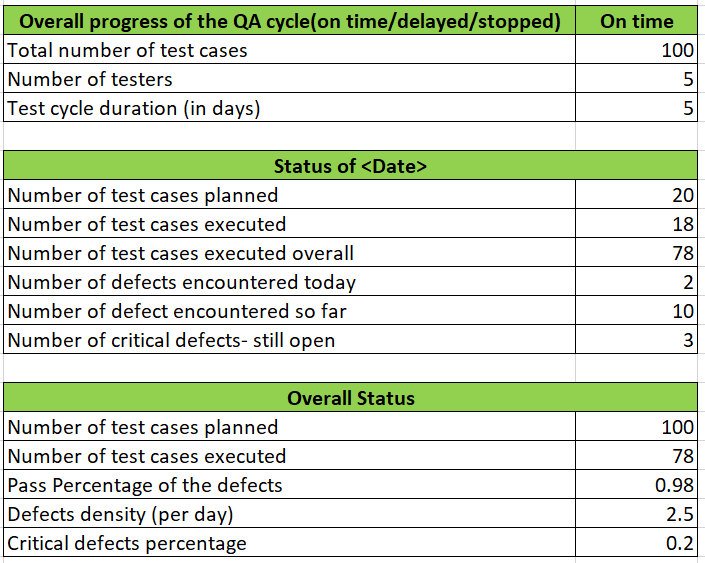
**Reporting test execution results is very important part of testing, whenever test execution cycle is complete, tester should make a complete test results report which includes the Test Pass/Fail status of the test cycle.**

**If manual testing is done then the test pass/fail result should be captured in an excel sheet and if automation testing is done using automation tool then the HTML or XML reports should be provided to stakeholders as test deliverable.**

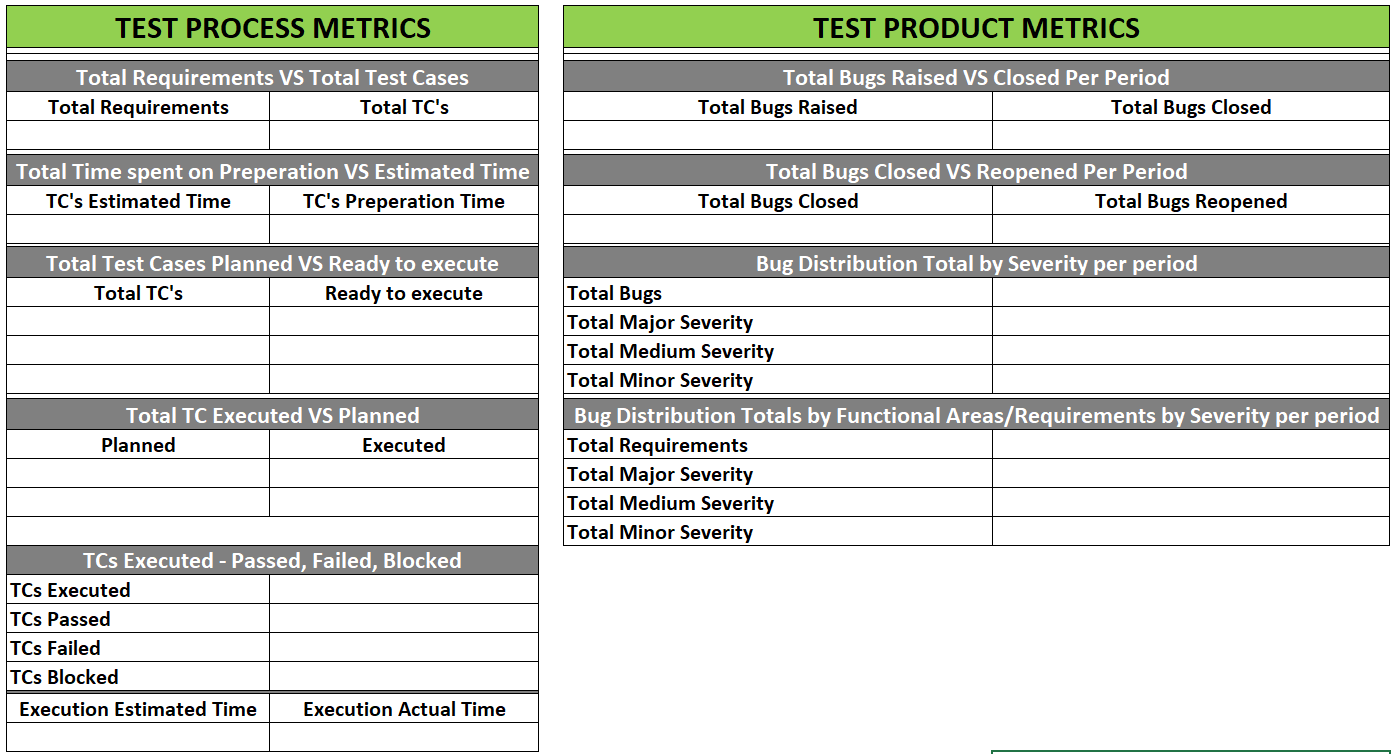
**Test Log: Is used by the test team to record what occurred during test execution.**

**Test Incident Report: describes any event that occurs during the test execution that requires further investigation. It might or might not be required to fix an incident, depending on whether it is a defect or just a mistake or some environmental issue. Incident needs to be investigated and based on the investigation the incident can be promoted to a defect. Most often it turns out to be a defect but sometimes it might occur due to different factors such as human mistake, missing or obscure documented requirement, environment issue such as no response from back-end server causing intermittent unexpected behavior or error.**

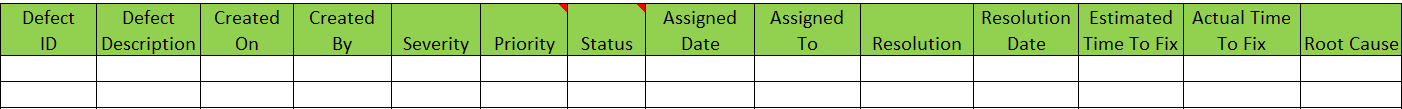
Test Status Report:



**Test Metrics:**



**8. Defect Tracking Log / Defect Triage Template:**



**Bug Life Cycle or Defect Life Cycle:**

**What is a Defect?**

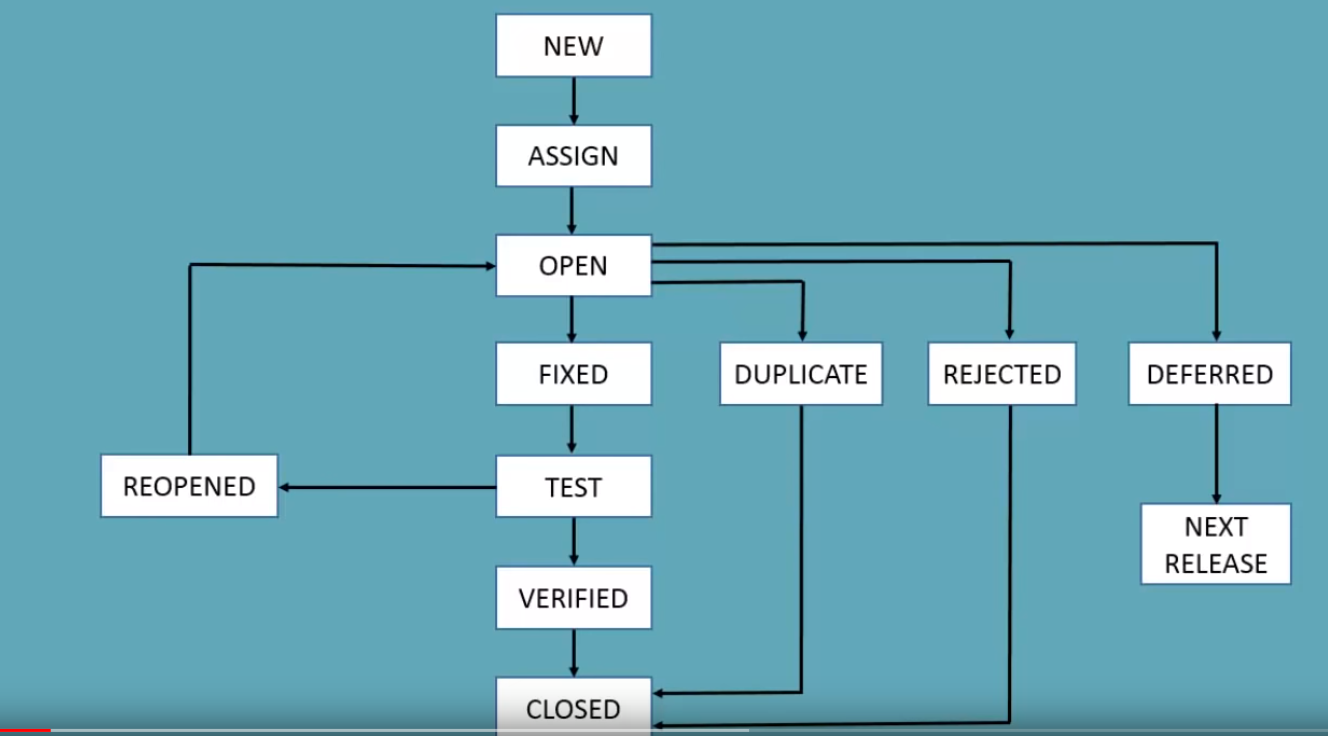
**The variation between the actual results and expected results is known as defect. If a developer finds an issue and corrects it by himself in the development phase then it’s called a defect. Click here for more details.**

**What is a Bug?**

**If testers find any mismatch in the application/system in testing phase then they call it as Bug.**

**Bug can be defined as an abnormal behavior of the software. But starts when defect is found and ends when a defect is closed, after ensuring it is not reproduced.**

**Bug status may vary depending on the tools (QC, JIRA, etc.) used and the process followed by the organization.**



***NEW:* When a tester finds a bug and posts it for the first time, the status of the defect is NEW.**

***ASSIGNED:* When the defect is assigned to the Development Team, the status changes to ASSIGNED.**

***OPEN:* When the Development Team starts working on the defect fix, the status changes to OPEN.**

***FIXED:* When developer makes necessary code changes and verifies the change, the status changes to FIXED.**

***TEST:* If the defect is fixed and ready to test, the status is TEST.**

***VERIFIED:* The tester retests the test, if there is no bug defected after retesting the software, the bug is fixed and the status assigned is VERIFIED.**

***CLOSED:* If the bug no longer exists after fixing it, the status is changed to CLOSED.**

***REOPEN:* If the defect remains the same after retesting, the tester changes the status to REOPEN.**

***DUPLICATE:* If the defect is repeated, the status is changed by development team to DUPLICATE.**

***DEFERED:* If the bug found during the end of the release and it is minor and not important to fix immediately, or if it is not related to current build, if it is expected to get fixed in the next release or if the customer is thinking to change the requirement, the status of the bug is changed to DEFERRED and it will be fixed in the next release.**

***REJECTED:* If the system is working according to the specifications and the bug is just due to some misinterpretations, the team lead may change the status to REJECTED.**

***NEED MORE INFORMATION:* If the developer is unable to reproduce the bug as per the steps provided by the tester, he/she can change the status to NEED MORE INFORMATION.**

**The two main parameters that form the basis for effective defect tracking and resolution are:**

* **Defect priority in Testing: indicates the urgency with which defect would need to be fixed. Priority is initially assigned by tester, but it is actually defined by Product Manager.**
* **Defect Severity in Testing: indicates the extent of impact of defect on the system. QA defines it.**

***Priority is associated with scheduling and severity is associated with standards.***

***Defect priority is generally set by the product manager in a “defect triage” meeting.* Defect Triage: is the process where each bug is prioritized based on its severity, frequency, risk, etc.**

**Priority of defects can be classified as follows:**

**Priority #1 Immediate / Critical (P1): Has to be fixed immediately within 24 hours. Like when an entire functionality is blocked and no testing can proceed as result of this.**

***All the critical severity defects fall under this category.***

**Priority #2 High (P2): These defects should be resolved before the release can be made. These are like a feature that is not usable. *All the Major severity defects fall into this category.***

**Priority #3 Medium (P3): Should be fixed after all the serious bugs are fixed. Like cosmetic errors such as expecting the right error message during the failure.**

***All the Minor severity defects fall into this category.***

**Priority #4 Low (P4): Like typing errors.**

***All the Low severity defects fall into this category.***

**Example for all: In the email service provider, if the system crashes and throws error message instead of logging in (Critical Defect), if the user can’t add more than one recipient in the CC section (Major Defect), one of the links in the terms and conditions of the website is not working (Minor Defect), there is a spelling mistake in the terms and conditions (Low Defect).**

**Different Levels:**

**High Priority High Severity: System crashes after you made the payment.**

**High Priority Low Severity: The logo of the company is wrong.**

**Low Priority High Severity: Defect found in the beta version of a new feature.**

**Low Priority Low Severity: The policy of a website has a spelling mistake.**

**JIRA**

Developed by Atlassian, JIRA is used for bug tracking, issue tracking and agile project management.

Creating defect/issue on JIRA:

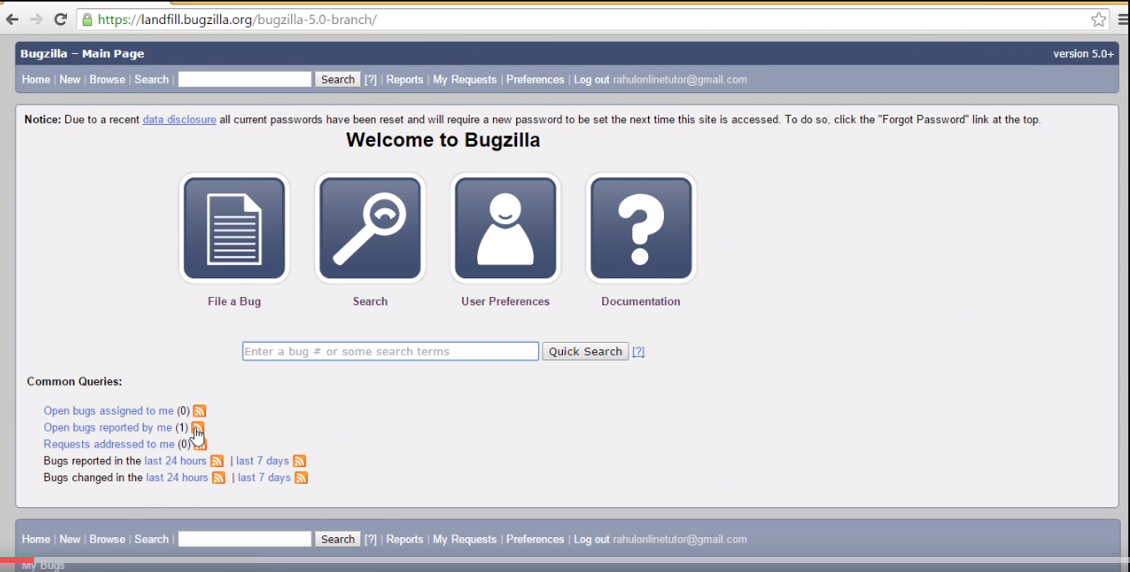
Click on create > select project name > select issue type i.e. bug > put something in summary > Choose Fix Version, it is the version in which you want the defect to be fixed > Choose Affects Versions, which is the version that is affected by the defect > select the priority (blocker, critical, major, trivial) > select estimates > select ‘linked issues’ if this defect is related to a defect observed in the past > assign ‘labels’ if you want to categorize issues for future reference > in the ‘Description’ section you can include the ‘Test Environment, Testing steps to reproduce, any assumptions, type of testing, version of software where it was working, Expected result, Actual Result’ > you can attach a screen shot or logs in the ‘Attachment’ section > add the name of ‘Reporter’ > click on ‘Create’.

Developer can: click on ‘In Progress’ in the upper menu if he has started working on it, or on ‘Workflow’ from the same menu and choose ‘Needs information’ or ‘resolved’.

Tester can: can choose Closed from the ‘Workflow’ if the issue is fixed and testing is passed, or can choose ‘in test’ if testing is in progress.

**Bugzilla – Defect Management Tool**

Bugzilla is an open source web based bug tracking and testing tool. There are other defect management tools like HP’s Quality Center and JIRA which are paid services.



***To report a bug:*** on Main Page click on File a Bug > select a product on which you want to log a defect > select the component of the product that has the bug, the version, summary, description, severity (blocker, critical, major, normal, minor, trivial, enhancement), hardware, OS > summary of the bug > description (steps to reproduce, expected results, actual results, timestamp > attachment (attach a file – i.e. screenshot) > submit bug.

You can also choose to edit a bug that is already reported and click on more details and assign it to someone, add a deadline, the QA contact details, keyword (to make future searches easy), etc.

When developer starts working on the bug, he will choose “in progress” as the status or he can choose “resolved” and then in the sub-status list he can choose either fixed, invalid, duplicate, wontfix, worksforme, moved.

In the bottom left corner, you can see a list of quick links to the bugs reported/assigned to you.

***To close a bug:*** when developer reports that he has fixed the bug, tester can then go and do the retesting and regression testing. If the issue is really resolved, he can then go to the defect and change the status to “verified” and choose “fixed” from the sub-status list.

In the general preferences section of Bugzilla, we can choose things like ‘display possible duplicates when reporting a new bug’. In the email preferences section, you can configure email settings i.e. who will receive emails for what. But setting up these preferences and configuration of Bugzilla is done by the IT department of the company you work for.

**Core Java:**

[] square brackets or brackets

<> angle brackets

{} braces

() parentheses, singular form is parenthesis

Java Keywords: public, static, void. We cannot use these keywords inside the blocks.

**Main:** is the name of method. The execution starts from the main method.

**Variables:** variables store data for processing. A variable is given a name (or identifier), such as area, age, height, and the like. The name uniquely identifies each variable, assigning a value to the variable and retrieving the value stored. Variables have types. Some examples are as follows:

**int:** for integers (whole numbers) such as 123 and -456.

**double:** for floating-point or real numbers with optional decimal points and fractional parts in fixed or scientific notations, such as 3.12 and -4.23

**string:** for text such as “Hello”. A String is an object that represents a sequence of characters. Text strings are enclosed within double quotes. We can also define an empty string. For example: string str =””;

**char:** stands for character and holds a single character.

**boolean:** it has two possible values: true and false. This data type is used for simple flags that track true/false conditions. For example: Boolean online = true.

You can declare a variable of a type and assign it a value. Example: String name = “David”; this creates a variable called name of type String, and assigns it the value “David”.

Note: we need to provide semicolon at the end of every step, except for while and for loops.

We can use a comma-separated list to declare more than one variable of the specified type. Ex: int a=42, b=1.

**The math operators:** Java provides a rich set of operators to use in manipulating variables. A value used on ehtier side of an operator is called an operand. For example: in the expression below, numbers 6 and 3 are operands of the plus operator. Int x = 6+3;

Java arithmetic operators:

**Addition +** : the + operator adds together two values, such as two constants, a constant and a variable, or a variable and a variable. Below are a few examples:

Int sum1 = 50 + 10;

Int sum2 = sum1+66;

Int sum3 = sum2 + sum2;

**Subtraction –** : The – operator subtracts one value from another. Examples are the same as in addition.

Note: just like in algebra, we can use both the operations in a single line. For example: int val = 10 + 5 – 2;

**Multiplication \*** : The \* operator multiples two values. Examples are the same as those of addition.

**Division /** : The / operator divides one value by another. Examples are the same as those of addition. If we use int as data type, the result of the division equation will be a whole number, we can use double to retrieve a value with a decimal point.

**Modulo %** : The modulo (or remainder) math operation performs an integer division of one value by another, and returns the remainder of that division. The operator for the modulo operation is the percentage (%) character.

Example: int value = 23; int res = value % 6; //res is 5

Dividing 23 by 6 returns a quotient of 3, with a remainder of 5. Thus, the value of 5 is assigned to the res variable.

Note in the examples above that arithmetic operators are used in mathematical expressions in the same way that they are used in algebraic equations.

**Assignment operators:**

The assignment operator (=) assigns a value to a variable. Int value =5; (this assigned the value 5 to a variable called value of type int).

Java provides a number of assignment operators to make it easier to write a code:

Addition and assignment (+=): int num1 = 4; int num2 = 8; num2 +=num1; //num2=num2+num1; //num2 is 12 and num 1 is 4

Subtraction and assignment (–=): int num1=4; int num2 = 8; num2 –= num1; //num2 = num2 – num1; //num2 is 4 and num1 is 4.

Similarly, Java supports multiplication and assignment (\*=), division and assignment (/=), and remainder and assignment (%=).

**String Concatenation:**

The + (plus) operator between strings adds them together to make a new string. The process is called concatenation. The resulted string is the first string put together with the second string. Example:

String firstName, lastName;

firstName = “David”;

lastName=”Williams”;

System.out.println(“My name is”+firstName+” “ +lastName); //prints: My name is David Williams

**Getting User Input:**

While Java provides many different methods for getting user input, the **Scanner** object is the most common, and perhaps the easiest to implement.

Import the Scanner class to use the Scanner object, as seen here: import java.util.Scanner;

In order to use the Scanner class, create an instance of the class by using the following syntax:

Scanner myVar=new Scanner (System.in);

You can now read in different kinds of input data that the user enters. Below are some methods that are available through the scanner class:

Read a byte – nextByte()

Read a short – nextShort()

Read an int – nextInt()

Read a float – nextFloat()

Read a long – nextLong()

Read a double – nextDouble()

Read a Boolean – nextBoolean()

Read a complete line – nextLine()

Read a word – next()

Example of a program used to get user input:

import java.util.Scanner;

class MyClass {

public static void main(String[] args) {

Scanner myVar= new Scanner (System.in);

System.out.println(myVar.nextLine());

}}

This will wait for the user to input something and print that input.

Example: printing name:

Scanner output = new Scanner (System.in);

System.out.print("What is your first name: ");

String fname = output.next();

System.out.print("What is your your last name: ");

String lname = output.next();

System.out.print("what is your date of birth: ");

String dob = output.next();

System.out.println("Hello \n"+fname+" "+lname);

Adding numbers:

Scanner input = new Scanner (System.in);

System.out.print("Input the first number: ");

int num1 = input.nextInt();

System.out.print("Input the second number: ");

int num2 = input.nextInt();

int sum = num1 + num2;

System.out.println();

System.out.println("Sum: "+sum);

Multiplication table up to 10:

Scanner in = new Scanner(System.in);

System.out.print("Input a number: ");

int num1 = in.nextInt();

for (int i=0; i< 10; i++){

System.out.println(num1 + " x " + (i+1) + " = " +

(num1 \* (i+1)));

}

**Decision Making**

**Conditional statements** are used to perform different actions based on different conditions.

The if statement is one of the most frequently used conditional statements.

If the if statement's condition expression evaluates to true, the block of code inside the if statement is executed. If the expression is found to be false, the first set of code after the end of the if statement (after the closing curly brace) is executed.

Syntax:

if (condition) {

//Executes when the condition is true

}

Any of the following comparison operators may be used to form the condition:

< less than

> greater than

!= not equal to

== equal to

<= less than or equal to

>= greater than or equal to

For example:

int x = 7;

if(x < 42) {

System.out.println("Hi");

}

Note: Remember that you need to use two equal signs (==) to test for equality, since a single equal sign is the assignment operator.

**if...else Statements**

An if statement can be followed by an optional else statement, which executes when the condition evaluates to false. For example:

int age = 30;

if (age < 16) {

System.out.println("Too Young");

} else {

System.out.println("Welcome!");

} //Outputs "Welcome!"

As age equals 30, the condition in the if statement evaluates to false and the else statement is executed.

**Nested if Statements:**

You can use one if-else statement inside another if or else statement. For example:

int age = 25;

if(age > 0) {

if(age > 16) {

System.out.println("Welcome!");

} else {

System.out.println("Too Young");

}

} else {

System.out.println("Error");

} //Outputs "Welcome!"

You can nest as many if-else statements as you want.

**else if Statements:**

Instead of using nested if-else statements, you can use the else if statement to check multiple conditions.

For example:

int age = 25;

if(age <= 0) {

System.out.println("Error");

} else if(age <= 16) {

System.out.println("Too Young");

} else if(age < 100) {

System.out.println("Welcome!");

} else {

System.out.println("Really?");

} //Outputs "Welcome!"

The code will check the condition to evaluate to true and execute the statements inside that block. You can include as many else if statements as you need.

**Logical Statements with Logical Operators:**

**The AND Operator:**

Logical operators are used to combine multiple conditions. Let's say you wanted your program to output "Welcome!" only when the variable age is greater than 18 and the variable money is greater than 500.

One way to accomplish this is to use nested if statements:

if (age > 18) {

if (money > 500) {

System.out.println("Welcome!");

}

}

However, using the AND logical operator (&&) is a better way:

if (age > 18 && money > 500) {

System.out.println("Welcome!");

}

If both operands of the AND operator are true, then the condition becomes true.

**The OR Operator:**

The OR operator (||) checks if any one of the conditions is true. The condition becomes true, if any one of the operands evaluates to true. For example:

int age = 25;

int money = 100;

if (age > 18 || money > 500) {

System.out.println("Welcome!");

} //Outputs "Welcome!"

The code above will print "Welcome!" if age is greater than 18 or if money is greater than 500.

**The NOT Operator:**

The NOT (!) logical operator is used to reverse the logical state of its operand. If a condition is true, the NOT logical operator will make it false. Example:

int age = 25;

if(!(age > 18)) {

System.out.println("Too Young");

} else {

System.out.println("Welcome");

} //Outputs "Welcome"

!(age > 18) reads as "if age is NOT greater than 18".

**The Switch Statement:**

A switch statement tests a variable for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each case.

Syntax:

switch (expression) {

case value1 :

//Statements

break; //optional

case value2 :

//Statements

break; //optional

//You can have any number of case statements.

default : //Optional

//Statements

}

- When the variable being switched on is equal to a case, the statements following that case will execute until a break statement is reached.

- When a break statement is reached, the switch terminates, and the flow of control jumps to the next line after the switch statement.

- Not every case needs to contain a break. If no break appears, the flow of control will fall through to subsequent cases until a break is reached.

The example below tests day against a set of values and prints a corresponding message.

int day = 3;

switch(day) {

case 1:

System.out.println("Monday");

break;

case 2:

System.out.println("Tuesday");

break;

case 3:

System.out.println("Wednesday");

break;

} // Outputs "Wednesday"

You can have any number of case statements within a switch. Each case is followed by the comparison value and a colon.

The default Statement: A switch statement can have an optional default case.

The default case can be used for performing a task when none of the cases is matched. For example:

int day = 3;

switch(day) {

case 6:

System.out.println("Saturday");

break;

case 7:

System.out.println("Sunday");

break;

default:

System.out.println("Weekday");

} // Outputs "Weekday"

No break is needed in the default case, as it is always the last statement in the switch.

**Loop Statements:**

A loop statement allows to repeatedly execute a statement or group of statements.

**while Loops:**

A while loop statement repeatedly executes a target statement as long as a given condition is true.

Example:

int x = 3;

while(x > 0) {

System.out.println(x);

x--;

}

/\*

Outputs

3

2

1

\*/

The while loops check for the condition x > 0. If it evaluates to true, it executes the statements within its body. Then it checks for the statement again and repeats.

Notice the statement x--. This decrements x each time the loop runs, and makes the loop stop when x reaches 0.

Without the statement, the loop would run forever.

When the expression is tested and the result is false, the loop body is skipped and the first statement after the while loop is executed. Example:

int x = 6;

while( x < 10 )

{

System.out.println(x);

x++;

}

System.out.println("Loop ended");

/\*

6

7

8

9

Loop ended

\*/

Notice that the last print method is out of the while scope.

**for Loops:**

Another loop structure is the for loop. A for loop allows you to efficiently write a loop that needs to execute a specific number of times.

Syntax:

for (initialization; condition; increment/decrement) {

statement(s)

}

Initialization: Expression executes only once during the beginning of loop.

Condition: Is evaluated each time the loop iterates. The loop executes the statement repeatedly, until this condition returns false.

Increment/Decrement: Executes after each iteration of the loop.

The following example prints the numbers 1 through 5.

for(int x = 1; x <=5; x++) {

System.out.println(x);

}

This initializes x to the value 1, and repeatedly prints the value of x, until the condition x<=5 becomes false. On each iteration, the statement x++ is executed, incrementing x by one.

Notice the semicolon (;) after initialization and condition in the syntax.

You can have any type of condition and any type of increment statements in the for loop.

The example below prints only the even values between 0 and 10:

for(int x=0; x<=10; x=x+2) {

System.out.println(x);

} /\* 0 2 4 6 8 10 \*/

**Some examples:**

Print the index of a number in array:

int[] a = {12, 3, 4, 1, 5, 7, 9, 3};

int sum = 0;

for (int i=0; i<a.length; i++)

if (a[i]==1)

{

System.out.println(i);

}

Sum numbers in array:

int[] set = {1, 2, 3, 4, 5, 56, 8};

int sum = 0;

for(int i=0; i<set.length; i++)

{

sum=sum+set[i];

}

system.out.print(sum);

Find the number of r's in a string:

String a = "FerozFerozFeroz";

int rCount = 0;

for (int i=0; i<a.length(); i++)

if (a.charAt(i)=='r')

rCount++;

System.out.println("this word has " +rCount +" r(s) in it");

Find out how many spaces, numbers, letters and other characters are there in a string:

String all = "adsflkahdf;l ahsdfklahdflka hjdflkajhsd'l;cxvkh3w 489703596789fc6n,lq34598ysdf";

count(all);

}

private static void count(String all) {

char[] ch = all.toCharArray();

int letters = 0;

int numbers = 0;

int spaces = 0;

int others = 0;

for(int i=0; i<all.length(); i++)

{

if(Character.isLetter(ch[i])) {

letters++;

}

else if(Character.isDigit(ch[i])) {

numbers++;

}

else if(Character.isSpaceChar(ch[i])) {

spaces++;

}

else {

others++;

}

}

System.out.println("The string is: " +all);

System.out.println("The letters are: " +letters);

System.out.println("The spaces are: " +spaces);

System.out.println("The numbers are: " +numbers);

System.out.println("Others are: " +others);

Print all odd numbers from 1 to 100:

for (int i = 1; i < 100; i++) {

if (i % 2 != 0) {

System.out.println(i);

}

}

**do...while Loops:**

A do...while loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.

Example:

int x = 1;

do {

System.out.println(x);

x++;

} while(x < 5); /\* output: 1 2 3 4 \*/

Notice that the condition appears at the end of the loop, so the statements in the loop execute once before it is tested.

Even with a false condition, the code will run once. Example:

int x = 1;

do {

System.out.println(x);

x++;

} while(x < 0); //Outputs 1

Notice that in do…while loops, the while is just the condition and doesn't have a body itself.

**Loop Control Statements:**

The break and continue statements change the loop's execution flow. The break statement terminates the loop and transfers execution to the statement immediately following the loop.

Example:

int x = 1;

while(x > 0) {

System.out.println(x);

if(x == 4) {

break;

}

x++;

} /\* Outputs: 1 2 3 4 \*/

The continue statement causes the loop to skip the remainder of its body and then immediately retest its condition prior to reiterating. In other words, it makes the loop skip to its next iteration.

Example:

for(int x=10; x<=40; x=x+10) {

if(x == 30) {

continue;

}

System.out.println(x);

}

/\* Outputs

10

20

40

\*/

As you can see, the above code skips the value of 30, as directed by the continue statement.

**Arrays:**

An array is a collection of variables of the same type. When you need to store a list of values, such as numbers, you can store them in an array, instead of declaring separate variables for each number.

To declare an array, you need to define the type of the elements with square brackets.

For example, to declare an array of integers:

int[ ] arr; The name of the array is arr. The type of elements it will hold is int.

Now, you need to define the array's capacity, or the number of elements it will hold. To accomplish this, use the keyword *new*.

int[ ] arr = new int[5];

The code above declares an array of 5 integers and allocates memory for the values.

In an array, the elements are ordered and each has a specific and constant position, which is called an index.

To reference elements in an array, type the name of the array followed by the index position within a pair of square brackets. Example: arr[2] = 42; This assigns a value of 42 to the element with 2 as its index.

Note that elements in the array are identified with zero-based index numbers, meaning that the first element's index is 0 rather than one. So, the maximum index of the array int[5] is 4.

**Initializing Arrays:**

Java provides a shortcut for instantiating arrays of primitive types and strings. If you already know what values to insert into the array, you can use an array literal.

Example of an array literal:

String[ ] myNames = { "A", "B", "C", "D"};

System.out.println(myNames[2]); // Outputs "C"

Place the values in a comma-separated list, enclosed in curly braces.

The code above automatically initializes an array containing 4 elements, and stores the provided values.

Sometimes you might see the square brackets placed after the array name, which also works, but the preferred way is to place the brackets after the array's data type.

**Array Length:**

You can access the length of an array (the number of elements it stores) via its length property. Example:

int[ ] intArr = new int[5];

System.out.println(intArr.length); //Outputs 5

Don't forget that in arrays, indexes start from 0. So, in the example above, the last index is 4.

**Calculate sum of all elements in an Array:**

The for loop is the most used loop when working with arrays, as we can use the length of the array to determine how many times to run the loop.

int [ ] myArr = {6, 42, 3, 7};

int sum=0;

for(int x=0; x<myArr.length; x++) {

sum += myArr[x];

}

System.out.println(sum); // 58

In the code above, we declared a variable sum to store the result and assigned it 0.

Then we used a for loop to iterate through the array, and added each element's value to the variable.

The condition of the for loop is x<myArr.length, as the last element's index is myArr.length-1.

**Enhanced for Loop:**

The enhanced for loop (sometimes called a "for each" loop) is used to traverse elements in arrays.

The advantages are that it eliminates the possibility of bugs and makes the code easier to read. Example:

int[ ] primes = {2, 3, 5, 7};

for (int t: primes) {

System.out.println(t);

} /\* output: 2 3 5 7 \*/

The enhanced for loop declares a variable of a type compatible with the elements of the array being accessed. The variable will be available within the for block, and its value will be the same as the current array element.

So, on each iteration of the loop, the variable t will be equal to the corresponding element in the array.

Notice the colon after the variable in the syntax.

**Multidimensional Arrays:**

Multidimensional arrays are arrays that contain other arrays. The two-dimensional array is the most basic multidimensional array. To create multidimensional arrays, place each array within its own set of square brackets. Example of a two-dimensional array: int[ ][ ] sample = { {1, 2, 3}, {4, 5, 6} };

This declares an array with two arrays as its elements.

To access an element in the two-dimensional array, provide two indexes, one for the array, and another for the element inside that array.

The following example accesses the first element in the second array of *sample*.

int x = sample[1][0];

System.out.println(x); // Outputs 4

The array's two indexes [][] are called row index and column index respectively.

int a[][] = new int [2][1]; //a longer way of initializing an array that has 3 rows and 2 columns ([row][column]):

a[0][0]=2;

a[0][1]=4;

a[0][2]=5;

a[1][0]=3;

a[1][1]=4;

a[1][2]=7;

system.out.println(a.[1][0]); //print the value of first column in second row.

//or we can create the array in a simple way as follows:

// Int b[][] = {{2, 4, 5}, {3, 4, 7} , {5, 2, 1}};

// System.out.println(b[2][1]);

For (int i=0; i<2; i++) //row

{

For (int j=0; j<3;j++) //column

{

System.out.println(a[i][j]);

}}}}

//Find and print the minimum number from the following matrix:

Int abc[][] = {{2, 4, 5}, {3, 4, 7} , {1, 2, 9}};

Int min = abc[0][0];

For (int i=0;i<3;i++)

{

For (int j=0; j<3;j++)

{

If (abc[i][j]<min)

{

Min=abc[i][j];

}}}

System.out.println(min);

}}

//Find the minimum number from the following matrix, then identify the column of that minimum number and then get the maximum number of the identified column:

int abc[][]={{2,4,5},{3,2,10},{1,2,0}};

int min=abc[0][0];

int mincoloumn = 0;

for(int i=0;i<3;i++)

{

for(int j=0;j<3;j++)

{

if(abc[i][j]<min)

{

min=abc[i][j];

mincoloumn=j;

}}}

int max=abc[0][mincoloumn];

int k = 0;

while(k<3)

//We go with while loop because we have a single iteration (like in this example we iterate through one column only), if we had multiple iterations, we would have used for loop.

{

if(abc[k][mincoloumn]>max)

{

max=abc[k][mincoloumn];

}

k++;

}

System.out.println(max);

}}}

//Print the current date and current time:

Note: Java has a class called ‘Date’ in the Util package. To use the date class, we need to import the Util package by simply creating object for the ‘Date’ class and then using the IDE suggestion to fix the error which will import the package.

Date d = new Date();

System.out.println(d.toString()); //we convert the date object to string to be able to visually see it in output.

//Print current date and time in mm//dd//yyyy HH:MM:SS format:

Java has another class named SimpleDateFormat, we can create object for it to be able to print date and time in our desired format.

Date d = new Date();

SimpleDateFormat sdf= new SimpleDateFormat(“M/d/yyyy hh:mm:ss”);

System.out.println(sdf.format(d));

//Retrieving the system date as per our requirements:

Calendar cal = Calendar.getInstance();

SimpleDateFormat sd= new SimpleDateFormat(“M/d/yyyy hh:mm:ss”);

System.out.println(sd.format(cal.getTime())); //prints the date and time in the format specified above

System.out.println(cal.get(Calendar.DAY\_OF\_MONTH)); //prints today’s day of the month

**Java Collections Framework:**

The Java platform includes a collections framework. A collection is an object that represents a group of objects. The Java Collections Framework is a collection of interfaces and classes which helps in storing and processing the data efficiently. This framework has several useful classes which have tons of useful functions which makes a programmer task super easy. All collection interfaces are enclosed in java.util package.

**List Collection** *(interface)*

A List is an ordered Collection (sometimes called a sequence). Lists may contain duplicate elements. Elements can be inserted or accessed by their position in the list, using a zero-based index. Classes that implement list interface: Arraylist, LinkedList, Vector.

**a) Arraylist:** ArrayList class implements List interface and it is based on an Array data structure. It is widely used because of the functionality and flexibility it offers. The limitation with array is that it has a fixed length so if it is full you cannot add any more elements to it, likewise if there are number of elements gets removed from it the memory consumption would be the same as it doesn’t shrink. But ArrayList can dynamically grow and shrink after addition and removal of elements. Apart from these benefits ArrayList class enables us to use predefined methods of it which makes our task easy.

ArrayList<String> alist=new ArrayList<String>(); //like the way we create object for any normal classes.

alist.add("Steve");

alist.remove("Steve");

alist.add(0, "Rahul") //Adding element "Rahul" at first position/ zeroth index

alist.remove(1); //Removes Second element from the List

system.out.println(alist); //prints all contents of ArrayList

system.out.println(alist.get(2)); //prints the value of third index

system.out.println(alist.contains(“Rahul”)); //the argument returns Boolean, so it will print True.

system.out.println(alist.indexof(“Rahul”)); //prints the index of Rahul.

system.out.println(alist.size); //prints the length of alist ArrayList.

**Array vs ArrayList in Java:**

Array is a fixed length data structure whereas ArrayList is a variable length Collection class. We cannot change length of array once created in Java but ArrayList can be changed.

We cannot store primitives in ArrayList, it can only store objects. But array can contain both primitives and objects in Java. Since Java 5, primitives are automatically converted in objects which is known as auto-boxing.

**To create an array:**

Int[] myArray = new int [10]; //this will create the int array of size 10.

Int[] myArray = {1, 2, 3, 4, 5, 6, 7, 8, 9., 10}; //this will also create an int array of size 10, but it will also initialize the values to the array from 1 through 10.

**To create arraylist:**

List<Integer> list=new ArrayList<>(); //you can also include the size of Arraylist in the ().

**Insert:**

myArray[3] = 7; //it doesn’t actually insert it, rather it assigns value 7 to index 3.

list.add(30); //adds 30 to the end of the list.

list.add(Integer.valueOf(10)); //storing Integer object

list.add(20); //Now compiler converts it into Integer.valueOf(20) which is object

**Get:**

X = myArray[3]; //gets item at index 3.

X = list.get(30); //gets item at index 3.

**Length:**

Len = myArray.length; //gets length

Len = list.size(); //gets length

**b) LinkedList:** LinkedList class is a doubly-linked list implementation of the List and Deque interfaces. The methods and results of LinkedList and ArrayList are almost identical. The insert and remove operations give good performance (O(1)) in LinkedList compared to ArrayList(O(n)). Hence if there is a requirement of frequent addition and deletion in application then LinkedList is a best choice. Search (get method) operations are fast in Arraylist (O(1)) but not in LinkedList (O(n)) so If there are less add and remove operations and more search operations requirement, ArrayList would be your best bet.

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

linkedlist.add("Item1");

**c) Vector:** Vector implements List Interface. Like ArrayList it also maintains insertion order but it is rarely used in non-thread environment as it is synchronized and due to which it gives poor performance in searching, adding, delete and update of its elements.

Vector vec = new Vector();

vec.addElement("Apple");

vec.remove("Harry");

ArrayList and Vector both use Array as a data structure internally. However there are few differences in the way they store and process the data:

Synchronization: ArrayList is non-synchronized which means multiple threads can work on ArrayList at the same time. For e.g. if one thread is performing an add operation on ArrayList, there can be an another thread performing remove operation on ArrayList at the same time in a multithreaded environment while Vector is synchronized. This means if one thread is working on Vector, no other thread can get a hold of it. Unlike ArrayList, only one thread can perform an operation on vector at a time.

Resize: Both ArrayList and Vector can grow and shrink dynamically to maintain the optimal use of storage, however the way they resized is different. ArrayList grow by half of its size when resized while Vector doubles the size of itself by default when grows.

Performance: ArrayList gives better performance as it is non-synchronized. Vector operations gives poor performance as they are thread-safe, the thread which works on Vector gets a lock on it which makes other thread wait till the lock is released.

**Set Collection** *(interface)*

A set is a collection that cannot contain duplicate elements. The classes that implement set interface are HashSet, LinkedHashSet and TreeSet.

**a) HashSet:** This class implements the set interface. HashSet doesn’t maintain any order, the elements would be returned in any random order. HashSet doesn’t allow duplicates. If you try to add a duplicate element in HashSet, the old value would be overwritten.

HashSet<String> hset = new HashSet<String>();

hset.add("Apple");

System.out.println(hset);

hset.remove("UK");

System.out.println(hset.isEmpty());

System.out.println(hset.size());

Iterator<String> i=hset.iterator(); //iterator is an interface. We use it here to iterate between elements in the set.

while(i.hasNext())

{

System.out.println(i.next());

} }

**b) LinkedHashSet:** This class is also an implementation of Set interface. It is similar to HashSet and TreeSet except the below mentioned differences:

* HashSet doesn’t maintain any kind of order of its elements.
* TreeSet sorts the elements in ascending order.
* LinkedHashSet maintains the insertion order. Elements gets sorted in the same sequence in which they have been added to the Set.

LinkedHashSet<String> lhset = new LinkedHashSet<String>();

lhset.add("Z");

**c) TreeSet:** TreeSet is similar to HashSet except that it sorts the elements in the ascending order while HashSet doesn’t maintain any order.

TreeSet<String> tset = new TreeSet<String>();

tset.add("ABC");

*To convert HashSet to a TreeSet:*

Set<String> tset = new TreeSet<String>(hset);

*To convert HashSet to an ArrayList:*

List<String> list = new ArrayList<String>(hset);

**Map Collection** *(interface)*

A Map is an object that maps keys to values. A map cannot contain duplicate keys. There are three main implementations of Map interfaces: HashMap, TreeMap, and LinkedHashMap.

**a) HashMap:** HashMap is a Map based collection class that is used for storing Key & value pairs, it is denoted as HashMap<Key, Value> or HashMap<K, V>. The key could be integer or string, the key is basically a reference to the value. This class makes no guarantees as to the order of the map. It is similar to the Hashtable class except that it is unsynchronized and permits nulls (null values and null key).

HashMap<Integer,String> hm=new HashMap<Integer,String>();

hm.put(0, "hello");

hm.put(1, "Goodbye");

hm.put(42, "morning");

hm.put(3, "evening");

System.out.println(hm.get(42));

hm.remove(42);

System.out.println(hm.get(42)); //since the value is removed, it should print Null

Set sn= hm.entrySet(); //converting HashMap into set.

Iterator it =sn.iterator(); //hashtable -pass table set collections

while(it.hasNext())

{

System.out.println(it.next()); //this won’t print, it will throw error, we have to follow the step below:

Map.Entry mp=(Map.Entry)it.next(); //this is how we separate the keys from values, so that we can print or pull out whether we want the key or the value as follows:

System.out.println(mp.getKey());

System.out.println(mp.getValue());

}}

**b) TreeMap:** implements Map interface similar to HashMap class. The main difference between them is that HashMap is an unordered collection while TreeMap is sorted in the ascending order of its keys. TreeMap is unsynchronized collection class which means it is not suitable for thread-safe operations until unless synchronized explicitly.

TreeMap<Integer, String> tmap = new TreeMap<Integer, String>();

tmap.put(1, "Data1");

**c) LinkedHashMap:** is a Hash table and linked list implementation of the Map interface, with predictable iteration order. This implementation differs from HashMap in that it maintains a doubly-linked list running through all of its entries. This linked list defines the iteration ordering, which is normally the order in which keys were inserted into the map (insertion-order).

* HashMap doesn’t maintain any order.
* TreeMap sort the entries in ascending order of keys.
* LinkedHashMap maintains the insertion order.

LinkedHashMap<Integer, String> lhmap = new LinkedHashMap<Integer, String>();

lhmap.put(22, "Abey");

**Difference between HashMap and HashTable:**

HashMap and Hashtable both classes implements java.util.Map interface

* HashMap is non-synchronized. This means if it’s used in multithread environment then more than one thread can access and process the HashMap simultaneously. Hashtable is synchronized. It ensures that no more than one thread can access the Hashtable at a given moment of time. The thread which works on Hashtable acquires a lock on it to make the other threads wait till its work gets completed.
* HashMap allows one null key and any number of null values. Hashtable doesn’t allow null keys and null values.
* HashMap implementation LinkedHashMap maintains the insertion order and TreeMap sorts the mappings based on the ascending order of keys. Hashtable doesn’t guarantee any kind of order. It doesn’t maintain the mappings in any particular order.
* Initially Hashtable was not the part of collection framework it has been made a collection framework member later after being retrofitted to implement the Map interface. HashMap implements Map interface and is a part of collection framework since the beginning.

*Note: the example of HashTable is the same as the script written above for HashMap, just change the name HashMap to HashTable and keep all of the remaining code as it is.*

**Printing unique number:**

public class collectiondemo {

public static void main(String[] args) {

int a[] ={ 4,5,5,5,4,6,6,9,4}; // Print unique number from the list //print the string in reverse

//We will use the Arraylist to solve this problem:

ArrayList<Integer>ab =new ArrayList<Integer>();

for(int i=0;i<a.length;i++)

{

int k=0;

if(!ab.contains(a[i]))

{

ab.add(a[i]);

k++;

for(int j=i+1;j<a.length;j++)

{

if(a[i]==a[j])

{

k++;

} }

// System.out.println(a[i]);

// System.out.println(k);

if(k==1)

System.out.println(a[i]+"is unique number");

} }

} }

**Object-Oriented Programming**

**Object-Orientation:**

Java uses Object-Oriented Programming (OOP), a programming style that is intended to make thinking about programming closer to thinking about the real world.

In OOP, each object is an independent unit with a unique identity, just as objects in the real world are.

An apple is an object; so is a mug. Each has its unique identity. It's possible to have two mugs that look identical, but they are still separate, unique objects.

Objects also have characteristics, which are used to describe them.

For example, a car can be red or blue, a mug can be full or empty, and so on. These characteristics are also called attributes. An attribute describes the current state of an object.

In the real world, each object behaves in its own way. The car moves, the phone rings, and so on.

The same applies to objects: behavior is specific to the object's type.

In summary, in object oriented programming, each object has three dimensions: identity, attributes, and behavior.

Attributes describe the object's current state, and what the object is capable of doing is demonstrated through the object's behavior.

**Classes:**

A class describes what the object will be, but is separate from the object itself.

In other words, classes can be described as blueprints, descriptions, or definitions for an object. You can use the same class as a blueprint for creating multiple objects. The first step is to define the class, which then becomes a blueprint for object creation.

Each class has a name, and each is used to define attributes and behavior.

**Methods:**

Methods define behavior. A method is a collection of statements that are grouped together to perform an operation. System.out.println() is an example of a method.

You can define your own methods to perform your desired tasks.

Let's consider the following code:

class MyClass {

static void sayHello() {

System.out.println("Hello World!");

}

public static void main(String[ ] args) {

sayHello();

}

} // Outputs "Hello World!"

The code above declares a method called "sayHello", which prints a text, and then gets called in main.

To call a method, type its name and then follow the name with a set of parentheses.

**Calling Methods:**

You can call a method as many times as necessary.

When a method runs, the code jumps down to where the method is defined, executes the code inside of it, then goes back and proceeds to the next line.

**Method Parameters:**

You can also create a method that takes some data, called parameters, along with it when you call it. Write parameters within the method's parentheses.

For example, we can modify our sayHello() method to take and output a String parameter.

class MyClass {

static void sayHello(String name) {

System.out.println("Hello " + name);

}

public static void main(String[ ] args) {

sayHello("David");

sayHello("Amy");

}

} // Hello David // Hello Amy

The method above takes a String called name as a parameter, which is used in the method's body. Then, when calling the method, we pass the parameter's value inside the parentheses.

Methods can take multiple, comma-separated parameters.

The advantages of using methods instead of simple statements include the following:

- code reuse: write a method once, and use it multiple times, without having to rewrite the code each time.

- parameters: Based on the parameters passed in, methods can perform various actions.

**The Return Type:**

The return keyword can be used in methods to return a value. For example, we could define a method named sum that returns the sum of its two parameters.

static int sum(int val1, int val2) {

return val1 + val2;

}

Notice that in the method definition, we defined the return type before we defined the method name. For our sum method, it is int, as it takes two parameters of the type int and returns their sum, which is also an int.

Now, we can use the method in our main:

class MyClass {

static int sum(int val1, int val2) {

return val1 + val2;

}

public static void main(String[ ] args) {

int x = sum(2, 5);

System.out.println(x);

}

} // Outputs "7"

As the method returns a value, we can assign it to a variable.

When you do not need to return any value from your method, use the keyword void.

Notice the void keyword in the definition of the main method - this means that main does not return anything.

Let's create a method that takes two parameters of type int and returns the greater one, then call it in main:

public static void main(String[ ] args) {

int res = max(7, 42);

System.out.println(res); //42

}

static int max(int a, int b) {

if(a > b) {

return a;

}

else {

return b;

}

}

public static void main(String[ ] args) //This definition indicates that the main method takes an array of Strings as its parameters, and does not return a value.

A method can have one type of parameter (or parameters) and return another, different type. For example, it can take two doubles and return an int.

**Creating Objects (Using methods of classes):**

An object is an instance of a class. To use a class and it's methods, we need to declare an object of that class.

public class Animal {

void bark() {

System.out.println("Woof-Woof");

}

}

class MyClass {

public static void main(String[ ] args) {

Animal dog = new Animal();

dog.bark();

}

}

// Outputs "Woof-Woof"

Now, dog is an object of type Animal. Thus we can call its bark() method, using the name of the object and a dot.

The dot notation is used to access the object's attributes and methods.

**Defining Attributes:**

A class has attributes and methods. The attributes are basically variables within a class. Let's create a class called Vehicle, with its corresponding attributes and methods.

public class Vehicle {

int maxSpeed;

int wheels;

String color;

double fuelCapacity;

void horn() {

System.out.println("Beep!");

}

}

maxSpeed, wheels, color, and fuelCapacity are the attributes of our Vehicle class, and horn() is the only method.

You can define as many attributes and methods as necessary.

**Creating Objects:**

Next, we can create multiple objects of our Vehicle class, and use the dot syntax to access their attributes and methods.

class MyClass {

public static void main(String[ ] args) {

Vehicle v1 = new Vehicle();

Vehicle v2 = new Vehicle();

v1.color = "red";

v2.horn();

}

}

**Access Modifiers:**

Access Modifiers (like *Public)* are used to set the level of access. You can use access modifiers for classes, attributes, and methods.

For classes, the available modifiers are public or default (left blank/ no modifier), as described below:

**public**: The class is accessible by any other class.

**default**: The class is accessible only by classes in the same package.

The following choices are available for attributes and methods:

**default**: A variable or method declared with no access control modifier is available to any other class in the same package.

**public**: Accessible from any other class.

**protected**: Provides the same access as the default access modifier, with the addition that subclasses can access protected methods and variables of the superclass.

**private**: Accessible only within the declared class itself.

Public class arrayListexample{

Public int i=5;

Public class ArraysDemo{

Public static void main (String [] args) {

arrayListexample ab = new arrayListexample();

ab.i

**Hint:** It's a best practice to keep the variables within a class private. The variables are accessible and modified using Getters and Setters.

**Getters & Setters:**

**Getters** and Setters are used to effectively protect your data, particularly when creating classes. For each variable, the get method returns its value, while the set method sets the value.

Getters start with get, followed by the variable name, with the first letter of the variable name capitalized.

**Setters** start with set, followed by the variable name, with the first letter of the variable name capitalized.

Example:

public class Vehicle {

private String color;

// Getter

public String getColor() {

return color;

//

}

// Setter

public void setColor(String c) {

this.color = c;

}

}

The getter method returns the value of the attribute.

The setter method takes a parameter and assigns it to the attribute.

The keyword this is used to refer to the current object. Basically, this.color is the color attribute of the current object.

Once our getter and setter have been defined, we can use it in our main:

public static void main(String[ ] args) {

Vehicle v1 = new Vehicle();

v1.setColor("Red");

System.out.println(v1.getColor());

} //Outputs "Red"

Getters and setters are fundamental building blocks for encapsulation.

**Constructors:**

Constructors are special methods invoked when an object is created and are used to initialize them. A constructor can be used to provide initial values for object attributes.

- A constructor name must be same as its class name.

- A constructor must have no explicit return type. (and this is the difference between constructor and method)

Example of a constructor:

public class Vehicle {

private String color;

Vehicle() {

color = "Red";

}

}

The Vehicle() method is the constructor of our class, so whenever an object of that class is created, the color attribute will be set to "Red".

A constructor can also take parameters to initialize attributes.

public class Vehicle {

private String color;

Vehicle(String c) {

color = c;

}

}

You can think of constructors as methods that will set up your class by default, so you don’t need to repeat the same code every time.

**Using Constructors:**

The constructor is called when you create an object using the new keyword.

Example:

public class MyClass {

public static void main(String[ ] args) {

Vehicle v = new Vehicle("Blue");

}

}

This will call the constructor, which will set the color attribute to "Blue".

A single class can have multiple constructors with different numbers of parameters. The setter methods inside the constructors can be used to set the attribute values.

Example:

public class Vehicle {

private String color;

Vehicle() {

this.setColor("Red");

}

Vehicle(String c) {

this.setColor(c);

}

// Setter

public void setColor(String c) {

this.color = c;

}

}

The class above has two constructors, one without any parameters setting the color attribute to a default value of "Red", and another constructor that accepts a parameter and assigns it to the attribute.

Now, we can use the constructors to create objects of our class.

//color will be "Red"

Vehicle v1 = new Vehicle();

//color will be "Green"

Vehicle v2 = new Vehicle("Green");

***Java automatically provides a default constructor, so all classes have a constructor, whether one is specifically defined or not.***

Rahul Shetty’s videos: A constructor executes a block of code when an object is created for the class. People use constructor blocks to define variables or initiate some properties. Name of constructor should be the class name.

public class MyClass {

public constructDemo () //this is the constructor block

{

System.out.println(“I’m in the constructor”);

}

public constructDemo (int a, int b) //this is a parameterized constructor block

{

System.out.println(“I’m in the parameterized constructor”);

Int c = a+b;

System.out.println(c); //output will be 6

}

public constructDemo (String str) //this is a parameterized constructor block

{

System.out.println(str);

}

Public void getdata()

{

System.out.println(“I’m the method”);

}

public static void main(String[ ] args) {

constructorDemo cd= new constructorDemo(); //whenever an object is created, the constructor is invoked.

constructorDemo c= new constructorDemo(2, 4); //we need to parameterize object to call a parameterized constructor

constructorDemo cde= new constructorDemo(“HELLO”); //output will be HELLO

In the example above, we didn’t call any method, we just created an object for the class and it automatically invoked the constructor block. Internally or when we work only inside the class, we usually don’t define a constructor, because the compiler calls default constructor when a constructor block is not defined.

**Super keyword:**

Super keyword provides reference to the super or parent class. Suppose that we have two classes; parentDemo and childDemo as follows:

Public class parentDemo{

String name = “Rahul”;

Public static void main (String [] args) {

}

Public class childDemo extends parentDemo {

String name =”QAClickAcademy”;

Public void getStringdata()

{

System.out.println(name); //this will print the local variable, because preference is always given to child class.

System.out.println(super.name); //because the variable is duplicated in child and parent classes, we need to use the super keyword to be able to avoid the local variable and call the method from parent class.

}

Public static void main (String [] args) {

ChildDemo cd = new childDemo();

cd.getStringdata();

}}

When there is a common method in parent and child classes:

Public class parentDemo{

String name = “Rahul”;

Public void getData() //this is a common method between child and parent class

{

System.out.println(“I’m in parent class”);

}

Public static void main (String [] args) {

}

Public class childDemo extends parentDemo {

String name =”QAClickAcademy”;

Public void getStringdata()

{

System.out.println(name); //this will print the local variable, because preference is always given to child class.

System.out.println(super.name); //because the variable is duplicated in child and parent classes, we need to use the super keyword to be able to avoid the local variable and call the method from parent class.

}

Public void getData() //this is a common method between child and parent class

{

Super.getData(); //we need this statement when there is a duplicate version of method in child and parent classes. This statement will help invoke the method of parent class.

System.out.println(“I’m in child class”);

}

Public static void main (String [] args) {

ChildDemo cd = new childDemo();

cd.getStringdata();

cd.getData(); //preference is given to the local method, so it will print: I’m in child class. When we use the super keyword as shown above, it will print ‘I’m in parent class’ as well.

}}

Super keyword with constructors:

Public class parentDemo{

Public parentDemo() //constructor

{

System.out.println(“parent class constructor”);

}

Public static void main (String [] args) {

}

Public class childDemo extends parentDemo {

Public childDemo()

{

Super(); //this should always be at first line. It will first invoke the parentDemo’s then the childDemo’s constructor.

System.out.println(“Child class constructor”);

}

Public static void main (String [] args) {

childDemo cd = new childDemo();

}}

**This keyword:**

Suppose that we define one global variable (class level variable) and also a method variable (local variable) with the same name as in the following example:

Public class thisDemo{

Int a=2; //this is global variable

Public void getData()

{

Int a=3; //this is local variable

Int b = a.this.a;

System.out.println(a); //this will print 3.

System.out.println(this.a); //this will print 2. //This keyword refers to current object (object that has class level scope).

System.out.println(b);

Public static void main (String [] args) {

thisDemo td = new thisDemo();

td.getData();

}

**Final keyword:**

Use the final keyword to mark a variable constant, so that it can be assigned a value only once.

class MyClass {

public static final double PI = 3.14;

public static void main(String[ ] args) {

System.out.println(PI);

}

}

PI is now a constant. Any attempt to assign it another value will cause an error.

Methods and classes can also be marked final. This serves to restrict methods (final void getData(){}) so that they can't be overridden and classes (final class finalDemo) so that they can't be extended.

**Try Catch Mechanism:**

When we suspect or expect that a block of script may throw an error or fail, but we want the execution to continue, then we put the script into try block so that upon failure the control goes to the catch block. For example, a website generates a pop up about specials, but that pop up is not always generated, we get the pop up only in some special days. So we know that if there is no special going on, there will be no pop up and our script for handling the popup will fail, therefore we put it in the try block, and in the catch block we can print the message – i.e. there is no special available today.

Note that one try can be followed by multiple catch blocks. And there shouldn’t be a different block between try and catch.

Public class exceptionsDemo{

Public static void main (String[] args) {

int b=7;

int c=0;

try

{

int k=b/c;

system.out.println(k);

}

Catch (ArithmeticException et) //There are many exceptions, each exception is for a specific type of error. If an arithmetic failure happens, the control will come to this block which addresses arithmetic exceptions.

{System.out.println(“caught arithmetic exception”); //control will come here because 7/0 throws arithmetic error.

}

Catch (Exception e) //this is the parent of exceptions and addresses all exceptions/errors.

{ System.out.println(“I caught an error”);

} //the above is how the catch blocks talk to each other in identifying and addressing the exceptions.

**Finally:**

Used to execute the block irrespective of script failing or not. Finally is used with try. For example, we want to delete cookies at the end of our test, no matter if the test fails at the middle or not, so we put our script for deleting cookies inside the finally block. That way, if the script passes or fails at any point, we will still be able to delete cookies as the control goes to finally block any way. The only time that finally block will not execute will be when you forcefully close JVM.

Public class exceptionsDemo{

Public static void main (String[] args) {

int b=7;

int c=0;

try

{

int k=b/c;

system.out.println(k);

}

Finally //we may or may not have catch block, but we must have try block.

{ System.out.println(“cookies deleted”); }

**Value Types:**

Value types are the basic types, and include byte, short, int, long, float, double, boolean, and char.

These data types store the values assigned to them in the corresponding memory locations.

So, when you pass them to a method, you basically operate on the variable's **value**, rather than on the variable itself. Example:

public class MyClass {

public static void main(String[ ] args) {

**int x = 5;**

addOneTo(**x**);

System.out.println(**x**);

}

static void addOneTo(int num) {

num = num + 1;

}

} // Outputs "**5**"

The method from the example above takes the **value** of its parameter, which is why the original variable is not affected and 5 remains as its value.

**Reference Types:**

A reference type stores a reference (or address) to the memory location where the corresponding data is stored.

When you create an object using the constructor, you create a reference variable.

For example, consider having a Person class defined:

public class MyClass {

public static void main(String[ ] args) {

Person j;

j = new Person("John");

j.setAge(20);

celebrateBirthday(j);

System.out.println(j.getAge());

}

static void celebrateBirthday(Person p) {

p.setAge(p.getAge() + 1);

}

} //Outputs "21"

The method celebrateBirthday takes a Person object as its parameter, and increments its attribute.

Because j is a reference type, the method affects the object itself, and is able to change the actual value of its attribute. Arrays and Strings are also reference data types.

**Static:**

When you declare a variable or a method as static, it belongs to the class, rather than to a specific instance. This means that only one instance of a static member exists, even if you create multiple objects of the class, or if you don't create any. It will be shared by all objects.

Example:

public class Counter {

public static int COUNT=0;

Counter() {

COUNT++;

}

}

The COUNT variable will be shared by all objects of that class.

Now, we can create objects of our Counter class in main, and access the static variable.

public class MyClass {

public static void main(String[ ] args) {

Counter c1 = new Counter();

Counter c2 = new Counter();

System.out.println(Counter.COUNT);

}

} //Outputs "2"

The output is 2, because the COUNT variable is static and gets incremented by one each time a new object of the Counter class is created. In the code above, we created 2 objects.

You can also access the static variable using any object of that class, such as c1.COUNT.

*It’s a common practice to use upper case when naming a static variable, although not mandatory.*

The same concept applies to static methods.

public class Vehicle {

public static void horn() {

System.out.println("Beep");

}

}

Now, the horn method can be called without creating an object:

public class MyClass {

public static void main(String[ ] args) {

Vehicle.horn();

}

}

***Also, the main method must always be static.***

**Packages:**

Packages are used to avoid name conflicts and to control access to classes.

A package can be defined as a group made up of similar types of classes, along with sub-packages.

Now, we need to import the classes that are inside a package in our main to be able to use them.

The following example shows how to use the Vehicle class of the samples package.

import samples.Vehicle;

class MyClass {

public static void main(String[ ] args) {

Vehicle v1 = new Vehicle();

v1.horn();

}

}

For example, import samples.\* will import all classes in the samples package.

Note that integer, array and other classes are all enclosed in java.lang package and we don’t import that package because it is built in Java compiler. For collection interfaces like ArrayList and other classes that are part of Java.util package, we need to import java.util package by simply hovering the mouse over the underlined statement and following the IDE suggestions, or by using the syntax (import packagename.classname;) ex: (import java.util.ArrayList;).

**Important:** suppose that we want to access a method of classA in the classB, we don’t need to extend classA, all we need to do is to create object for classA in classB:

classA a = new classA;

a.methodname();

If both classA and classB are in the same package, the statements above will simply help access the required method, but if the classes are in two different packages, IDE will prompt us to import the classA package.

**Core Concepts in OOP:**

There are 4 core concepts in OOP: encapsulation, inheritance, polymorphism and abstraction.

OOP concepts in Java are the main ideas behind Java’s Object Oriented Programming. They let programmers create components that can be reused in different ways, but still maintain security.

**Encapsulation:**

The idea behind encapsulation is to ensure that implementation details are not visible to users. The variables of one class will be hidden from the other classes, accessible only through the methods of the current class. This is called data hiding.

To achieve encapsulation in Java, declare the class' variables as private and provide public setter and getter methods to modify and view the variables' values.

For example:

class BankAccount {

private double balance=0;

public void deposit(double x) {

if(x > 0) {

balance += x;

}

}

}

This implementation hides the balance variable, enabling access to it only through the deposit method, which validates the amount to be deposited before modifying the variable.

In summary, encapsulation provides the following benefits:

- Control of the way data is accessed or modified

- More flexible and easily changed code

- Ability to change one part of the code without affecting other parts

**Inheritance:**

Inheritance is the process that enables one class to acquire the properties (methods and variables) of another. With inheritance, the information is placed in a more manageable, hierarchical order.

The class inheriting the properties of another is the subclass (also called derived class, or child class); the class whose properties are inherited is the superclass (base class, or parent class).

To inherit from a class, use the extends keyword. Ex: login and logout methods for every scenario.

This example shows how to have the class Dog to inherit from the class Animal.

class Dog extends Animal {

// some code}

When one class is inherited from another class, it inherits all of the superclass' non-private variables and methods.

Example:

class Animal {

protected int legs;

public void eat() {

System.out.println("Animal eats");

}

}

class Dog extends Animal {

Dog() {

legs = 4;

}

}

As you can see, the Dog class inherits the legs variable from the Animal class.

We can now declare a Dog object and call the eat method of its superclass:

class MyClass {

public static void main(String[ ] args) {

Dog d = new Dog();

d.eat();

}

}

If you inherit from grandfather class to son and to grandson, it is called multi-level inheritance.

Rahul’s example:

Public class Father{

Public static void main (String [] args) {

Public void city ()

{

System.out.println(“Hyderabad”);

}

Public void country()

{

System.out.println(“India”);

}

Public class Son extends Father {

Public static void main (String [] args) {

Son s = new Son ();

S.city();

**Polymorphism:**

Polymorphism, which refers to the idea of "having many forms", occurs when there is a hierarchy of classes related to each other through inheritance. Briefly, Polymorphism is one method with different implementations.

A call to a member method will cause a different implementation to be executed, depending on the type of the object invoking the method.

Here is an example: Dog and Cat are classes that inherit from the Animal class. Each class has its own implementation of the makeSound() method.

class Animal {

public void makeSound() {

System.out.println("Grr...");

}

}

class Cat extends Animal {

public void makeSound() {

System.out.println("Meow");

}

}

class Dog extends Animal {

public void makeSound() {

System.out.println("Woof");

}

}

As all Cat and Dog objects are Animal objects, we can do the following in main:

public static void main(String[ ] args) {

Animal a = new Dog();

Animal b = new Cat();

}

We've created two reference variables of type Animal, and pointed them to the Cat and Dog objects.

Now, we can call the makeSound() methods.

a.makeSound();

//Outputs "Woof"

b.makeSound();

//Outputs "Meow"

As the reference variable a contains a Dog object, the makeSound() method of the Dog class will be called.

The same applies to the b variable.

This demonstrates that you can use the Animal variable without actually knowing that it contains an object of the subclass. This is very useful when you have multiple subclasses of the superclass.

**Method Overriding (aka Runtime Polymorphism):** In method overriding, the child class can use the OOP polymorphism concept to override a method of its parent class. That allows a programmer to use one method in different ways depending on whether it’s invoked by an object of the parent class or an object of the child class.

class Animal {

public void makeSound() {

System.out.println("Grr...");

}

}

class Cat extends Animal {

**public void makeSound() {**

System.out.println("Meow");

}

}

In the code above, the Cat class overrides the **makeSound**() method of its superclass Animal.

**Rules for Method Overriding:**

* + Should have the same return type and arguments
  + The access level cannot be more restrictive than the overridden method's access level (Ex: If the superclass method is declared public, the overriding method in the sub class can be neither private nor protected)
  + A method declared final or static cannot be overridden
  + If a method cannot be inherited, it cannot be overridden
  + Constructors cannot be overridden

**Method Overloading (aka Compile-Time Polymorphism):**

When methods have the same name, but different parameters, it is known as method overloading.

This can be very useful when you need the same method functionality for different types of parameters.

The following example illustrates a method that returns the maximum of its two parameters.

int max(int a, int b) {

if(a > b) {

return a;

}

else {

return b;

}

}

The method shown above will only work for parameters of type integer.

However, we might want to use it for doubles, as well. For that, you need to overload the max method:

double max(double a, double b) {

if(a > b) {

return a;

}

else {

return b;

}

}

Now, our max method will also work with doubles.

An overloaded method must have a different argument list; the parameters should differ in their type, number, or both.

**Abstraction:**

Data abstraction provides the outside world with only essential information, in a process of representing essential features without including implementation details.

A good real-world example is a book. When you hear the term book, you don't know the exact specifics, such as the page count, the color, or the size, but you understand the idea, or abstraction, of a book.

The concept of abstraction is that we focus on essential qualities, rather than the specific characteristics of one particular example.

In Java, abstraction is achieved using abstract classes and interfaces.

**Abstract Class:** any class that contains an abstract method should be defined as abstract.

* + An abstract class is defined using the abstract keyword.
  + If a class is declared abstract it cannot be instantiated (you cannot create objects of that type).
  + To use an abstract class, you have to inherit it from another class.

An **abstract method** is a method that is declared without an implementation (without braces, and followed by a semicolon): abstract void walk();

For example, we can define our Animal class as abstract:

**abstract** class Animal {

int legs = 0;

**abstract** void makeSound();

}

The makeSound method is also abstract, as it has no implementation in the superclass.

We can inherit from the Animal class and define the makeSound() method for the subclass:

class Cat extends Animal {

public void makeSound() {

System.out.println("Meow");

}

}

Every Animal makes a sound, but each has a different way to do it. That's why we define an abstract class Animal, and leave the implementation of how they make sounds to the subclasses.

***This is used when there is no meaningful definition for the method in the superclass.***

**Interfaces:**

An interface is a completely abstract class that contains only abstract methods. In interface, we only have methods signature, but no code for implementing the methods, for implementing those methods we need classes.

Some specifications for interfaces:

* + Defined using the interface keyword.
  + May contain only static final variables.
  + Cannot contain a constructor because interfaces cannot be instantiated.
  + Interfaces can extend other interfaces.
  + A class can implement any number of interfaces, we just need to separate the interface names with comma

An example of a simple interface:

interface Animal {

public void eat();

public void makeSound();

}

Interfaces have the following properties:

* + An interface is implicitly abstract. You don’t need to use the abstract keyword while declaring an interface.
  + Each method in an interface is also implicitly abstract, so the abstract keyword is not needed. All methods are public, you can’t make them private.
  + Methods in an interface are implicitly public.

***A class can inherit from just one superclass, but can implement multiple interfaces!***

Use the implements keyword to use an interface with your class.

interface Animal {

public void eat();

public void makeSound();

}

class Cat implements Animal {

@Override

public void makeSound() {

System.out.println("Meow");

}

@Override

public void eat() {

System.out.println("omnomnom");

}

}

***When you implement an interface, you need to override all of its methods.*** Implementing an interface is like an agreement, where you agree to override, declare and implement all methods of that interface.

**Type Casting:**

Assigning a value of one type to a variable of another type is known as Type Casting.

To cast a value to a specific type, place the type in parentheses and position it in front of the value.

Example:

double a = 42.571;

int b = (int) a;

System.out.println(b); //Outputs 42

The code above is casting the value 42.571 to an integer, with 42 as the resulting value.

For classes, there are two types of casting.

**Upcasting:**

You can cast an instance of a subclass to its superclass.

Consider the following example, assuming that Cat is a subclass of Animal.

Animal a = new Cat();

Java automatically upcasted the Cat type variable to the Animal type.

**Downcasting:**

Casting an object of a superclass to its subclass is called downcasting.

Example:

Animal a = new Cat();

((Cat)a).makeSound();

This will try to cast the variable a to the Cat type and call its makeSound() method.

Why is upcasting automatic, downcasting manual? Well, upcasting can never fail. But if you have a group of different Animals and want to downcast them all to a Cat, then there's a chance that some of these Animals are actually Dogs, so the process fails.

**Inner Classes:**

Java supports nesting classes; a class can be a member of another class.

Creating an inner class is quite simple. Just write a class within a class. Unlike a class, an inner class can be private. Once you declare an inner class private, it cannot be accessed from an object outside the class.

Example:

class Robot {

int id;

Robot(int i) {

id = i;

Brain b = new Brain();

b.think();

}

private class Brain {

public void think() {

System.out.println(id + " is thinking");

}

}

}

The class Robot has an inner class Brain. The inner class can access all of the member variables and methods of its outer class, but it cannot be accessed from any outside class.

**Diamond Problem?**

For example, Classes B and C extend Class A, and Class D extends both classes B and C. Now, if you create a function of (f1) in class A, a function of (f1) in class B and another function of (f1) in class C and then add lines of codes to each function in each class, you can simply create object for class A to call the function in class A, you can create object of class B to call the function in class B and you can create the object of class C to call the function in class C. Now, if you are calling the function (f1) of class D (d.f1()), which (f1) should D class refer to, the one in class B or the one in class C?

That is the problem because of which Java doesn’t support ***multiple inheritance***.

**To call a method from one class in another class:**

First create object for the class that consists the methods:

Classb cb = new Classb ();

Then:

//Classobject.methodname();

//example: cb.validateheader();

**Text Validations:**

String is a predefined class in Java, used to manipulate string. To create an object for the string class:

//String str = new String (“Payment $100 paid”); //general way

String str = “Payment $100 paid”; //simplified way only for String

System.out.println(str.charAt(8)); //what character is at the 8th index.

System.out.println(str.indexOf(“$”)); //at what index is ‘$’ located.

System.out.println(str.substring(8)); //print the string from 8th index // output will be: $100 paid

**//Print the string in reverse:**

String s = “rahul”;

String t=””;

For (int i=s.length()-1; i>=0; i--) //we have -1 because it starts from zeroth index.

{

T = t + s.charAt(i);

}

System.out.println(t);

**//Sum of numbers in the array:**

Int sum = 0;

Int c[] = {2, 5, 8, 7, 4};

For(int i=0; i<c.length; i++)

{

Sum = sum+c[i];

}

System.out.println(sum); //output will be: 26

**//At what index the number 8 is located:**

Int sum = 0;

Int c[] = {2, 5, 8, 7, 4};

For(int i=0; i<c.length; i++)

{

If(c[i]==8)

{

System.out.println(i) //output will be: 2

Break;

}}}

**OOPS Interview questions**

**What are the core concepts of OOPS?**

OOPS core concepts are: Abstraction, Encapsulation, Polymorphism, Inheritance, Composition, Association, Aggregation.

**What is Abstraction?**

Abstraction is an OOPS concept to construct the structure of the real world objects. During this construction only the general states and behaviors are taken and more specific states and behaviors are left aside for the implementers.

**What is Encapsulation?**

Encapsulation is an OOPS concept to create and define the permissions and restrictions of an object and its member variables and methods. A very simple example to explain the concept is to make the member variables of a class private and providing public getter and setter methods. Java provides four types of access level modifiers: public, protected, no modifier and private.

**What is the difference between Abstraction and Encapsulation?**

“Program to interfaces, not implementations” is the principle for Abstraction and “Encapsulate what varies” is the OO principle for Encapsulation.

Abstraction provides a general structure of a class and leaves the details for the implementers. Encapsulation is to create and define the permissions and restrictions of an object and its member variables and methods.

Abstraction is implemented in Java using interface and abstract class while Encapsulation is implemented using four types of access level modifiers: public, protected, no modifier and private.

**What is Polymorphism?**

Polymorphism is the occurrence of something in various forms. Java supports various forms of polymorphism like polymorphic reference variables, polymorphic method, polymorphic return types and polymorphic argument types.

**What is Inheritance?**

A subclass can inherit the states and behaviors of it’s super class is known as inheritance.

**What is multiple inheritance?**

A child class inheriting states and behaviors from multiple parent classes is known as multiple inheritance.

**What is the diamond problem in inheritance?**

In case of multiple inheritance, suppose class A has two subclasses B and C, and a class D has two super classes B and C.If a method present in A is overridden by both B and C but not by D then from which class D will inherit that method B or C? This problem is known as diamond problem.

**Why Java does not support multiple inheritance?**

Java was designed to be a simple language and multiple inheritance introduces complexities like diamond problem. Inheriting states or behaviors from two different type of classes is a case which in reality very rare and it can be achieved easily through an object association.

**What is Static Binding and Dynamic Binding?**

Static or early binding is resolved at compile time. Method overloading is an example of static binding.

Dynamic or late or virtual binding is resolved at run time. Method overriding is an example of dynamic binding.

**What is a Class?**

A class is the specification or template of an object.

**What is an Object?**

Object is instance of class.

**What is Runtime Polymorphism?**

Runtime polymorphism or dynamic method dispatch is a process in which a call to an overridden method is resolved at runtime rather than at compile-time.

In this process, an overridden method is called through the reference variable of a super class.

**What is the difference between abstraction and encapsulation?**

Abstraction hides the implementation details whereas encapsulation wraps code and data into a single unit.

**What is abstract class?**

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

**Can there be any abstract method without abstract class?**

No, if there is any abstract method in a class, that class must be abstract.

**Can you use abstract and final both with a method?**

No, because abstract method needs to be overridden whereas you can't override final method.

**Is it possible to instantiate the abstract class?**

No, abstract class can never be instantiated.

**What is interface?**

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

**Can you declare an interface method static?**

No, because methods of an interface is abstract by default, and static and abstract keywords can't be used together.

**Can an Interface be final?**

No, because its implementation is provided by another class.

**What is marker interface?**

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

**What is difference between abstract class and interface?**

|  |  |
| --- | --- |
| Abstract Class | Interface |
| An abstract class can have method body (non-abstract methods). | Interface have only abstract methods. |
| An abstract class can have instance variables. | Cannot have instance variables. |
| An abstract class can have constructor. | Interface cannot have constructor. |
| An abstract class can have static methods. | Interface cannot have static methods. |
| You can extend one abstract class. | You can implement multiple interfaces. |

**Can we define private and protected modifiers for variables in interfaces?**

No, they are implicitly public.

**When can an object reference be cast to an interface reference?**

An object reference can be cast to an interface reference when the object implements the referenced interface.

**Difference between int and integer?**

**Int:** int is a primitive data type which means it stores numeric values but not much more.

**Integer:** Integer is a class, also known as wrapper class because it wraps primitive data type inside it.

The same is the difference between char and Character, Boolean and Boolean, float and Float, double and Double.

**Primitive type variables vs. Reference type variables?**

Primitive types are the basic types of data: byte, short, int, long, float, double, boolean, char. Primitive variables store primitive values. Reference variables store addresses to locations in memory for where the data is stored. In short, the main difference between the two types is that primitive types store actual values but reference type stores handle to object in the heap.

**What is Selenium:**

Selenium is an open source web automation testing tool or framework that allows you to execute test cases across different browsers.

**Selenium automation framework:** *Data Driven, Keyword Driven & Hybrid*

Selenium Framework is a code structure that helps to make the code maintenance easy.

Using frameworks, produce beneficial outcomes like increased code reusage, higher portability, reduced script maintenance cost, higher code readability, etc.

There are mainly three types of frameworks created by Selenium WebDriver to automate test cases.

**Data Driven Test Framework:**

Data-driven is a test automation framework which stores test data in a table or spread spreadsheet format. This allows automation engineers to have a single test script which can execute tests for all the test data in the table.

In this framework, input values are read from data files and are stored into a variable in test scripts. Ddt (Data Driven testing) enables building both positive and negative test cases into a single test.

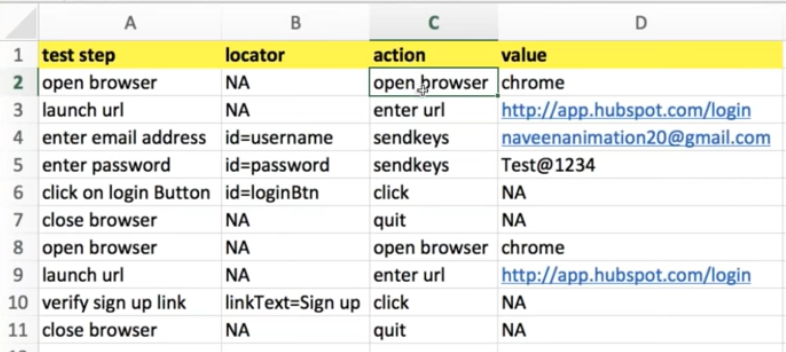
In Data-driven test automation framework, input data can be stored in single or multiple data sources like xls, XML, csv, and databases.To read or write an Excel, Apache provides a very famous library called POI.

**Keyword Driven Test Framework:**

In keyword driven test framework, all the operations and instructions are written in some external file like Excel worksheet. Keywords are basically the actions you want to perform on the browser.

Components: BasePage.java class (where we initialize the driver and properties), config.properties file (where we type the browser, url, username, password), Execution Data Excel file (as shown in the picture bellow), ExecutionEngine.java class (where we write all the execution script or logic and this class will read the data from the execution data excel file).

Advantages of KDF: anyone can use it, less components are required. Disadvantages: maintenance requires more efforts, too many methods, method for every action, flow changes break the tests. To write a really stable test, we can use the Page Object Pattern that helps us to resolve many issues that we face in the keyword driven framework, and the record and replay methodology (just typing the scripts).



**Hybrid Test Framework:**

Hybrid Test framework is a concept where we are using the advantage of both Keyword and Data driven framework. Here for keywords, we will use Excel files to maintain test cases, and for test data, we can use data provider of TestNG framework.

**Locator Techniques:**

* //tagName[@attribute=’value’] – xpath syntax
* tagName[attribute=’value’] – CSS
* tagName#id – CSS
* tagname.classname – CSS
* //tagName[contains(@attribute=’value’)] – xpath regular expression
* tagName[attribute\*=’value’] – CSS regular expression
* When locating using Classname, we should remove spaces and replace them with dots(.)
* We can convert the value of Classname to CSSSelector by simply putting dot (.) in front of the value of classname. Ex: driver.findElement(By.cssSelector(“.asdfasdf”)…
* To convert ID value to CSS, put # before the code.
* Parent to Child Traverse Technique: define xpath for parent/tagnames

Example: //tagname[@attribute=’value’]/tagname/tagname/tagname[2]/tagname

For CSS we replace the forward slash with space.

* Parent Child Relationship Technique: parent xpath child xpath (simply followed, no slashes)
* Identify siblings: define xpath for sibling/following-sibling::tagname[2].

Example: //\*[@id=’tabha4’]/following-sibling::li[2]

* Child to Parent Traverse Technique: define xpath for child/parent::tagname
* Find element using the visible text: //\*[text()=’yourvisibletexthere’]
* If there are two matching nodes, and we intend to choose the second: (xpath of the element)[2]
* CSS syntax for traversing to child element in table (i.e. specific column in a row): parent’s css followed by space then followed by tagname:nth-child(number of child). Example: ("div[class='cb-col cb-col-100 cb-scrd-itms'] div:nth-child(3)")
* To verify the locator in Chrome Console:
  + $x(“”) for Xpath
  + $(“”) for CSS

**Difference between Relative and Absolute xpath?**

Relative xpath: use a specific locator to identify an element directly without relying on parent nodes. Relative xpath is more preferred.

Absolute xpath: identifying an element by traversing technique.

**Handling static dropdowns with Select webdriver API:**

Select s = new select (driver.findElement(By.id(“sdfs”)));

s.selectByValue(“2”);

**Dropdown looping:**

1. We use while loop, which keeps looping until the condition becomes falls.

With while loop, we have to initialize first, then compare, then put condition to increment it every time.

Int i=1;

While (i<5)

{

Driver.findElement(By…..).click();

i++;

}

2. Use For Loop:

For(int i=1; i<5;i++)

{

Driver.findElement(By…..).click();

}

**Handling Dynamic dropdowns:**

We can use the parent to child traverse technique or parent child relationship technique to target the intended element.

**Handling Auto-Suggestive dropdowns:**

The type of dropdowns where you type something first to get the suggestions.

WebElement source=driver.findElement(By.id("hp-widget\_\_sfrom"));

source.clear();

source.sendKeys("MUM");

Thread.sleep(2000);

source.sendKeys(Keys.ENTER);

WebElement destination=driver.findElement(By.id("hp-widget\_\_sTo"));

destination.clear();

destination.sendKeys("DEL");

Thread.sleep(2000);

destination.sendKeys(Keys.ARROW\_DOWN);

destination.sendKeys(Keys.ENTER);

**Handling Dynamic Radio-buttons:**

int count =driver.findElements(By.xpath("//input[@name='group1']")).size();

for(int i=0;i<count;i++)

{

String text=driver.findElements(By.xpath("//input[@name='group1']")).get(i).getAttribute("value");

if(text.equals("Cheese"))

{

driver.findElements(By.xpath("//input[@name='group1']")).get(i).click();

}

}

**Handling Alerts:**

* Web related pop ups can be handled by Selenium, we simply get the xpath and perform our action on it – i.e. click();.
* Non HTML alerts which are JAVA alerts or pop ups, can’t be handled by Selenium and we can’t spy on it to get the Xpath code, so we use JAVA methods:
  + To accept: driver.switchTo().alert().accept();
  + To cancel: driver.switchTo().alert().dismiss();
  + To enter something in the alert: driver.switchTo().alert().sendkeys(“xxx”);

**Assertions:**

Verify that the state of the application is same to what we are expecting.

When using TestNG framework, there are two types of assertions in WebDriver:

1. Hard assertions: when assertion fails, it terminates the test (method).
   1. assertEquals
   2. assertNotEQuals
   3. assertTrue
   4. assertFalse
   5. assertNull
   6. assertNotNull

Example: Assert.assertTrue(driver.findElement(By.cssSelector(“xxx”)).isSelected());

1. Soft assertions: the failed assertions will be reported in TestNg report and not making the test to abort anywhere.

To use a soft assertion in TestNg, we have to include it’s corresponding class (as SoftAssert()) in the script.

Ex: softAssert softAssert = new softAssert();

softAssert.assertNull(“assertion”);

**Validating if UI Elements are disabled or enabled with Attributes:**

System.out.println(driver.findElement(By.name("ctl00$mainContent$view\_date2")).isEnabled());

System.out.println(driver.findElement(By.id("Div1")).getAttribute("style"));

driver.findElement(By.id("ctl00\_mainContent\_rbtnl\_Trip\_1")).click();

System.out.println(driver.findElement(By.id("Div1")).getAttribute("style"));

if(driver.findElement(By.id("Div1")).getAttribute("style").contains("1"))

{

System.out.println("its enabled");

Assert.assertTrue(true);

}

else

{

Assert.assertTrue(false);

}

**To autoformat code in Selenium:** Press control+shift+f

**Debugging:** Start and watch your execution step by step.

**Naming conventions:**

Variable and method names should always start with lowercase. Example: name

Camel case standard: If variable name consists of several words, the first word should start with small letter, all subsequent words should start with capital letters.

Class names are opposite: first name should start with uppercase and subsequent with lowercase.

**Selenium Web Driver Waits (Synchronization):**

**1. Implicit Wait:** Implicit wait define time globally. It doesn’t necessarily wait for the specified amount of time, rather the specified number is the maximum it will wait.

driver.manage().timeouts().implicitlyWait(10,TimeUnit.SECONDS);

**2. Explicit Wait:** You can target specific element or scenario. Explicit wait can be achieved in two ways:

1. WebDriverWait:

Syntax: WebDriverWait wait = new WebDriverWait(WebDriverRefrence,TimeOut);

Example: WebDriverWait wait = new WebDriverWait(driver, 20);

Wait.until(Expectedconditions.visibilityofElementLocated(Locator)

Replace locator with the xpath of element.

1. Fluent Wait:

Finds web elements repeatedly at regular intervals until the timeout or till the object gets found.

Wait<WebDriver> wait = new FluentWait<WebDriver>(driver) ….. (too long code)

Where FluentWait is a class implementing Wait which is an interface.

**Keyboard and Mouse Interactions:**

Selenium WebDriver allows us to perform keyboard and mouse events by using Actions class. We need to create an instance of the Actions class and pass the instance of driver class in it.

*To move the mouse cursor to a specific web element:*

Webdriver driver = new ChromeDriver();

Driver.get(“www.amazon.com”);

Actions a = new Actions(driver);

a.moveToElement(driver.findElement(By.cssSelector(“sss”))).build().perform(); //just moves mouse cursor

*Type in uppercase letters:*

a.moveToElement(driver.findElement(By.id(“x”))).click().keyDown(Keys.SHIFT).sendKeys(“hello”).build().perform();

*Type in uppercase letters and then double click to select it:*

a.moveToElement(driver.findElement(By.id(“x”))).click().keyDown(Keys.SHIFT).sendKeys(“hello”).doubleClick().build().perform();

*To move the mouse cursor to a specific web element and then right click on it:*

a.moveToElement(driver.findElement(By.cssSelector(“sss”))).contextClick().build().perform();

**Handling Multiple Windows:**

*(switch from one window to another window of browser)*

Webdriver driver = new ChromeDriver();

Driver.get(“www.amazon.com”);

Driver.findElement(By.xpath(“xxx”)).click();

System.out.println(driver.getTitle());

Set<String> ids=driver.getWindowHandles(); //this methods gives us the ids of all open windows.

Iterator<String> it = ids.iterator(); //iterator is a java method

String parentID = it.next(); //by default the iterator starts from the top, zeroth row, it.next takes it to

first row, and any additional it.next will gradually bring it down.

String childID = it.next();

Driver.switchTo().window(childID);

System.out.println(driver.getTitle());

*To drag and drop:*

Actions a = new Actions(driver);

WebElement source = Driver.findElement(By.id(“xxx”));

WebElement target = Driver.findElement(By.id(“yyy”));

a.dragAndDrop(source, target).build().perform();

**How to Handle Frames?**

Frame is a part of a webpage which displays content independent of its container, with the ability to load content independently. Frame servers as a container or box in a webpage. The HTML or media elements that go in a frame may or may not come from the same website as the other elements of content on display. We have to tell Selenium to switch to the specific frame in order for Selenium to handle its elements. So to work with IFrame elements in Selenium WebDriver, first of all we need to select that IFrame:

Webdriver driver = new ChromeDriver();

Driver.get(“www.xyz.com”);

Driver.switchTo().frame(driver.findElement(By.cssSelector(“iframe[class=’demo-frame]”)));

Driver.findElement(By.id(“xxx”)).click();

*To know how many frames are there in the app:*

System.out.println(driver.findElements(By.tagName(iframe”)).size());

*To switch back from frame to webpage elements:*

Driver.switchTo().defaultContent();

*Working with multiple frames on the same page:*

If there are multiple frames on a single webpage, you can’t directly navigate from one frame to another frame. For that, you need to switch from one frame to the webpage first and then switch to the second frame.

**Limiting WebDriver Scope:**

public class Scope {

public static void main(String[] args) throws InterruptedException {

//1. Give me the count of links on the page. HTML way of defining links is to use anchor or tagname ‘a’.

System.setProperty("webdriver.chrome.driver", "C:\\work\\chromedriver.exe");

WebDriver driver=new ChromeDriver();

driver.get("http://qaclickacademy.com/practice.php");

System.out.println(driver.findElements(By.tagName("a")).size());

//2. Limit scope of WebDriver (like creating a mini WebDriver) and count links in footer section:

WebElement footerdriver=driver.findElement(By.id("gf-BIG"));

System.out.println(footerdriver.findElements(By.tagName("a")).size());

//3. Limit the scope of FooterDriver (a mini WebDriver) to a specific column:

WebElement coloumndriver=footerdriver.findElement(By.xpath("//table/tbody/tr/td[1]/ul"));

System.out.println(coloumndriver.findElements(By.tagName("a")).size());

//4. Click on each link in the column and check if the pages are opening:

for(int i=1;i<coloumndriver.findElements(By.tagName("a")).size();i++)

{

String clickonlinkTab=Keys.chord(Keys.CONTROL,Keys.ENTER);

coloumndriver.findElements(By.tagName("a")).get(i).sendKeys(clickonlinkTab); //sendKeys(keys.ENTER) will hit enter, but here we need a combination of keys, so we put it in string variable.

Thread.sleep(5000L);

} // opens all the tabs

5. Print the title of each page:

Set<String> abc=driver.getWindowHandles();

Iterator<String> it=abc.iterator();

while(it.hasNext())

{

driver.switchTo().window(it.next());

System.out.println(driver.getTitle());

} //iterates to each and every tab and prints its title.

**Handling Calendar UI in Travel Websites:** *(select a specific day of a specific month)*

public class calendar {

public static void main(String[] args) {

System.setProperty("webdriver.chrome.driver", "C://work//chromedriver.exe");

ChromeDriver driver=new ChromeDriver();

driver.get("https://www.path2usa.com/travel-companions");

//May 21

driver.findElement(By.XPath(".//\*[@id='travel\_date']").click();

while(!driver.findElement(By.cssSelector("[class='datepicker-days'] [class='datepicker-switch']")).getText().contains("May")) //while accepts a true or false argument. If it is true, it will go inside the block and execute it, and it will keep on executing until it becomes false. (!) is negation, negation of true is false and vice versa. In this example, when it reaches ‘May’, since ‘May’ contains ‘May’ (condition is true) and its negation is false, we will come out of loop and we will have the month of May on our calendar UI.

{

driver.findElement(By.cssSelector("[class='datepicker-days'] th[class='next']")).click();

}

List<WebElement> dates= driver.findElements(By.className("day"));

//Grab common attribute//Put it into list and iterate

int count=driver.findElements(By.className("day")).size();

for(int i=0;i<count;i++)

{

String text=driver.findElements(By.className("day")).get(i).getText();

if(text.equalsIgnoreCase("21"))

{

driver.findElements(By.className("day")).get(i).click();

break;

}

}

}

}

**Working with multiple frames:**

*Click on “I’m not a robot” (which is located in one frame) and then click on “verify” (which is in another frame):*

package basics;

public class Miscelleanous {

\*Here

public static void main(String[] args) {

WebDriver driver=new FirefoxDriver();

driver.get("https://fantasycricket.dream11.com/IN/");

int number = findFrameNumber(driver, By.xpath(“.//\*[@id=’recaptcha-anchor’]/div[5]”));

driver.switchTo().frame(number);

driver.findElement(By.xpath(“.//\*[@id=’recaptcha-anchor’]/div[5]”)).click();

driver.switchTo().defaultContent(); //if you are working with frames inside frames, don’t use this method. This is only used when you have two independent frames, where you have to get from one frame to the main page first before working on the other independent frame.

//Now the following is how we click on “verify button”:

int number1 = findFrameNumber(driver, By.xpath(“//\*[@id=’recaptcha-verify-button’]”));

driver.switchTo().frame(number1);

driver.findElement(By.xpath(“//\*[@id=’recaptcha-verify-button’]”)).click();

}

Public static int findFrameNumber(WebDriver driver, By by) //we want to create a reusable method. We either have to define the webdriver instance globally, outside of main and other methods (like shown above=marked with \*Here), or we need to pass it along with locator as argument here.

{

Int i;

Int framecount=driver.findElements(By.tagName(“iframe”)).size();

For(i=0; i<framecount;i++)

{

Driver.switchTo().frame(i);

int count = Driver.findElements(by).size();

if(count>0)

{

Break;

}

Else

{

System.out.println(“continue looping”); //you don’t have to have anything here.

}

}

//the above is how we first want to know how many frames are there on our webpage, then we want to know which of the frames contain the web element we are looking for. We shouldn’t confuse the isDisplayed method with the procedure we are doing here. isDisplayed is used only to know if something is visible or not, but here we are looking for things that are available in a frame or not.

//the following is how we need to come out of one independent frame (like the above) to the main page and then switch back to another independent frame. Note: if the frames have parent-child relationship, we don’t have to go out of the frame, we can simply switch from one frame to the other.

Driver.switchTo().defaultContent();

Return I;

**Handling Auto-suggestive Dropdowns:**

*(when you type something, you will get a dropdown list of suggestions)*

Assume that our requirement is to validate if we get a dropdown menu when we type ‘BENG’ in the search box:

Public class Autosuggestive{

public static void main(String[] args) {

System.setProperty("webdriver.chrome.driver", "C:\\work\\chromedriver.exe");

WebDriver driver=new ChromeDriver();

Driver.get(“https://www.ksrtc.in”);

Driver.findElement(By.xpath(“//input[@id=’fromPlaceName’]”)).sendKeys(“BENG”);

Driver.findElement(By.xpath(“//input[@id=’fromPlaceName’]”)).sendKeys(Keys.DOWN);

Driver.findElement(By.xpath(“//input[@id=’fromPlaceName’]”)).sendKeys(Keys.DOWN);

//by typing ‘BENG’ and pressing the key down two times, ‘BENGALURU DARSHINI’ will appear in the search box, but if we want to print it using the method below, we will get an error message:

System.out.println(driver.findElement(By.xpath(“//input[@id=’fromPlaceName’]”)).getText());

//the reason is that Selenium doesn’t support hidden elements. And this is a **disadvantage of Selenium**. And to handle such kind of situation, we use Javascipt.

**Javascript Executor:**

Because Selenium cannot identify hidden elements (Ajax implementation), we use Javascript DOM which can extract hidden elements. Document Object Model (DOM) is a programming interface for HTML and XML documents.

Question: we are working with Java, how can we switch the programming language to Javascript? To address this question, Selenium has come up with an API called Javascript Executor which can be used to execute the Javascript functions. To use it, we need to define it as follows (continuing the code above):

Public class Autosuggestive{

public static void main(String[] args) {

System.setProperty("webdriver.chrome.driver", "C:\\work\\chromedriver.exe");

WebDriver driver=new ChromeDriver();

Driver.get(“https://www.ksrtc.in”);

Driver.findElement(By.xpath(“//input[@id=’fromPlaceName’]”)).sendKeys(“BENG”);

Driver.findElement(By.xpath(“//input[@id=’fromPlaceName’]”)).sendKeys(Keys.DOWN);

JavascriptExecutor js = (JavascriptExecutor)driver;

String script = “return document.getElementById(\“fromPlaceName\”).value;”;

//in Javascript, if we want to retrieve the value, we have to use ‘return’ which is works the way how ‘system.out.println’ of Java works. like in the console tab of Chrome. Return will take the value and put it in the ‘string’ variable.

//we use the forward slash as shown above because Java doesn’t support double quotes inside double quotes.

String text = (String) Js.executeScript(script);

System.out.println(text);

Int I = 0;

While (!text.equalsIgnoreCase(“BENGALURU INTERNATION AIPORT”))

{

i++;

Driver.findElement(By.xpath(“//input[@id=’fromPlaceName’]”)).sendKeys(Keys.DOWN);

text = (String) Js.executeScript(script);

System.out.println(text);

if (i>10)

{

break;

}

}

if (i>10)

{

System.out.println(“Element not found”);

}

Else

{

System.out.println(“Element found”);

}

**Handling Table Grids in a Webpage (Automating Tables):**

*Validate if the total of values in a table column is correct.*

package basics;

public class Tableexercises {

public static void main(String[] args) {

int sum=0;

System.setProperty("webdriver.chrome.driver", "C://work//chromedriver.exe");

WebDriver driver = new ChromeDriver();

driver.get("http://www.cricbuzz.com/live-cricket-scorecard/18970/pak-vs-sl-2nd-t20i-pakistan-v-sri-lanka-

in-uae-2017");

WebElement table=driver.findElement(By.cssSelector("div[class='cb-col cb-col-100 cb-ltst-wgt-hdr']"));

//in Java when we create a variable (i.e. table in this example), it is mandatory to declare its return type,

i.e. here the return type is WebElement //this is CSS of the table

int rowcount= table.findElements(By.cssSelector("cb-col cb-col-100 cb-scrd-itms")).size();

int count=table.findElements(By.cssSelector("div[class='cb-col cb-col-100 cb-scrd-itms'] div:nth-

child(3)")).size(); //one of css syntaxes for traversing to child element is: parent’s css followed by space

and followed by tagname:nth-child(number of child)

for(int i=0;i<count-2;i++) //iteration starts from zeroth index and continues until the count, except the last

two rows (total and another one) which we don’t want to be considered in the sum operation.

{

String value=table.findElements(By.cssSelector("div[class='cb-col cb-col-100 cb-scrd-itms'] div:nth- child(3)")).get(i).getText();

int valueinteger= Integer.parseInt(value); //to convert string into integer in Java, there is a simple

function: integer.pareInt

sum=sum+valueinteger;

}

String Extras=driver.findElement(By.xpath("//div[text()='Extras']/following-sibling::div")).getText();

int extrasValue=Integer.parseInt(Extras);

int TotalSumValue=sum+extrasValue;

System.out.println(TotalSumValue);

String ActualTotal=driver.findElement(By.xpath("//div[text()='Total']/following-sibling::div")).getText();

int ActualTotalVAlue=Integer.parseInt(ActualTotal);

if(ActualTotalVAlue==TotalSumValue)

{

System.out.println("Count Matches");

}

else

{

System.out.println("count fails");

}

}

}

**Handling HTTPS Certifications with Selenium WebDriver:**

Secure Sockets Layer (SSL) certificates, sometimes called digital certificates, are used to establish an encrypted connection between a browser or user's computer and a server or website. The SSL connection protects sensitive data, such as credit card information, exchanged during each visit, which is called a session, from being intercepted from non-authorized parties.

We can use the *ChromeOptions* class of Selenium to set our local browser settings - i.e. handle SSL certificates.

*Desired Capabilities* is a class that helps you create a general profile to customize your browser.

public class qw {

public static void main(String[] args) {

//the intention here is to accept ssl certifications:

DesiredCapabilities ch=DesiredCapabilities.chrome(); //I’m adding the desired capabilities to Chrome browser. So far it is a general chrome profile.

//ch.acceptInsecureCerts();

ch.setCapability(CapabilityType.ACCEPT\_INSECURE\_CERTS, true); //setting capabilities for the Chrome browser and then in the following steps, merging them to the local browser.

ch.setCapability(CapabilityType.ACCEPT\_SSL\_CERTS, true);

ChromeOptions c= new ChromeOptions(); //belongs to your local browser

c.merge(ch);

System.setProperty("webdriver.chrome.driver", "xxxxx");

WebDriver driver=new ChromeDriver(c); //after we injected the capabilities to our local browser, we declare it as argument to our ChromeDriver.

}

}

**Maximize the Browser Window and Delete Cookies:**

package basics;

public class Miscelleanous {

public static void main(String[] args) {

System.setProperty("webdriver.chrome.driver", "C://work//chromedriver.exe");

WebDriver driver = new ChromeDriver();

driver.manage().window().maximize();

driver.manage().deleteAllCookies();

// driver.manage().deleteCookieNamed("sessionKey"); //to delete only one specific cookie – i.e.

delete the session key cookie to end the login session and log you out. The testcase could be: delete the

session key cookie, click on any link and verify that you are redirected to the login page.

driver.get("http://google.com");

}

}

**Take Screenshots in Selenium:**

//To take a screenshot, we have to change the webdriver object behavior to capture screenshots.

File src= ((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);

FileUtils.copyFile(src,new File("C:\\Users\\rahul\\screenshot.png"));

//fileutils is a method used to copy the file from src object to local machine.

**Selenium Grid:**

Selenium grid allows you to run your tests on different machines against different browsers.

The Hub is the central point that receives all the test requests and distributes them to the right nodes. Test is triggered from the hub.

Nodes will execute the tests.

**To setup and register hub with nodes:**

* We need to download the Selenium Standalone Server JAR from seleniumhq.org website.
* Command line -> navigate to the folder where Selenium Standalone Server JAR is located.
* cd (change directory). To go to the parent folder, type (cd..). To go to a child folder, type (cd nameofchilddirectory)
* Now invoke the JAR file by typing (java -jar nameoftheJARfile.jar -role hub)
* Command prompt will give you the URL to register nodes.
* Switch to the computer that you want to use as a node
* Download Selenium Standalone Server JAR to the node machine
* Download chrome webdriver to the node and save it in the same folder as Grid JAR
* Open command prompt, invoke the JAR file by typing:
  + java -Dwebdriver.chrome.driver=”c:\chromedriver.exe” -jar nameoftheJARfile.jar -role webdriver -hub urlToRegisterNodes -port 5566
  + Make sure that JAVA is properly configured in system variables of the Node
* To see the Selenium Grid setup: paste the URL in browser and replace (register) with (console)

**Desired Capabilities:**

* Import the Selenium Standalone Server JAR to the library of your project on Eclipse
* To run test cases remotely/on server we have to import DesiredCapabilities class:
  + DesiredCapabilities dc= new DesiredCapabilities();
  + Dc.setBrowserName(“chrome”);
    - You can also include the browser version (optional)
  + Dc.setPlatform(Platform.WINDOWS);
  + Webdriver driver = new RemoteWebDriver(new URL(<http://localhost::4444/wd/hub>,dc);

Note: we can do cross browser testing using Selenium Grid.

**TestNG**

It is one of the most important components in building framework.

It is an automation testing framework in which NG stands for Next Generation.

Using TestNG we can generate a proper report, and we can easily come to know how many test cases are passed, failed and skipped.

We can execute failed test cases separately. Suppose, you have five test cases, one method is written for each test case (Assume that the program is written using the main method without using TestNG). When you run this program first, three methods are executed successfully, and the fourth method is failed. Then correct the errors present in the fourth method, now you want to run only fourth method because first three methods are anyway executed successfully. This is not possible without using TestNG.

The TestNG provides an option, i.e., testng-failed.xml file in test-output folder. If you want to run only failed test cases means you run this XML file. It will execute only failed test cases.

Multiple test cases can be grouped more easily by converting them into testng.xml file. In which you can make priorities like which test case should be executed first.

The same test case can be executed multiple times without loops just by using keyword called 'invocation count.'

Using TestNG, you can execute multiple test cases on multiple browsers, i.e., cross browser testing.

The TestNG framework can be easily integrated with tools like Maven, Jenkins, etc.

Annotations used in the TestNG are very easy to understand ex: @BeforeMethod, etc.

WebDriver has no native mechanism for generating reports. TestNG can generate the report in a readable format.

TestNG simplifies the way the tests are coded. There is no more need for a static main method in our tests. The sequence of actions is regulated by easy-to-understand annotations that do not require methods to be static.

Uncaught exceptions are automatically handled by TestNG without terminating the test prematurely. These exceptions are reported as failed steps in the report.

@Test is used to tell that the method under it is a test case. In this case, we have set the verifyHomepageTitle() method to be our test case, so we placed an '@Test' annotation above it.

**Conclusion**

* TestNG is a testing framework that is capable of making Selenium tests easier to understand and of generating reports that are easy to understand.
* The main advantages of TestNG over JUnit are the following.
  + Annotations are easier to use and understand.
  + Test cases can be grouped more easily.
  + TestNG allows us to create parallel tests.
* The Console window in Eclipse generates a text-based result while the TestNG window is more useful because it gives us a graphical output of the test result plus other meaningful details like:
  + Runtimes of each method.
  + The chronological order by which methods were executed
* TestNG is capable of generating HTML-based reports.
* Annotations can use parameters just like the usual Java methods. Ex: @Test (priority = 1)
* Every method after the @Test annotation is treated as a test case
* After the annotation, the test case should be wrapped in the method name: Public void ftest()
* With TestNG you can have multiple test cases executed from a single class.

**Configure TestNG:**

* Go to testing.org, click on Eclipse link, click on installation (just to view all the guidance).
* In Eclipse click on Help, Install new software, type (<http://beust.com/eclipse>) in Work With cell, check TestNG and proceed with installation. This process will install the TestNG plugin in Eclipse.

**Running test cases in TestNG without JAVA compiler:**

* Create a Java project, then create a package inside it
* Create a class inside package, but don’t check (public static void main) because we don’t need to depend on Java compiler, TestNG itself is doing the job of a compiler.

**Create testing.xml file:**

Right click on the TestNG project, TestNG, convert to TestNG, next, finish.

Inside the XML file:

<suite Name = “Loan Department”>

<test name=”Personal Loan”>

<classes>

<class name=”test.day1”/>

<class name=”test.day1”/>

</classes>

</test>

</suite>

If we want to group test cases (classes) and run them with one click only, we pass all the class names in the xml file as shown above: <class name=”packageName.className”/>

In the xml script, the relation of *Suite-Test-Class* is like *Test Suite – Test Folder (module) – Test Cases*.

**Include and Exclude Mechanism to control test cases execution / methods execution:**

<suite Name = “Loan Department”>

<test name=”Personal Loan”>

<classes>

<class name=”test.day1”/>

<methods>

<exclude name=”MobileLogincarLoan”/>

</methods>

</class>

</classes>

</test>

</suite>

Include is declared the same way.

**Note:** When we have a single line, we can add the slash at the end of it, that will close the line, otherwise we have to declare it with slash in front of the name -i.e. </class>

**Note:** we should have a naming convention for test methods (test cases), so that it makes it easy to use the include, exclude feature for bigger number of test cases. i.e. <exclude name = “mobile.\*/> which means exclude all test cases names of which start with ‘Mobile’.

**Note:** If we want to exclude an entire package, we can simply declare <packages> and inside it <package name = “test”/> in between the <test> </test> lines.

**TestNG annotations:**

TestNG executes test cases based on the alphabetic order of method names, unless we use annotations:

@BeforeTest (execute the method declared here before all methods in the relevant test folder)

@AfterTest (execute the method declared here after all methods in the relevant test folder)

Note: scope of BeforeTest and AfterTest annotations is limited to one test folder only.

@BeforeSuite (execute this method before all methods in the relevant suite.)

@AfterSuite (execute this method after all methods in the relevant suite.)

Note: scope of BeforeSuite and AfterSuite annotations is applied to the entire suite or the entire xml file.

@BeforeMethod (execute this method before each and every other method in the relevant class)

@AfterMethod (execute this method after each and every other method in the relevant class)

Note: BeforeMethod and AfterMethod will repeatedly execute before/after each and every method in the relevant class. Practical example: delete cookies after every execution.

@BeforeClass (This method will execute before executing any methods in the relevant class)

@AfterClass (This method will execute after executing all the methods in the relevant class)

**Groups in TestNG:**

For example in smoke testing, we want to execute a few selected test cases that come from different class files (selected out of hundreds of test cases), we use groups tag in front of the @Test annotation:

@Test(groups={“Smoke”}) -we use this for all the selected test cases.

And then in the xml file, we declare all classes in between <classes> </classes> and the following between <test name=”regression”> and <classes>:

<groups>

<run>

<include name =”Smoke”/>

</run>

</groups>

Note: we can also exclude the specific group tagged ‘Smoke’ simply by replacing ‘include’ with ‘exclude’.

**TestNG Helper Attributes:**

**Execute one test case after another specific one:**

If we want to execute a specific method right after executing another method: @Test(dependsOnMethods={“WebloginCarLoan”})

If we want two methods to be executed before our intended method, then:

@Test(dependsOnMethods={“WebloginCarLoan”, “MobilessignoutcarLoan”})

**Skip a test case:** usually used when a test case fails, you are aware of it and you want to skip or comment it: @Test(enabled=false)

**Wait for a specific time period before failing the test:** @Test(timeOut=4000)

**Parametrization in TestNG xml file:**

If you want to avoid hard coding a step that is used again and again in several test cases, you can parameterize global environment variables or in other words, drive data from xml file to classes.

In xml file: write the following above folders (test) and below the ‘suite’ (this will create a suite level parameter, we can also create a test folder level parameter):

<Parameters name=”URL” value=[www.facebook.com](http://www.facebook.com)/>

In the actual class, write:

@Parameters({“URL”})

@Test

Public void WebloginCarLoan(String urlname)

In the above example (and always), the Parameter is applicable only to the test case below it.

Here, the URL is passed from the xml file to the URL declared in parameters, and then passed and stored in the String urlname. And from there we can further use it – i.e. driver.get(urlname).

**Note:** We have to run the test from xml file, it won’t work directly by running the class.

**Note:** If we want to parameterize two or three values, simply write the same statement below the above one in xml file with intended name and value, and in the actual class in parameters line, simply separate the parameter names by comma inside the braces, and in argument section in this fashion: (String urlname, String username).

**Parameterizing with multiple data sets by running tests with multiple combinations:**

You can achieve it by the help of *data provider annotations*.

Example: we want to test login feature with three different sets of username and password:

First define a multi-dimensional object array in the test case or method:

@DataProvider

Public Object[][] getData()

{

Object [][] data = new Object [3][2] //defining array with three rows and two columns

//each combination is a row, column is values

//first set:

Data[0][0] = “firstsetusername”;

Data[0][1] = “firstpassword”;

//second set:

Data[1][0] = “secondsetusername”;

Data[1][1] = “secondpassword”;

//third set:

Data[2][0] = “thirdsetusername”;

Data[2][1] = “thirdpassword”;

Return data;

Now in another method/test case:

@Test(dataProvider=”getData”) //go and catch ‘dataProvider which has name of ‘getData’

Public void MobilesignoutcarLoan (String username, String password)

{

System.out.println(username);

System.out.println(password);

}

Interview question: how do you achieve parameterization in TestNG?

Answer: you can do it in two ways. (1) drive the data from xml file and (2) you can parameterize using dataProvider annotations.

**TestNG Listeners:**

Scenario: take a screenshot only when a test case fails. With TestNG we can reroute our execution to a specific block when a test fails, and there in that block we can write screenshot script.

Listeners listen to your execution results. When it hears that something fails, it will invoke a particular method, when it hears that something succeeded it will invoke another particular method.

To implement Listeners, we use ITestListeners interface which has all TestNG listeners.

Start with creating a new class and call it ‘Listeners’.

In the class, we use a keyword called ‘implements’:

Public class Listeners implements ITestListener {

//hover the mouse over ‘Listeners’ and click on ‘Add unimplemented methods’ and you will get all the listeners methods.

Now in the onTestFailure method, we can write the script for taking screenshot.

Now, very important, we need to tell xml file where our Listeners class is located, if we don’t, it will not redirect to Listeners upon test failure. And we can do this by typing the following after the ‘suite’ and below the ‘test’ folder in xml file:

<listeners>

<listener name=”test.Listeners”/>

</listeners>

**Note:** After a test is failed, execution will be rerouted to listeners and to the onTestFailure block. Now if we want to know exactly after which test case execution was failed and routed to the onTestFailure block, we write the following in the onTestFailure block of Listeners class:

System.out.println(“execution of this method failed:” +result.getName());

Interview question: when you catch listeners, how do you know the test case name?

Answer: we have a class called iTestResult: which describes the result of a test.

**Running Tests Parallelly with TestNG:**

(It is not preferred way in real work environment. Because the performance will be affected.)

Reword the ‘Suite’ line of the xml file as follows:

<suite name=”Loan Department” parallel=”tests” thread-count=”2”>

Which means run test cases of test folders parallelly, two tests at a time.

We can also run test cases of classes parallelly, in that case we declare the same code in the test line and just change “tests” to “classes”.

**The Test Output Folder (Test Results):**

* Right click on the TestNG project and click on refresh, the test-output folder will appear.
* Expand the test-output folder.
* Right click on index.html file.
* Click on properties.
* Copy the entire location path.
* Paste it in the address bar of your browser to see the Test Results.

**Global Environmental Variables & Reusable Components:**

***Parameterize username, password and URL:***

This kind of globalized parameterization is done using Java.

First create a file: right click on the project name, click on new, click on other, select file, next, name it ‘*datadriven.properties’* (we must enter *.properties* extension), finish.

In that file type the followings (using syntax: key=value)

Username=selenium

Password=Rahul123

url=www.facebook.com

Now create a new class *and name it something like TestBase,* type:

Properties prop=new Properties();

//properties is a class in Java, we call it here.

FileInputStream fis = new FileInputStream (“path of file, in Java you need to type double slashes”)

//This is a class in Java and we provide the path of the file we created above as argument to this class.

//Now we connect properties and FileInputStream classes, and we use the “load” method of Properties class as follows:

Prop.load(fis);

System.out.println(prop.getProperty(“username”));

***Parameterization of browsers:***

In the ‘*datadriven.properties’* file that we created above, we type:

Browser=firefox

Now go to the class we created above (which has imported the properties and FileInputStream classes) and type:

If(prop.getProperty(“browser”).equals(“firefox”))

{

driver = new FirefoxDriver();

//we usually type: webdriver driver = new FirefoxDriver();

//but since the scope of statements are limited inside the if clause, we are going to remove ‘webdriver’ from here and we declare it globally above the @Test annotation/below the public class TestBase{ line to expand its scope outside of the if clause. So this is how we declare it: *Public WebDriver driver = null;*

}

else if (prop.getProperty(“browser”).equals(“chrome”))

{

driver = new ChromeDriver();

}

else

{

driver = new InternetExplorerDriver();

}

Driver.get(prop.getProperty(“url”));

Now we can inherit the properties of the class above (TestBase) in other classes in order to avoid all that coding:

So in another class we type:

Public class DependencyAnnot extends TestBase {

@Test

Public void OpeningBrowser()

{

Login();

}

**MAVEN**

Apache Maven is a software project management and build management tool for Java Frameworks (i.e. Selenium Framework, Rest Assured Framework, Apium Framework).

**Why Maven:**

* Central repository to get dependencies: JARs of all famous softwares are available in Maven repository. We just copy the dependencies, and Maven manages to download their JARs. It also helps to maintain the right version of JARs.
* Maintaining common structure across the organization: Maven gives us its own standard template.
* Flexibility integrating with CI (Continuous Integration) tools -i.e. Jenkins: we need a build management tool to provide data to CI and Maven does it.
* Plugins for test framework execution: it supports TestNG and Junit
* Helps to execute framework from command prompt
* Can be used to create exe files

**1. Install Maven:**

* Go to maven.apache.org/download.cgi
* Download the bin.zip fine and unzip it

**2. Set system variables to recognize Maven:**

* After following the Install Maven steps, copy the path of the unzipped file, go to Advanced System Settings, Environment Variables, New, put name as “Maven\_Home, and paste the file path there.
* Then go to the bin file and copy its path, then go to system variables and find the path variable and paste the path there.
* Now go to command prompt and type: mvn –version: if you see all the related info, it means that Maven is properly configured in your system.

**Understanding Maven terminologies:**

Artifact: An artifact is a file, usually a JAR, that gets deployed to a Maven repository.

Groupid: groupid will identify your project uniquely across all projects.

Archetype:generate: it is a keyword used to create a new project from an archetype.

**3. Creating Maven Project:**

There are two ways to create a Maven project: (1) in Eclipse and (2) via command prompt.

*It is better to know the command technique because if the employer is not using Eclipse, we will still be able to create a Maven project.*

1. Creating Maven Project via Eclipse:

Click on file > new > Maven Project > Create a simple project > next > Group Id (PackageName) Artifact ID (ProjectName) > Finish.

1. Creating Maven Project Via Command Prompt: to to command prompt and type:

mvn archetype:generate -DgroupId=qaclickacademy -DartifactId=Mavenjava -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

In the above syntax: -D stands for parameter, ArchetypeArtifactId is the template and quickstart is one those templates. artifactId will be the project name and groupId the project-packaging.

All utilities go into the main folder, all your test cases will go to the test folder.

**4. Integrate Maven with Eclipse:**

Since we have created the Maven project in command prompt and it is just a skeleton/template of Maven project, we need to create .classpath and .project files to make the project compatible with eclipse:

* When you created the Mavenjava project via command prompt in the steps above, the project folder was saved somewhere in the computer
* Navigate to the Mavenjava project folder in command prompt
* Type mvn eclipse:eclipse and hit enter. It will create two files (.classpath and .project)
* In Eclipse go to File, import, expand Maven, Existing Maven Projects, next, browse > navigate to Maven folder and choose it > check the checkbox next to pom.xml… Finish.

**Understanding POM.xml file and its dependencies:**

* Maven is used across Java based frameworks (RestAPI, Appium and Selenium testing tools).
* First, we need to copy Selenium dependencies from Maven repository (their website) into the <dependencies> tag of pom.xml file.
* Second, we have to copy TestNG dependencies to the POM.xml file.
* Third, copy REST Assured dependencies and Appium dependencies to the POM.xml file.
* Maven runs all test cases at once. Maven Surefire Plugin is used to execute all test cases in Maven project: we need to copy its dependencies and paste it above the <dependencies> in POM.xml file.
* Even when Eclipse is not open, Maven runs the tests via command prompt. The three important MVN commands are:
  + **mvn clean** (it will delete all temporary files or in other words cleans the project. Example: if we have build errors and fix them, then we have to use the clean command.
  + **mvn compile** (it compiles the code and tells you if there is a syntax error.)
  + **mvn test** (it actually triggers execution of test cases. If we directly run test command without running the ‘clean’ and ‘compile’ commands, by default it will first do those two commands and then do testing.
* Maven always reads POM.xml file first, then it looks for the JARS in the local folder (local maven repository), if some are not available, it will get them from the online repository.
* We must have “Test” word at the end of the class name for Maven to recognize and runt it.

**TestNG Integration with Maven:**

* In the previous section, we learnt how to execute all Test Cases with ‘mvn test’ command
* Here we want to use the TestNG features of running selected test cases. For that purpose we need to create the TestNG.xml file.
* So right click on Mavenjava project, TestNG, Convert to TestNG, next, finish.
* Now we need to tweak the surefire plugin dependencies inside the POM.xml file and paste the TestNG.xml file dependencies inside it in order to make Surefire Plugin run the TestNG.xml file (flexible configuration of tests to be run) instead of executing all test cases of Maven by default:
  + For that purpose, copy the following from the maven website and paste it after the version line of Surefire Plugin dependencies:

<configuration>

<suiteXmlFiles>

<suiteXmlFile>testng.xml</suiteXmlFile>

</suiteXmlFiles>

</configuration>

* + Now, if we run tests from command line by typing ‘mvn test’, the TestNG.xml file will be executed.
* If we want to run one single test case, we can use the (mvn -Dtest=AppiumTest test) command (also described in the maven website) in command prompt.

**Profiling in Maven:**

Lets say we need two different TestNG.xml files (one for regression testing and the other for smoke testing). It is not ideal to change the name of the TestNG.xml file inside the Surefire Plugin dependencies in POM.xml file for each requirement. Rather, we can use the profiling technique.

So we create the following inside of POM.xml file above the <build> tag of Surefire Plugin:

<profiles>

<profile>

<id>Regression</id>

</profile>

</profiles>

Now we cut the entire dependencies of Surefire Plugin and paste it inside of the profiles after the <id>Regression</id> line as shown below:

<profiles>

<profile>

<id>Regression</id>

<build>

<pluginManagement>

<plugins>

All of the remaining until:

</build>

</profile>

</profiles>

And we can create another profile and name it <smoke>, copy paste the above dependencies and only change the name of the TestNG.xml file.

Now, to run a specific profile via command line, type the following in command prompt:

Mvn test -PRegression //p stands for profile, followed by profile’s name

**Note: In real work environment, we don’t run test cases from command prompt, rather we usually run it via Jenkins.**

**JENKINS:**

* Jenkins is a continuous integration tool
* With Jenkins we can have one common URL for our tests that we can share with our team and that will work irrespective of what the TestNG versions others have on their computers.
* Jenkins runs on its own server, so we need to set Java and Maven versions in it and Jenkins will run based on those versions. (this is another feature of Jenkins as it eliminates the issues of version incompatibilities that arise across different machines of different staff members when they don’t use Jenkins and solely rely on local machine programs like Eclipse).
* Jenkins helps schedule test runs
* Jenkins automatically sends email to all users when there is any compilation errors

**Download Jenkins:**

*(In real work environment DevOps configure server host so we don’t have to install Jenkins).*

* Go to Jenkins website and download Generic Java Package.war file.
* Now open Command Prompt and navigate to the path where the downloaded Jenkins.war file is located.
* Now type: java -jar Jenkins.war -httpPort=9090 //on which port you want to initialize it
* Now in the URL box of browser, type: localhost:9090 and it will take you to Jenkins sign in page

**Configure Jenkins:**

* After you sign in to Jenkins, click on Manage Jenkins
* Click on Global Tool Configuration
* Click on JDK installation, copy the path of Java JDK from your machine (the one you set in system variables for Java) and provide it here, or you can simply select ‘Install automatically’
* Click on Maven installations, copy the path of Maven from system variables and give it here
* Save the settings

**Jenkins Job Configuration:**

* Click on New Item > add item name i.e. “MavenJob” > choose Freestyle Project (which is common) > click on OK.
* In real job environment, companies upload their codes to GitHub, so we can pass the link and Jenkins will automatically get the code from GitHub.
* But to upload code from our local computer, we click on advanced > use custom workspace.
* Now on the actual computer, navigate to Maven directory and copy the entire folder of Mavenjava and paste it in the .jenkins directory.
* Now back to Jenkins website: type the following in the directory field:
  + ${JENKINS\_HOME}/Mavenjava
  + This means: go to .jenkins directory and then look for pom.xml in Mavenjava directory
* Now under Build Triggers > Build Periodically > we can schedule test execution:
  + M (minute) H (Hour) D (Day) M (Month) DAYWeek
  + 0 7 \* \* \*
  + It means: Run at zeroth minute, At 7 am, every day, every month, every day of the week
  + Test execution will automatically be executed at the above specified time of the days.
* Under build click on build step > Invoke top level maven targets > choose maven from dropdown menu (this is what you had set up earlier using the maven path).
  + Under goals, type the mvn command: test -PRegression
* Now when you click on build now, it will run the test.

**TestNG Reporting Plugin for Jenkins Jobs:**

*In eclipse, if you expand the target folder (where pom.xml and TestNG.xml along some others files are located) of Mavenjava project and then expand ‘surefire reports’ you will see index.html. If you right click > properties > copy the path and paste it in URL, you will get the test report, but that report doesn’t look very good. We need a better report, for that purpose we install TestNG plugin in Jenkins:*

* In Jenkins, go to Manage Jenkins > Manage Plugins > search for testng > install TestNG Results Plugin.
* Go back to Jenkins main screen > click on your project ‘MavenJob’ > click on configure (this is where you have set your project job and maven commands > click on Post-built Actions > select Publish TestNGResults > it will prepopulate the file ‘testng-results.xml’ which is created by TestNG in the folder explained in the first paragraph.
* Now, if you execute your test from Jenkins (build now), you will get a link for TestNG results graph and TestNG Results link, click on it you will get a detailed report that will include the names of all classes as well as methods, the date and time when test took place, the duration of test and the result.

**APACHE POI API:**

It is a Java API to access Microsoft format files.

**Downloading POI Jars:**

Go to maven central repository website and copy the dependencies of ‘poi’ as well as ‘poi ooxml’, paste the dependencies in the Maven POM.xml file.

**Strategy to Access Excel Data:**

* Create and save your Excel document – i.e. we saved it as ‘Demodata’.
* There is a class named SXXFWorkbook in the POI API which has all the required methods used to drive data from Excel. So we create object for that class:

SXXFWorkbook workbook = new SXXFWorkbook ();

* The SXXFWorkbook accepts a fileInputStream argument inside the parentheses. fileInputStream is a Java class which can read many types of documents. So we need to create object for the fileInputStream class too:

Public ArrayList<String> getData (String testcaseName) //it is a method

ArrayList<String> a = new ArrayList<String>(); //comes from one of the steps below.

FileInputStream fis = new fileInputStream(“c://users//nwol//documents//Demodata.xlsx”);

SXXFWorkbook workbook = new SXXFWorkbook (fis);

// Here we know that the fis object is able to access the ‘Demodata’ Excel file and that ability is passed to ‘workbook’ object.

// Now we want to access ‘testdata’ sheet of the Demodata Excel File, so we scan all the available sheets to identify and target the intended sheet:

Int sheets = Workbook.getNumberOfSheets();

For(int i=0; i<sheets; i++)

{

If(workbook.getSheetName(i).equalsIgnoreCase(“testdata”))

{

XSSFSheet sheet = Workbook.getSheetAt(i);

//here when we type workbook.getSheetAt… the editor suggests the return type of ‘XSSFSheet sheet’ to us and we simply follow the suggestion. The same is the case in the following steps:

//Now, identify testcases column by scanning the entire first row:

Iterator<Row> rows = sheet.iterator(); //sheet is a collection of rows

Row firstrow=rows.next();

Iterator<cell> ce =firstrow.cellIterator(); //row is a collection of cells

Int k=0;

Int column = 0;

While (ce.hasNext())

{

Cell value = ce.next();

If(value.getStringCellValue().equalsIgnoreCase(“Testcases”))

{

Column = k; //desired column

}

K++;

System.out.println(column);

//Once column is identified then scan entire testcase column to identify purchase testcase row:

While (rows.hasNext())

{

Row r=rows.next();

If(r.getCell(column).getStringCellValue().equalsIgnoreCase(“Purchase”))

{

//After you identify the purchase testcase row, pull all the data of that row and feed it into test:

Iterator<cell> cv = r.cellIterator();

While(cv.hasNext())

{

//Now we are going to create an arraylist that accepts only string (created at the first line of the code) and then to put all the values in it:

Cell c = cv.next();

If(c.getCellTypeEnum()==CellType.STRING) //cellTypeEnum finds if it is string, number …

{

a.add(c.getStringCellValue());

else {

//like parsing, we need to convert the integer into string to be able to put it in the arraylist that accepts only string. POI API has come up with a converter:

a.add(NumberToTestConverter.toText(c.getNumericCellValue()));

}

}

}

}

}

Return a;

}

//we decide to add this whole code in a method, then create a separate class, create object of this class in it to call this method.

Now we create another class and name it ‘testSample’. In this case, we create object for the dataDriven class in order to use its method:

dataDriven d=new dataDriven();

ArrayList data = d.getData(“Add Profile”);

//For example, you are using it with Selenium, and let’s assume that the first index has the usernames and you want to pass it:

Driver.findElement(by.xpath(“xxxx”)).sendKeys(data.get(1));

**APACHE LOG4J**

It is a reliable, fast and flexible logging framework (APIs) written in Java, which can be integrated with Selenium framework to log all of the execution output.

Log4j can be used for all Java based frameworks (Selenium, Appium, REST API) and its usage is the same across all of the mentioned three Java based testing frameworks.

**Why Log4j?**

*Client questions:*

* Send all the logs to a file.
* For Package A, log only errors (Don’t print the success logs like: clicked on sign in button, navigated to homepage …)
* Log everything for Package B.
* I want logs with Timestamp (at what particular time something happened)
* I want the log file of last week.
* How will I know if there is an error by just looking at logs? (error logs vs. success logs)

**Importing Apache Log4j jars into Eclipse project:**

* Google ‘download Apache Log4j)
* Download bin.zip and unzip it
* Right click on your project > properties > Java Build Path > Add external JARs > only add Log4j-core and Log4j-API (these two are enough) > import them.

**Setting up logs:**

LogManager is an API and GetLogger is a function or method inside it. We create an object for it and use it to log any step we want to.

Private static Logger Log = LogManager.getLogger(Demo.class.getName());

//demo is the name of the class //getLogger takes the class argument

Log.debug(“I have clicked on button”);

Log.info(“Button is displayed”);

Log.error(Button is not displayed”);

Log.fatal(“Button is missing”);

Use ***log. Error()*** and ***log.fatal()*** to log when elements are not displayed in page or any validations fail.

Use ***Log. Debug()*** when each Selenium action is performed like click, SendKeys, getText().

Use ***log.info()*** when operation is successfully completed ex: After loading page, or after any successful validations.

**Importance of Log4j xml configuration file:**

**Log4j has 3 basic components**: Loggers, Appenders, and Layouts:

Where to log? -- ***Appenders tag***: if you give the console tag, the output will go to console, if you give a file tag, the output will go to the file.

What to log? -- ***Logger tag:*** if you have <Root level=”error”> it means we want to log errors only. If we replace ‘error’ with ‘trace’ it will log/print everything.

How to log? -- ***Layout tag:*** the patter or the format of how the logs are presented.

**Create Log4j xml configuration file:**

* Right click on your package > click on new > other > expand XML > XML file > next > name it “Log4j2.xml > next > finish.
* Copy all the configuration script from Log4j website and paste it in the Log4j2.xml file.
* You can bring changes in the Appenders, Logger and Layout tags.
* Important: right click on the project > Build Path > Configure Build Path > Source > Add Folder > check “resources” which is the folder where your Log4j2.xml file is located > OK.

**Client wants to log errors for “Demo1” class and everything for “Demo2” class:**

* In the Log4j.xml file:
* <Logger name = “*alpha.Demo1*” level=”*error*” additivity=”false”>
* Where ‘alpha’ is the package name and ‘Demo1’ is the class name
* In the above line we have asked xml file to log only errors for the class “Demo1”, now we have another line in xml file which is: <root level = “trace”> which basically applies to all packages and classes. So if we have “error” for Demo1 class and “trace” for root level, it means that we want to log only errors for Demo1 and for all of the remaining classes we want to log everything.
* Additivity=”false” means that we don’t want the root level to log the ‘Demo1’ again because we have already set properties for logging it in the logger section.

**How to print all the logs in a file rather than in console:**

* Paste the following code below the <Appenders> tag of Log4j.xml file:

<RollingFile name="File" fileName="${basePath}/prints.log" filePattern="${basePath}/prints-%d{yyyy-MM-dd}.log">

<PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>

<SizeBasedTriggeringPolicy size="500" />

</RollingFile>

* In this code, the patternLayout which refers to the formatting of logs, is the same as what we get by default. The <sizeBasedTriggeringPolicy size=”500”/> basically means that when the size of the file reaches 500kb, automatically create another file for the remaining logs, and for those subsequent files follow the naming format specified in filePattern section.
* Now that we have both ‘file’ and ‘console’ destinations for our logs, we can assign either or both to a specific class in the <AppenderRef> sections under <Logger> tag or to the entire project in the <AppenderRef> section under <Root Level> tag.
* **To create a file for passing all the logs into it:** Right click on your package > new > other > General > United Text File > Finish. Type something in it and save it under the name of ‘prints.logs’.
* Now above <Appenders> tag in xml file, pass the path of the ‘prints.logs’ file by typing:

<Properties>

<Property name=”basePath”>./logs</Property>

</Properties>

*Here ‘basePath’ is the project level, from which we want to navigate to the logs file.*

* Then pass the name of the logs file in filename section of <RollingFile> tag.
* The complete Log4j.xml file will look like the following after we bring all our desired changes:

<?xml version="1.0" encoding="UTF-8"?>

<Configuration status="WARN">

<Properties>

<Property name="basePath">./logs</Property>

</Properties>

<Appenders>

<RollingFile name="File" fileName="${basePath}/prints.log" filePattern="${basePath}/prints-%d{yyyy-MM-dd}.log">

<PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>

<SizeBasedTriggeringPolicy size="500" />

</RollingFile>

<Console name="Console" target="SYSTEM\_OUT">

<PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>

</Console>

</Appenders>

<Loggers>

<Root level="trace">

<AppenderRef ref="File"/>

</Root>

</Loggers>

</Configuration>

**Page Object Model:**

Why do we adopt Page Object Pattern for our framework design?

* It is easy to maintain
* Easy readability of scripts
* Reduce or eliminate duplication
* Reusability of code
* Reliability

Keyword driven, hybrid and other frameworks are outdated.

The pattern is the same for Web platform and Mobile platform.

For example: we are creating two packages one named “objectrepository” for storing the objects or web elements and second named “Testcases” for storing our test cases. Inside objectrepository package we create one Java class named “homepage” and defining all the objects of home page in it and another class named “loginpage” and adding the login page objects in it. The goal is, for example when we need the username object in a test case, we simply call it from the objectrepository (specifically: loginpage class).

The Loginclass (RediffLoginpage.java class) will look like this:

package objectrepository;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

public class RediffLoginpage {

WebDriver driver;

public RediffLoginpage(WebDriver driver) {

// TODO Auto-generated constructor stub

this.driver=driver;

}

By username=By.xpath(".//\*[@id='login1']");

By Password=By.name("passwd");

By go=By.name("proceed");

By home=By.linkText("Home");

public WebElement Emaild()

{

return driver.findElement(username);

}

public WebElement Password()

{

return driver.findElement(Password);

}

public WebElement submit()

{

return driver.findElement(go);

}

public WebElement Home()

{

return driver.findElement(home);

}

}

*The meaning of the constructor we have used in the loginpage class (of objectrepository package): I’m passing my testcase driver into the loginpage, and assigning the testcase driver to local driver, so that the local driver behaves as testcase driver.*

The Testcase (*Loginapplication.java* class) will look like this:

package Testcases;

import objectrepository.RediffHomepage;

import objectrepository.RediffLoginpage;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

import org.testng.annotations.Test;

public class Loginapplication {

@Test

public void Login()

{

System.setProperty("webdriver.chrome.driver", "C:\\chromedriver.exe");

WebDriver driver=new ChromeDriver();

driver.get("https://mail.rediff.com/cgi-bin/login.cgi");

RediffLoginpage rd=new RediffLoginpage(driver);

rd.Emaild().sendKeys("hello");

rd.Password().sendKeys("hello");

//rd.submit().click();

rd.Home().click();

RediffHomepage rh=new RediffHomepage(driver);

rh.Search().sendKeys("rediff");

rh.Submit().click();

*Note: some objects are coming from another class of repository package.*

**Page Object Factory:**

The page object classes can be designed using a style called Page Object Factory which is more simplified. So if we change the same class (RedfinLoginpage.java) to PF, it will look like this:

package objectrepository;

public class RediffLoginpagePF {

WebDriver driver;

public RediffLoginpagePF(WebDriver driver) {

// TODO Auto-generated constructor stub

this.driver=driver;

PageFactory.initElements(driver, this);

}

@FindBy(xpath=".//\*[@id='login1']")

WebElement username;

@FindBy(name="passwd")

WebElement Password;

@FindBy(linkText="Home")

WebElement home;

@FindBy(name="proceed")

WebElement go;

public WebElement Emaild()

{

return username;

}

public WebElement Password()

{

return Password;

}

public WebElement Home()

{

return home;

**BUILDING FRAMEWORK FROM SCRATCH**

Agenda:

* Create Maven Project
* Create base and utility functions
* Organize Page Objects
* Add Tests
* Datadriving/parameterizing tests
* Convert into TestNG Framework
* Log4j API for logging inside Framework
* TestNG Listeners
* Screenshot on Test Failures
* Create HTML reports for Test Results
* Jenkins continuous Integration

**Create Maven Project:**

Create a Maven project via command line and import it to Eclipse by following the steps in Maven section of this note.

**Configure Framework dependencies in POM.xml**

Simply copy all the jar files dependencies from the maven central repository (their website) and paste them inside the Maven POM.xml file.

**Create Base and Utility functions / design base test with global variables setup:**

In the project, we have the ‘test’ and ‘main’ folders. All of our utilities and base files will go to ‘main’ folder and all of the test cases in the ‘test’ folder.

So, first create a file: right click on the ‘main’ folder name, click on new, click on other, select file, next, name it ‘*data.properties’* (we must enter *.properties* extension), finish. It is better to first create a package and call it resources and then create this file as well as the base file inside it. It will help make the visibility and readability better.

Inside the file, type the following:

Browser = chrome

*//after completing the parameterization, if we want to use a specific browser, we simply type its name in the file above – i.e. replace ‘chrome’ with ‘firefox’.*

Then, create a class named ‘base’ in the main folder and parameterize the browsers inside it:

package resources;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileNotFoundException;

import java.io.IOException;

import java.util.Properties;

import java.util.concurrent.TimeUnit;

import org.apache.commons.io.FileUtils;

import org.openqa.selenium.OutputType;

import org.openqa.selenium.TakesScreenshot;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

public class base {

public static WebDriver driver;

public Properties prop;

public WebDriver initializeDriver() throws IOException

{

prop= new Properties();

FileInputStream fis=new FileInputStream("C:\\Users\\ data.properties"); //path of data.properties file.

prop.load(fis);

String browserName=prop.getProperty("browser");

System.out.println(browserName);

if(browserName.equals("chrome"))

{

System.setProperty("webdriver.chrome.driver", "C://chromedriver.exe");

driver= new ChromeDriver();

//execute in chrome driver

}

else if (browserName.equals("firefox"))

{

driver= new FirefoxDriver();

//firefox code

}

else if (browserName.equals("IE"))

{

// IE code

}

driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

return driver;

}

public void getScreenshot(String result) throws IOException

{

File src=((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);

FileUtils.copyFile(src, new File("C://test//"+result+"screenshot.png"));

}

Now, create a class in the ‘test’ folder and type the following in it:

public class HomePage extends base{

@Test

public void basePageNavigation() throws IOException

{

driver = initializeDriver();

driver.get(“[http://qaclickacademy.com”](http://qaclickacademy.com));

**Implement Page Object Mechanism (centralized data) to Drive Tests:**

Create a new package in the main folder and call it pageObjects. Inside it, create a separate class for every page and write all the objects that we collect from each page in the relevant class.

The following is how our Page Object class will look like:

Public class LandingPage{

Public WebDriver driver;

//when the driver has no life you will get ‘Null point exception” we need to give life to the driver. Like assign Chromedriver to it. Now our actual class/test case (i.e. HomePage class) has the driver which has life, we can pass that driver as an argument for the method. The place there is marked with word ‘//\*here’.

By signin=By.cssSelector(“a[href\*=’sign\_in’]”);

//The best practice in designing page object is to define all objects at the top (like here) and methods at the bottom.

Public LandingPage(WebDriver driver) {

This.driver =driver; //1 //the first driver here is the variable of this class which doesn’t have life, the second one which has life comes from the HomePage class.

}

Public WebElement getLogin()

// Note that a method cannot be without a return type. Since we are sending WebElement, our return type will be WebElement instead of ‘void’.

{

Return driver.findElement(signin);

}

The actual class (test case) will look like this:

public class HomePage extends base{

@Test

public void basePageNavigation() throws IOException

{

driver = initializeDriver();

driver.get(“[http://qaclickacademy.com”](http://qaclickacademy.com));

//There are two ways to call / invoke objects of a class:

//1. Inheritance method, like the way how we are extending the base class.

//2. Creating object to that class and invoking its methods, like the following:

LandingPage l = new LandingPage(driver); //\*here

//now if you want to send any thing with arguments, you need to come up with a constructor.

Just move mouse over the argument and click on create constructor.

You will see that there is a new method created in the LandingPage and the green text below it

says ‘auto generated constructor stub’. Now go to LandingPage and pickup from there. //1

//now when you just type ‘l.’ you will get a list of all the methods declared in LandingPage class.

l.getLogin().click();

Note the Selenium errors:

* Null pointer exception: when there is no life
* File not found exception: when the path is wrong
* No such element exception: when there is a mistake in object locator (like mistake in xpath or cssSelector)

Note that for the objects of every page, we create a separate class – i.e. LandingPage, LoginPage. But the actual class (used to run the testcases) is one (i.e. HomePage in these examples).

After creating the LandingPage and LoginPage object classes and mapping them to the test cases, the actual class (test case) will look like this:

public class HomePage extends base{

@Test

public void basePageNavigation() throws IOException

{

driver = initializeDriver();

driver.get(“[http://qaclickacademy.com”](http://qaclickacademy.com));

LandingPage l = new LandingPage(driver);

l.getLogin().click();

LoginPage lp = new LoginPage(driver);

Lp.getEmail().sendKeys(abc@qw.com”);

Lp.getPassword().sendKeys(“asdfhzxl23”);

Lp.getLogin().click();

**We can further eliminate the need for hardcoding data by Parameterization:**

i.e. in the HomePage class above, we can parameterize the URL, username and password. Both data parameterization and data driving is achievable with the help of @DataProvider annotation of TestNG:

So the HomePage class will look like the following:

public class HomePage extends base{

@Test(dataProvider=”getData”) //we are saying go and find the dataProvider called getData.

public void basePageNavigation(String Username, String Password, String Text) throws

IOException //the count of arguments we are sending per each test should be equal to the

Count of the values per each row.

{

driver = initializeDriver();

driver.get(“[http://qaclickacademy.com”](http://qaclickacademy.com));

LandingPage l = new LandingPage(driver);

l.getLogin().click();

LoginPage lp = new LoginPage(driver);

Lp.getEmail().sendKeys(Username”);

Lp.getPassword().sendKeys(“password”);

System.out.println(text); //just added to make our values number to 3 for better

explanation of the example

Lp.getLogin().click();

@DataProvider

Public Object[][] getData() //we had first typed (public void getData() but since we are passing

array here, we have to change the return type.

{

//row stands for how many different data types test should run

//for example we want to run this test for two types of data: one for a restricted student

//the other for a non-restricted student.

//column stands for how many values per each test.

Object[][] data = new object [2][3]; //actual count of rows and columns not the index count

//0th row

Data[0][0] = “nonrestricteduser@qw.com”;

Data[0][1] = “323aksh”;

Data[0][2] = “Non-restricted User”;

//1st row

Data[1][0] = “restricteduser@qw.com”;

Data[1][1] = “g23aksh”;

Data[1][2] = “restricted User”;

Return data;

//So now that we have mapped the getData set of values to the test, it will pick the first row and

complete the full test, then it will pick the second row and complete a full test.

//that is how we achieve both parameterization and driving data with getData.

}

Clean code:

public class HomePage extends base{

@Test(dataProvider=”getData”)

public void basePageNavigation(String Username, String Password) throws IOException

{

driver = initializeDriver();

driver.get(“[http://qaclickacademy.com”](http://qaclickacademy.com));

LandingPage l = new LandingPage(driver);

l.getLogin().click();

LoginPage lp = new LoginPage(driver);

Lp.getEmail().sendKeys(Username”);

Lp.getPassword().sendKeys(“password”);

Lp.getLogin().click();

@DataProvider

Public Object[][] getData()

{

Object[][] data = new object [2][2];

Data[0][0] = “nonrestricteduser@qw.com”;

Data[0][1] = “323aksh”;

Data[1][0] = “restricteduser@qw.com”;

Data[1][1] = “g23aksh”;

Return data;

}

**Asserting the Testcases with validations:**

We create a new class in the test folder of our project and name it ‘validateTitle’, but we declare the object in the previous LandingPage class. Code inside the class validateTitle is as follows:

public class validateTitle extends base{

@Test(dataProvider=”getData”)

public void basePageNavigation() throws IOException

{

driver = initializeDriver();

driver.get(“[http://qaclickacademy.com”](http://qaclickacademy.com));

LandingPage l = new LandingPage(driver);

//compare the text from the webpage with actual text. – if not matching: Error ..

Assert.assertEquals(l.getTitle().getText(), “FEATURED COURSES”);

Other assertions:

Assert.assertTrue(l.getNavigationBar().isDisplayed()); //use when we expect something to be true.

Assert.assertFalse(false);

We shouldn’t hardcode the URL, we want to define it in one place, so we put it in the data.properties file:

Browser = chrome

url = “http://qaclickacademy.com”

And the validateTitle.java class will look like this:

public class validateTitle extends base{

@Test

public void basePageNavigation() throws IOException

{

driver = initializeDriver();

driver.get(prop.getProperty(“url”));

LandingPage l = new LandingPage(driver);

//compare the text from the webpage with actual text. – if not matching: Error ..

Assert.assertEquals(l.getTitle().getText(), “FEATURED COURSES”);

**Converting Framework into TestNG and configuring TestNG XML file:**

Right click on your project > TestNG > Convert to TestNG > Next > Finish. This will convert the project to TestNG and will also automatically create the TestNG XML file for us.

We can use the @AfterTest annotation of TestNG for closing the web browser after test execution:

@AfterTest

Public void teardown()

{

Driver.close();

}

**TestNG and Maven Integration:**

The concept is to include the TestNG XML file details into the POM.xml file, so that when we run POM.xml file, it triggers the TestNG xml file and TestNG xml file will trigger all the test cases.

Copy the TestNG suite xml files code from Maven website, create a ‘build’ tag in the POM.xml above the dependencies tag and paste the code inside it, which will look like the following:

<build>

<plugins>

<plugin>

….

</plugin>

</plugins>

</build>

**Execute testcases via Command Prompt:**

* Navigate to the maven project folder (in Eclipse right click on project name > properties > get the path, then in command prompt type cd followed by the path of Maven project and hit enter
* Mvn compile (it will tell us if we have any syntax errors, it also downloads the jars that are not available).
* Mvn test (it will execute the tests).

**Integrating Log4j Logging API into Framework:**

First copy the dependencies of Log4j from the official Apache website and paste them inside the dependencies tag of the maven project’s pom.xml file.

In the Log4j section we had created the Log4j2.xml file, we can paste that somewhere on computer and copy and then paste it in our project by right clicking on the resources package in main folder of our project and then clicking on paste.

We need to the following in every test case in order to be able to use the logging mechanism:

Private static Logger Log = LogManager.getLogger(base.class.getName());

And then we can declare the logs, so the class will look like this:

public class validateTitle extends base{

Private static Logger Log = LogManager.getLogger(base.class.getName());

@BeforeTest

Public void initialize () throws IOException

{

driver = initializeDriver();

log.info(“Driver is initialized”);

driver.get(prop.getProperty(“url”));

log.info(“Navigated to Homepage”);

}

@Test

public void basePageNavigation() throws IOException

{

LandingPage l = new LandingPage(driver);

Assert.assertEquals(l.getTitle().getText(), “FEATURED COURSES”);

Log.info(“successfully validated Text Message”);

Now, for our pom.xml to know where exactly the log4j2.xml file is located, otherwise Maven will not be able to locate and run the file:

Google ‘**maven resource filtering**’ to copy the plugin code. Get the resources code (as follows), create <build> tag, inside it create <resources tag and inside it paste the code, so that it will look like:

<build>

<resources>

<resource>

<directory>src/main/resources</directory>

<filtering>true</filtering>

</resource>

</resources>

</build>

That step basically tells maven to consider the files located in source > main > resources folder.

**Refresh Project to See the Logs Folder: by right clicking on the project name and clicking on Refresh.**

Question**:** though we have used the @AfterTest annotation for closing browser after executing each testcase in each class, why do the browsers still stay open until all testcases are executed and then at the end they all close at once?

Answer**:** because in the TestNG.xml file we have wrapped all the classes under a single test tag. And the solution is to wrap each class inside separate test tag.

**Screenshots on Failure – TestNG Listeners:**

We want to define our screenshot method in base.java class so that it is applicable to all testcases.

public void getScreenshot(String) throws IOException

{

File src=((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);

//so now we need to copy that screenshot into our local computer: we use the FileUtils.copyfile method:

FileUtils.copyFile(src, new File("C://test//"screenshot.png"));

}

Now we want to trigger this method only when a test fails. So we use the TestNG listeners concept:

Create a class and name it ‘Listeners’. We need to implement the ITestListeners interface to be able to utilize all TestNG listeners:

Public class listeners implements iTestListener {

Base b = new base(); //we create base class object to be able to access all the methods of that class here.

Now type the following in the onTestFailure block (all the blocks come automatically after we import the interface.):

b.getScreenshot();

Now we need to tweak TestNG.xml file so that it considers the listeners in our listeners class: So inside the TestNG.xml file, we type the following below <suite> tag:

<listeners>

<listener class-name=”Academy.listeners”/>

</listeners>

Now, we don’t want the screenshots to be overwritten when other test cases fail. So we have to assign dynamic names to our screenshots. And we do that by passing ‘result’ as argument in the onTestFaulure block (*public void onTestFailure(ITestResult result) {* ) of listeners class and pass result.getName() as argument inside the block. So the class will look like:

Public void onTestFailure(ITestResult result) {

Try {

b.getScreenshot(result.getName());

} catch (IOException e) {

e.printStackTrace();

}

}

And then we tweak the screenshot block of the base.java class as follows:

public void getScreenshot(String result) throws IOException

{

File src=((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);

//so now we need to copy that screenshot into our local computer: we use the FileUtils.copyfile method:

FileUtils.copyFile(src, new File("C://test//"+result+"screenshot.png"));

}

**Reports on Test Execution:**

XSLT Report: it is one way of reporting. You open index.html in browser and see the TestNG results.

Extent Report: it beautifully presents the results with filters and charts, and it covers log and screenshot as well.

* Go to extentreports.com, click on Maven and copy the dependencies. Paste the dependencies in the dependencies section of in POM.xml file of your Maven project.
* Copy the extent report class file from the extentreports.com website and simply paste it in the resources package of main folder in your maven project.
* Tell TestNG.xml file where exactly the ExtentReporterNG file is located by typing the following in the <listeners> tag, just like we did for the TestNG listeners:

<listener class-name=”resources.ExtentReporterNG”/>

* After executing the project via maven, you can refresh your project by right clicking on the project name and selecting refresh. Then go to target folder > surefire reports and find ExtentReportsTestNG. Copy its path paste it in the browser URL box, it will show the results.

The following is the entire code of ExtentReportsTestNG class: (according to Rahul Shetty, we don’t need to bother understanding the methods inside it).

public class ExtentReporterNG implements IReporter {

private ExtentReports extent;

public void generateReport(List<XmlSuite> xmlSuites, List<ISuite> suites, String outputDirectory) {

extent = new ExtentReports(outputDirectory + File.separator + "ExtentReportsTestNG.html", true);

for (ISuite suite : suites) {

Map<String, ISuiteResult> result = suite.getResults();

for (ISuiteResult r : result.values()) {

ITestContext context = r.getTestContext();

buildTestNodes(context.getPassedTests(), LogStatus.PASS);

buildTestNodes(context.getFailedTests(), LogStatus.FAIL);

buildTestNodes(context.getSkippedTests(), LogStatus.SKIP);

}

}

extent.flush();

extent.close();

}

private void buildTestNodes(IResultMap tests, LogStatus status) {

ExtentTest test;

if (tests.size() > 0) {

for (ITestResult result : tests.getAllResults()) {

test = extent.startTest(result.getMethod().getMethodName());

/\*test.getTest(). = getTime(result.getStartMillis());

test.getTest().endedTime = getTime(result.getEndMillis());\*/

for (String group : result.getMethod().getGroups())

test.assignCategory(group);

String message = "Test " + status.toString().toLowerCase() + "ed";

if (result.getThrowable() != null)

message = result.getThrowable().getMessage();

test.log(status, message);

extent.endTest(test);

}

}

}

private Date getTime(long millis) {

Calendar calendar = Calendar.getInstance();

calendar.setTimeInMillis(millis);

return calendar.getTime();

}

}

**Integrating Project with Jenkins CI Tool:**

All process has been discussed in the Jenkins section.

**Latest Selenium Interview Questions:**

**Difference between get and navigate method in Selenium?**

GET method will get a page to load or get page source or get text that’s all. GET will wait till the whole page gets loaded i.e. the onload event has fired before returning control to our test or script. If our pages uses lot of AJAX then we can’t know that when our pages has completely loaded. To overcome this we can use WAIT.

NAVIGATE will just redirect to our required page and will not wait. It will guide us through the history like refresh, back, forward. For example if we want to move forward and do some functionality and back to the home page then this can be achieved through navigate() only.

GET and NAVIGATE do exactly the same thing, the only difference between them is that one is easier to type than other.

**Difference between quit and close methods in Webdiver?**

close( ) WebDriver command closes the Browser window which is in focus.

If there are more than one Browser window opened by the Selenium Automation, then the close( ) command will only close the Browser window which is having focus at that time. It won’t close the remaining Browser windows.

Whereas quit( ) WebDriver command is generally used to shut down the WebDrivers instance. Hence it closes all the Browser windows that are opened by the Selenium Automation.

close( ) and quit( ) work in the similar way when Selenium Automation opens only single Browser window. They differ in their functionality when there are more than one Browser windows opened by the Selenium Automation.

**What is implicit wait? Difference between implicit and explicit wait?**

Implicit Wait: During Implicit wait if the Web Driver cannot find it immediately because of its availability, it will keep polling (around 250 milli seconds) the DOM to get the element. If the element is not available within the specified Time an NoSuchElementException will be raised. The default setting is zero. Once we set a time, the Web Driver waits for the period of the WebDriver object instance.

Explicit Wait: There can be instance when a particular element takes more than a minute to load. In that case you definitely not like to set a huge time to Implicit wait, as if you do this your browser will going to wait for the same time for every element.

To avoid that situation you can simply put a separate time on the required element only. By following this your browser implicit wait time would be short for every element and it would be large for specific element.

**In how many ways we can handle frames in the application using webdriver methods?**

With FrameID, Frame name, Frame webElement

**Code to handle 3rd child window?**

String parent=driver.getWindowHandle();

// This will return the number of windows opened by Webdriver and will return Set of St//rings

Set<String>s1=driver.getWindowHandles();

// Now we will iterate using Iterator

Iterator<String> I1= s1.iterator();

I1.next()// Gets child window id

I1.next() // Get 2nd child window

I1.next() // Gets 3rd child window

**How to handle https certifications?**

DesiredCapabilities cap=DesiredCapabilities.chrome();

// Set ACCEPT\_SSL\_CERTS variable to true

cap.setCapability(CapabilityType.ACCEPT\_SSL\_CERTS, true);

// Set the driver path

System.setProperty("webdriver.chrome.driver","Chrome driver path");

// Open browser with capability

WebDriver driver=new ChromeDriver(cap);

**Different type of locators present in webdriver?**

ID, Xpath, CSS, ClassName, Name, Linktext, tagName

**Write Syntax for xpath and css if id and tagnames are given**

Xpath = //tagName[@attribute=‘value’]

CSS = tagName[attribute=‘value’]

**How to use Contains regular expression to xpath?**

//tagname[contains(@attribute,’value’)]

**How to use regular expression to CSS?**

tagName[attribute\*=‘value’]

**What is the class available in Selenium to handle drop downs?**

Select class

**What is the method to check if checkbox is selected?**

isSelected()

**How to validate if element is visible or hidden in webpages?**

isDisplayed()

**How to get the count of similar objects list in the web page?**

driver.findElements(By.xx(“”)).size();

**Importance of desired capabilities mechanism?**

The desired capability is a series of key/value pairs that stores the browser properties like browser name, browser version, the path of the browser driver in the system, etc. to determine the behavior of the browser at run time.

**How to enter the text in caps lock?**

driver.findelement("YOURELEMENTLOCATOR").sendKeys(Keys.SHIFT,"yourtexttobetyped")

**How to mouse over on the web element on page?**

Actions a =new Actions(driver);

a.moveToElement().build().perform()

**Methods to handle Java Alert?**

driver.switchTo.Alert();

**How to get links count in the page?**

driver.findElements(By.tagName(“a”).size();

**How to validate if we are navigated to child window successfully?**

By checking the title of child window

**Difference between relative and absolute xpath?**

Absolute Xpath: It uses Complete path from the Root Element to the desire element.

Relative Xpath: You can simply start by referencing the element you want and go from there.

Always Relative Xpaths are preferred as they are not the complete paths from the Root element. (//html//body) .. Because in future any of the webelement when added/Removed then Absolute Xpath changes. So Always use Relative Xpaths in your Automation.

**Write down the sample xpath syntax to handle parent from child object?**

//tagName[@attribute=value]/parent::tag name

**What driver is must to run tests in Firefox driver?**

Geckodriver

**What driver is must to run tests in Chrome driver?**

ChromeDriver

**How do you set driver in Firefox and chrome drivers through script?**

Using System.setProperty(“webdriver.chrome.driver”, “path to .exe”)

**Difference between findElement and findELements?**

FindElement identifies first object which matches with provided locator on the screen with top left scanning.

FindElements gets all the objects which matched and takes into List.

**List out any 2 methods available in explicit wait**

visibilityOfElementLocated, PresenceOfElementLocated

**How to take screenshots with selenium webdriver?**

driver.get("http://www.google.com");

// Take screenshot and store as a file format

File src= ((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);

try {

// now copy the screenshot to desired location using copyFile //method

FileUtils.copyFile(src, new File("C:/selenium/error.png"));

}

catch (IOException e)

{

System.out.println(e.getMessage());

}

}

**How to hit enter from webdriver commands?**

sendKeys(“Keys.Enter”)

**CUCUMBER FRAMEWORK**

Cucumber is a commonly used tool for BDD (Behavior Driven Development). BDD is a popularly used template in Agile projects. In BDD our requirements are in the form of test cases. Developers and Testers use the same test cases for developing and testing purposes. In traditional models used in waterfall methodologies, Business Analyst prepares requirements and developers and testers interpret it to adjust their work with it. But there was a risk of mis interpretation associated with that method. To overcome that problem, requirements are defined in business level by a ubiquitous language.

Ubiquitous Language is a semi-formal language that is shared by all members of a software development team -both software developers and nontechnical personnel.

Step 1: define business requirements:

*Syntax / template for business scenarios:*

In order to (achieve something) -> business outcome

As a (role)

I want to (do this)

*Example:*

In order to pay credit card balance

As a net banking sole owner who has credit card section access

I want to navigate to credit card section, enter the amount and process the payment

Step 2: derive test cases from the above scenarios / requirements:

*Syntax for writing testcases:*

Given (what you need to have to perform an action) – prerequisite

When (perform action) – action

Then (the desired outcome for the user) – validation

*Example:*

Given an account with zero balance

When I navigate to credit card payment section and click on submit button without providing the amount

Then It should through error message

**Advantages of BDD:**

* BDD approach introduces a Standard Template where all QA team can stick to that one common standard to define testcases.
* Each Scenario reflects a Business Value
* We can estimate the test coverage for test scenarios
* We can tag these Annotations to Selenium Automation and execute the Business Testcases
* Common Standardized Testcase template for both Manual and Automation testing.

**Install Gherkin plugin:**

*We need Gherkin plugin in Eclipse in order for it to support Gherkin language.*

In Eclipse click on Help > Eclipse Marketplace > Type Cucumber in search box and install Natural 0.7.6.

**Create a cucumber project:**

Cucumber expects a standardized skeleton for projects. We can manually create that skeleton, but it is a lot of work, so we use Maven to easily create the skeleton for us.

* In Eclipse, go to File > New > Other > expand Maven > maven project > quick start template (cucumber supports a quick start template).
* Artifact ID is project name
* Group ID is package name

**Cucumber needs Maven for two reasons:**

* To get the JAR files / dependencies
* To use the template facilities of maven

**Setup Maven’s pom.xml file for Cucumber:**

*We need to give cucumber knowledge to our Maven project.*

* Copy cucumber jars/ dependencies (cucumber java and cucumber JUnit dependencies) from maven online repository and paste them inside the dependencies tag of pom.xml file of Maven project.

**Components of Cucumber Framework:**

Feature file: this is where we define our test cases. They can be related to multiple scenarios.

StepDefination class: contains automation code. Keywords (given, when, then, etc) of feature file are defined here.

Junit Test Runner class: to connect the dots, we use this runner class to execute the testcases.

**To Create Feature File:**

* Right click on src/test/java folder and create a new package called features.
* Right click on the newly created features package > file > File name: Login.feature (we have to give the .feature extension).

Inside feature file:

**Feature**: Application Login //our requirement

**Scenario**: Home page default login

**Given** User is on NetBanking landing page

**When** User logs in to the application with username and password

**Then** Home page is populated

**And** Cards are displayed

Now, we need to map all those test cases of feature file with the code to automate them for us:

**To Create StepDefination Class:**

* Create a package named stepDefination under the test folder of maven project and then create a class named stepDefination inside it that package.

Inside stepDefination class:

Package stepDefination;

Public class stepDefination {

@Given(“^User is on NetBanking landing page$”)

Public void user\_is\_on\_NetBanking\_Landing() {

//code to navigate to login url

}

@When(“^User logs in to the application with username and password$”)

Public void User\_logs\_in\_to\_the\_application\_with\_username\_and\_password() {

//code to login

}

@Then(“^Home page is populated$”)

Public void Home\_page\_is\_populated() {

//home page validation code

}

@And(“^Cards are displayed$”)

Public void Cards\_are\_displayed() {

//validating the cards code

}

}

Note: hold down control and click on the a line in the feature file, it will take you to the relevant block in stepdefinition file, this is how we validate if the mapping is properly done.

Mapping: the tagname (i.e. Given, when, then) followed by the description (Gherkin line) in Feature and stepDefination files should match. We can use Tidy Gherkin to automate it for us.

**Tidy Gherkin:**

* Install Tidy Gherkin plugin for Chrome.
* Then go to chrome, open a new tab and in the address bar type: chrome://apps and click on Gherkin.
* Copy all the test cases from Feature file and paste it in the first box of Tidy Gherkin, in the second box, click on Java Steps and we will get the code converted for stepDefination file.
* There is another plugin called **Pretty Gherkin** that can also do the same job.
* When we don’t have the scenarios defined in the stepDefination class, and run the testrunner class, Eclipse will suggest in the entire template will all missing steps in the console section, we can simply copy it to our stepDefination file and then add our codes in the relevant blocks. This way, we won’t need to purely rely on Tidy Gherkin and Pretty Gherkin plugins.

**Test Runner Class:**

* Create a new package named cucumberOptions under the test folder of maven project and inside it create a class named TestRunner.
* Here we provide the location of feature file and stepDefinition class. So that when we run this class with **JUnit**, it will first map the steps of features file with the code blocks of stepDefination class and execute the testcases.

Inside testRunner class:

Package cucumberOptions;

@RunWith(Cucumber.class)

@CucumberOptions(

Features = “src/test/java/features”, //the path for features package, which will trigger all its classes

Glue=”stepDefinations”) //the name of stepDefinations package, because it is the child of

the same folder as feature package is

Public class TestRunner {

}

**Regular Expressions for different data but the same implementation:**

Assume that our feature file has two scenarios as follows:

**Feature**: Application Login //our requirement

**Scenario**: Home page default login

**Given** User is on NetBanking landing page

**When** User logs in to the application with username “john” and password “1234”

**Then** Home page is populated

**And** Cards are displayed

**Scenario**: Home page default login

**Given** User is on NetBanking landing page

**When** User logs in to the application with username “jin” and password “4321”

**Then** Home page is populated

**And** Cards are not displayed

There is no point in writing the implementation steps in stepDefination file two times. When we pass something in double quotes, cucumber understand that the value is dynamic, so with the help of Tidy Gherkin we can get regular expressions for the dynamic values (values that change) across the similar scenarios.

So we will keep the implementation steps for all other similar gherkin steps the same as the above form of stepDefination class, and only for the dynamic steps use the regular expressions, as follows:

@When(“^User logs in to the application with username \”([^\”]\*)\” and password \”([^\”]\*)\”$”)

Public void User\_logs\_in\_to\_the\_application\_with\_username\_something\_and\_password\_something(String strArg1, String strArg2) {

//code to login

}

And, for the ‘And’ gherkin steps, we can write them in a reusable way – i.e.:

**And** Cards displayed “true”

**And** Cards displayed “false”

And then define implementation steps with regular expressions and argument.

**Selenium Integration with Cucumber:**

We had already created an end to end framework from scratch, now we want to add cucumber knowledge to it. The framework was created as a Maven project, TestNG and Log4j and other stuff like page objects and etc. were added to it, so now we are going to add the last thing which is cucumber.

So, back to our previous E2E Maven project developed in the process of Building Framework from Scratch, we keep everything of that project and just in the test folder of the project we create a package named features and inside it a file named feature as well as a package named stepDefinations and inside it a class named stepDefination and also another package named cucumberOptions and inside it a class named cucumberOptions.

But before that step, we first need to give the knowledge of cucumber to our E2E Maven project by adding cucumber jars and dependencies to the POM.xml file. Now, right click on the project and click on update to refresh the project with new dependencies.

Now, proceed with creation of the above mentioned packages, classes and file. The feature file, stepDefination class and TestRunner class will look as follows.

The stepDefination class will look like this after we add our code to it too:

package stepDefinations;

public class stepDefination extends base {

@Given("^Initialize the browser with chrome$")

public void initialize\_the\_browser\_with\_chrome() throws Throwable {

// Write code here that turns the phrase above into concrete actions

driver =initializeDriver();

}

@Given("^Navigate to \"([^\"]\*)\" Site$")

public void navigate\_to\_Site(String arg1) throws Throwable {

// Write code here that turns the phrase above into concrete actions

driver.get(arg1);

}

@Given("^Click on Login link in home page to land on Secure sign in Page$")

public void click\_on\_Login\_link\_in\_home\_page\_to\_land\_on\_Secure\_sign\_in\_Page() throws Throwable {

LandingPage l=new LandingPage(driver);

if(l.getPopUpSize()>0)

{

l.getPopUp().click();

}

l.getLogin().click(); //driver.findElement(By.css()

}

@Then("^Verify that user is succesfully logged in$")

public void verify\_that\_user\_is\_succesfully\_logged\_in() throws Throwable {

// Write code here that turns the phrase above into concrete actions

portalHomePage p=new portalHomePage(driver);

Assert.assertTrue(p.getSearchBox().isDisplayed());

}

@When("^User enters (.+) and (.+) and logs in$")

public void user\_enters\_and\_and\_logs\_in(String username, String password) throws Throwable {

LoginPage lp=new LoginPage(driver);

lp.getEmail().sendKeys(username);

lp.getPassword().sendKeys(password);

lp.getLogin().click();

}

@And("^close browsers$")

public void close\_browsers() throws Throwable {

driver.quit();

}}

Since we are planning to parameterize our feature file, the final version of the file is in the following section of *Parameterizing Selenium tests with Cucumber.*

*Note: after integration of cucumber with our framework, we should have eliminated the need for the actual test of E2E Maven project (i.e. HomePage.java class that served as actual test prior to the project’s integration with cucumber). And after integration of cucumber with our E2E project, our actual testcase is now stepDefination class.*

*Note: the difference between assertion and printLn is that printLn will only print the failure, but assertion will force the test to stop upon failure.*

Because we are converting our TestRunner to TestNG, the final version is written in the following section.

**Parameterizing Selenium tests with Cucumber:**

When we are parameterizing our Gherkin file (features file), instead of simply typing ‘Scenario:’ we need to type ‘Scenario Outline:’ so that our test understands that we are performing parameterization in our Gherkin file.

Also, note that if we aren’t parameterizing our feature’s file, we can simply type, for instance, username and password values inside double quotes (like: When user enters username “Ahmad” and password “124abc” and logs in) and in that case since we are not parameterizing we don’t have to have a table of values below the scenario section. But if we are parameterizing our feature’s file, then we should type username and password and etc. inside angle brackets (like: When user enters <username> and <password> and logs in). And then we must provide a table of values below the scenario section. The headers or column names of the table must be exactly the same as the how we define it inside the state angle brackets in our step.

Note that cucumber parameterization eliminates the need to use the data provider or other parameterization features of TestNG. This doesn’t mean that TestNG is bad.

After parameterization, our feature file will look like this:

**Feature:** Login into Application

**Scenario Outline:** Positive test validating login

**Given** Initialize the browser with chrome

**And** Navigate to “http://qaclickacademy.com” site

**And** Click on Login link in home page to land on secure sign in page

**When** User enters <username> and <password> and logs in

**Then** Verify that user is successfully logged in

**And** close browsers

**Examples:**

|username |password |

|test99@gmail.com |123abcd |

|test123@gmail.com |127alcjk |

*Note: defining the Gherkin language (i.e. what to write for given, when, then, and) depends upon user perception. He can write it the way he wants.*

So here with cucumber, we are doing Data Driven Testing by sending data from feature file, and we are also parameterizing it by giving *Examples* table.

**Integrating Cucumber with TestNG:**

As of the completion of last steps of cucumber integration with Maven, our code has imported JUnit and hence running based on JUnit library, which is in confrontation with TestNG (if we ask this test case to run as part of the TestNG.xml file it will fail, because TestNG.xml file doesn’t support JUnit test files), so we have to convert the JUnit runner to TestNG runner:

* Google ‘Cucumber TestNG Maven Dependency’ and get the dependencies and copy it to pom.xml file.
* Comment the (@runWith(Cucumber.class)) line in TestRunner class.
* Then, as per the instructions available on Maven website, our TestRunner class should extend ‘AbstractTestNGCucumberTests’ class (Note: the class is not available in our project, it is imported by extending it).
* Update the project by right clicking and clicking on update project.

So now we can run the TestNG.xml file to run our TestRunner and execute our testcases. We can also execute our test cases by running the pom.xml file.

The TestRunner class will look like the following after converting it to TestNG runner:

package cucumberOptions;

//@RunWith(Cucumber.class)

@CucumberOptions(

features = "src/test/java/features",

glue="stepDefinations")

public class TestRunner extends AbstractTestNGCucumberTests {

}

The TestNG.xml file will look like the following after cucumber integration:

<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">

<suite name="Suite">

<!-- Suite -->

<listeners>

<listener class-name="Academy.listeners"/>

<listener class-name="resources.ExtentReporterNG"/>

</listeners>

<test name="Validating User login">

<classes>

<class name="cucumberOptions.TestRunner"/>

</classes>

</test>

<!-- Test -->

</suite>

Note that a lot of companies are using both TestNG and cucumber. They use cucumber to represent test cases in terms of business scenarios.

The combination of TestNG and cucumber can be referred to us a hybrid framework. And through this combination we can achieve the grouping, include, exclude and other many features of TestNG with the great features of cucumber to obtain a robust framework.

**Git and GitHub**

Git is a version control system. It allows us to work together and collaborate in a more efficient manner. It tracks changes in a file.

* Download git from their official website. In the process of installation, choose “use git from the windows command prompt”.

GitHub is the central repository where we save files to share with other team members.

* We need to know the git commands to use them using the git software to talk to the GitHub.
* So we need to have access to the git software as well as a GitHub account to access the central repository.
* Example: user A works on lines 1-5 of a code for a project and user B on lines 6-10. GitHub has the intelligence to collect and merge all the code and come up with one collection of all the code.

**Create GitHub Repository:**

On github.com after signing in, click on New repository > name the repository > If you choose public, anyone can see the repository and you choose who can commit, but if you choose Private, You can choose who can see and commit to the repository. In real time project you choose Private. > click on create.

**Git commands:**

*There are different ways to communicate with GitHub. For instance, there is a UI called Tortoise Git, Eclipse also has a plugin for Git and there are other tools too. But we will use the command prompt in order to know the basic concepts so that even if tools change we will still be able to execute our commands via the standard / global tool of command prompt.*

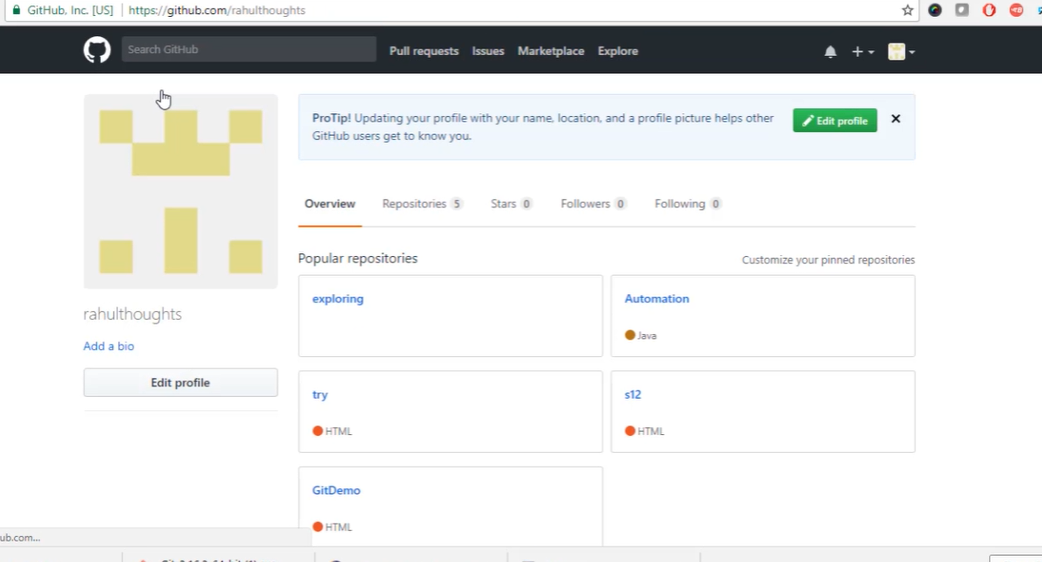
Below are the Git commands with the task they do. They should be followed in this order to do the described task:

|  |  |  |
| --- | --- | --- |
| Git command | Task it does | Details |
| Git config --global user.name “Nazir” | Tell git who you are (important because Git needs to know who is talking with the repository or who is the author of a commit) | Config is used for entering username and email address. And we set it globally to be able to access it from everywhere in the system. |
| Git config -- global user.email “nwolusmal@hotmail.com” |  | We hit enter after typing every command to execute it. |
| Cd followed by the path of folder/file we want to upload | To navigate to the folder we want to work on | It is not a Git command, this is a standard of Command Prompt. |
| Git init | Initialize Git repository in your folder. | This command creates a .git file which is hidden somewhere inside the folder you want to upload to GitHub. The purpose of the file is to make GitHub understand the folder. |
| Git add  Git add \* | Add a file to staging  Add all files to staging | If you want to add a specific file to staging you write “Git add” followed by space and name of the file, otherwise followed by \* |
| Git status | See staging status | To see which files are added to staging |
| Git commit -m “first commit” | *Commit* changes to *head* (but not yet to the remote repository) | -m means you are adding a message/ comment to your commit. |
| git commit -a | *Commit* any files you've added with git add, and also *commit* any files you've changed since then |  |

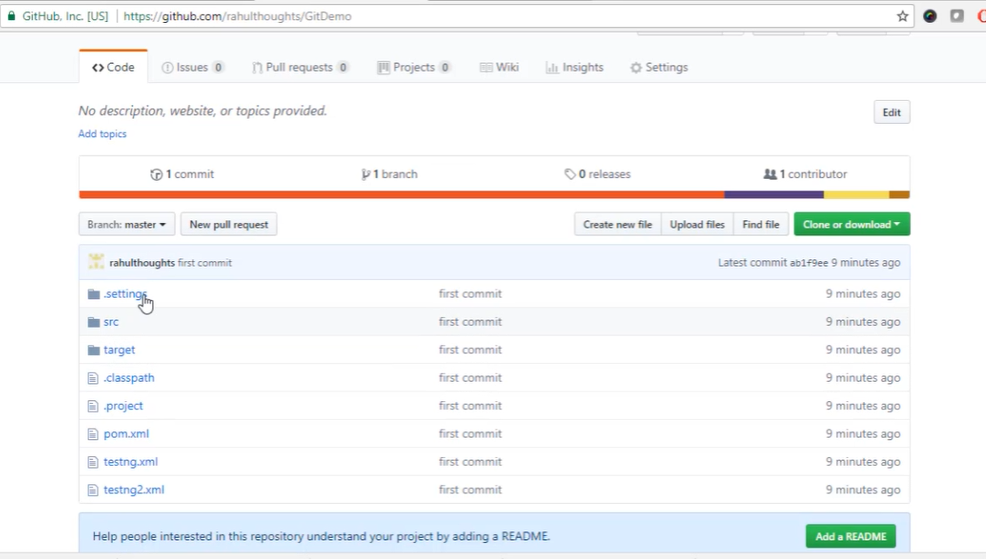
*You first need to add your file to staging (index, it is like first level of commit). Then Commit the files which are in staging, and then connect the local repository with the remote server (repository) and then upload the commits (files that were committed) to the server (GitHub).*

|  |  |  |
| --- | --- | --- |
| Git command | Task it does | Details |
| Git remote add origin <server> | Connect to a remote repository (considered *‘Origin*’) | Add the server to be able to push to it. Instead of <server> type the URL of your remote repo – i.e. https://github.com/rahul/GitDemo.git |
| Git push origin master | Upload files to the ‘master’ branch of ‘origin’. | By default, we are working on ‘Master’ branch, unless we create a specific branch. |

*After hitting enter, command prompt will ask us for the GitHub account credentials. Files will be pushed to remote server after providing the credentials. You can view all the repositories in GitHub by clicking on ‘view profile’.*



*By clicking on a repository, we can view the number of commits, branches, releases and contributors, all the files that are committed, time they were committed them and the comments.*



*Assume that the user “Rahul” uploaded his files as shown above to the main server. Now Nazir wants to get those files of Rahul and continue working on them from where Rahul had left. The first step will be to Clone the files. Cloning is used as the first step when we have none of the files of main server in the local repository. But if we already have some files and we want to download only the updates, then we use Pull command.*

|  |  |  |
| --- | --- | --- |
| Git command | Task it does | Details |
| Cd .. | Go to previous directory. | Note that it is a command of Command Prompt.  We use this to navigate to the location where we want to clone the directory. |
| Git clone https://github.com/rahul/GitDemo.git | Clone/ download entire directory from main server | In GitHub click on Clone or Download, the URL for the main repository will be shown, we can copy it. |

*After hitting enter, it will ask for credentials. Provide the credentials and the entire repository will be downloaded to the local machine.*

*Now let’s assume that Nazir wants to work on that project, so he has to import it to his IDE because it is the first time he is downloading the project and working on it. So in Eclipse click on > Import > expand General > existing projects into workspace > next > browse > chose the project “GitDemo > OK > Finish. Then Nazir brings some changes, updates some codes and saves them (File > save all). Now he wants to upload the updated project to GitHub: first navigate to the folder in Command Prompt with the help of “cd” command, then add all files to staging. Now to see which files you have brought changes to, use the command “git status”. If we use ‘git status’ command before adding our project to staging, command prompt will list all the files that have changed or updated in red color. So once we add all the files to staging after completion of working on them, we commit the files and (connect local repository with main repository if we haven’t already done so) and finally push the repository to server.*

*Now assume that Rahul has come back and wants to pickup from where Nazir left. He will need to pull the changes (not clone the entire repository, because he already has the repository, he just needs the updates made by Nazir), so he will first need to navigate to the local repository’s path in command prompt and use pull command as follows:*

|  |  |  |
| --- | --- | --- |
| Git command | Task it does | Details |
| Git pull origin master | Download the changes and updates | Command simply means that we want to download the updates from the Master branch of the Origin (which is the remote server or main repository) |

*And after running the command above, Rahul will see Nazir’s changes in his IDE without having to import the project again from repository to Eclipse or doing anything extra in Eclipse.*

**Branching:**

Importance: as an instance, an Architect is assigned to change the Framework, doing so will require a couple of weeks. If he directly starts working on the Master branch (the entire remote repository), tests will fail until the Framework is completely set up. To avoid doing so, architect will take advantage of the branching concept and will take a branch. Then after completing his work and making sure that it is ready to implement, he will merge it.

|  |  |  |
| --- | --- | --- |
| Git command | Task it does | Details |
| Git checkout -b developing | Create a branch and switch to it | Creates a branch named developing and automatically switches to it. |
| Git branch | To know which branch you are in | It will show you the list of all branches in your repo and also tell you what branch you are currently in. |

***Note:*** *when you create a branch and switch to it, your Eclipse will also automatically switch to that branch and whatever changes you make in it, it will affect the relevant branch, not the master branch.*

Now that you are in the developing branch (for the purpose of this example) you can use the staging, committing and pushing commands, all of them will affect the developing branch only, not the master branch.

|  |  |  |
| --- | --- | --- |
| Git command | Task it does | Details |
| Git push origin developing | Upload files to the ‘developing’ branch of ‘origin’ =’server’. | The developing branch will appear in GitHub upon utilizing this command. |
| Git checkout developing | Switch to developing branch | When you switch in command prompt, your IDE – i.e. Eclipse will switch automatically. |
| Git branch -d developing | Delete the developing branch |  |
| Git push --all origin | Push all branches to your remote repository |  |

Now let’s assume that the *developing* branch is ready and the team wants to merge it with the master branch: first switch to the master branch (git checkout master), then make sure that we have the most recent files of master from the server (git pull origin master), now merge by typing:

|  |  |  |
| --- | --- | --- |
| Git command | Task it does | Details |
| Git merge developing | Merge the branch named ‘*developing’* into active branch | In the scenario details above we switched to master branch to make it active directory |

**Resolve Merge Conflicts:**

Scenario: two users are working on the same file at the same time, one has deleted some stuff, the other has kept them, both have added some extra stuff. When they try to push their commits, command prompt will display ‘Merge Conflict” error, so we have to fix the code manually.

**Execute Git Commands from Eclipse:**

Right click on project name in Eclipse > Team > you will be able to execute many commands of Git here.

**Summary:**

1- Git tracks changes to the files in a folder.

2- Allows you to decide whether or not to track those changes.

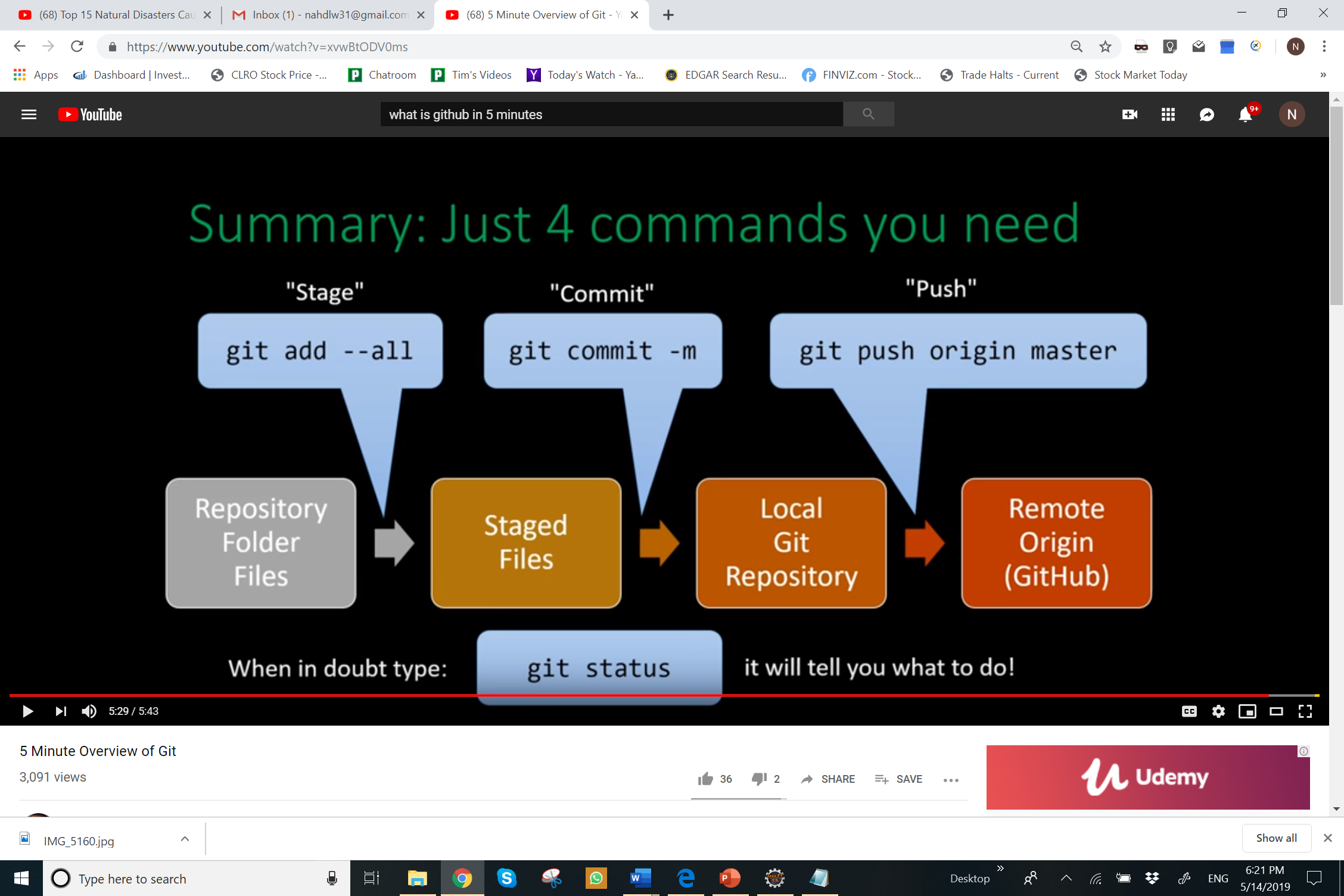
* The process of deciding which file changes to track is called staging.
* The command is: git add \* (star for all, or if a single file, then replace \* with the path of a that specific file)

3- Provides a complete history of changes to the files you decide to track.

* The process of telling Git to save the current status of changes is called a commit.
* The command is: git commit -m "your message"

4- Allows you to sync those changes (commits) to remote sites over a network.

* The process of telling Git to take all commits and save them to a remote Git directory is called a push.
* The command is: git push origin master



- Git Repository is on the cloud (Github).

- The process of copying the repository is called cloning.

- After cloning, the git folder on local machine is called repository folder or working directory.

- You must type the git commands from the command line from within the repository folder.

**Cloud Testing with Selenium:**

What is cloud-based testing: using cloud infrastructure for software testing, or testing on VMs due to the lack of required infrastructures, like a specific OS, a specific version of OS or browser, etc. We can do cloud computing with the help of Sauce Labs (Sauselabs.com).

**Sauce Labs:**

Sauce Labs allows users to run tests in the cloud on more than 700 different browser platform, operating system and device combinations, providing a comprehensive test infrastructure for automated and manual testing of desktop and mobile applications using Selenium, Appium and JavaScript unit testing frameworks. Sauce Labs offers automated cross browser testing, so we can test any OS and browser combination.

**Desired Capabilities class:**

In cloud computing, where we run our testcases outside the scope of our local machines - on a server or some VMs, we need to use Desired Capabilities class. This class serves as a mean of communicating our requirements with Sauce Labs. We define all our requirements in the desired capabilities class. For example, we want to run our testcases on version x of Firefox browser on OS Windows 7: we define these requirements in desired capabilities class and when we start running the Selenium testcases, the objects read and understand the capabilities that the host is expecting to run the testcases – i.e. what OS is it expecting and in that OS what browser is it expecting. Then it will go and connect to the Sauce Labs and check all the available VMs there and wherever these requirements are satisfied, it will give our Selenium testcase access to that specific VM.

**Remote Web Driver class:**

Instead of chromedriver or firefoxdriver, we use RemoteWebDriver class and pass two arguments inside it (i.e. the Sauce Labs URL as well as the desired capabilities class object).

Use the Platform Configurator page of Sauce Labs website to create code snippets for the desired capabilities of our automated testing scripts in our required language – i.e. the proper names of OS, browser, their versions, etc.

We also use the documentation page of Sauce Labs website to get the code snippets for the URL definition, simply paste the code in our Selenium class and put the Sauce Labs username, password and access key (that we generate in our Sauce Labs account user settings page on their website) in the relevant fields.

When we create our class and want to execute the test, we first go to the dashboard section of our user profile on Sauce Labs website, and when we run the test in Eclipse then immediately switch our screen to Sauce Labs website, we will see the a test link added in the dashboard, just click on it and it will show the VM running our test.

Our class for running could based testing will look like the following:

import java.net.MalformedURLException;

import java.net.URL;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.remote.DesiredCapabilities;

import org.openqa.selenium.remote.RemoteWebDriver;

public class SauceLabsTest {

public static final String USERNAME = "rahul2";

public static final String ACCESS\_KEY = "xxxxx";

public static final String URL = "http://" + USERNAME + ":" + ACCESS\_KEY + "@ondemand.saucelabs.com:80/wd/hub";

public static void main(String[] args) throws MalformedURLException {

DesiredCapabilities caps = DesiredCapabilities.chrome();

caps.setCapability("platform", "Windows 7");

caps.setCapability("version", "51.0");

WebDriver driver=new RemoteWebDriver(new URL(URL), caps);

driver.get("http://google.com");

System.out.println(driver.getTitle());

}

}

**Performance Testing:**

How can we put load on testcases that are written in Selenium.

Right click on the project name in Eclipse > Export > expand Java folder > select JAR file > Next > Select the export destination > name it > save.

Now, on the actual computer (not on Eclipse) navigate to the exported project, move the entire project to the Junit folder which is a subfolder of lib folder of JMeter’s main folder.

Restart JMeter, right click on Test Plan > add > Thread Users > Thread Group > number of threads ‘5’. Then right click on Thread Group > Add > Sampler > JUnit Request (generally we go with HTTP sampler) > select the class name from the drop down menu (the class name is the actual name given to your class in Eclipse) > select the method you want to execute > right click on the sampler created before (i.e. JUnit Request) and listeners and choose something (i.e. graph). Run the test by clicking on the green play icon and see the results. It will show us the load time for first request, second and etc. And we can see that for first request it took less time, but the time kept increasing with every additional request.

Since the practice above will open number of browsers equal to the number of threads we choose, it will affect the performance of our testing, so the right approach for performance testing with JMeter will be to run selenium testcases on headless browser.

**Run Selenium Testcases on Headless Browser:**

There is an HTML Unit Driver in Selenium with which you can run your scripts without opening the browser. Now, since the main intention of Selenium is to test the UI and avoiding to open the browser by using HTML Unit Driver makes UI testing impossible, hence we don’t use HtmlUnitDriver for functional testing, we only use it for performance testing.

Our class will look like the following when we use headless browser:

Public class UpdatedTest {

@Test

Public void testing()

{

Webdriver driver = new HtmlUnitDriver();

Driver.get(“http://ebay.com”);

System.out.println(driver.getTitle());

}}

**Apache JMeter**

(note beyond the course curriculum)

It is a Java open source software that is used as a load testing tool for analyzing and measuring the performance of a variety of services.

**Performance Testing:** is defined as a type of software testing to ensure software applications perform well under expected workload. It focuses on certain factors of a software program, such as the speed, scalability and stability.

Load Testing: testing to see whether the system or application under test is able to handle the required number of concurrent user accesses on web server without any failure.

Stress Testing: testing to see how the web server copes up with high load and limited resources. Here we just determine what the maximum load is that the web server can handle.

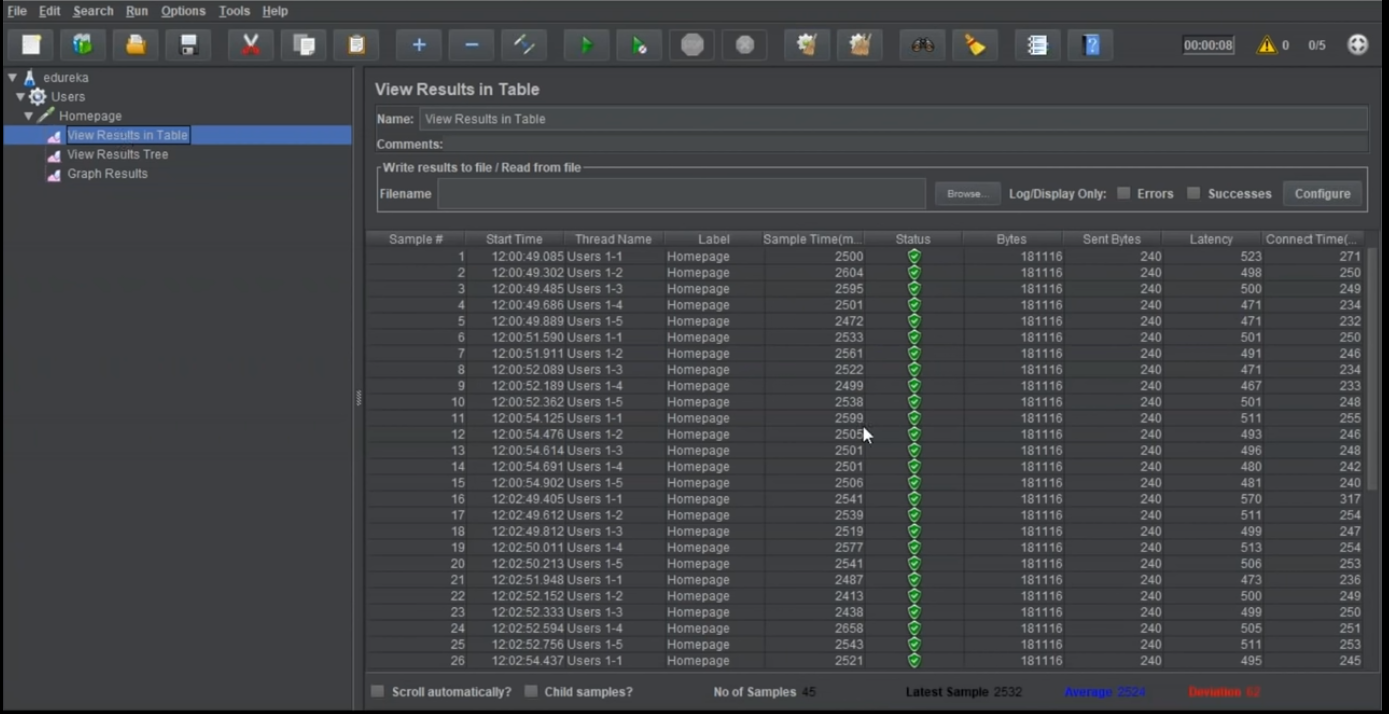
**JMeter Installation:** Download JMeter from its official website, unzip the files > and open the JMeter.bat file located in the bin folder: it will open JMeter app.

**Elements of JMeter:**

1. **Thread Group:** is a collection of threads. Each thread represents one user using the application under test and simulates one real user request to the server. For instance, if you set the number of threads to 100, JMeter will create and simulate 100 user requests to the server under test
2. **Samplers:** how does a thread group know which type of request it needs to make. The answer is samplers. The user request could be FTP Request (File Transfer Protocol Request: lets you send and FTP download file or upload file request to an FTP server), HTTP Request (lets you send and HTTP or HTTPS ‘Hypertext Transfer Protocol Secure’ request to a web server to retrieve html files or images from the website), JDBC Request (lets you execute database performance testing. It basically sends a JDBC request to a database), BSF Sampler (allows you to write a sampler using BSF scripting language), Access Log Sampler (allows you to read access logs and generate HTTP requests) or SMTP Sampler (used when you want to test a Mail server. Used when you want to send emails using the SMTP protocol).
3. **Listeners:** show the results of test execution. Results can be displays in different formats – i.e. Graphs, table, Tree, Log.
4. **Configuration:** data set configuration, login configuration, etc.

**Create Test Plan:**

First click on test plan and assign a name to it, if you want (i.e. edureka in the screenshot below). Then right click on it > add > threads > thread group. In ‘Actions to be taken after a sampler error’ (you can choose ‘continue’ so that it continues the test even if an error occurs), set the number of threads. Now right click on Thread Group > add > sampler > choose your sampler – i.e. http request > rename it – i.e. homepage > put the website address. Now right click on your sampler (homepage) > add > listener > chose your listener – i.e. view results in table. Now save it if you want to be able to reuse it. Now you can start the test by clicking on the green play icon.



**SQL / MySQL and Database Connection to Selenium**

**What is a Database:**

A Database is a collection of information that is organized so that it can be easily accessed, managed and updated.

**What does database consist of?**

* Database is collection of tables.
* A table is a collection of related data entries and it consists of columns and rows.
* Each row is treated as Record

**What is SQL?**

* It stands for structured Query language.
* It is used to communicate with a database.
* It can execute queries against a database
* It can retrieve data from a database
* It can insert records to a database
* It can update records in a database
* It can delete records from a database.
* It can create view in a database.
* It can create a new database.

**What is Database Management System?**

It is a software through which we execute SQL queries.

There are many such databases like: Oracle, Microsoft SQL server, SQLlite and many others are examples of such.

**MySQL:**

* MySQL is one of the Database Management Systems. It is an open source DBMS and you can use it for free.
* Runs on various platforms, and has good performance results.

**Downloading and Installing MySQL:**

*(from* [*https://dev.mysql.com/downloads/windows/installer/*](https://dev.mysql.com/downloads/windows/installer/)*)*

* During the installation process, you will be prompted to create password for your database, username by default will be root. Also, during the installation process, when we are clicking on next-next it is automatically creating a MySQL server for us.
* When installation is completed, we go to the MySQL folder on the computer and open MySQL Workbench.
* Then we click on “Local instance MySQL…” which is supposedly our server created during installation. After signing in, the MySQL editor will launch.
* By default, we will see some databases – i.e. sakila, test, world. We can create our own too.
* By expanding the Database names, we get tables inside it, expanding them will show columns.

**Creating a Database:**

To create Database, just write ‘create database’ and assign a name to it followed by semicolon:

create database QADBT;

Then select the statement and execute it by clicking on the yellow lightening icon next to save button.

*To comment something in MySQL, we simply use # in front of it.*

**Pointing to a Database and Creating Tables in it:**

use QADBT; #QADBT is the name of the database #We point to database using ‘use’ command

create table employeeinfo (name varchar (20), id int, location varchar (20), age int);

#in MySQL, data type of String is varchar #20 is the maximum number of characters

describe employeeinfo; #to see the outer layout or schema of the table

**Inserting Data into a Table:**

Insert into employeeinfo values (‘khomar’, 1, ‘DC’, 26);

Insert into employeeinfo values (‘amin, 2, ‘VA’, 35);

Insert into employeeinfo values (‘warin, 3, ‘PA’, 25);

Insert into employeeinfo values (‘Loudin, 4, ‘CA’, 44);

**Retrieving Records from a Table:**

Select name,id,location,age from employeeinfo; #retrieves all values of the specified rows

Select name,age from Employeeinfo; #if you want the values of name and age columns only

Select \* from employeeinfo; #if you want retrieve the values of the entire table

***Select distinct* keyword in queries:**

Select distinct name from employeeinfo; (we want to retrieve unique names and avoid duplicates.)

**SQL *where* command usage:**

Select name from employeeinfo where location=’New York’; (get names of employees based in NY)

Select name from employeeinfo where location=’New York’ and age>23; (we only want the information about employees who are based in New York and whose ages are more than 23 years.)

Select name from employeeinfo where (location=’Florida’ and age<23) or location=’charlotte’; (we want to retrieve information of the employees based in Florida whose ages are less than 23 years, or otherwise we want employees based in Charlotte.)

Select name from employeeinfo where age in (’22,23,24,25); (use this to find employees whose ages match those written inside the parentheses).

Select name from employeeinfo where location in (‘florida’, ‘newyork’); (use this to find employees whose ages match those written inside the parentheses)

Select name from employeeinfo where id between 1 and 1000; (we can replace ‘between’ with ‘not between’ as well if we want to get the opposite result).

**To print the employee names whose id is less than 25:**

Age<25

In(1,2,3…25)

Between 0 and 25

**Regular expressions:**

Underscore(\_) stands for one letter only

Percentage sign (%) stands for infinite number of letters

a% (print names that start with a, no matter how many or what letters follow it)

\_b% (find names where second letter is b, no matter what letter is before it, but there is only one letter before it, and no matter what letters follow it, but there could be any number of letters after ‘b’)

\_\_c% (find names where the third letter is c, and there are only two letters before it).

D\_\_ (print names that have total of 3 letters where first letter is d)

*Note: We should use ‘like’ operator with regular expressions.*

Examples:

Select name from city where countrycode like ‘A%’;

Select countrycode from city where countrycode like ‘A%’;

Select countrycode from city where countrycode like ‘\_\_T%’;

Select countrycode from city where countrycode like ‘\_A%’;

**Database Connection to Selenium Testcases:**

We can do such by the help of JDBC (Java Database Connectivity) which is a Java API for connecting programs written in Java to the data in relational databases.

*We created a Database in the MySQL part of this note. Now we want to connect it to Selenium.*

* First we need to download “MySQL Connector Java” JAR file.
* Right click on your project in Eclipse > properties > Java Build Path > Add External JARS > navigate to the relevant folder and choose MySQL Connector Java jar.
* Create new class in your project in Eclipse and inside it type:
  + //Connection URL syntax is: “jdbc:mysql://”+host+”:”+port+”/databasename”;
  + //Connection URL example is: “jdbc:mysql://”+localhost+”:”+3306+”/QADBT”;
  + *//For simplicity purpose, we have created objects for ‘host’ and ‘port’.*
  + *//first “root” is the username for the Database, second is password, which could be different based on how we have set it in MySQL installation process.*
  + *//connection is like a route to the destination (database) and statement is like vehicle*

String host="localhost";

String port= "3306";

Connection con=DriverManager.getConnection("jdbc:mysql://" + host + ":" + port + "/demo", "root", "root");

Statement s=con.createStatement();

ResultSet rs=s.executeQuery("select \* from credentials where scenario ='rewardscard'");

*//this is the query we use in MySQL to select a row in a table.*

while(rs.next())

{

WebDriver driver= new FirefoxDriver();

driver.get("https://login.salesforce.com");

driver.findElement(By.xpath(".//\*[@id='username']")).sendKeys(rs.getString("username"));

driver.findElement(By.xpath(".//\*[@id='password']")).sendKeys(rs.getString("password"));

}

**SQL (Structured Query Language)**

SQL is the database language by the use of which we can perform certain operations on the existing database and also we can use this language to create a database. SQL uses certain commands like Create, Drop, Insert etc. to carry out the required tasks.

These SQL commands are mainly categorized into four categories as discussed below:

**DML (Data Manipulation Language):** The SQL commands that deals with the manipulation of data present in database belong to DML and this includes most of the SQL statements. For all DML we need to commit.

Examples of DML:

SELECT – is used to retrieve data from the a database.

INSERT – is used to insert data into a table.

UPDATE – is used to update existing data within a table.

DELETE – is used to delete records from a database table.

**DDL (Data Definition Language):** DDL actually consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in database. All DDL are auto committed (saved).

Examples of DDL commands: DR CAT (Drop, Rename, Create, Alter, Truncate)

CREATE – is used to create the database or its objects (like table, index, function, views, store procedure and triggers).

DROP – is used to delete objects from the database.

ALTER-is used to alter the structure of the database.

TRUNCATE–is used to remove all records from a table, including all spaces allocated for the records are removed.

COMMENT –is used to add comments to the data dictionary.

RENAME –is used to rename an object existing in the database.

**DCL (Data Control Language):** DCL includes commands such as GRANT and REVOKE which mainly deals with the rights, permissions and other controls of the database system.

Examples of DCL commands:

GRANT-gives user’s access privileges to database.

REVOKE-withdraw user’s access privileges given by using the GRANT command.

clear screen

connect

hr

hr

select table\_name from user\_tables

/

select \* from employees

/

set linesize 400

Set pagesize 400

clear screen

select \* from departments

select employee\_id, first\_name, last\_name from employees (selecting specific columns from employees).

spool C:\name.docx (spool means save, which is followed by the destination or address of the file where it is going to be saved with the extension.).

spool off (saves everything to the file).

ed (for editing)

select \* from employees where employee\_id=100

- string = any types of text, characters, symbols

- numbers = 1234

- when we type in any type of string we have to add ''"" i.e. 'skj'

- for numbers we don't need to add ""'' i.e. 1234

To find specific person by name: note that it is case sensitive

select \* from employees where first\_name='Neena'

select \* from employees where lower(first\_name)='neena' (this means find Neena whether it is writtin in lower or upper case, but I want to type it in lower case here. you can replace "lower" with "upper" or "initcap" which means first letter capital, too).

select employee\_id, first\_name, salary from employees where lower(first\_name)='steven'

select employee\_id, first\_name, salary from employees where lower(first\_name)='steven' and employee\_id=100

q. find out employees whose first name is peter or steven:

select \* from employees where lower(first\_name)='peter' or initcap(first\_name)='Steven'

q. find out all employees whose first name starts with the letter N:

select \* from employees where lower(first\_name)like'n%'

q. find out all employees whose phone number starts with 515:

select \* from employees where phone\_number like '515%'

q. find out all employees who got hired in the year 96:

select \* from employees where hire\_date like '%96'

q. find out all employees who got hired in the month of jul:

select \* from employees where lower(hire-date) like '%jul%'

q. find out all employees info and sort the info with their salary in descending order.

select \* from employees order by salary desc

q. find out employees first\_name, last\_name, salary, commission\_pct (percentage).

select first\_name, last\_name, salary, commission\_pct from employees

q. find out only those employees who don't have commission

select \* from employees where commission\_pct is null

q. find out only those employees who have commission

select \* from employees where commission\_pct is not null

q. find out employees yearly salary

select first\_name, last\_name, salary\*12 from employees

q. find out employees weekly salary

select first\_name, last\_name, salary/4 from employees

q. find out employees total monthly salary with commission

select first\_name, last\_name, salary+salary\*commission\_pct from employees

q. find out employees total monthly salary wih commission and those that have no commission, show me their total salary:

select first\_name, last\_name, salary+salary\*NVL(commission\_pct,0) from employees

NVL = Null Value

q- Find out employees whose first name starts with 'R':

Select \* from employees where lower(first\_name) like 'r%'

q- Find out employees total salary with commission:

select first\_name, last\_name, salary, commission\_pct, salary+(salary\*commission\_pct) from employees

q- Find out employees that has commission only:

select first\_name, last\_name, commission\_pct from employees where commission\_pct is not null

q- Find out employees who don't have commission

select first\_name, last\_name, commission\_pct from employees where commission\_pct is null

q- Create a specific file at a specific destination:

spool "c:\10July18.docx"

q- Save to the file above:

spool off

q- Find out employees yearly salary:

select first\_name, last\_name, Salary\*12 from employees

q- Rename the Salary\*12 column to "yearly salary" :

select first\_name, last\_name, salary\*12 yearly\_salary from employees

q- People who have commission show their exact values for commission and total salary, but people who don't have commission

show zero in the commission column for them:

Select first\_name, last\_name, salary, commission\_pct, salary+salary\*NVL(commission\_pct,0) from employees

NVL = NullValue

q- Find out employees total monthly salary with or without commission and add 20% commission for this month to the people that don't

have any commission:

select first\_name, last\_name, commission\_pct, salary+salary\*NVL(commission\_pct,.2) from employees

q- Find out employees total yearly salary with commission:

select first\_name, commission\_pct, (salary+salary\*NVL(commission\_pct,0))\*12 from employees

q- Find out the average salary that gets paid each month:

select avg(salary) from employees

Note: We can't use group functions for multiple columns at the same time.

Group Functions:

AVG = average

Count = count

SUM = total

MAX = maximum

MIN = minimum

STDDEV = Standard deviation

Variance =

q- Find out the employee who gets more than average salary:

This query has two parts, so split it and execute each part as follows:

i) find out avg salary

ii) find out who gets more than avg

select avg(salary) from employees

for example the result is: 6545.79

select \* from employees where salary > 6545.79

Or we can clearly describe each query, where we put one query in parentheses, as follows:

select \* from employees where salary>(select avg(salary) from employees)

Note: a query that is inside () will execute first, then the other query will execute.

Note: we have to use the keyword "select" to get the value of the first query

q- Find employees' maximum salary:

select max(salary) from employees

q- Find out employees that get less than maximum salary:

Solution 1:

i) find out the max salary: select max(salary) from employees

ii) find out employees who get less than max: select \* from employees where salary <24000

Solution 2:

select \* from employees where salary<(select max(salary) from employees)

q- Find all employees who get the same salary as Peter Tucker:

This query has two parts:

i) find out Peter Tucker's salary: select \* from employees where lower(first\_name)='peter' and lower(last\_name)='tucker'

ii) find out everyone else that gets the same salary as Peter Tucker does: select \* from employees where salary=10000

A combined query is as follows:

Select \* from employees where salary=(select salary from employees where lower(first\_name)='peter' and lower(last\_name)='tucker')

Note: when we are comparing a specific column from a subquery to a specific column in the primary query,

we cannot call more than one column in the subquery.

And that single column has to be the same as whatever on the left side of the equal (=) sign.

Conditions of sub query:

- A query that goes inside () that query runs / executes first

- A query that is located in the bottom will run / execute first

q- How many people are getting commissions:

select count(commission\_pct) from employees

q- Find out first 10 employees info from employees table:

select \* from employees where rownum<11

or

select \* from employees where rownum<=10

or

select \* from employees where employee\_id between 100 and 109

Note: rownum= row number

Note: Varchar = various characters = string = text = adlfhadsf123

Varchar2 = includes all different types of characters that varchar don't.

q- Find out employees from row number 11 to 20:

select \* from employees where rownum<=20 minus select \* from employees where rownum<=11

Note: A query that is dependent on another query to be executed is known as sub query.

A query that can run independently is known as primary query.

q- Find out employees information where employees ID is in descending order:

select \* from employees order by employee\_id desc

q- Find out the last 10 records of employees:

It has two portions:

i) Make the table in a way that the last records become the first, or make it in descending order: select \* from employees order by rownum desc

ii) get the first ten rows of the flipped table: select \* from employees where rownum<=10

Now to combine them:

select \* from (select \* from employees order by rownum desc) where rownum<=10

q- Find out the 10 least paid employees:

select \* from (select \* from employees order by salary asc) where rownum<=10

q- To find what columns the table of employees is made of:

describe employees

**SQL JOIN:**

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

To join more than one table, one of the conditions below should be true:

There has to be a matching PK and FK.

PK (Primary Key): uniquely identifies each record in a table.

- PK has to be unique

- PK cannot have repeated value

- PK cannot have Null Value

- A table can have only one PK

FK (Foreign Key): A FK in one table points to a primary key in another table.

FK has to become PK in another table, the opposite is not possible.

- FK can have null value

- In one table we could have multiple FK

- FK can be repeated

- FK will match with a PK in another table

- A FK in one table cannot match with another FK in the other table

There are various types of joining: (a good example can be Venn Diagram).

* 1. Simple Join (Inner Join): the join that brings common data from both tables. Or the common part of the two tables. Or Inner Join returns records that have matching values in both tables.
  2. Left (Outer) Join: returns all records from the left table, and the matched records from the right table.
  3. Right (Outer) Join: returns all records from the right table, and the matched records from the left table
  4. Full (Outer) Join: all from right and left tables including the common part.
  5. Self join: xxxx bring data from the same table to two different tables.





Example: Find out employees name, salary and department name.

select employees.first\_name, employees.last\_name, employees.salary, departments.department\_name from employees, departments

where employees.department\_id=departments.department\_id

We can semplify the above query as follows:

select first\_name, last\_name, salary, department\_name from employees, departments

where employees.department\_id=departments.department\_id

We can also write just e instead of employees and d instead of departments like in the query below:

q. find out employees name, salary, department name, city:

select first\_name, salary, department\_name, city from employees e, departments d, locations l

where e.department\_id=d.department\_id

and

d.location\_id=l.location\_id

*The above example was simple joining.*

Q. Find out employees name, start\_date.

select e.first\_name, jh.start\_date from employees e, job\_history jh, jobs j

where

e.job\_id=jobs.job\_id

and

jobs.job\_id=jh.job\_id

In the above example "start date" is coming from Job History table and "Employee Name" is coming from Employee table,

and there is no FK in one table relevant to Employee Name and Start Date that points to PK in the other table,

so we go through another table i.e. Jobs table.

Q. Find out employees name and region:

select e.first\_name, r.region\_name

from employees e, regions r, departments d, locations l, countries c

where

e.department\_id=d.department\_id

and

d.location\_id=l.location\_id

and

l.country\_id=c.country\_id

and

c.region\_id=r.region\_id

Q. Find out employees name, department name and department id. Only retrieve the info of the employees that has a department id.

select e.first\_name, d.department\_name, e.department\_id

from employees e, departments d

where

e.department\_id=d.department\_id

The following example is for left outer join:

Q. Find out employee name, department name and retrieve all employees with or without department id.

select e.first\_name, d.department\_name, e.department\_id

from employees e, departments d

where

e.department\_id=d.department\_id(+)

Note: Plus sign on the right side of the equation is left outer join, on the left side is for right outer join.

But the solution above will not work with all databases. The best solution is as follows:

select e.first\_name, d.department\_name, e.department\_id

from employees e

LEFT JOIN departments d

ON

e.department\_id=d.department\_id

*On (in the above solution) = based on*

The following example is for right outer join:

Q. find out employees name, department name and bring all the department names including the departments that don't have any employees.

select e.first\_name, d.department\_name

from employees e

RIGHT JOIN departments d

ON

e.department\_id=d.department\_id

The following example is for full outer join:

Q. find out employees name and department name and bring data for the employees that do not belong to any department.

select e.first\_name, d.department\_name, d.department\_id

from employees e

left join employees d

on

e.department\_id=d.department\_id

minus

select e.first\_name, d.department\_name, d.department\_id

from employees e

right join employees d

on

e.department\_id=d.department\_id

Q. Find out all the departments that doesn't have any employees.

Select e.first\_name, d.department\_name, d.department\_id

from employees e

Right join departments d

ON

e.department\_id=d.department\_id

minus

Select e.first\_name, d.department\_name, d.department\_id

from employees e

Left join departments d

ON

e.department\_id=d.department\_id

Q. Find out all the departments including the employee info that doesn't have any department.

(Select e.first\_name, d.department\_name, d.department\_id

from employees e

Right join departments d

ON

e.department\_id=d.department\_id

minus

Select e.first\_name, d.department\_name, d.department\_id

from employees e

Left join departments d

ON

e.department\_id=d.department\_id)

UNION

(Select e.first\_name, d.department\_name, d.department\_id

from employees e

Left join departments d

ON

e.department\_id=d.department\_id

minus

Select e.first\_name, d.department\_name, d.department\_id

from employees e

Right join departments d

ON

e.department\_id=d.department\_id)

*(some repetition in the following notes:)*

Simple Join: The join that we do to return only the common values from multiple tables.

Q. Find out first\_name, dept\_name, dept\_id of the employees that belong to a dept:

select e.first\_name, d.department\_name, d.department\_id from employees e, departments d

where

e.department\_id=d.department\_id

Q. find out employee first\_name, department name and find the departments that don't have any employees:

select e.first\_name, d.department\_id, d.department\_name

from employees e

right join

departments d

ON

e.department\_id=d.department\_id

Q. Find out employees name, department name, department ID, retrieve all the info of employees that belong to a department or that don't belong to a department:

select e.first\_name, e.department\_id, d.department\_name from employees e

left join

departments d

on

e.department\_id=d.department\_id

Q. Find out emp name, dept name and bring all the employees info with or without dept, and bring

all the department names with or without employees:

select e.first\_name, d.department\_name, d.department\_id

from employees e

Full join

departments d

ON

e.department\_id=d.department\_id

Q. Find out employees job title and country name:

select j.job\_title, c.country\_name

from jobs j, countries c, job\_history jh, departments d, locations l

where

j.job\_id=jh.job\_id

and

jh.department\_id=d.department\_id

and

d.location\_id=l.location\_id

and

l.country\_id=c.country\_id

Q. Find out employee name and managers name:

select e.first\_name, e.last\_name, m.first\_name, m.last\_name

from employees e, employees m

where

m.employee\_id=e.manager\_id

Q. Increase salary of employees that has job\_id as IT programming by 20% and increase salary of employees that has job ID as ST clerk by 25%:

select first\_name, last\_name, job\_id, salary, salary+salary\*0.2 inc\_salary from employees

where

lower(job\_id)='it\_prog'

UNION

select first\_name, last\_name, job\_id, salary, salary+salary\*0.25 inc\_salary from employees

where

lower(job\_id)='st\_clerk'

Q. increase IT prog salary by 20%, increase st clerk salary by 25% , all other there is no increase:

select first\_name, last\_name, job\_id, salary,

DECODE(job\_id,

'IT\_PROG',salary+salary\*0.2,

'ST\_CLERK',salary+salary\*0.25,

salary) inc\_salary

from employees

Note: make sure you put a comma before the word decode

Decode=apply

Q. Find out unique/distinct department\_ID:

select distinct (department\_id) from employees

select count (distinct (department\_id)) from employees

Q. create a table in the database:

create table mytable (id number (3), name varchar2 (20), salary number (6))

Q: insert into my table:

Insert into mytable

values

(103, 'John', 3500);

Note: you can insert one row at a time by typing the above as command, but if you want to add more entries at the same time, follow the query below:

Q. Insert multiple entries to the table from a doc: create your doc with the table contents as follows:

Insert into mytable

values

(101, 'John', 3500);

Insert into mytable

values

(103, 'Zeb', 3400);

Insert into mytable

values

(104, 'Tom', 3300);

And then type in the command:

@"c:\file address \file name and format"

Q. Update entries:

update mytable

set salary=7000

where ID=104

Q. Delete entries:

delete from mytable where id=105

Q. Undo deleting an entry:

rollback

Note: we cannot rollback on truncate or drop

Note: when we insert, update or delete anything from a table, we have to save the work.

To do that we have to write the following every time we insert, update or delete: commit

Q. Create a table named mytable2 and copy everything from mytable into it:

create table mytable2

as

select \* from mytable

Q. Drop the whole table (deletes the entire table i.e. the contents and the structure of table):

Drop table mytable2

Q. Truncate a table (delete the contents but leave the structure):

Truncate table mytable

Q. Show the structure of a table:

describe mytable

Q. create a new table where we want to copy the structure of another table only not the contents then we write:

create table mytable4

as

select \* from mytable

where 1=2

DELETE:

1. Using Delete we can delete one row or all rows

2. We can Rollback in Delete

3. When Delete all rows, table structure still in the database

DROP:

1. Using DROP we cannot drop one row but all rows

2. We cannot Rollback in DROP.

3. When DROP all rows, table structure will not in the database anymore

TRUNCATE:

1. Using TRUNCATE we cannot truncate one row BUT all rows

2. We cannot Rollback in TRUNCATE.

3. When TRUNCATE all rows, table structure will in the database

**PL/SQL:**

PL/SQL is Oracle’s procedural extension to industry standard SQL. It is a combination of SQL along with the procedural features of programming languages.

Q. Insert a row in mytable and it should also insert the same row into mytable2:

Create or replace TRIGGER xyz

AFTER INSERT

ON mytable

for each row

begin

Insert into mytable2

Values

(:new.id, :new.name, :new.salary);

end;

Note: in the formula above, (xyz) is the name for the trigger which could be named anything.

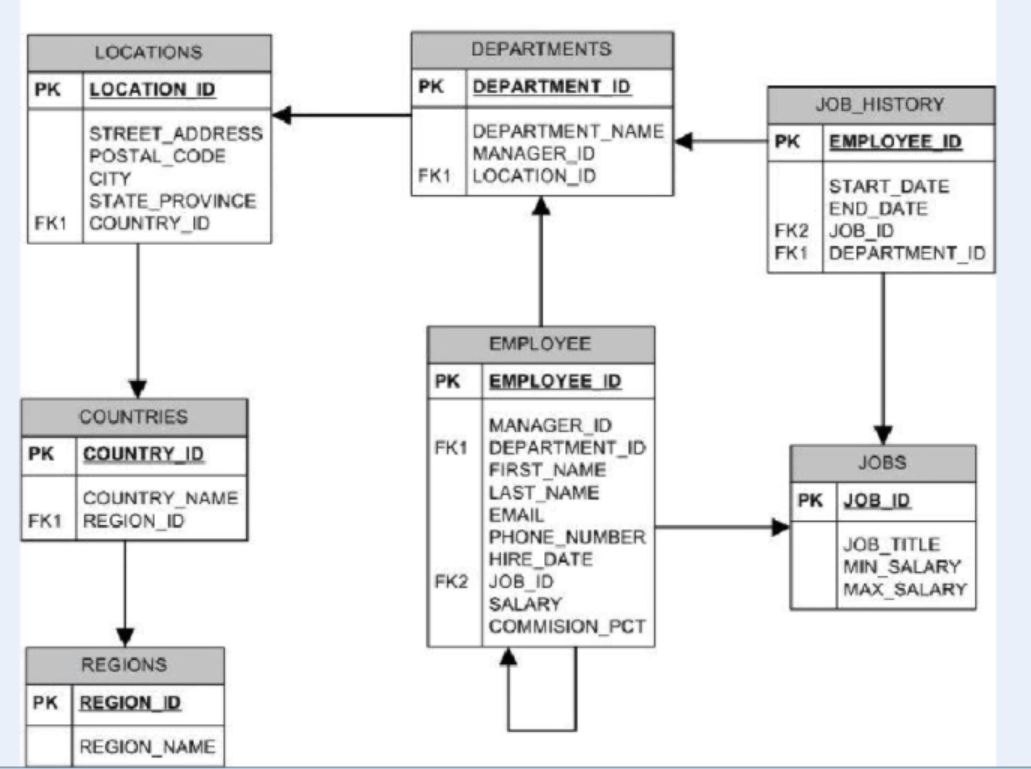
Note: "replace" in the formula above is not required, but in case another formula is created with the same name, it will replace it.

Q. Create an update trigger:

Create or replace trigger update\_triggger

before update

on mytable

for each row

begin update mytable2

set id=:new.id,

name=:new.name,

salary=:new.salary

where

id=:old.id;

end;

Now update an entry:

update mytable

set salary=9000

where id=103

*And see that the salary is*

*updated for the same entry*

*in both tables.*

Q. Create a delete trigger:

create or replace trigger del-tr

before delete

on mytable

for each row

begin

delete from mytable2

where

id=:old.id;

end;

Q. Create a procedure:

Create or replace procedure my\_pr

(id number, raisedpercent number)

is

begin update employees

set salary=salary+(salary\*.1)

where

employee\_id=id;

end my\_pr;

Now you can just type:

execute my-pr (101, 10)

**REST API Testing with SoapUI:**

**Webservices:**

A web service is a method of communication between two applications or electronic devices over the World Wide Web (www). Webservice is a subset of API.

Example (all based on assumptions): British Airways’ wants to show hotel.com functionalities after confirmation of ticking booking. Now, their website is written in Java but hotels.com website is written in .Net. since integration of these two different language is an issue, webservices plays a role in solving this problem. So hotels.com will create a webservice (wrap its code as a webservice) for the required functionality required by British Airways. (i.e. showing hotels at the destination to which ticket is booked). And that webservice will connect to the real code of hotel.com website.

A real example: twitter webservices on different news websites.

* Web Service is an application or business logic that is accessible using standard internet protocols via standardized XML messaging system.
* Web Services allows you to expose the functionality of your existing code over the network. Once it is exposed on the network, other application can use the functionality of your program.
* Because all communication is in XML, web services are not tied to a specific operating system or programming language.
* Example: Java can talk with Perl; Windows applications can talk with Unix applications.
* The language used for communication (request and responses) is either XML (for SOAP webservice) or Json (for REST API).

Endpoint is where the webservice is hosted.

We can say that webservice is a collection of operations – i.e. search for hotels with destination name could be one operation, search for rental calls with destination name could be other operation and search tour packages at the destination with provided details could be another operation.

SOAP requests are sent via an HTTP request and SOAP responses are returned within the content of the HTTP response.

The basic Webservices communication platform is XML + HTTP.

**Types of Webservices:**

1. Simple Object Access Protocol (SOAP).
2. Representational State Transfer (REST) – it is newer. It uses Json language.

If a webservice accepts messages in SOAP protocol, we can say that that webservice is a SOAP webservice.

**What is a SOAP protocol:**

SOAP is a protocol for sending and receiving messages between applications without confronting interoperability issues. The underlying language used by Soap is XML.

**Scenario:**

Application “A” bundles Employee information into a Soap Message and sends it to Webservice “B” over HTTP. *(request)*

Webservice “B” unpacks the SOAP request and converts it into a command that the .Net application can understand.

Next, the Webservice B packages up the response into another SOAP message, which it sends back to the Application “A” in response to its HTTP request. *(response)*

The Application “A” unpacks the SOAP message to obtain the results of the account registration process.

**The structure of SOAP message:**

*Envelope: (Mandatory)*

The SOAP envelope indicates the start and the end of the message so that the receiver knows when an entire message has been received. The SOAP envelope solves the problem of knowing when you're done receiving a message and are ready to process it.

* + Every SOAP message has a root Envelope element.
  + Envelope element is mandatory part of SOAP Message.

*Header: (Optional)*

Contains any optional attributes of the message used in processing the message, either at an intermediary point or at the ultimate end point.

Headers are intended to add new features and functionality

*Body: (Mandatory)*

Contains the XML data comprising the message being sent.

The SOAP body is a mandatory element which contains the application-defined XML data being exchanged in the SOAP message. The body must be contained within the envelope and must follow any headers that might be defined for the message.

*Fault: (Optional)*

An optional Fault element that provides information about errors that occur while processing the message. When an error occurs during processing, the response to a SOAP message is a SOAP fault element in the body of the message, and the fault is returned to the sender of the SOAP message.

**Calculator Add Operation Soap Request:**

<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope" xmlns:tem="http://tempuri.org/">

<soap:Header/>

<soap:Body>

<tem:Add>

<tem:a>7</tem:a>

<tem:b>6</tem:b>

</tem:Add>

</soap:Body>

</soap:Envelope>

**WSDL**

*WSDL stands for Web Services Description Language.*

To test any webservice, we need to refer to the contract of that webservice which is WSDL.

Example: you work as a test engineer for hotels.com and your manager tells you to go and test all our webservices. The first thing you do is to ask for the contract of the webservice. That contract is WSDL. WSDL is a file (i.e. the calculator webservice’s WSDL is referred to as the ‘service description’), it enlists all the *operations* that are supported by the webservice.

* WSDL is the standard format for describing a webservice.
* WSDL is pronounced as 'wiz-dull' and spelled out as 'W-S-D-L'.
* WSDL definition describes how to access a webservice and what operations it will perform. WSDL specifies the location of the service and the methods of the service using ‘elements’.
* WSDL is often used in combination with SOAP and XML Schema to provide webservices over the Internet.
* A client program connecting to a webservice can read the WSDL to determine what functions are available on the server.

WSDL is an XML document with a <definitions> element at the root and the child elements:

<types>,

<message>,

<portType>

<binding>.

*Definitions:* the <definitions> element must be the root element of all WSDL documents. It defines the name of the web service. The definitions element is a container of all other elements.

*Types:* the WSDL’s <types> element takes care of defining the data types (integer, string ..) that are used by the webservice. WSDL allows the types to be defined in separate elements so that the types are reusable with multiple webservices.

*Message:* the <message> element describes the data being exchanged between the webservice providers and consumers. Each webservice has two messages: input and output.

*portType:* the <portType> can combine one request and one response message into a single request/response operation. This is most commonly used in SOAP services. A portType can define multiple operations.

*Binding:* the <binding> element provides specific details on how a portType operation will actually be transmitted over the wire. The bindings can be made available via multiple transports, including HTTP GET, HTTP POST, or SOAP.

Binding is a term which will help you see how many operations the webservice supports.

*Service:* webservice clients can learn from the service element where to access the service, through which port to access the webservice, and how the communication messages are defined.

We don’t have to bother reading the WSDL, we just import it to SaopUI and it will simplify it for us.

Process: we create a soap message and pass our inputs, send it to the relevant operation of the webservice, webservice reads the code and gives back response. And that response is populated on our website.

**Differences between Soap and WSDL:**

A Soap message is an XML document which is used to transmit your data. WSDL is an XML document which describes how to connect and make requests to your webservice.

Basically, Soap messages are the data you transmit, WSDL tells you what you can do and how to make the calls.

Soap is a structure you apply to your message/data for transfer. WSDLs are used only to determine how to make calls to the service in the first place.

Example: a restaurant menu is WSDL and your order is Soap.

**SoapUI:**

SoapUI is a testing tool for Webservices and APIs (both REST and SOAP). It has a free and full version.

*To create a project:*

* File > new SOAP project, name the project (Calculator), and;
* Import the WSDL file by navigating in the computer or pasting the URL of WSDL, then click OK.

You will see that in the navigation window (on the left hand side), SoapUI lists the projects and under each project, the operations that the WSDL enlists.

On the right hand side, we have the endpoint address (which is the URL for the location of webservice) and the request and response fields.

We read the code and data type to see what it is supporting for submission – i.e. if it is taking integers, we add it and submit the request. The request hits the endpoint and the response is generated in the response field.

**Creating Test Suites and Testcases:**

We need to create test suites and test cases to be able to get the features for and use assertions and other features for automating our tests.

* Right click on the project name > new TestSuite > name it (Calculator Testing) > click on OK. Its icon will look like the windows icon (four squares).
* Right click on the TestSuite name > new TestCase > name it (Adding Function) > click on OK.
* Expand Test Case and right click on Test Steps > click on Add Step > Soap Request > Name it (Add) > it will ask you to select the operation to invoke for the request. We can choose the ‘Add’ operation. And that is how we ***map the operation with our test steps***.

**Configure Webservice in Local Machine:**

* Download the webservice (-i.e. webservice named Axis2 created by Instructor Rahul Shitty) to your computer, unzip it, open it and copy its path.
* Go to System Environment Variables > New > name it (i.e. AXIS\_HOME) and paste the path of the webservice folder.
* Now open the bin folder inside your webservice folder > copy its path > go to path in environment variables and add a button and paste the path of bin folder.

We also need backend for our webservice, so for that we download and install MySQL from <https://dev.mysql.com/downloads/installer/>

* During the installation process, make sure to choose ‘Developer Default’ as the setup type. And in Accounts and Roles screen, put ‘root’ as your username and ‘root’ as your password.
* Copy the following code and paste it in MySQL Workbench and execute it:

create database Employeeportal;

use Employeeportal;

CREATE TABLE IF NOT EXISTS Employeeinfo(

name VARCHAR(100) NOT NULL,

id VARCHAR(200), dept varchar(20),

age INT NOT NULL,

PRIMARY KEY(name));

Now we need to invoke / configure the webservice, so:

* Open the webservice folder (i.e. Axis2) > open bin folder and then open axis2server (the type Windows Batch File only, not the second one).
* If we got error message because the port number (8080 in the case of Axis2 webservice) is already taken by other service, then go to conf folder inside the Axis2 folder and open axis2.xml file with Notepad++ and change the port number to something like 8082.
* Now to start the server in local host: go to your browser and in the address bar type: localhost:8080. Under the deployed services you will see “EmployeeManagementService” by clicking it you will open the WSDL file for the webservice.
* Copy the URL, create a project in SoapUI > name it EmployeePortal, paste URL in the Initial WSDL box and click OK.
* You will see that there are two bindings available, if we expand the 12Binding of it, we will see three operations available in it – i.e. addEmployee, deleteEmployee, getEmployeeDetails.
* We can send the four parameters inside the request as an instance to add an employee into our database. And when we execute it, in the response section we will see “true” which basically means that the execution was successful and the employee is added.
* Now, to validate if the employee is actually added along with his/her other details into our database, we can go to MySQL Workbench and execute: use Employeeportal; and then execute select \* from Employeeinfo where name=’john’;

**Assertions:**

To be able to add assertions, we must first create test suites, then test cases and then we can add assertions by clicking the ‘plus’ button at the top of the request pane.

Ex: click the plus button to add assertions > select ‘compliance, status and standards’ > click on ‘SOAP Response’ > add. And the assertions will be added.

Scenario: we are asked to add an employee and:

* Verify that the response is returning ‘true’: achieved by using the Contains assertion: write true
* Verify that the response is not having null or false value: Not Contains
* Verify that the response is a Valid Soap Response: Soap assertion
* Verify that the response has Valid HTTP Code: Valid HTTP status codes assertion: add 200.
* Verify that the response time is within boundaries: Response SLA: add 310 (it means we want the response to take less than 3.1 seconds.
* Verify Security Threat: add assertions button > security > Sensitive Information Exposure assertion > add a token … (usage: we want to make sure that some particular info or details shouldn’t be exposed in our response)

Completed 1st video of section 3.

**C#** is a simple, modern, object-oriented, and type-safe programming language.

**.NET** is a free, cross-platform, open source developer platform for building many different types of applications.

**Visual Studio IDE:** is a development environment that is used for writing C# code. It is the number one preferred way of C#.

For references and questions:

* All topics: <https://wp.me/p7absY-PM>
* Instructor’s email address: [admin@ultimateqa.com](mailto:admin@ultimateqa.com)
* Twitter: @Nikolay\_A00
* All codes on GitHub: <https://github.com/nadvolod/csharplesons>

Solution: a solution is a container. It doesn’t contain any code itself. But it contains info about files and structures.

Variable: a name given to a storage area that our program can manipulate.