Automation Testing :

1. Manual Testing
   1. Different type of testing
   2. TestCase / Test Plan
   3. SDLC / STLC
   4. Defect LC
2. JAVA
   1. OOPS Concept
   2. Why Java is popular
   3. Installation
      1. Download
      2. Set the environments ( Windows and MAC )
      3. IDE (**Eclipse**, Netbeans, **intelllij**.. )
      4. Create a project
   4. Variables
      1. Local variables
      2. Global variables
      3. Constants
   5. Methods
      1. Methods without arguments
      2. Methods with arguments
      3. Methods without return type
      4. Methods with return type
   6. Access specifies
      1. Private
      2. Default
      3. Protected
      4. Public
   7. Access modifiers
      1. Static
      2. Final
      3. Abstract
      4. Synchronized
   8. Classes
      1. Abstract class
      2. Final class
      3. *Inner classes*
   9. Interfaces
   10. Conditional Statements
       1. If
       2. If ,else
       3. If ,elseif ,else
       4. Switch
   11. Looping Statements
       1. For
       2. While
       3. Do-while
       4. Foreach
   12. Operators (arithmetic, logical operators, increment and decrement Operators, conditional operators )
   13. String
       1. String functions
       2. Programs
       3. StringBuffer and StringBuilder
   14. Threads
       1. Why ? Advantages ????
       2. Thread class
       3. Runnable interface
   15. Exception Handling
       1. Try
       2. Catch
       3. Throws
       4. Throw
       5. Finally
       6. How to write UDE
   16. File Handling
       1. How to read and write files
          1. Txt
          2. Properties
          3. Excel
   17. Collections
       1. List
       2. Set
       3. Map
   18. Generics
3. Automation with Selenium:
   1. Why Automation is required
   2. What are the different tools available
   3. Why Selenium is Popular
   4. Different versions of selenium
      1. Selenium IDE -> version-1
      2. Selenium ~~RC -> Selenium 1.0~~
      3. Selenium Web Driver -> Selenium 2.0 / Selenium 3.0 / Selenium 4.0- alpha
      4. Selenium GRID
   5. Framework
      1. Function driven Automation Framework
      2. Keyword driven Automation Framework
      3. Data driven Automation Framework
      4. Hybrid driven Automation Framework
      5. **TestNG**
      6. **POM**
4. **Maven/ Gradle (Build Automation Tool )**
5. **GIT (Version Control Tool)**
6. **Jenkins ( CI and CD )**
7. **BDD (Cucumber)**

GIT

Version Control System



Advantages :

1. We can allow multiple developers to work simultaneously
2. Any deleted files you can get it back
3. Remove any newly added code from the project - can be done easily

Types:

1. LVC – Local Version controlling
2. Centralized version control tool - SVN, Perforce, TFS



1. Distributed Version Control -> GIT, Big bucket,..





Installation:

<https://git-scm.com/download/win>

Configuration:

git config --global user.name "Aravinda HB"

git config --global user.email [aru03.info@gmail.com](mailto:aru03.info@gmail.com)

Creating local repository and making changes

1. go to any folder which you want to make as a repository and execute
   1. git init
2. Add files in that and commit the changes to local repository
   1. git add <<filename>>
   2. git commit –m “<<any valid message>>”
3. Create a Remote Repository (gitlab, github,bigbucket……)

git remote add origin git@github.com:AravindaHB/ITAutomation\_June\_Batch.git

git push -u origin master

java

Installation

- JDK

- JRE

Setting Environment Variables –

- JAVA\_HOME

- PATH

Data types in JAVA :



Variables –

- Local Variables :

- any variable which we declare inside a method is called local variable.

- scope of the variable is only inside the method outside the method we cannot access local variables.

- We cannot specify any access specifies or access modifiers to local variables

-Global Variables :

1. Instance Variable

2. Static Variable

3. Constants

**Methods** :

/\*

\* Methods describes the ***Behavior*** of an Object

\* reusable entities - you write once and use it many times

\* /

Syntax :

[AS] [AM] return\_type name\_of\_method([arguments list])

{

}

return\_type - can be void OR any data\_type

if the return\_type is other than void - last statement in your method should be return statement

**TYPES:**

1. Methods without return type and with out arguments

2. Methods without return type and with arguments

3. Methods with return type and with arguments

**Polymorphism :**

1. Overloading / Static polymorphism / Compile Time polymorphism
   * Two or more functions having **same name** but **different signature**
   * Signature
     1. Type of argument
     2. Number of arguments
     3. ***Return is not part of signature***
   * Because at the time of compilation Compiler will check is there a method which can be used! if it is not found it throws CTE.
2. Overriding / Dynamic polymorphism / Runtime polymorphism
   * Two or more functions having **same name** but **same signature** one in parent Class and another one is in child class
   * **Why :** if the super class function is not giving a correct output then we override the super class function in child class to give a new implementation.

**NOTE**

non-static means instance (Object)

**Access specifies :** specifies the accessibility of a member



NOTE :

*1. You can write as many classes as you want in a same file*

*2. You should have only one public class in one file*

*3. File name should be same as the class name which contain main method*

*4. main method should be present inside the public class only*

*5. In Java we* ***cannot*** *have* ***private*** *or* ***protected***  *Class*

**Access Modifiers :** Changes the behavior of a member(variable or method)

**static –**

1. *Value will be stored inside a class memory not inside a Object*
2. *Same value will be copied to all Objects*
3. *No need to create an Object to access static member*

**final –**

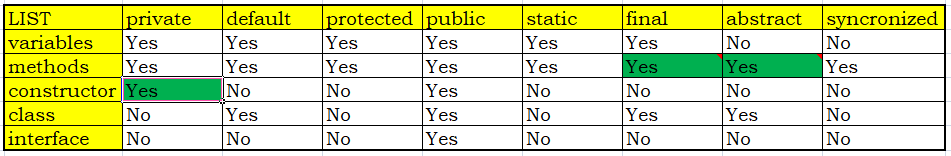
1. We declare variables as final -> To make variables as constants
2. We can also declare methods as final -> It can not be override (we can not give new implementation in child class )
3. We can also declare class as final🡪 Then you can not make final class as a Super class.

**abstract –**

1. is a KeyWord which can be given only to class or methods
2. class can be declared as abstract in 2 scenarios
   1. if a class contain any abstract methods
   2. if you want to restrict creating Object to your class

**synchronized –**

1. Discuss when we start THREADS



**Constructors :**

1. **Default**
2. **Parameterized**

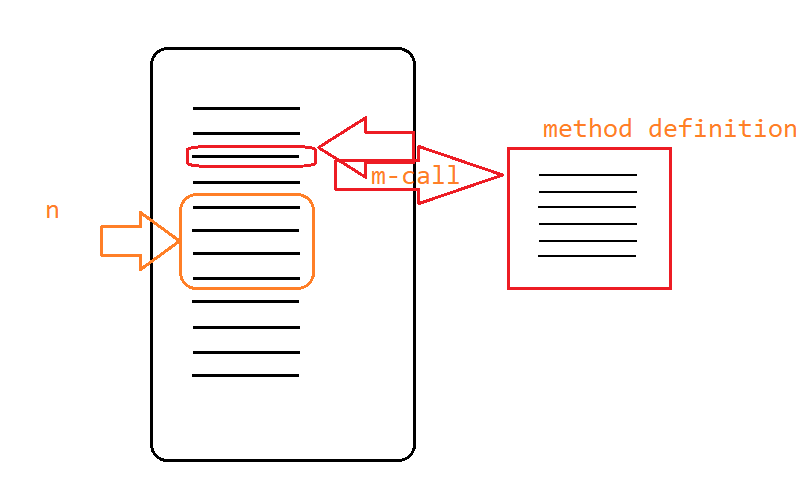
**Strings:**

String functions

**Programs**:

1. WJP - to Reverse a given String
2. WJP - to replace all ‘a’ with ‘s’
3. WJP - to find the given string is palindrome
   1. NitiN,MOM,DAD
4. WJP – to reverse a word
   1. split – using spaces
   2. call your reverse function and pass word
5. WJP – to count the number of occurrence of a given word
   1. read a String word by word
   2. check if it is a given word if yes increment the counter
   3. print the count with word
6. WJP – to replace all special characters in a given Date function with ‘\_’
   1. Tue Jun 16 22:17:01 IST 2020

**Looping Statements :**



1. for
   * Syntax

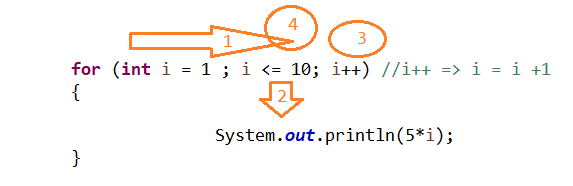
for(initialization; condition; increment/decrement)

{

-----

-----

}



* When we know the number of iterations we use for loop,
* execution is 0 OR n

1. while -> When we don’t know the exact number of iterations

while(condition)

{

--- // **we can put any valid java code ex - another loop, condition etc**

---

}

1. do,while
   * Similar to while loop
   * first it executes loop Once, if condition is satisfied then it continues the execution
   * execution is 1 OR n
2. enhanced for loop / Extended for loop / for each loop
   * Can be used only with Objects or with String Array
   * Can be used only with Collections not with Primitive Data types
   * Syntax:

for(Local\_Variable\_of\_Array\_DT : ArrayName)

{

====

====

}

**Conditional Statements:**

1. **if**

if(condition)

{

--

**}**

1. **if,else**

if(condition)

{

--

**}**

else

{

---

}

1. **if,elseif,else**

if(condition1)

{

--

**}**

else if(condition2)

{

--

**}**

elseif(condition1)

{

--

**}**

else

{

---

}

1. **switch**

switch(key)

{

case 1 :

----

----

break;

case 2 :

----

----

break;

case 3 :

----

----

break;

**default:**

**---**

**---**

**break;**

}

**Ternary Operator:**

(Conditon)?<<true>>:<<false>>

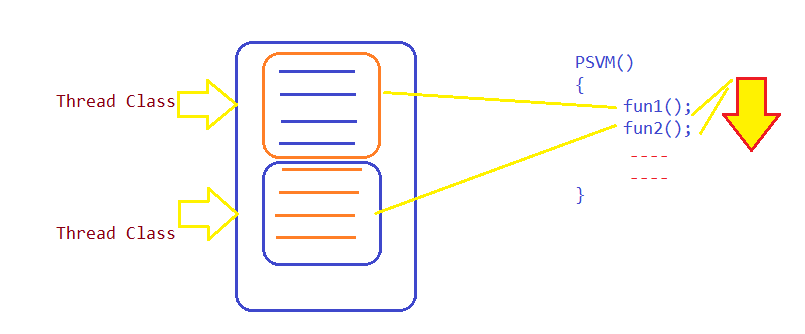
**Reading Values at Runtime:**

**Scanner - java.util**

* To Read Values from the Keyboard at the run time .
* Scanner class has several method, based on the data type we have to call the corresponding class

**Threads:**

Multithreading programming language



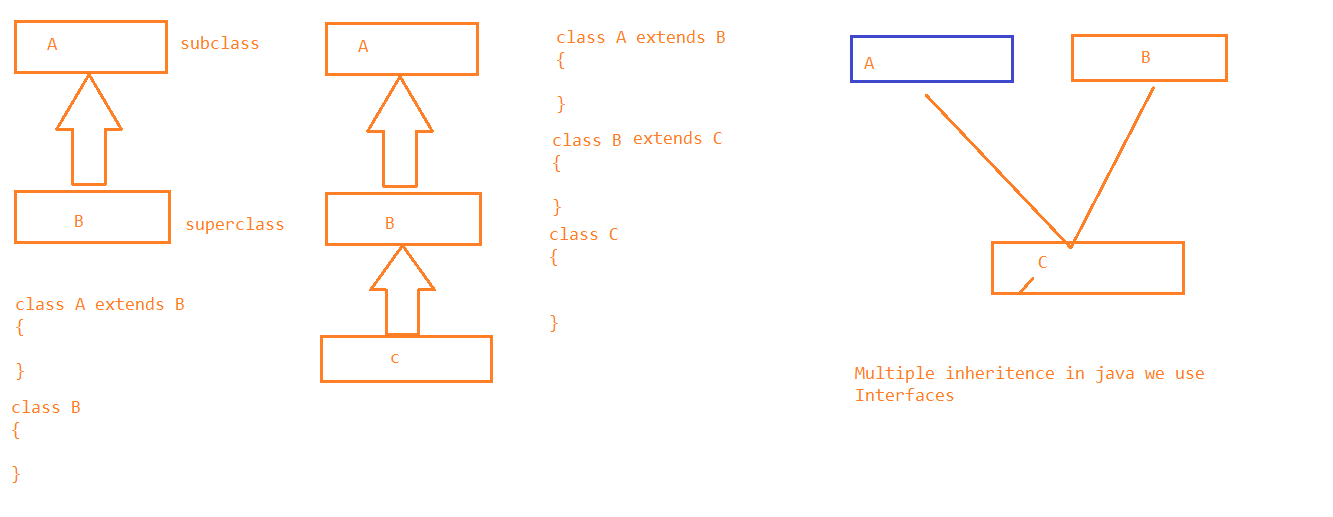
**Ways to Create Multiple Threads:**

1. Thread
   1. write a class(ChildTread) by making Thread as a super class
   2. Override run method and keep the thread logic
   3. Create an object to ChildThread class and call start method to execute the logic in parallel
2. Runnable Interface
   1. write a class(ChildTread) by making Runnable Interface as a super class
   2. run method which is abstract
   3. Create a Object to Thread class and pass Runnable object

**Interface:**

Interfaces are Similar to Class, but it is 100% pure abstract Class.

Inside interface all variable by default are public static final variables and all methods are abstract by default.

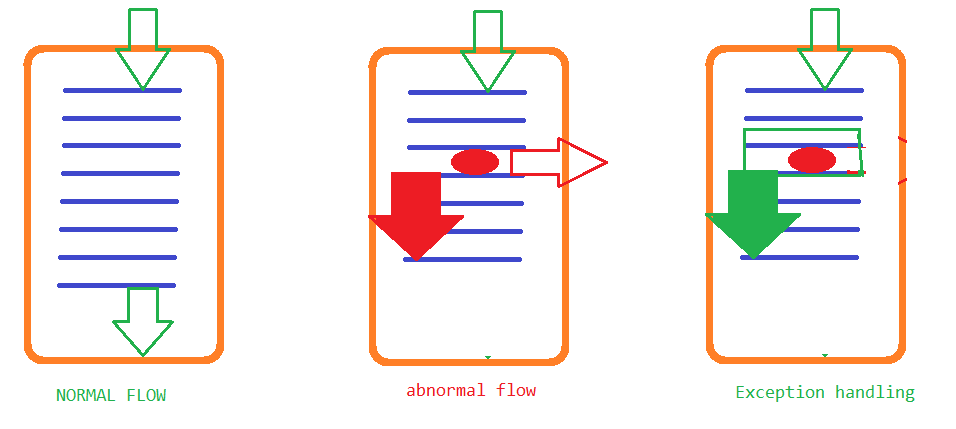


**Exceptions:**

Exceptions are abnormal statements because of which program will be terminated abruptly.

Errors and Exceptions are different .

**Error –** Syntax mistake done by programmer

****

****

**- **

- To Handle the exception we have to use try and catch blocks

- for one try block you can keep as many catch blocks as you want

- there should not be any valid java code between try and catch

1. try and catch
2. throws
3. throw
4. finally

**User Defined Exception**:

1. Your class should be a subclass of Exception Class
2. You should have at least one constructor inside the class(one default and another one is parameterized)

**Type Casting**

Converting one data type to another data type.

Usually from smaller data type to larger will be automatically done Auto Type casting.

Larger Data type to smaller data type we have to explicitly convert, Explicit type casting.

(data\_type)Variable;

ASSIGNMENTS:

- add a switch block with different option

1. Credit

2. Debit

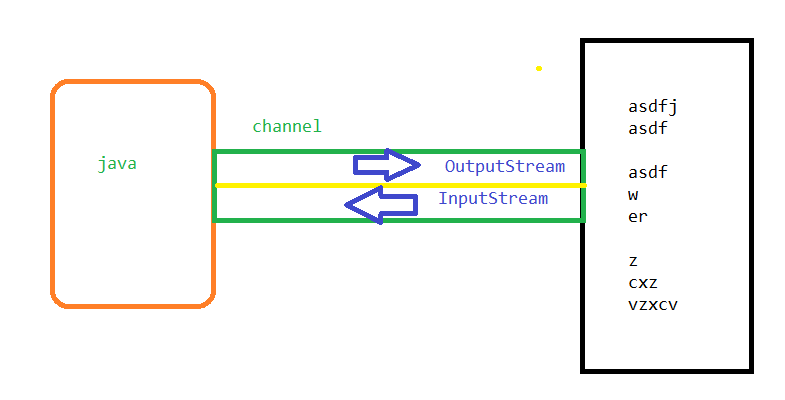
3. Show Balance

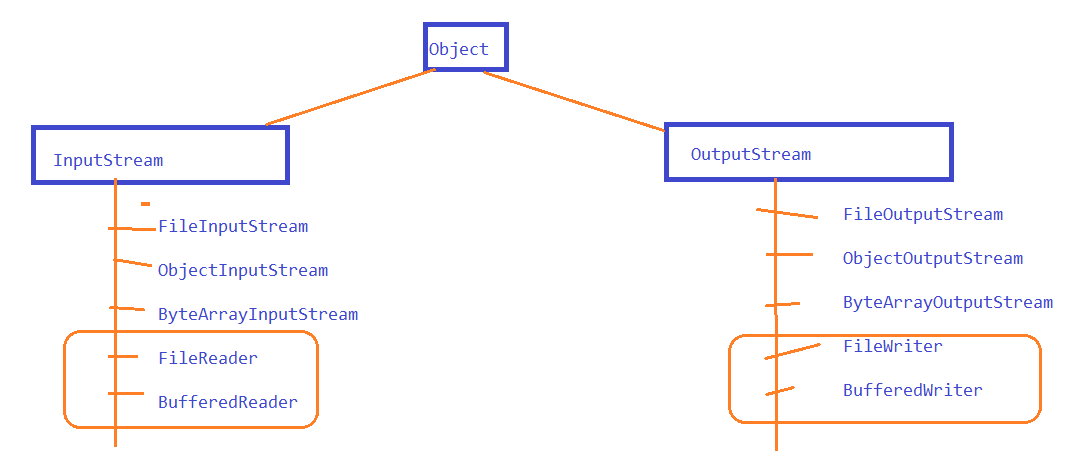
4. Display Personal Information

**File Handling :**

**- java.io –**We have to make use the classes and interfaces Present in java.io package.

- Using java libraries, we can read only text file or properties file ( we cannot read pdf, doc, excel etc)

****

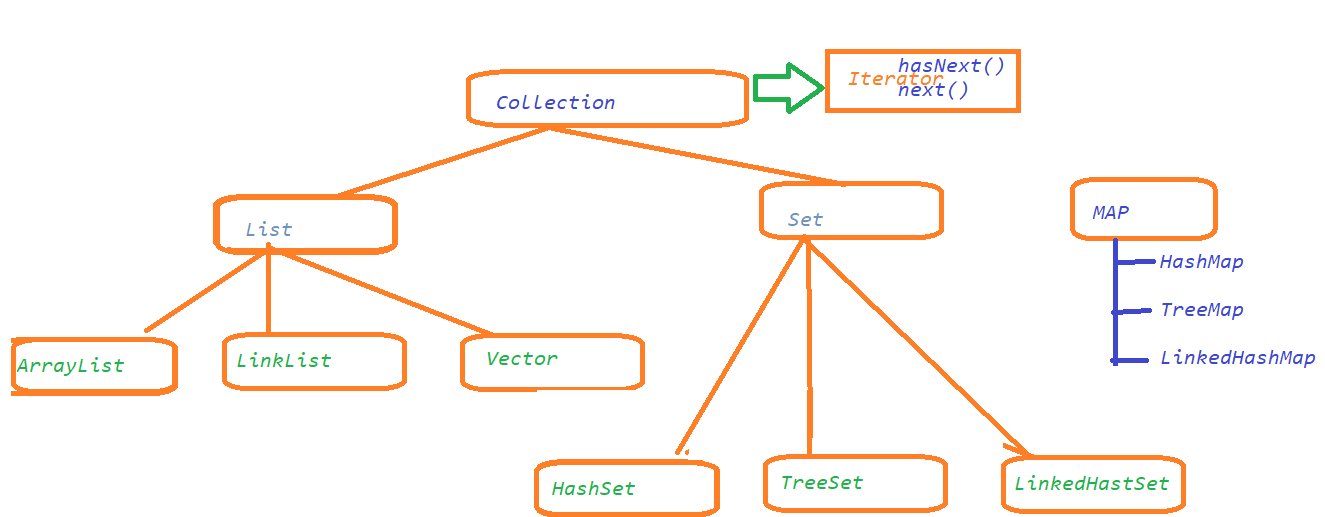
****

**Collections:**

*Drawbacks of Arrays:*

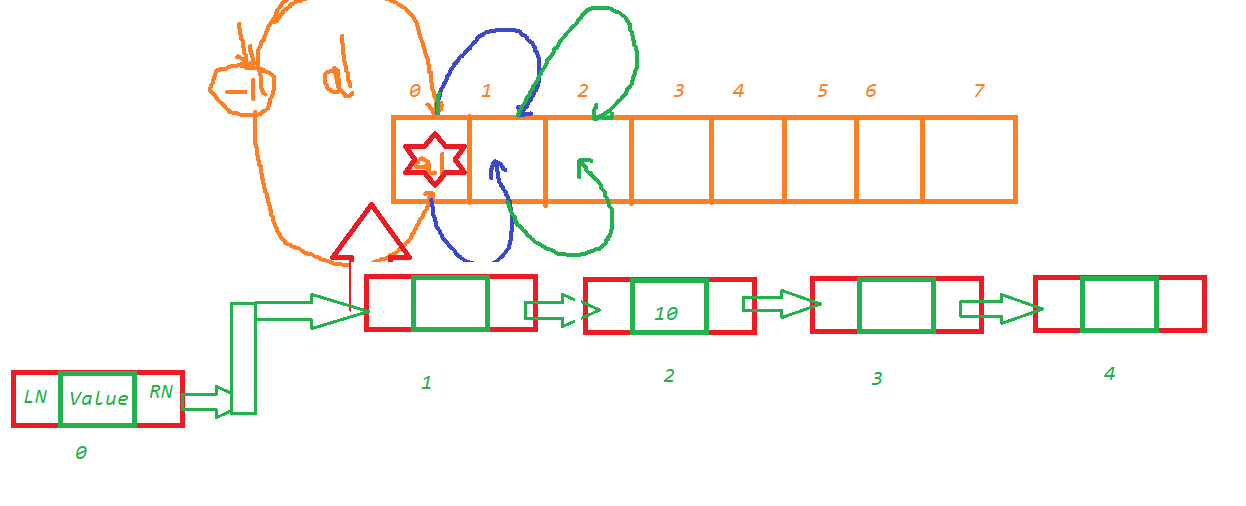
1. we have to specify the size.
2. Only one type of data can be stored
3. if you want to insert or delete elements in between
4. if you want to avoid storing duplicate values – NOT POSSIBLE
5. Sorted order NOT POSSIBLE
6. Only Primitive datatypes we can store

**Collection API –**



Collections are containers ( Bag ), which can be used to store anything ( Primitive data type or derived data type )

ArrayList / LinkedList / Vector :



HashSet/TreeSet/LinkedHashSet



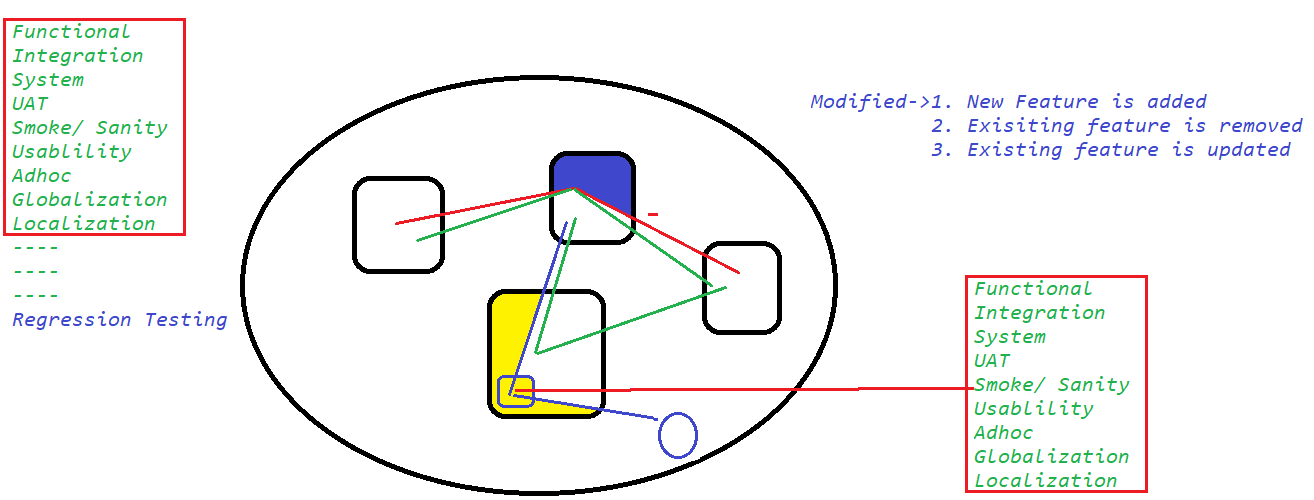
HashMap/TreeMap/LinkedHashMap

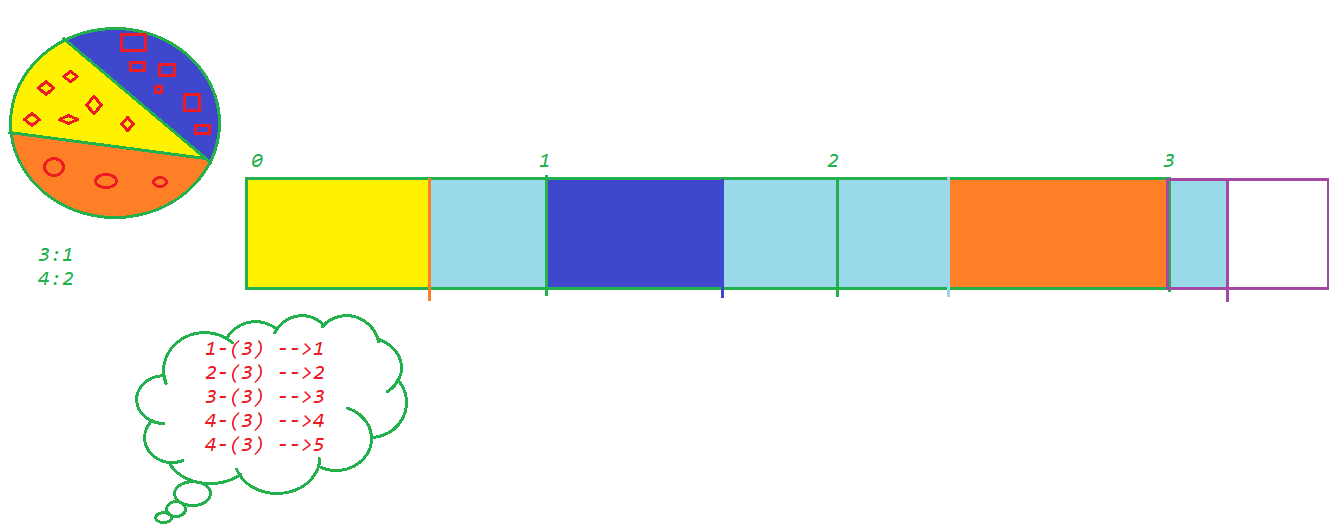


Wrapper Classes

1. convert Primitive data types to Derived data types (Objects).
2. Convert one data type to other Ex: “5”=>5

**Automation Testing**

****

****

**Drawbacks of Manual Testing :**

1. We depend on **Person** not on Process.
2. Mood of a Tester
3. Slow- > Productive time we spend is 5-6 hrs
4. Testing depends on the Product / Domain Knowledge

**Automation Testing**

1. We rely on tool to test the application
2. Tools
   1. TestComplete
   2. UFT
   3. RFT
   4. Saahi
   5. EggPlant
   6. **Selenium**
   7. AutoIT
   8. ----
   9. ---
   10. ---

**Selenium –**

1. It is Open Source free software
2. **No dedicated machine** is required to execute the tests
3. Only tool which supports multiple programming languages – ***Java****, c#, ruby, python, perl, php, javascript.*
4. Only tool which supports almost all the popular browser
5. Parallel Execution
6. Distributed execution
7. Any platform

Selenium :

- Selenium IDE ->( Java Script code )

Same origin policy issue.

- ~~Selenium RC -> Remote Control ->->( Java Script code )~~

- ***Selenium WebDriver 2.0,*** Oct, 2016 ->***3.0 (3.141.59) / Alpha 4.0 (JAVA)***

- Selenium Grid

Selenium IDE

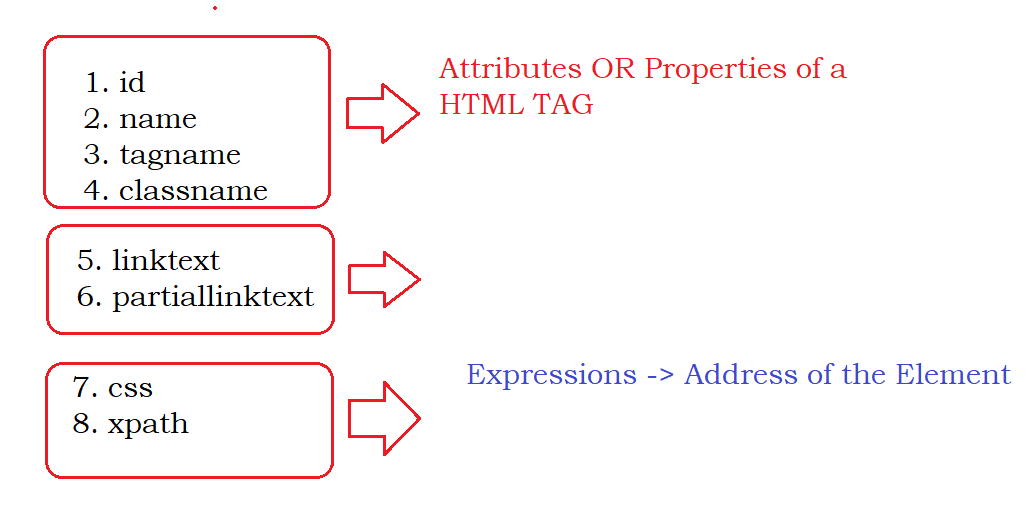
Before Selenium 3.0 -> IDE was only on Mozilla Firefox

After/When Selenium 3.0 -> IDE was available on both Mozilla Firefox /Google Chrome as an Addon.

Chrome Addon :

***8-Ways we can identify the WebElement (Anything we see on a webpage) :***

1. id
2. name
3. tagname
4. classname
5. linktext
6. partiallinktext
7. css
8. xpath



Verification Statements in IDE :

1. Verify
   1. Continue even though there is a failure
2. Assert
   1. Stops the execution

Expressions:

css :

1. htmltag[propertyname=’propertyvalue’]

ex : input[name='username']

1. if ID ->
   1. htmltag#idvalue
   2. #idvalue
   3. EX :
      1. input[id='username']
      2. input#username
      3. #username
2. Parent to child traverse
   1. >
   2. Ex :
      1. a#loginButton > div

NOTE - don’t use id which starts with **ext-gen** or **auto-gen**

**xpath:**

**XML Path**

1. **Absolute Xpath**
   1. **/html/body/../../../../../../../../../.././input**
2. **Relative Xpath**
   1. **BASIC**
      1. //htmltag
   2. **BASIC-WITH FILTER**
      1. //htmltag[@propertytype=’propertyValue’]
   3. **XPATH – more than one property**
      1. **Using Logical Operator – and / or**
         1. //input[@name='username' **or** @name='pwd']
         2. //input[@name='username' and @id='username
      2. **Complex Xpath**
         1. //td[(@class='current day' or @class='wd day' or@class='we day') and text()='29']

OR

* + - 1. //td[not(@class='past day') and text()='29']
    1. **Functions:**
       1. **text()**
          1. //htmltag[text()=’value’]
          2. //div[text()='Login ']
       2. **contains(arg1, arg2)**
          1. arg1 – any attribute(any attribute inside html tag) or function – text()
          2. arg2 - partial value corresponds to arg1
          3. EX:

//label[text()='Keep me logged in']

//label[contains(text(),'me')]

//img[contains(@src,'timer')]

* + - 1. **starts-with(age1, arg2)**

//button[starts-with(@id,'ext-gen')]

//h3[starts-with(text(),'IT Automation')]

* + 1. **Parent To Child**
       1. //a[@id='loginButton']/div
       2. //td[@id='loginButtonContainer']//a
    2. **Child to Parent**

Whenever we have dependent and independent elements then we have to traverse from child to parent

* + - 1. Write an xpath for independent element
      2. traverse (In HTML DOM) until both dependent and independent elements are highlight in the application
      3. Once it is done(STEP2) traverse to its child
      4. EX:
         1. **//tr[th[text()='Directed by']]/td**
         2. **//div[div[div[p[text()='SG-253 | SG-516']]]]//span[@class='actual-price']**
    1. **LOGICAL OPERATORS :**
       1. **and -> if both the properties are matched**

1. //span[@class='product-discountedPrice' and (text()='395')]
   * + 1. **or ->if any one property is matched** 
          1. //span[@class='product-discountedPrice' or text()='Rs. ']
       2. **not-> if not matched**
          1. **//span[@class='product-discountedPrice' and not(text()='395')]**
     1. **Xpath using Axes Functions**