A keyword based Hybrid Test Automation Training using Selenium

November 2016

Curated exclusively for UHC

By Tekstrom Inc.,

TEKSTROM
Nurturing quality. Cultivating talent.

Selenium Training with Qualitree

We strive to offers a 360 degree experience of learning

Blend 'Onsite-Online' with Hands ON experience

Content curated and developed by Industry practitioners

Designed to suit the need of every professional within the organization

Passion to enable customers to have consistent & Best Practices in the ever challenging Test and Automation Domains.

Trainer Profile – Krishna Sapare

Krishna Sapare

- Test Automation Architect and Geek.
- Over 16+ years in spanning - Retail, Mobile, Telecom, Networking industry.
- Cross functional leadership
- Currently leading Test Automation Engineering and DevOps Coach.
- Global facilitator and Practitioner of DevOps
- Part of select Global Technology Transformation team. Part of defining and transitioning best practices in Test Automation for the Corporate.
- Designed and Deployed several strategic product and application test assets with high emphasis on re-use, Compliance and Conformance in the mobility Industry.
- In depth experience in Open source tools – Selenium
- Hands-On approach to solve critical challenges.
- Automation, DevOps, Mobile Application Development are his key strengths and areas of interest.
- Integral part of establishing a Test Centre of Excellence (TCoE)
- Presented several papers in leading forums of Test and Automation.
- Certifications from leading bodies for Software and Agile standards

Trainer Profile – Naveen Kumar

- Test Automation Architect
- Over 12 years of Development and Coach experience
- Test Automation and Tools Specialist.
- Consulting practitioner experience across Retail, Telecom and Supply Chain domains.
- Drove tools evaluation and standardization across the enterprise
- Cross functional expertise across diverse development and test teams.
- Part of Global Test Experts pool within the Corporate.
- Rendered several case studies and approach papers for vital decision making across the Corporate.
- Helped setup a ToD – Test On Demand infrastructure to help leverage common assets.
- Currently - Technical Coach(across Agile, Infra and Test Automation practices)
- His expertise and interest include Automation across SDLC, consulting on Automation Solutions as core practices and Coaching

Introduce Yourself

- About You
- Your Experience in Selenium
- Expectations from the Session

Learning by Examples

We will learn incrementally each day

- Learning new concepts
- Understanding the motive behind them
- Building on the example as we learn

We will use a sample application for the learning

Code walkthrough and execution to help appreciate the concepts

From Simple Tests to Building Frameworks...

Agenda for the Session

DAY 1	DAY 2	DAY 3	DAY 4	DAY 5
❖ Introduction to • Selenium • OOPS • Java ❖ IntelliJ Setup ❖ Selenium WebDriver ❖ Simple Tests	❖ Web Elements ❖ Locators ❖ Browser plugins for Object Identification • Firefox • Chrome	❖ Object Synchronization ❖ Test Data Management ❖ Parameterization ❖ Exception Handling ❖ Reporting • Test NG • JUNIT • log4j	❖ Framework Design Patterns • Keyword • Data • Hybrid Model • Page Object Model ❖ DB Interactions ❖ Selenium Grid	❖ Overview of • Jenkins • Maven ❖ Jenkins and Selenium Integration ❖ Framework code Walkthrough ❖ Insights into • Groovy • Geb & Spock • Cucumber

Learning Objectives– Day 1

- Introduction to Automation
- Automation Tools
- Selenium History and Current
- Object Oriented Programming
- JAVA Basics
- Introduction to IntelliJ
- Setting up Projects for Se Automation
- Selenium WebDriver
- Exercise : Launching/Controlling Browser, Page Navigation

Automation Principles

- Building code to execute a manual test process. It is an Investment, to see long term value
- Investments include Human Resource (effort), Tool and Execution effort.
- Returns are also multi-dimensional, spread across dimensions of : Effort reduction , Expedited schedule , Quality
- Test Automation Pyramid, represents the logical model of implementing Automation in an Agile program

Test Automation is a multi-step process

What

Right Candidates

- Test Pyramid Implementation
- Unit, API, Sanity
- Regression, Cross Browser
- Any repeatable test process

Bad Candidates

- One time, OS tests
- High risk , Edge cases
- Third party API tests
- Alpha – Beta Tests

How

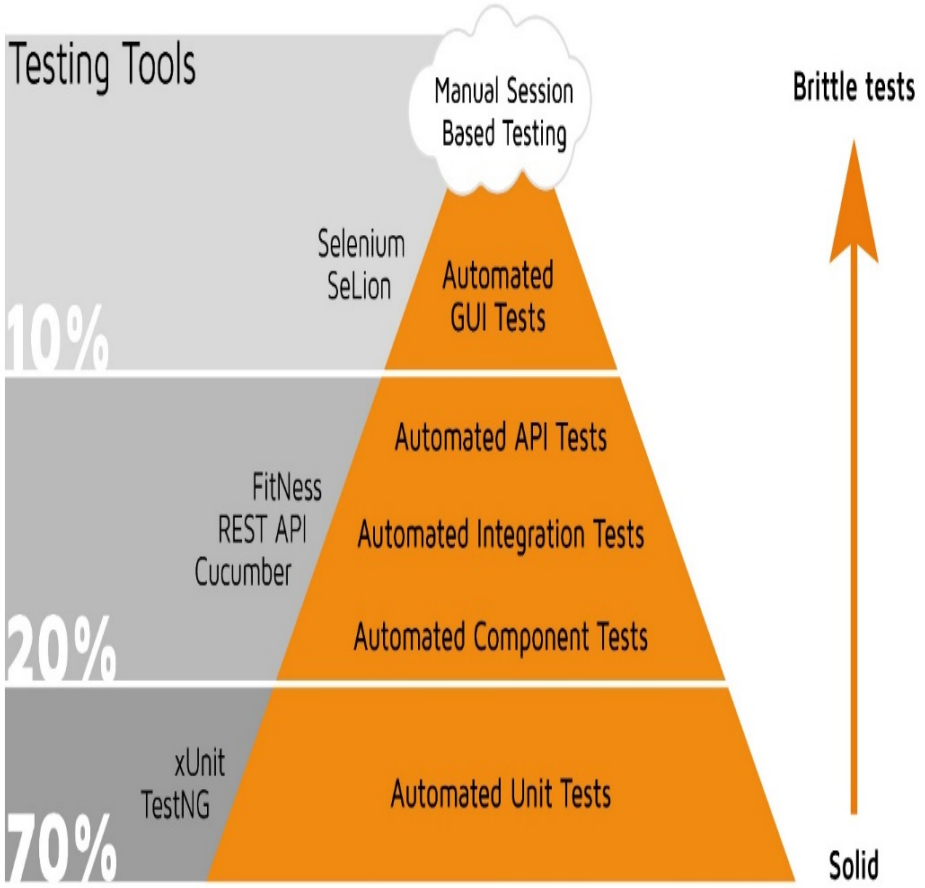
- Goal based
- Right tests
- Robust framework and tools
- Monitor via Constant feedback
- Right Team

How Much

- Incremental coverage
- 100% coverage not achievable and not needed
- Right coverage based on goals

Test Automation Pyramid

- Automated GUI tests, offer great value, as they replace time consuming manual tests, though not fully efficient
- Quality should be built-in and Automation should be across all layers (refer pic) in Agile
- Unit & API tests are:
  - Faster
  - Defined scope
  - Less/No dependencies
  - Less expensive to automate
  - Great value
  - Solid and Stable
- Ideal distribution of test coverage through phases (Unit, API and UI) be 70%, 20% and 10%
- Open source tools are recommended against expensive commercial tools
- Emphasize on disciple and focus on XP practices like Test Driven development (see other chapter for TDD) &
- Pair programming



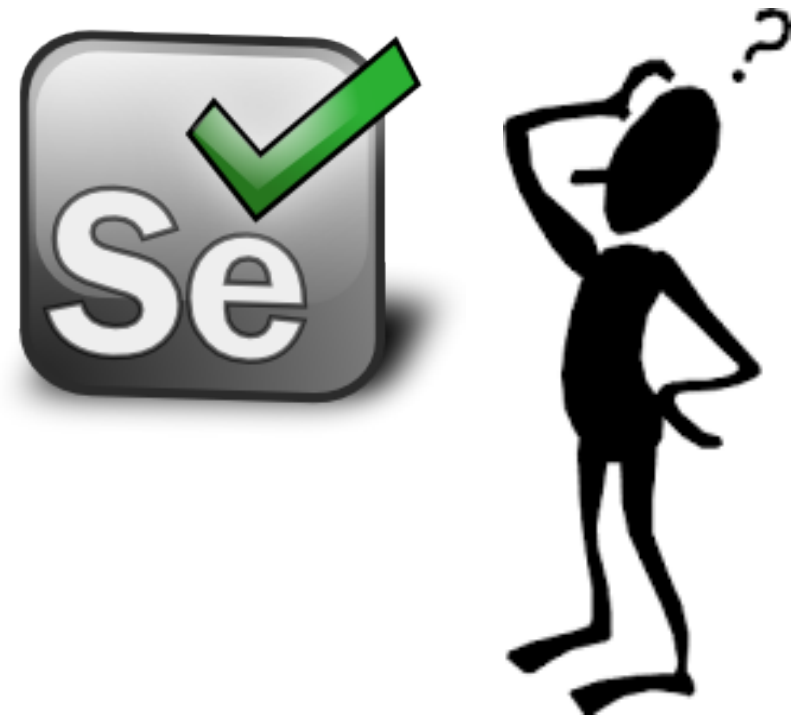
Why Selenium

Flexible and Seamless automation

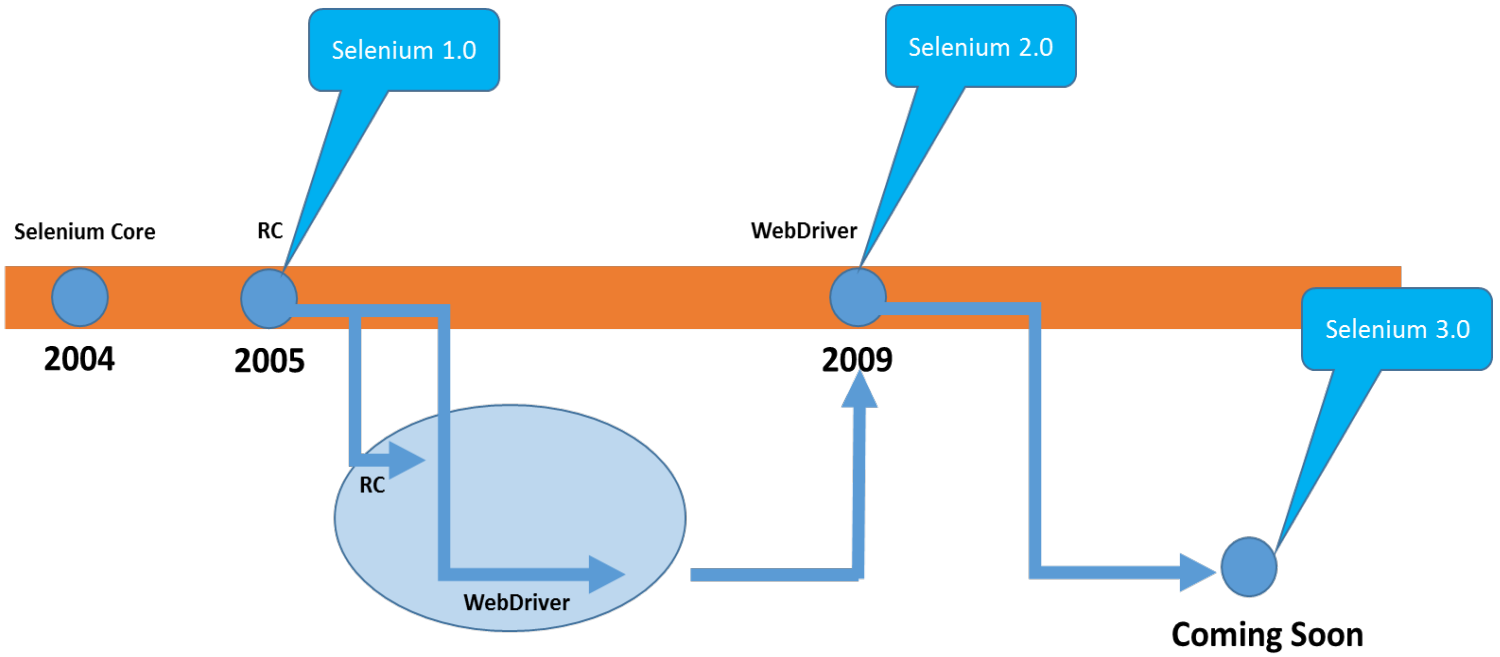
- It comes without any price tag
- Freedom of language
- If WEB then Selenium
- Large Community of developers

Promotes testing to next level

- Faster feedback loops
- Extensive tool/ framework integrations
- Customizable Frameworks can be created
- Portability of other open source plugin is seamless



History of Selenium



"Why are the projects merging? Partly because WebDriver addresses some shortcomings in selenium (by being able to bypass the JS sandbox, for example. And we've got a gorgeous API), partly because selenium addresses some shortcomings in WebDriver (such as supporting a broader range of browsers) and partly because the main selenium contributors and I felt that it was the best way to offer users the best possible framework."

– Simon Stewart

Selenium - Simplified

Version	What it is?
Selenium IDE	Selenium IDE is a recording tool for automating Firefox, with the ability to generate simple RC/WebDriver code
Selenium 1.x aka Selenium RC	First version of Selenium API
Selenium 2.x aka Selenium Web driver	Selenium WebDriver is the core library to drive web browsers on a single machine
Selenium Grid	Selenium Grid is a tool to execute Selenium tests in parallel on different machines.
Selenium 3	Its in beta version

Object Oriented Programming

OOP is mainly a program design philosophy

The Focus of Object Oriented Programming is not on Structure, but on Modelling Data

Programmers code using “blueprints” of data models called classes

Everything in OOP is grouped as self-sustainable “objects”
Re-usability is gained by means of 4 main OOP concepts

OOP/Java Basic Terminology

Object

Usually a person, place or thing *(noun)*

Method

An action performed by an object *(verb)*

Property or Attribute

Characteristics of certain object

Class

A category of similar objects (such as automobiles), does not hold any values of the object's attributes/properties

An object is a class when it comes alive

Messages and Interfaces

Messages

A request for an object to perform one of its operations (methods)

All communication between objects is done via messages

Interfaces

Set of operations (methods) that given object can perform

The interfaces provide abstractions – No need to know implementation

Everything an object can do is represented by message interface

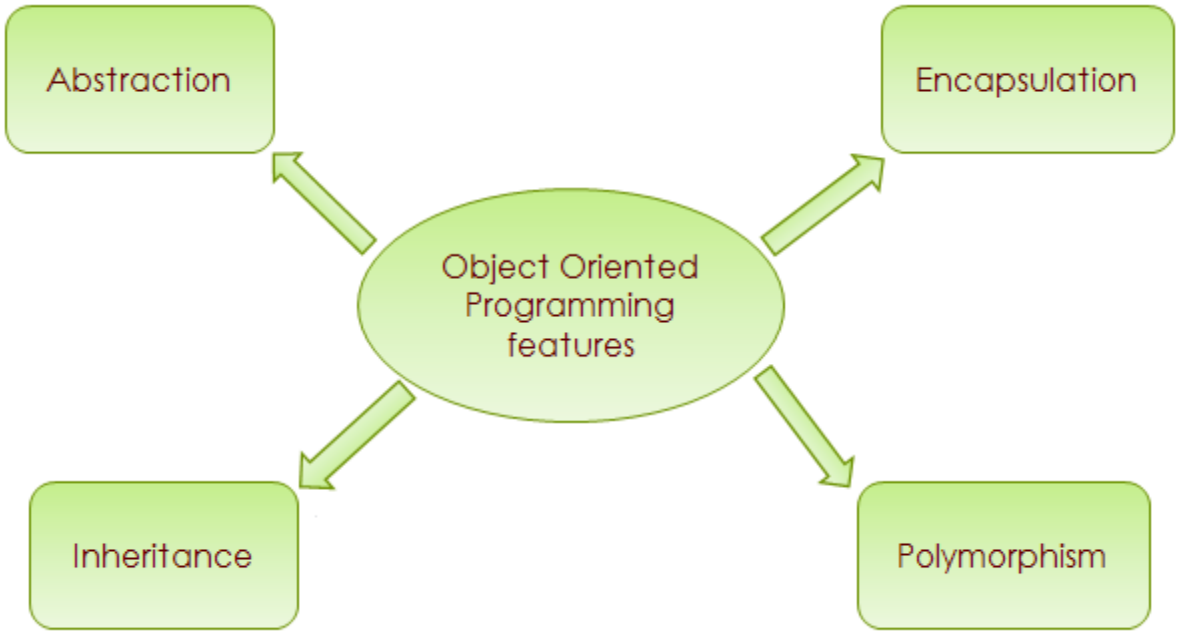
OOP Basic Concepts

Encapsulation

Inheritance

Abstraction

Polymorphism

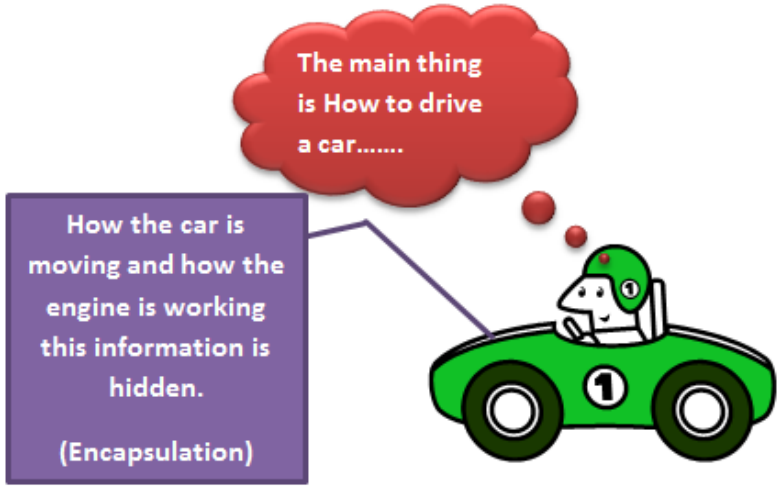


Encapsulation

Inclusion of Property and Method within a class/object in which it needs to function

Enables re-usability of an instance of an already implemented class in a new class while hiding & protecting them from client classes

Allows a class to change its internal implementation without hurting the overall functioning of the system



Example: Power steering mechanism of Car. There are lot of complexity inside the system, but for external world its only one interface. The power steering unit as a whole itself is complete and independent

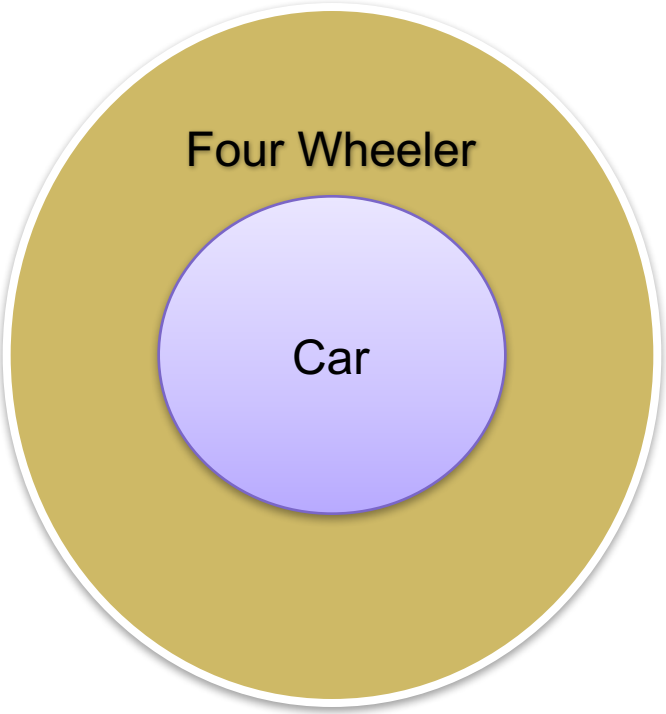
Inheritance

A way of organizing classes

Classes with common properties can be grouped so that their common properties are defined only once in parent class

Inheritance allows child classes to inherit the characteristics of existing parent class – *(Eg. Attributes and Properties)*

Child class can extend the parent class *(add new fields, refine methods)*



Example: Car is a classification of Four Wheeler. Car (Sub Class) acquires properties of Four Wheelers (Super Class). Other classification would be Jeep, Van etc. OR small car, open car acquiring properties of Car

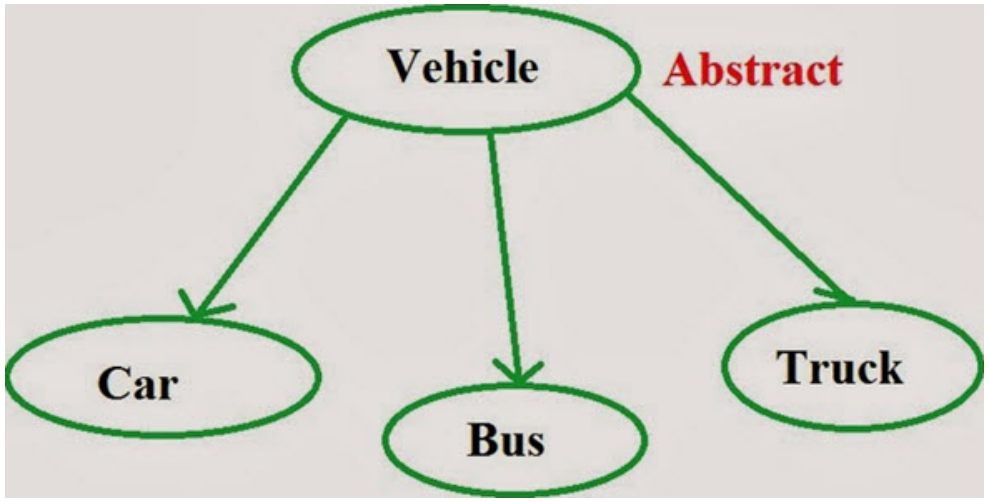
Abstraction

A Design principle, is the process of removing characteristics from something in order to reduce it to a set of essential characteristics

A programmer can hide all but relevant data about a class in order to reduce complexity and increase re-usability

Represent necessary features without representing background details

An abstract class may not have any direct instances



Example: We would always buy a BWM, Ford or Toyota and not just "a car". Here brands are concrete things and "Car" is Abstract

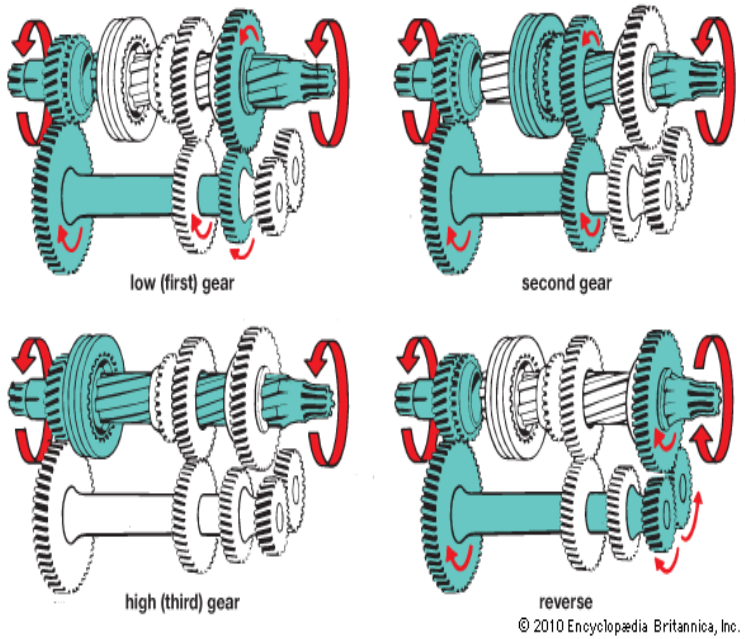
Polymorphism

Means “Many Shapes”. Ability to request that the same methods be performed by a wide range of different type of things

An object has multiple identities based on its class inheritance tree

Achieved by different techniques namely method overloading, operator overloading and method overriding

Example : Car Transmission System. It will have forward and reverse gears. When the engine is accelerated then depending upon which gear is engaged different amount power and movement is delivered



Encapsulation Example

```
public class EncapTest {
   private String name;
   private String idNum;
   private int age;

   public int getAge() {
      return age;
   }
   public String getName() {
      return name;
   }

   public String getIdNum() {
      return idNum;
   }
```

```
      public void setAge( int newAge)
      {
         age = newAge;
      }

      public void setName(String
                  newName)
      {
         name = newName;
      }

      public void setIdNum(String
                              newId)
      {
         idNum = newId;
      }
}
```

```
public class RunEncap {

   public static void main(String args[]) {
      EncapTest encap = new
                        EncapTest();
      encap.setName("James");
      encap.setAge(20);
      encap.setIdNum("12343ms");

      System.out.print("Name : " +
            encap.getName() + " Age : "
             + encap.getAge());
   }
}
```

Inheritance Example

```
class Calculation {
   int z;

   public void addition(int x, int y) {
      z = x + y;
      System.out.println("The sum of the given
                                    numbers:"+z);
   }

   public void Subtraction(int x, int y) {
      z = x - y;
      System.out.println("The difference between the
                                    given numbers:"+z);
   }
}
```

```
public class MyCalculation extends Calculation {

        public void multiplication(int x, int y) {
                z = x * y;
                System.out.println("The product of the
                        given numbers:"+z);
        }

        public static void main(String args[]) {
                int a = 20, b = 10;
                MyCalculation demo = new
                                MyCalculation();
                demo.addition(a, b);
                demo.Subtraction(a, b);
                demo.multiplication(a, b);
        }
    }
```

Abstraction Example

```
public abstract class Employee {
   private String name;
   private String address;
   private int number;

   public Employee(String name, String address, int number) {
      System.out.println("Constructing an Employee");
      this.name = name;
      this.address = address;
      this.number = number;
   }

   public double computePay() {
     System.out.println("Inside Employee computePay");
     return 0.0;
   }
```

```
   public void mailCheck() {

      System.out.println("Mailing a check to " + this.name
                                    + " " + this.address);
   }

   public String toString() {
      return name + " " + address + " " + number;
   }

   public String getName() {
      return name;
   }

   public String getAddress() {
      return address;
   }

   public void setAddress(String newAddress) {
      address = newAddress;
   }

   public int getNumber() {
      return number;
   }
}
```

Abstraction Example, continued...

```
public class Salary extends Employee {
   private double salary;   // Annual
                                salary


   public Salary(String name, String
address, int number, double salary)
   {
      super(name, address, number);
      setSalary(salary);
   }
   public void mailCheck() {
      System.out.println("Within mailCheck
               of Salary class ");
      System.out.println("Mailing check to "
+ getName() + " with salary " + salary);
   }
   public double getSalary() {
      return salary;
   }
```

```
public void setSalary(double
newSalary) {
      if(newSalary >= 0.0) {
         salary = newSalary;
      }
   }

   public double computePay() {

System.out.println("Computing
               salary pay for " +
               getName());
      return salary/52;
   }
}
```

```
public class AbstractDemo {

   public static void main(String [] args) {

      Salary s = new Salary("Emp1",
               "Suite 1, IL", 3, 3600.00);
      Employee e = new Salary("Emp2", "Boston,
                                    MA", 2, 2400.00);
      System.out.println("Call mailCheck using Salary
                                    reference –");
      s.mailCheck();
      System.out.println("\n Call mailCheck using
                           Employee reference–");
      e.mailCheck();
   }
}
```

Polymorphism: Overloading and Overriding

```
class Overload
{
   void demo (int a)
   {
      System.out.println ("a: " + a);
   }
   void demo (int a, int b)
   {
      System.out.println ("a and b: " + a + "," + b);
   }
   double demo(double a) {
      System.out.println("double a: " + a);
      return a*a;
   }
}
class MethodOverloading
{
   public static void main (String args [])
   {
      Overload Obj = new Overload();
      double result;
      Obj .demo(10);
      Obj .demo(10, 20);
      result = Obj .demo(5.5);
      System.out.println("O/P : " + result);
   }
}
```

```
class Override
{
   public void method()
   {
      System.out.println(”Method Called");
   }
}

class OverrideOne extends Override{
   public void method(){
      System.out.println(”Method Inside");
   }
   public static void main( String args[]) {
 OverrideOne obj = new OverrideOne();
      obj.method();
   }
}
```

Why JAVA?

JAVA is an OOPs based Internet Programming Language

Features

Platform Independent
Secured
Simple
Robust
Multithreaded
Distributed

IDE and use of it

An integrated development environment (IDE) is a Software Application that provides comprehensive facilities to computer programmers for software development

An IDE normally consists of a source code editor, build automation tools and a debugger

Most modern IDEs have intelligent code completion

Many modern IDEs also have a class browser, an object browser, and a class hierarchy diagram, for use in object-oriented software development

IntelliJ IDE

- Developed by JetBrains and is available as an Apache 2 Licensed community edition which is opensource.

- Features:
  - Deep insight into your code
    - Smart Completion
    - Chain Completion
    - Data flow analysis
  - Developer Ergonomics
    - Editor-centric environment
    - Shortcuts for everything
    - Ergonomic user interface
    - Inline debugger

- **Built-in developer tools**
  - Version Control
  - Build Tools
  - Test runner & Coverage
  - Terminal
  - Database tools
  - Application servers

Setting up project for Selenium using IntelliJ

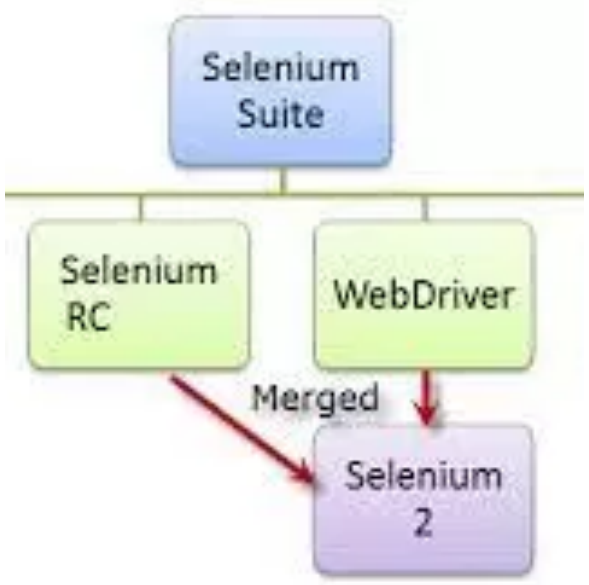
Web Driver Evolution

- Selenium RC runs on JavaScript libraries that drives interactions with the browsers.
  - Browsers apply security limitations to Javascript which makes many things impossible to do.
  - Web applications became more powerful overtime and super cool features were added which made automation more painful using Selenium RC.

- Web driver interacts directly to the browser
  - Uses 'native' method for the browser and operating system,
  - Avoids the restrictions of a sandboxed Javascript environment.

- Selenium WebDriver fits in the same role as RC did, and has incorporated the original 1.x bindings.

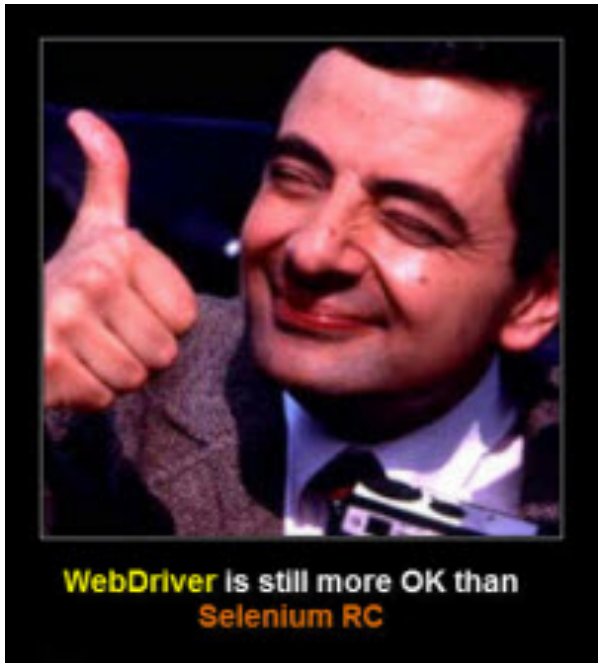
Selenium 1.0 + WebDriver = Selenium 2.0



Web Driver Continued…

- WebDriver is a compact Object Oriented API

- Limitations of Selenium-RC API is overcome by WebDriver.
  - Drives the browser much more effectively
  - Designed in a simpler and more concise programming interface.
  - Selenium-WebDriver makes direct calls to the browser using each browser's native support for automation



Exercise

- Launch Browser
- Navigate to Website
- Close Browser
- Exercise: https://github.com/UHC-SeleniumTraining/Selenium.git - Day1 Folder

Day 1 - Summary

Today we learnt :

- Basics of Automation
- Automation Tools Overview
- Why Selenium is preferred
- Basics of Object Oriented Programming
- Basics of JAVA
- IntelliJ IDE
- Setting up Projects for Se Automation
- Selenium WebDriver