

Java

Day - 1

Java Basics

Java Basics

Java Installation

OOPS Concepts

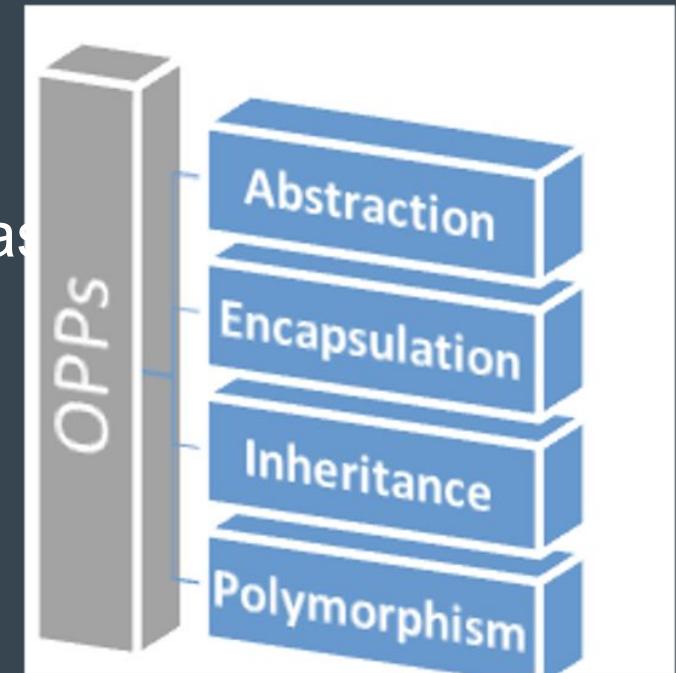
Class and Object

Method and Constructor

Java Access Modifiers

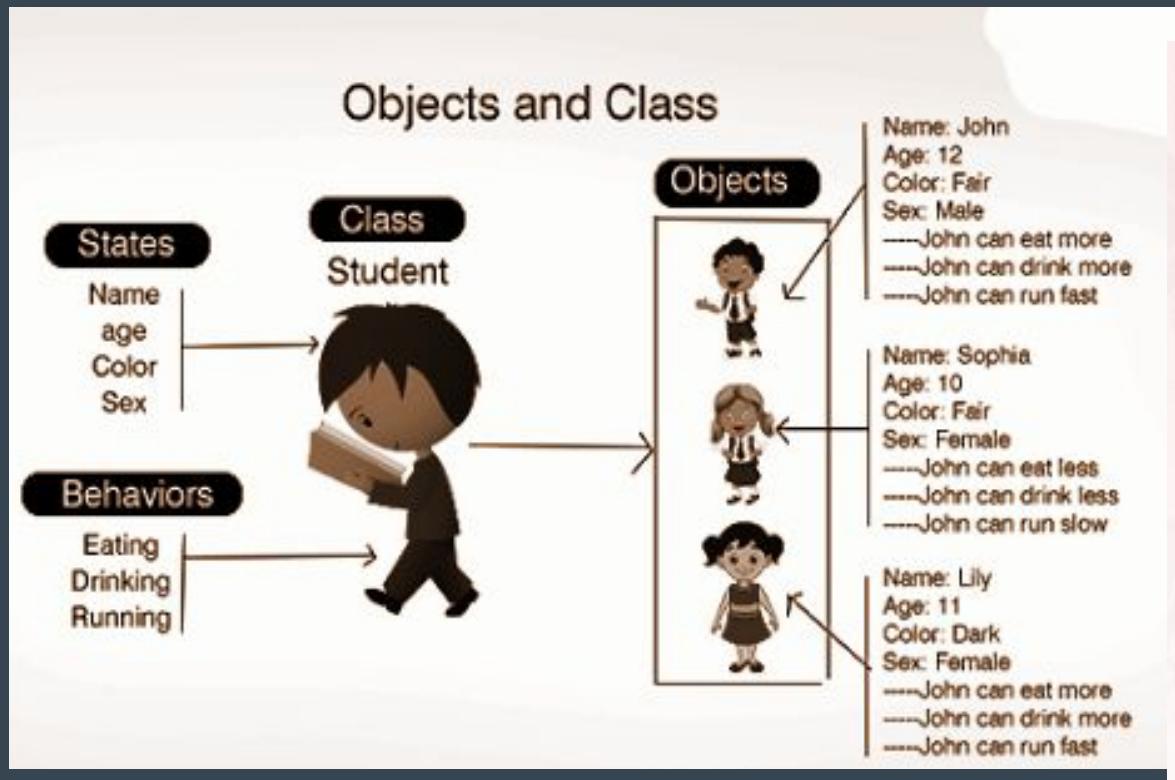
OOPS Concepts

- **Polymorphism:** One task can be performed in many ways
 - In Java it is mainly divided into two types:
 - 1) Static Polymorphism : Method and Operator overloading
 - 2) Runtime Polymorphism : Method Overriding
- **Inheritance:** Object acquires all the properties of parent class
- **Encapsulation:** Hiding data and property in one unit e.g. class
- **Abstraction:** Hiding data and showing functionality
 - Can be achieved using Interfaces and Abstract classes in java
- **Class:** Blueprint for creating projects
- **Object:** have states and behaviours

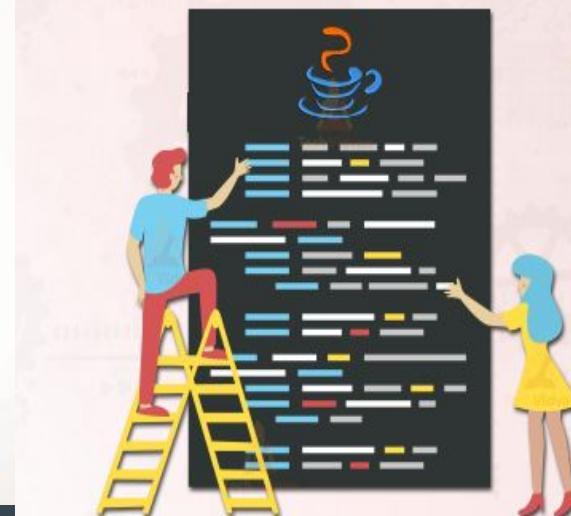


Class and Object

- **Class:** Blueprint for creating projects
- **Objects:** have states and behaviours



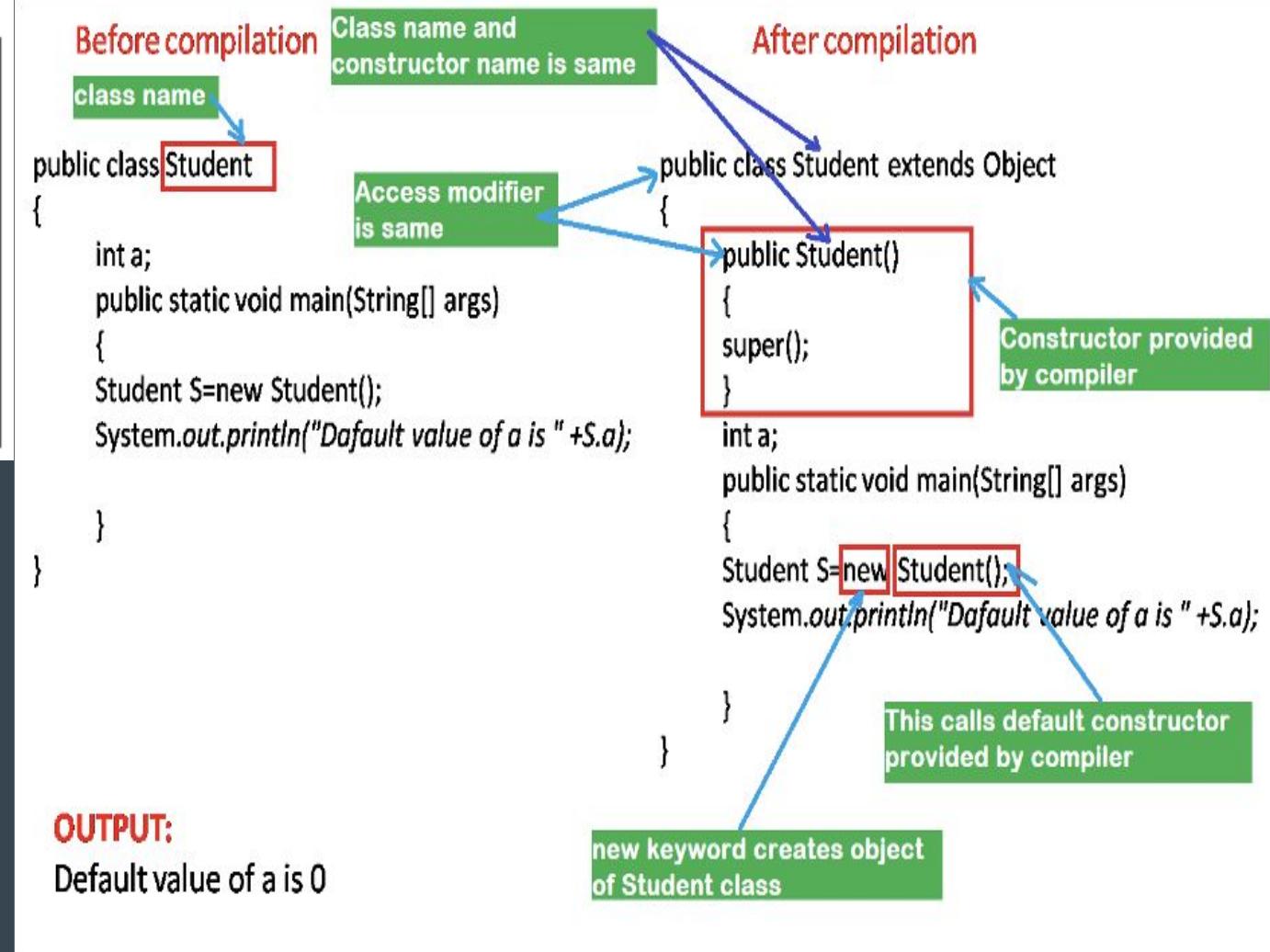
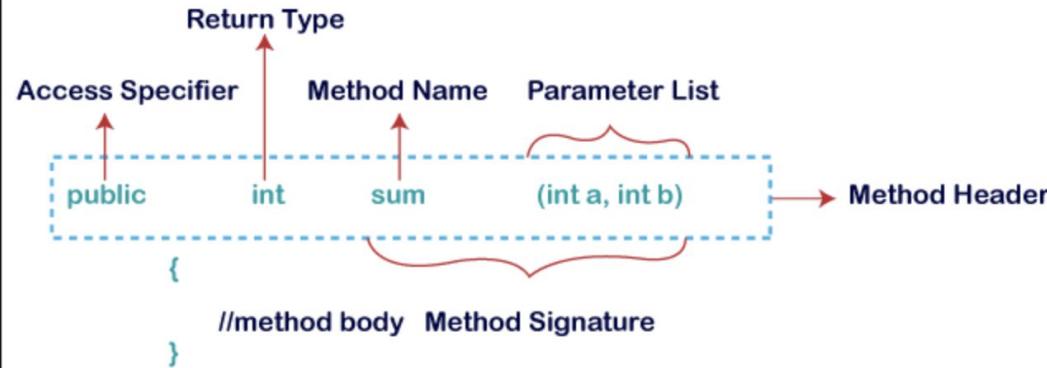
Ways to Create Object in Java



- 01 **By new keyword**
- 02 **By newInstance() method**
- 03 **By clone() method**
- 04 **By deserialization**
- 05 **By factory method**

Method and Constructor

Method Declaration



Java Access Modifiers

	Default	Private	Protected	Public
Within class	Yes	Yes	Yes	Yes
Sub Class within package	Yes	No	Yes	Yes
Sub Class different package	No	No	Yes	Yes
Others	No	No	No	Yes

Handson topics/details

- <https://www.codesdope.com/practice/java-subclass/>

Reference Links

- Reference:
 - [Java-basics](#)
 - [Beginners book](#)
 - Core java [cheat-sheet](#)
- Java installation [guidelines](#)
- OOPs Concepts, class and object
 - Basics [here](#)
 - OOP [with example](#)
- Java Constructor & Methods:
 - [Constructor](#) in java
 - [Method](#) in java
 - Constructor [chaining](#)
 - Method [overding & overloading](#)

Data Type and Operators

Data Types and Operators

Auto-boxing and Unboxing

Wrapper Classes

Control Flow - conditions, Loops

Keywords - final, static etc

Operators

Java provides many types of operators which can be used according to the need. They are classified based on the functionality they provide.

Operators	Associativity	Type
<code>++ --</code>	<code>Right to left</code>	<code>Unary postfix</code>
<code>++ -- + - ! (type)</code>	<code>Right to left</code>	<code>Unary prefix</code>
<code>/ * %</code>	<code>Left to right</code>	<code>Multiplicative</code>
<code>+ -</code>	<code>Left to right</code>	<code>Additive</code>
<code>< <= > >=</code>	<code>Left to right</code>	<code>Relational</code>
<code>== !=</code>	<code>Left to right</code>	<code>Equality</code>
<code>&</code>	<code>Left to right</code>	<code>Boolean Logical AND</code>
<code>^</code>	<code>Left to right</code>	<code>Boolean Logical Exclusive OR</code>
<code> </code>	<code>Left to right</code>	<code>Boolean Logical Inclusive OR</code>
<code>&&</code>	<code>Left to right</code>	<code>Conditional AND</code>
<code> </code>	<code>Left to right</code>	<code>Conditional OR</code>
<code>?:</code>	<code>Right to left</code>	<code>Conditional</code>
<code>= += -= *= /= %=</code>	<code>Right to left</code>	<code>Assignment</code>

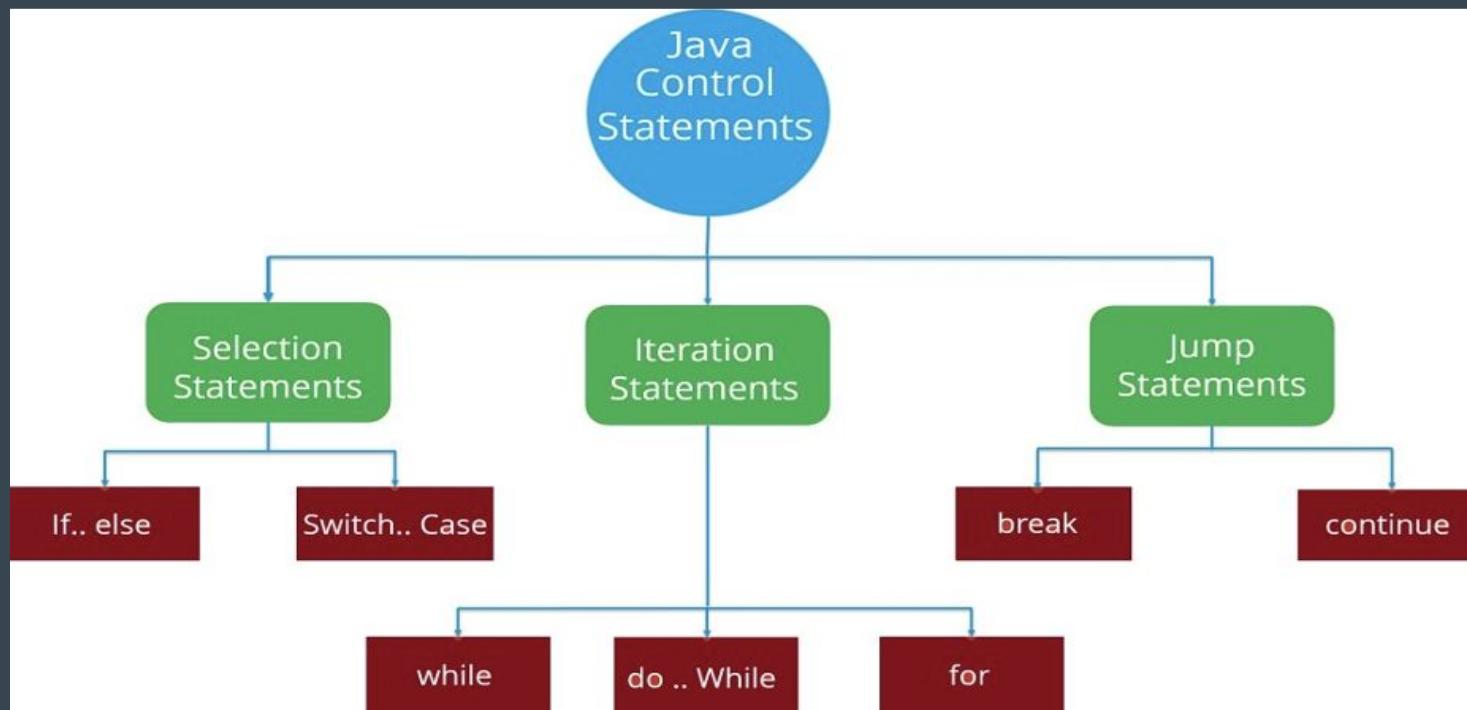
Operators

Java provides many types of operators which can be used according to the need. They are classified based on the functionality they provide.

Operators	Associativity	Type
<code>++ --</code>	<code>Right to left</code>	<code>Unary postfix</code>
<code>++ -- + - ! (type)</code>	<code>Right to left</code>	<code>Unary prefix</code>
<code>/ * %</code>	<code>Left to right</code>	<code>Multiplicative</code>
<code>+ -</code>	<code>Left to right</code>	<code>Additive</code>
<code>< <= > >=</code>	<code>Left to right</code>	<code>Relational</code>
<code>== !=</code>	<code>Left to right</code>	<code>Equality</code>
<code>&</code>	<code>Left to right</code>	<code>Boolean Logical AND</code>
<code>^</code>	<code>Left to right</code>	<code>Boolean Logical Exclusive OR</code>
<code> </code>	<code>Left to right</code>	<code>Boolean Logical Inclusive OR</code>
<code>&&</code>	<code>Left to right</code>	<code>Conditional AND</code>
<code> </code>	<code>Left to right</code>	<code>Conditional OR</code>
<code>?:</code>	<code>Right to left</code>	<code>Conditional</code>
<code>= += -= *= /= %=</code>	<code>Right to left</code>	<code>Assignment</code>

Control Flow - conditions, Loops

Generally the statements inside your java code are executed from top to bottom, in the order that they appear. Control flow statements, change or break the flow of execution by implementing decision making, looping, and branching your program to execute particular blocks of code based on the conditions.

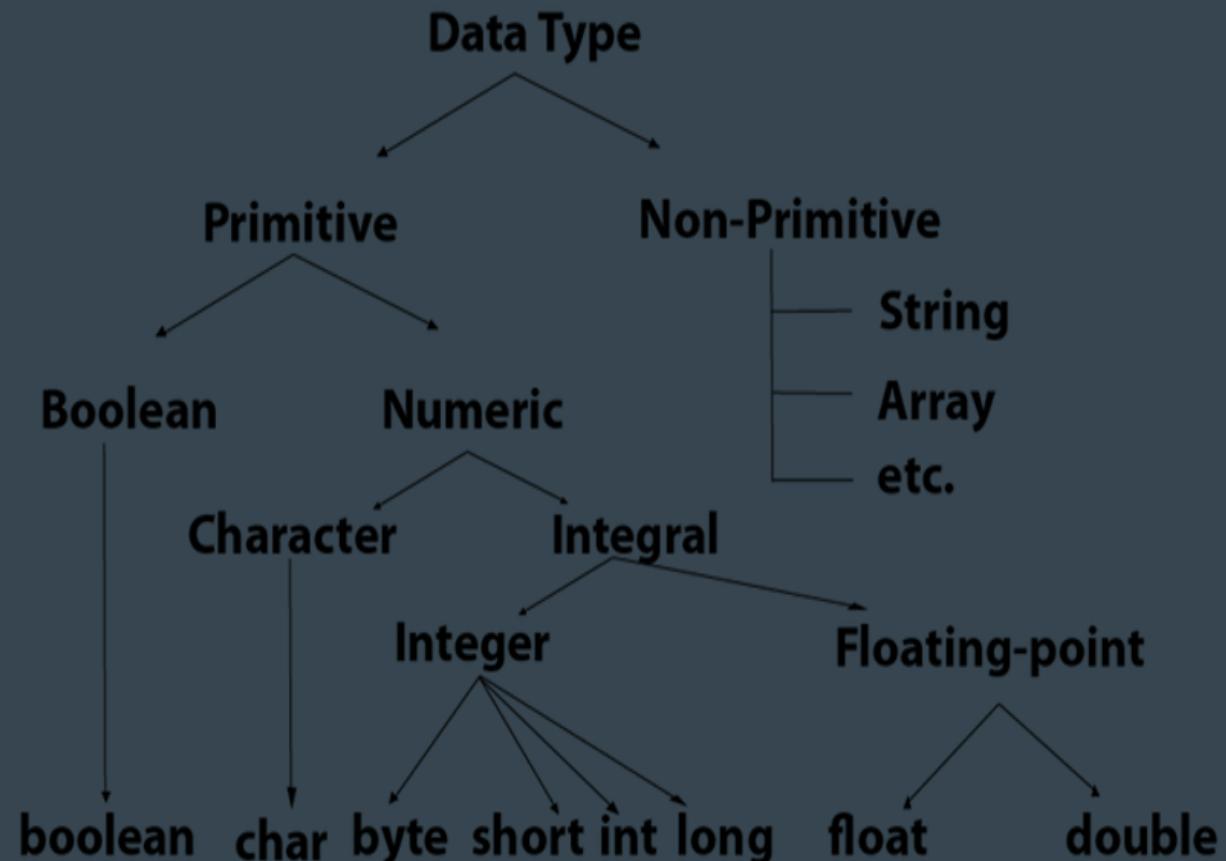


Data Types

Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java

Primitive data types: The primitive data types include boolean, char, byte, short, int, long, float and double.

Non-primitive data types: The non-primitive data types include Classes, Interfaces, and Arrays.



Auto-boxing and Unboxing and Wrapper Class

A Wrapper class is a class whose object wraps or contains primitive data types. When we create an object to a wrapper class, it contains a field and in this field, we can store primitive data types. In other words, we can wrap a primitive value into a wrapper class object.

Need of Wrapper Classes: Change the value in Method, Serialization, Synchronization, java.util package, Collection Framework

Autoboxing: Automatic conversion of primitive types to the object of their corresponding wrapper classes is known as autoboxing. For example – conversion of int to Integer, long to Long, double to Double etc.

Unboxing: It is just the reverse process of autoboxing. Automatically converting an object of a wrapper class to its corresponding primitive type is known as unboxing. For example – conversion of Integer to int, Long to long, Double to double, etc.

Auto-boxing and Unboxing and Wrapper Class

```
//Java program to convert primitive into objects  
//Autoboxing example of int to Integer  
public class WrapperExample1{  
public static void main(String args[]){  
    //Converting int into Integer  
    int a=20;  
    Integer i=Integer.valueOf(a);//converting int into Integer explicitly  
    Integer j=a;//autoboxing, now compiler will write Integer.valueOf(a) internally  
  
    System.out.println(a+" "+i+" "+j);  
}}
```

Output : 20 20 20

Auto-boxing and Unboxing and Wrapper Class

```
//Java program to convert object into primitives  
//Unboxing example of Integer to int  
public class WrapperExample2{  
public static void main(String args[]){  
    //Converting Integer to int  
    Integer a=new Integer(3);  
    int i=a.intValue();//converting Integer to int explicitly  
    int j=a;//unboxing, now compiler will write a.intValue() internally  
  
    System.out.println(a+" "+i+" "+j);  
}}
```

Output: 3 3 3

Keywords - Final and Static

Java has a set of keywords that are reserved words that cannot be used as variables, methods, classes, or any other identifiers:

Ex: final, static, while, try, void.

The Final keyword in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

Variable
Method
Class



Keywords - Final and Static

The static keyword in Java is used for memory management mainly. We can apply static keyword with variables, methods, blocks and nested classes.

Static method: A static method belongs to the class rather than the object of a class and can be invoked without the need for creating an instance of a class. A static method can access static data member and can change the value of it.

Static block: Static block is used for initializing the static variables. This block gets executed when the class is loaded in the memory. A class can have multiple Static blocks, which will execute in the same sequence in which they have been written into the program.

Static variable: When you declare a variable as static, then a single copy of the variable is created and divided among all objects at the class level. Static variables are, essentially, global variables. Basically, all the instances of the class share the same static variable. Static variables can be created at class-level only.

Static class: A class can be made static only if it is a nested class. Nested static class doesn't need a reference of Outer class. In this case, a static class cannot access non-static members of the Outer class

Handson topics/details

- <https://www.codesdope.com/practice/java-know-data-types/>
- <https://www.codesdope.com/practice/java-operators/>
- <https://www.codesdope.com/practice/java-decide-if-or-else/>
- <https://www.codesdope.com/practice/java-loop-loop-loop/>

Reference Links

- https://www.w3schools.com/java/java_data_types.asp
- <https://www.geeksforgeeks.org/operators-in-java/>
- <https://www.geeksforgeeks.org/autoboxing-unboxing-java/>
- <https://www.javatpoint.com/static-keyword-in-java>
- <https://www.geeksforgeeks.org/final-keyword-java/>
- <https://www.w3adda.com/java-tutorial/java-control-flow-statements>

Day - 3

Abstract Class and Interface

Abstract Class and Interface

Abstract Class

Interface

Interface vs Abstract Class

New features introduced in Interface

Abstract Class

A class which is declared as **abstract** is known as an abstract class. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.

An abstract class must be declared with an **abstract** keyword.

It can have abstract and non-abstract methods.

It cannot be instantiated.

It can have constructors and static methods also.

It can have final methods which will force the subclass not to change the body of the method.

```
abstract class Base {  
    abstract void fun();  
}  
  
// Class 2  
class Derived extends Base {  
    void fun()  
    {  
        System.out.println("Derived fun() called");  
    }  
}
```

Interface

An interface in Java is a blueprint of a class. All fields are public, static, and final and the abstract methods are public by default.

The interface in Java is a mechanism to achieve abstraction and multiple inheritance in Java.

If a class implements an interface and does not provide method bodies for all abstract functions specified in the interface, then the class must be declared abstract

From Java 8 we can have default and static methods in Interfaces. Before this we can only have abstract methods i.e. methods with their definition and no implementation.

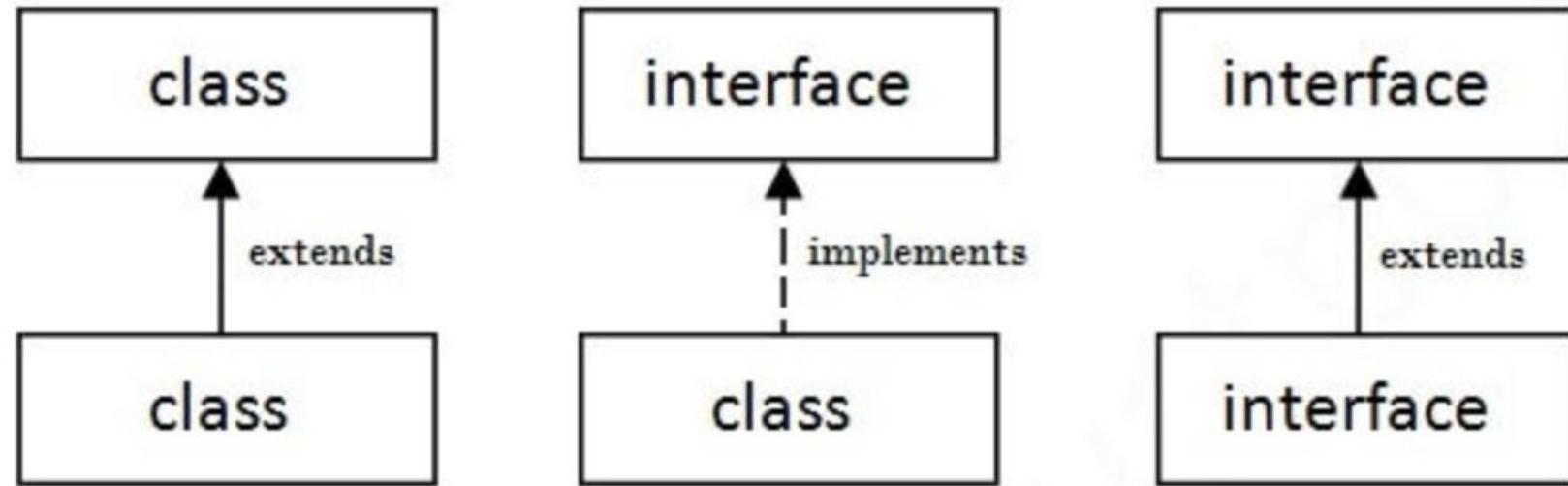
```
interface In1
{
    final int a = 10;
    // public and abstract
    void display();
}

// A class that implements the interface.
class TestClass implements In1
{
    // Implementing the capabilities of
    // interface.
    public void display()
    {
        System.out.println("Geek");
    }
}
```

Interface vs Abstract Class

Abstract class	Interface
1) Abstract class can have abstract and non-abstract methods.	Interface can have only abstract methods. Since Java 8, it can have default and static methods also.
2) Abstract class doesn't support multiple inheritance.	Interface supports multiple inheritance.
3) Abstract class can have final, non-final, static and non-static variables.	Interface has only static and final variables.
4) Abstract class can provide the implementation of interface.	Interface can't provide the implementation of abstract class.
5) The abstract keyword is used to declare abstract class.	The interface keyword is used to declare interface.
6) An abstract class can extend another Java class and implement multiple Java interfaces.	An interface can extend another Java interface only.
7) An abstract class can be extended using keyword "extends".	An interface can be implemented using keyword "implements".
8) A Java abstract class can have class members like private, protected, etc.	Members of a Java interface are public by default.
9) Example: <pre>public abstract class Shape{ public abstract void draw(); }</pre>	Example: <pre>public interface Drawable{ void draw(); }</pre>

Extend vs Implement



New features introduced in Interface

Since Java 8, we can have default and static methods in an interface.
Since Java 9, we can have private methods in an interface

Handson topics/details

<https://www.codesdope.com/practice/java-abstract-class/>

<https://www.tutorialspoint.com/when-to-use-an-abstract-class-and-when-to-use-an-interface-in-java>

Reference Links

- <https://www.geeksforgeeks.org/interfaces-in-java/>
- <https://www.geeksforgeeks.org/abstract-classes-in-java/>
- <https://www.geeksforgeeks.org/difference-between-abstract-class-and-interface-in-java/>
- <https://www.tutorialspoint.com/when-to-use-an-abstract-class-and-when-to-use-an-interface-in-java>
- <https://www.javatpoint.com/abstract-class-in-java>
- <https://www.javatpoint.com/interface-in-java>

Day - 4

String Manipulations

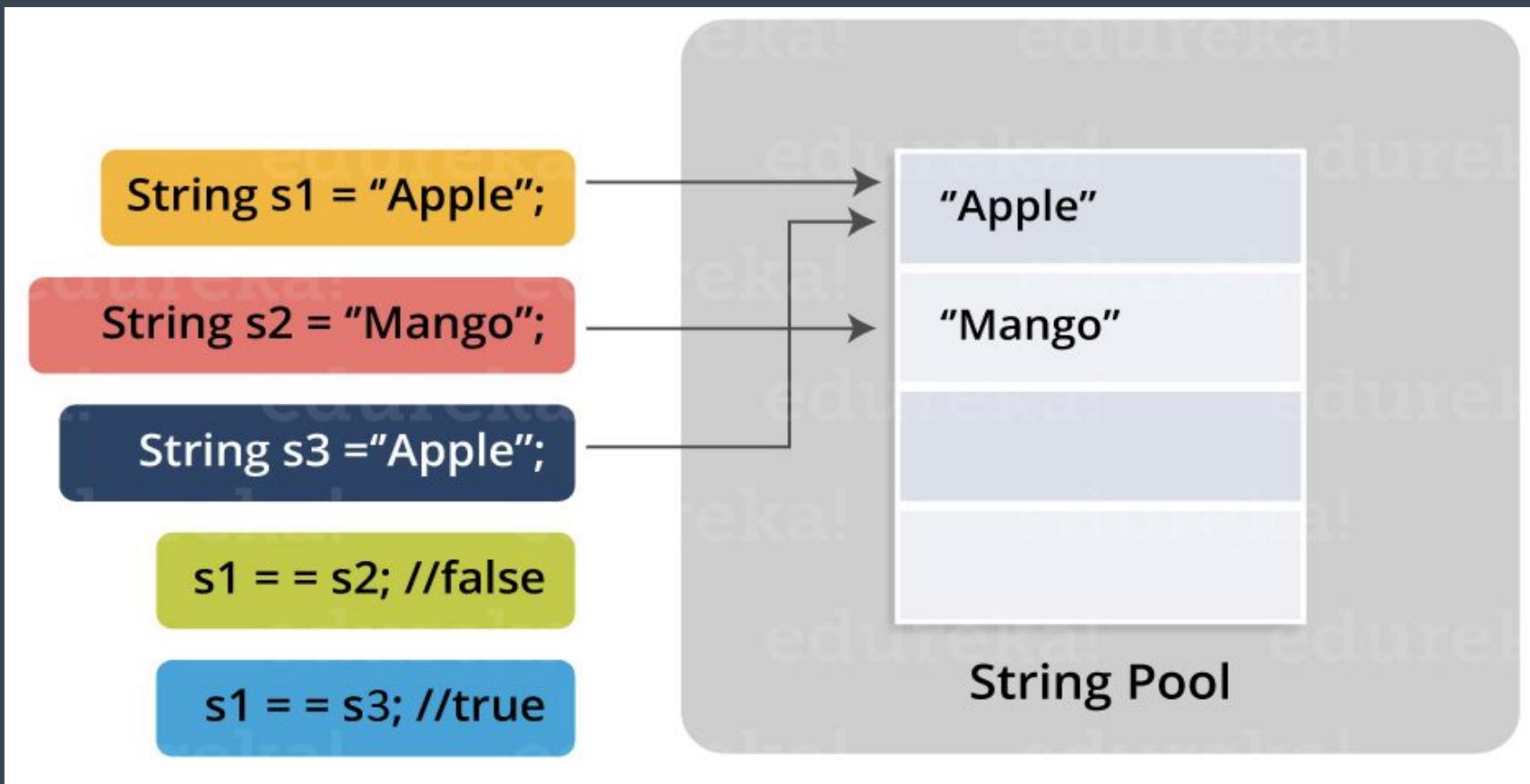
String Manipulations

Understanding Java String Pool
String Operations
StringBuffer and StringBuilder

Understanding Java String Pool

Immutable

Why java allows us to change the value, if string is immutable ?



String Operations

charAt(int index) : It returns char value for the particular index

length() : It returns string length

substring(int beginIndex) : It returns substring for given begin index.

isEmpty() : It checks if string is empty.

equals(Object another) : It checks the equality of string with the given object.

contains(CharSequence s) : It returns true or false after matching the sequence of char value.

toUpperCase() : It returns a string in uppercase.

toLowerCase() : It returns a string in lowercase.

indexOf(int ch) : It returns the specified char value index.

String Builder & String Buffer

StringBuffer

vs

StringBuilder

StringBuffer is synchronized i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously.

StringBuilder is non-synchronized i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously.

StringBuffer is less efficient than StringBuilder.

StringBuilder is more efficient than StringBuffer.

Handson topics/details

- <https://www.codesdope.com/practice/java-characters-and-string/>
- <https://www.java67.com/2018/04/21-string-programming-and-coding-interview-questions-answers.html>

Reference Links

- <https://study.com/academy/lesson/java-string-constant-pool-concept-mechanism.html>
- <https://www.javatpoint.com/java-string>
- https://www.youtube.com/watch?v=UKG1xgh_tbQ
- <https://www.youtube.com/watch?v=47-kDPWBBM>
- <https://www.youtube.com/watch?v=ZhQxRyyqil8>

Day - 5

Arrays, Map and Collections

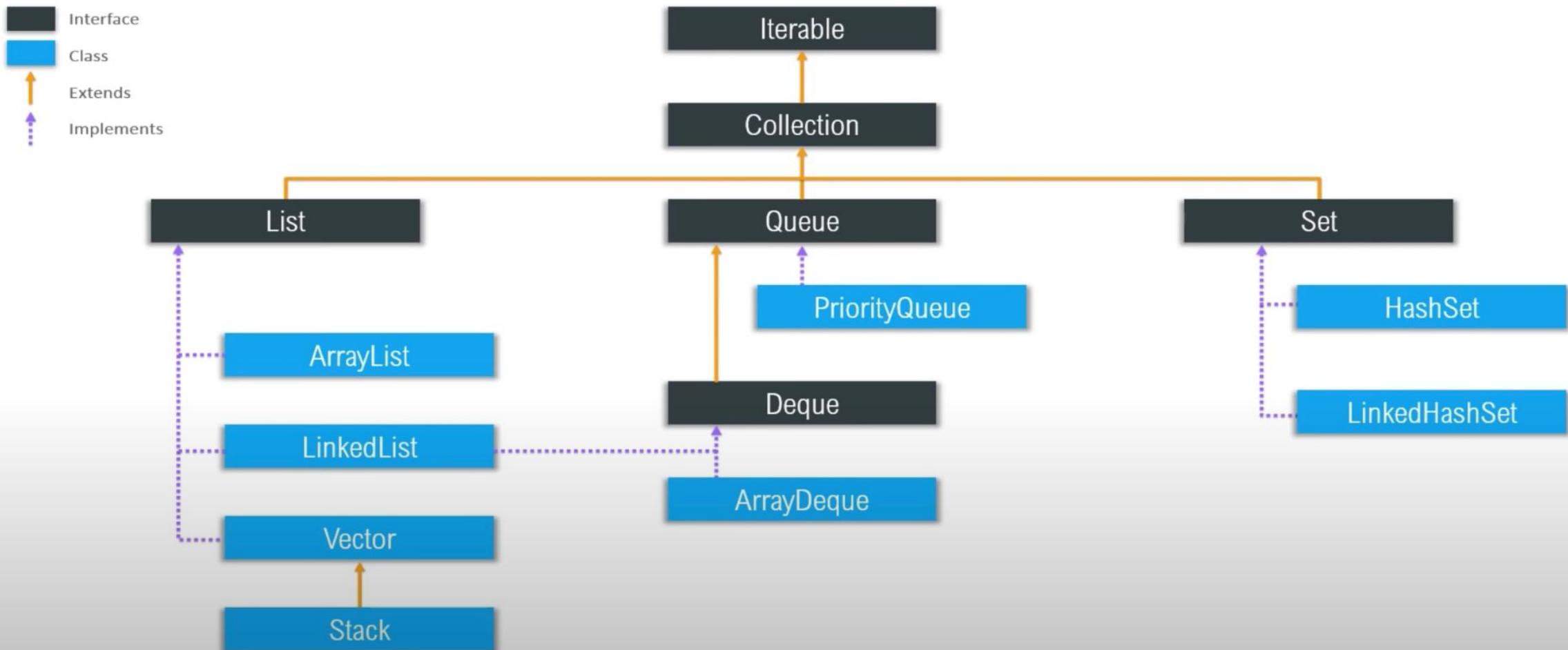
Arrays, Map and Collections

Arrays, Map and Collections
Java Collections Framework
Initialising an Array in Java

Arrays, Map and Collections

Arrays, Map and Collections
Java Collections Framework
Initialising an Array in Java

Collection Framework Hierarchy



Initialising an Array in Java

```
int a[]={};//declaration and instantiation
a[0]=10;//initialization
a[1]=20;
a[2]=70;
a[3]=40;
a[4]=50;
//traversing array
for(int i=0;i<a.length;i++)//length is the property of array
    System.out.println(a[i]);
```

Arrays, Map and Collections

Arrays : An array is a container object that holds a fixed number of values of a single type.

Map & Collection : The Collection in Java is a framework that provides an architecture to store and manipulate the group of objects. Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.

A map contains values on the basis of key, i.e. key and value pair. Each key and value pair is known as an entry. A Map contains unique keys

Sr. No.	Key	Arrays	Collection
1	Size	Arrays are fixed in size i.e once the array with the specific size is declared then we can't alter its size afterward.	The collection is dynamic in size i.e based on requirement size could be get altered even after its declaration.
2	Memory Consumption	Arrays due to fast execution consumes more memory and has better performance.	Collections, on the other hand, consume less memory but also have low performance as compared to Arrays.
3	Data type	Arrays can hold the only the same type of data in its collection i.e only homogeneous data types elements are allowed in case of arrays.	Collection, on the other hand, can hold both homogeneous and heterogeneous elements.
4	Primitives storage	Arrays can hold both object and primitive type data.	On the other hand, collection can hold only object types but not the primitive type of data.
5	Performance	Arrays due to its storage and internal implementation better in performance.	Collection on the other hand with respect to performance is not recommended to use.

List

Set

Map

The list interface allows duplicate elements.

The list maintains insertion order.

We can add any number of null values.

List implementation classes are [Array](#), [List](#), [LinkedList](#).

The list provides get() method to get the element at a specified index.

If you need to access the elements frequently by using the index then we can use the list.

To traverse the list elements by using ListIterator.

Set does not allow duplicate elements.

Set do not maintain any insertion order.

But in set almost only one null value.

Set implementation classes are [HashSet](#), [LinkedHashSet](#), and [TreeSet](#).

Set does not provide get method to get the elements at a specified index

If you want to create a collection of unique elements then we can use set

Iterator can be used traverse the set elements

The map does not allow duplicate elements

The map also does not maintain any insertion order.

The map allows a single null key at most and any number of null values.

Map implementation classes are [HashMap](#), [HashTable](#), [TreeMap](#), [ConcurrentHashMap](#), and [LinkedHashMap](#).

The map does not provide get method to get the elements at a specified index

If you want to store the data in the form of key/value pair then we can use the map.

Through keyset, value, and entry set.

Handson topics/details

<https://www.codesdope.com/practice/java-array/>

<https://www.codesdope.com/practice/java-array-of-objects//practice/java-array-of-objects/>

<https://www.w3resource.com/java-exercises/collection/index.php>

Reference Links

- https://www.w3schools.com/java/java_arrays.asp
- <https://www.vogella.com/tutorials/JavaCollections/article.html>

Generics

```
// A Simple Java program to show working of user defined
```

```
// Generic classes
```

```
// We use < > to specify Parameter type
```

```
class Test<T>
```

```
{
```

```
    // An object of type T is declared
```

```
    T obj;
```

```
    Test(T obj) { this.obj = obj; } // constructor
```

```
    public T getObject() { return this.obj; }
```

```
}
```

```
// Driver class to test above
```

```
class Main
```

```
{
```

```
    public static void main (String[] args)
```

```
{
```

```
    // instance of Integer type
```

47

```
    Test <Integer> iObj = new Test<Integer>(15);
```

```
    System.out.println(iObj.getObject());
```

15

Hello Java Generics

Day - 6

Generics and Enums

Generics and Enums

Generics

The Java Generics programming is introduced in J2SE 5 to deal with type-safe objects. It makes the code stable by detecting the bugs at compile time.

Before generics, we can store any type of objects in the collection, i.e., non-generic. Now generics force the java programmer to store a specific type of objects.

Generics mean parameterized types. The idea is to allow type (Integer, String, ... etc, and user-defined types) to be a parameter to methods, classes, and interfaces. Using Generics, it is possible to create classes that work with different data types.

An entity such as class, interface, or method that operates on a parameterized type is called a generic entity.

HashSet, ArrayList, HashMap, etc use generics very well

```
// To create an instance of generic class  
BaseType <Type> obj = new BaseType <Type>()
```

Note: In Parameter type we can not use primitives like 'int', 'char' or 'double'.

```
// A Simple Java program to show multiple // type parameters in Java Generics
```

```
// We use < > to specify Parameter type
```

```
class Test<T, U>
```

```
{
```

```
    T obj1; // An object of type T
```

```
    U obj2; // An object of type U
```

```
    // constructor
```

```
    Test(T obj1, U obj2)
```

```
{
```

```
    this.obj1 = obj1;
```

```
    this.obj2 = obj2;
```

```
}
```

```
    // To print objects of T and U
```

```
50 public void print()
```

```
{
```

Output:

GfG

15

```
// A Simple Java program to show working of user defined Generic functions
```

```
class Test

{
    // A Generic method example

    static <T> void genericDisplay (T element)

    {
        System.out.println(element.getClass().getName() +
                           " = " + element);
    }

    // Driver method

    public static void main(String[] args)
    {
        // Calling generic method with Integer argument

        genericDisplay(11);

        // Calling generic method with String argument

        genericDisplay("Java Generic Functions");
    }
}
```

```
// A Simple Java program to show working of user-defined Generic classes
```

```
// We use < > to specify Parameter type

class Test<T>
{

    // An object of type T is declared

    T obj;

    Test(T obj) { this.obj = obj; } // constructor

    public T getObject() { return this.obj; }

}
```

```
// Driver class to test above
```

```
class Main
{

    public static void main (String[] args)

    {

        // instance of Integer type

        Test <Integer> iObj = new Test<Integer>(15);

        System.out.println(iObj.getObject());
    }
}
```

Even though iObj and sObj are of type Test, they are the references to different types because their type parameters differ. Generics add type safety through this and prevent errors.

```
error:
incompatible types:
Test cannot be converted to Test
```

Advantages of Generics

1. Code Reuse:

We can write a method/class/interface once and use it for any type we want.

2. Type Safety:

Generics make errors to appear compile time than at run time (It's always better to know problems in your code at compile time rather than making your code fail at run time). Suppose you want to create an ArrayList that store name of students and if by mistake programmer adds an integer object instead of a string, the compiler allows it. But, when we retrieve this data from ArrayList, it causes problems at runtime.

3. Individual Type Casting is not needed:

If we do not use generics, then, in the above example every time we retrieve data from ArrayList, we have to typecast it. Typecasting at every retrieval operation is a big headache. If we already know that our list only holds string data then we need not typecast it every time.

4. Generics promotes code reusability.

5. Implementing generic algorithms:

By using generics, we can implement algorithms that work on different types of objects and at the same, they are type safe too.

53

```
// A Simple Java program to demonstrate that NOT using generics can cause run time exceptions
import java.util.*;

class Test
{
    public static void main(String[] args)
    {
        // Creatinga an ArrayList without any type specified
        ArrayList al = new ArrayList();

        al.add("Sachin");
        al.add("Rahul");
        al.add(10); // Compiler allows this

        String s1 = (String)al.get(0);
        String s2 = (String)al.get(1);

        // Causes Runtime Exception
        String s3 = (String)al.get(2);
    }
}

Exception in thread "main" java.lang.ClassCastException:
java.lang.Integer cannot be cast to java.lang.String
at Test.main(Test.java:19)
```

Wildcard in Java Generics

- The ? (question mark) symbol represents the wildcard element. It means any type. If we write <? extends Number>, it means any child class of Number, e.g., Integer, Float, and double. Now we can call the method of Number class through any child class object.
- We can use a wildcard as a type of a parameter, field, return type, or local variable. However, it is not allowed to use a wildcard as a type argument for a generic method invocation, a generic class instance creation, or a supertype.

```
1. import java.util.*;  
2. abstract class Shape{  
3.     abstract void draw();  
4. }  
5. class Rectangle extends Shape{  
6.     void draw(){System.out.println("drawing rectangle");}  
7. }  
8. class Circle extends Shape{  
9.     void draw(){System.out.println("drawing circle");}  
10. }  
11. class GenericTest{  
12.     //creating a method that accepts only child class of Shape  
13.     public static void drawShapes(List<? extends Shape> lists){  
14.         for(Shape s:lists){  
15.             s.draw();//calling method of Shape class by child class instance  
16.         } }  
17.     public static void main(String args[]){  
18.         List<Rectangle> list1=new ArrayList<Rectangle>();  
19.         list1.add(new Rectangle());  
20.         List<Circle> list2=new ArrayList<Circle>();  
21.         list2.add(new Circle());  
22.         list2.add(new Circle());  
23.         drawShapes(list1);  
24.         drawShapes(list2);  
25.     } }  
26. }
```

Unbounded Wildcards Example

- The unbounded wildcard type represents the list of an unknown type such as List<?>. This approach can be useful in the following scenarios: -
 - When the given method is implemented by using the functionality provided in the Object class.
 - When the generic class contains the methods that don't depend on the type parameter.

```
1. import java.util.Arrays;
2. import java.util.List;
3.
4. public class UnboundedWildcard {
5.     public static void display(List<?> list)
6.     {
7.         for(Object o:list)
8.         {
9.             System.out.println(o);
10.        }
11.    }
12.
13.    public static void main(String[] args) {
14.
15.        List<Integer> l1=Arrays.asList(1,2,3);
16.        System.out.println("displaying the Integer values");
17.        display(l1);
18.        List<String> l2=Arrays.asList("One","Two","Three");
19.        System.out.println("displaying the String values");
20.        display(l2);
21.    }
22. }
```

displaying the Integer values

1
2
3

displaying the String values

One
Two
Three

Lower Bounded Wildcards

- The purpose of lower bounded wildcards is to restrict the unknown type to be a specific type or a supertype of that type. It is used by declaring wildcard character ("?") followed by the super keyword, followed by its lower bound.
- List<? super Integer>
- ? is a wildcard character.
- super, is a keyword.
- Integer, is a wrapper class.
- Suppose, we want to write the method for the list of Integer and its supertype (like Number, Object). Using List<? super Integer> is suitable for a list of type Integer or any of its superclasses whereas List<Integer> works with the list of type Integer only. So, List<? super Integer> is less restrictive than List<Integer>.

```
1. public class LowerBoundWildcard {  
2.     public static void addNumbers(List<? super Integer> list) {  
3.         for(Object n:list)  
4.             {  
5.                 System.out.println(n);  
6.             }  
7.     }  
8.     public static void main(String[] args) {  
9.         List<Integer> l1=Arrays.asList(1,2,3);  
10.        System.out.println("displaying the Integer values");  
11.        addNumbers(l1);  
12.        List<Number> l2=Arrays.asList(1.0,2.0,3.0);  
13.        System.out.println("displaying the Number values");  
14.        addNumbers(l2);  
15.    }  
16. }
```

displaying the Integer values

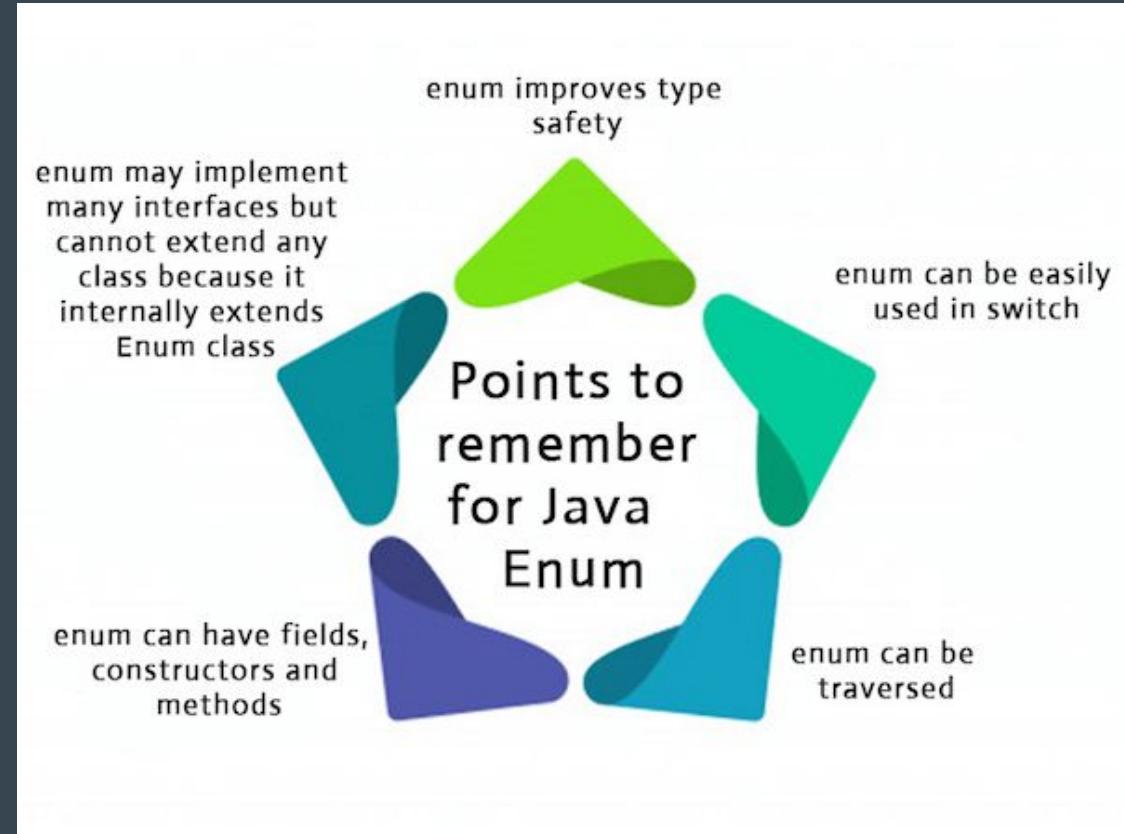
1
2
3

displaying the Number values

1.0
2.0
3.0

Enums

- **Enums**
 - The **Enum** in Java is a data type which contains a fixed set of constants.
 - Java Enums can be thought of as classes which have a fixed set of constants (a variable that does not change).
 - The Java enum constants are static and final implicitly.
 - It is available since JDK 1.5.



Enums Examples

```
1. class EnumExample1{  
2.     //defining the enum inside the class  
3.     public enum Season { WINTER, SPRING, SUMMER, FALL }  
4.     //main method  
5.     public static void main(String[] args) {  
6.         //traversing the enum  
7.         for (Season s : Season.values())  
8.             System.out.println(s);  
9.     }  
}
```

WINTER

SPRING

SUMMER

FALL

```
class EnumExample1{  
  
    //defining enum within class  
  
    public enum Season { WINTER, SPRING, SUMMER, FALL }  
  
    //creating the main method  
  
    public static void main(String[] args) {  
  
        //printing all enum  
  
        for (Season s : Season.values()){  
  
            System.out.println(s);  
  
        }  
  
        System.out.println("Value of WINTER is: "+Season.valueOf("WINTER"));  
  
        System.out.println("Index of WINTER is: "+Season.valueOf("WINTER").ordinal());  
  
        System.out.println("Index of SUMMER is: "+Season.valueOf("SUMMER").ordinal());  
  
    }  
  
    WINTER  
  
    SPRING  
  
    SUMMER  
  
    FALL  
  
    Value of WINTER is: WINTER  
  
    Index of WINTER is: 0  
  
    Index of SUMMER is: 2
```

Enums

Java compiler internally adds values(), valueOf() and ordinal() methods within the enum at compile time. It internally creates a static and final class for the enum.

- purpose of the values() method in the enum

The Java compiler internally adds the values() method when it creates an enum. The values() method returns an array containing all the values of the enum.

- purpose of the valueOf() method in the enum

The Java compiler internally adds the valueOf() method when it creates an enum. The valueOf() method returns the value of given constant enum.

- purpose of the ordinal() method in the enum

The Java compiler internally adds the ordinal() method when it creates an enum. The ordinal() method returns the index of the enum value.

The enum can be defined within or outside the class because it is similar to a class. The semicolon (;) at the end of the enum constants are optional. For example:

```
enum Season { WINTER, SPRING, SUMMER, FALL }
```

Or,

```
enum Season { WINTER, SPRING, SUMMER, FALL; }
```

Both the definitions of Java enum are the same.

```
enum Season{  
  
    WINTER(10), SUMMER(20);  
  
    private int value;  
  
    Season(int value){  
  
        this.value=value;  
  
    }  
}
```

Constructor of enum type is private.
If you don't declare private compiler
internally creates private
constructor. We cannot create
Enum using new Keyword.

```
final class Season extends Enum
```

```
{  
  
    public static Season[] values()  
  
    {  
  
        return (Season[])$VALUES.clone();  
  
    }
```

```
    public static Season valueOf(String s)  
  
    {  
  
        return (Season)Enum.valueOf(Season, s);  
  
    }
```

```
    private Season(String s, int i, int j)  
  
    {  
  
        super(s, i);  
  
        value = j;  
  
    }
```

```
    public static final Season WINTER;
```

Handson topics/details

- <https://courses.cs.washington.edu/courses/cse341/12au/java/mini1.pdf>

Reference Links

- <https://javarevisited.blogspot.com/2011/09/generics-java-example-tutorial.html>
- <https://docs.oracle.com/javase/tutorial/java/generics/types.html>
- <https://www.geeksforgeeks.org/generics-in-java/>
- <https://www.javatpoint.com/enum-in-java>

Day - 7

Exception Handling

Exception Handling

- Overview of Exception Handling
- Types of Exception
- Exceptions Hierarchy
- Building blocks of Exception Handling
- MultiCatch
- Finally Block
- Exception Handling Example

Overview of Exception Handling

- Exception are anomalies or unusual conditions that a java program may encounter during execution.
- Flow of exception Handling should be :
 - If there is any possibility of such unusual conditions then add throws declaration on that method
 - Found an anomaly or unusual condition then throw an exception
 - For exception handling, catch exception and complete processing then throw exception with or without manipulation again if its required

Types of Exception

- Checked Exception
 - These are the exceptions that are checked at compile time. If some code within a method throws a checked exception, then the method must either handle the exception or it must specify the exception using the throws keyword.
 - Ex: IOException, InterruptedException
 - Inherited from java.lang.Exception.java
- Unchecked Exception
 - These are the exceptions that are not checked at compile time.
 - It is not forced by the compiler to either handle or specify the exception.
 - Ex: ArithmeticException, NullPointerException

In Java, exceptions under Error and RuntimeException classes are unchecked exceptions, everything else under throwable is checked.

MultiCatch

```
Line1> class MultiCatch {  
Line2>     void myMethod(Connection con, String fileName) {  
Line3>         try {  
Line4>             File file = new File(fileName);  
Line5>             FileInputStream fin = new FileInputStream(file); ←  
Line6>             Statement stmt = con.createStatement(); ←  
Line7>         }  
Line8>         catch (FileNotFoundException | SQLException e) {  
Line9>             System.out.println(e.toString()); ←  
Line10>        }  
Line11>    }  
Line12>}
```

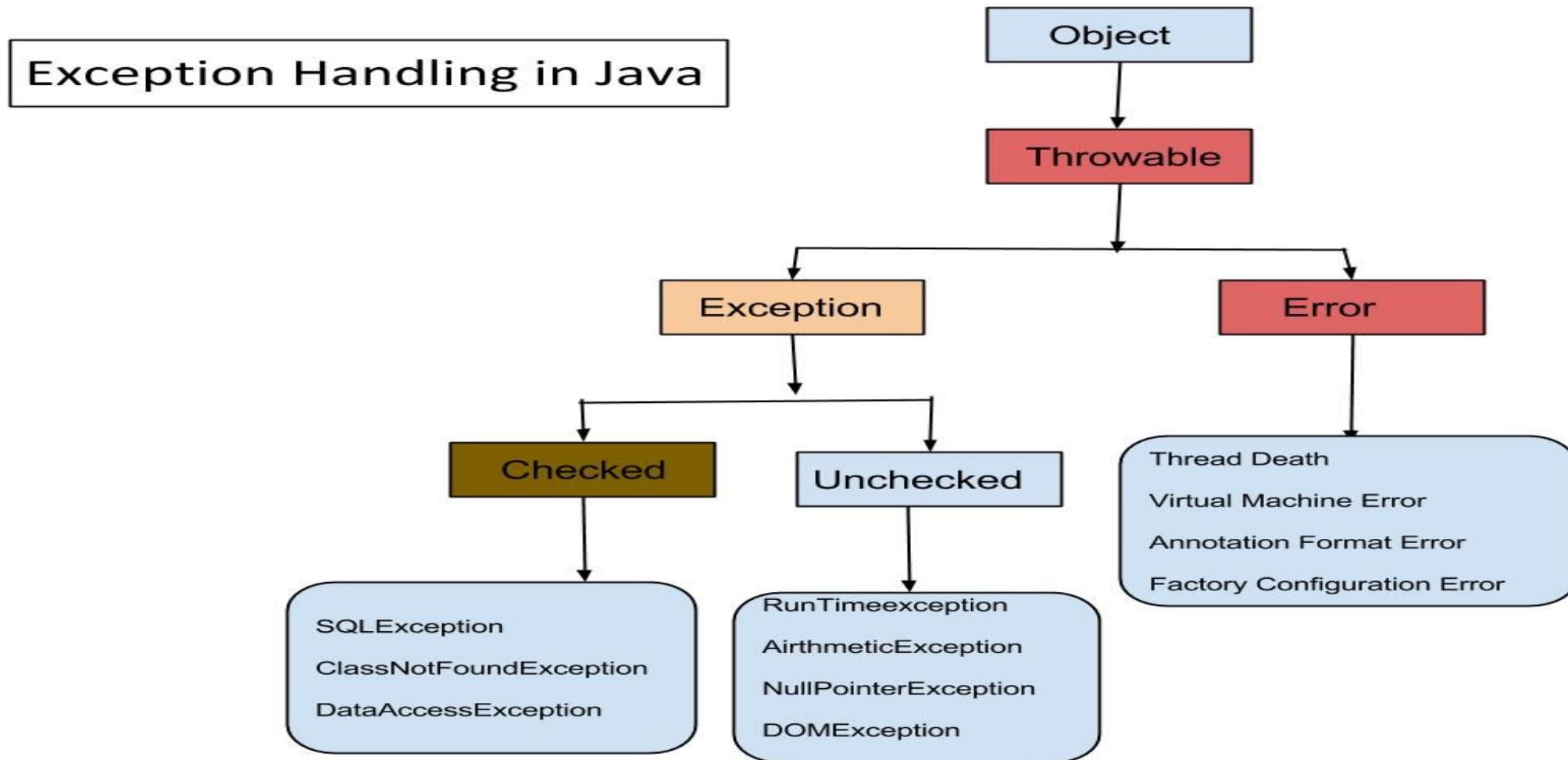
Might throw
FileNotFoundException

Might throw
SQLException

Executes if line 5 throws
FileNotFoundException or line
6 throws SQLException or any
of their subclasses.

finally block can follow multi-catch block

Exceptions Hierarchy



Building blocks of Exception Handling

try - Block where execution happens

catch - Block where exceptions are handled

throw - Used to throw exception in execution flow

throws - Used to indicate which exceptions can be encountered

*void somemethod() **throws** Exception*

try {

... normal program code

throw new Exception();

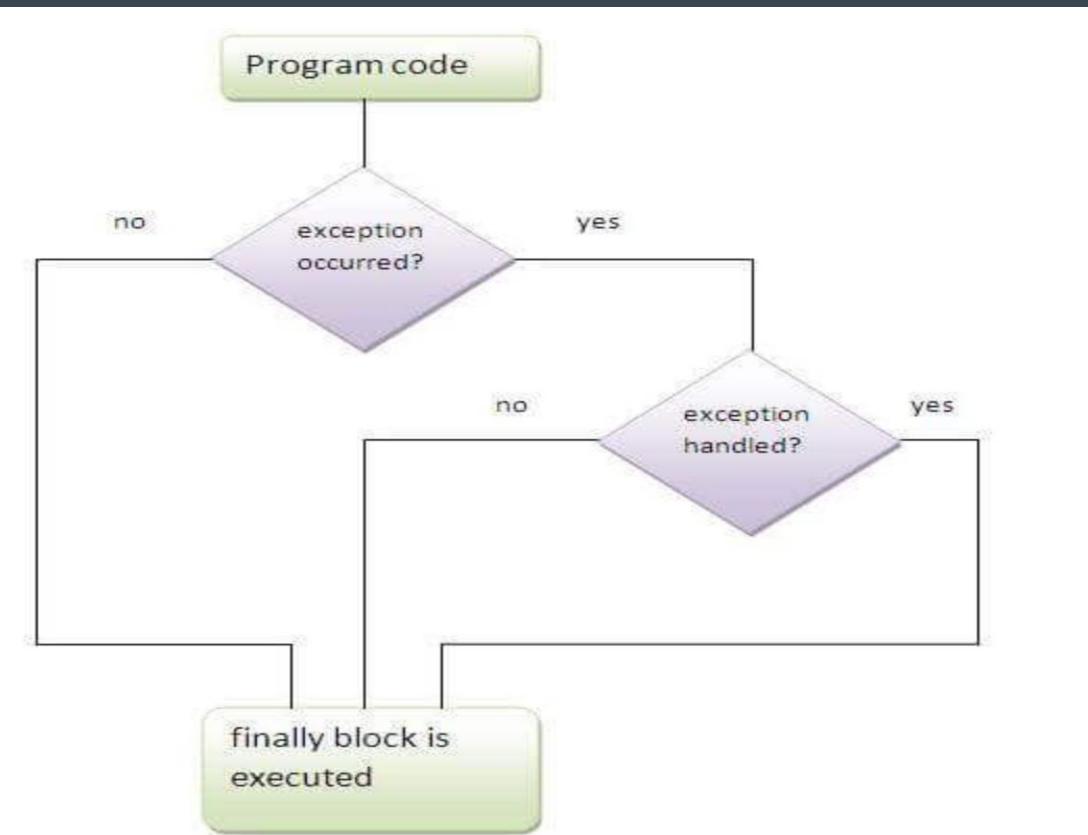
}

catch(Exception e) {

... exception handling code

Finally Block

- finally block is a block that is used to execute important code such as closing connection, stream etc.
- finally block is always executed whether exception is handled or not.
- finally block follows try or catch block.



Handson topics/details

- <https://courses.cs.washington.edu/courses/cse341/12au/java/mini1.pdf>
- Catch a runtime exception - divide a number by zero
- Add a readList method to ListOfNumbers.java (in a .java source file). This method should read in int values from a file, print each value, and append them to the end of the vector. You should catch all appropriate errors. You will also need a text file containing numbers to read in.
- Keeping the above, Create your own exception
- Create your own exception class hierarchy
- Keeping problem statement 2 in mind catch an exception and throw different exception

Exception Handling Example

```
//This method reads a string from the keyboard

public String input() throws MyException { // Use throws to warn
    // that the method may throw a MyException

    BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

    String s = null;

    // We use a try block to wrap code that might create an exception. In this case,
    // the compiler tells us that the readLine() method in the
    // BufferedReader class might throw an I/O exception

    try {
        s = reader.readLine();

        // We use a catch block to wrap the code that handles an IOException
    } catch (IOException e) {
        System.out.println(e.getMessage());
    }
    // We close the read stream in the finally block
} finally {
```

Reference Links

<https://www.javatpoint.com/exception-handling-in-java>

<https://www.javatpoint.com/finally-block-in-exception-handling>

<https://www.javatpoint.com/custom-exception>

Day - 8

IO Operations

IO Operations

Input/Output Streams

File Read and Write

Java Input/Output

- Java I/O (Input and Output) is used to process the input and produce the output.
- Java uses the concept of a stream to make I/O operation fast. The `java.io` package contains all the classes required for input and output operations.
- We can perform file handling in Java by Java I/O API.

Stream

A stream is a sequence of data. In Java, a stream is composed of bytes. It's called a stream because it is like a stream of water that continues to flow.

In Java, 3 streams are created for us automatically. All these streams are attached with the console.

- 1) System.out: standard output stream
- 2) System.in: standard input stream
- 3) System.err: standard error stream

Input / Output Stream

- **Input Stream**
 - Java application uses an input stream to read data from a source; it may be a file, an array, peripheral device or socket.
- **Output Stream**
 - Java application uses an output stream to write data to a destination; it may be a file, an array, peripheral device or socket.

Stream

Let's see the code to print output and an error message to the console.

```
System.out.println("simple message");
```

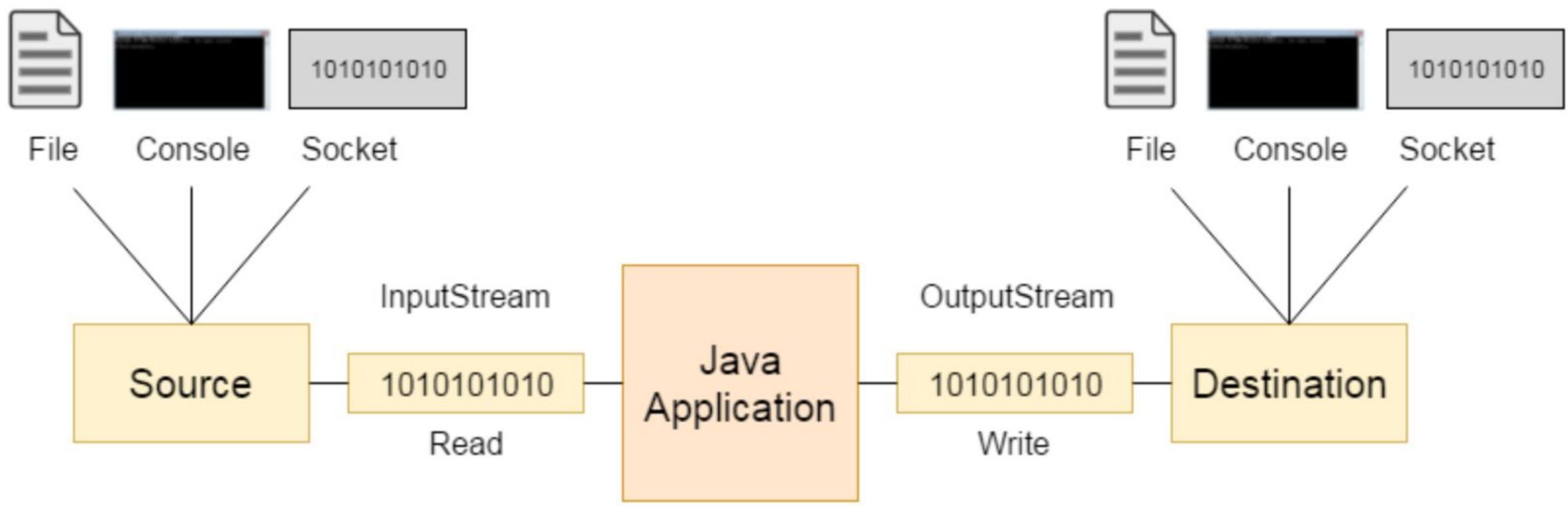
```
System.err.println("error message");
```

Let's see the code to get input from console

```
int i=System.in.read(); //returns ASCII code of 1st character
```

```
System.out.println((char)i); //will print the character
```

Input / Output Stream



Input / Output Stream

Code to copy content of one file to another

```
import java.io.*;
public class BStream {
    public static void main(
        String[] args) throws IOException
    {

        FileInputStream sourceStream = null;
        FileOutputStream targetStream = null;

        try {
            sourceStream
                = new FileInputStream("sorcefile.txt");
            targetStream
                = new FileOutputStream("targetfile.txt");

            // Reading source file and writing
            // content to target file byte by byte
            int temp;
            while ((temp = sourceStream.read()) != -1)
                targetStream.write((byte)temp);
        }
        finally {
            if (sourceStream != null)
                sourceStream.close();
            if (targetStream != null)
                targetStream.close();
        }
    }
}
```

Input / Output Stream

Code to read a file using File Reader

```
import java.io.*;
public class GfG {
    public static void main(
        String[] args) throws IOException
    {
        FileReader sourceStream = null;
        try {
            sourceStream
                = new FileReader("test.txt");

            // Reading sourcefile and
            // writing content to target file
            // character by character.
            int temp;
            while ((temp =
                    sourceStream.read()) != -1)
                System.out.println((char)temp);
        }
        finally {
            // Closing stream as no longer in use
            if (sourceStream != null)
                sourceStream.close();
        }
    }
}
```

Handson topics/details

- Generate CSV file from DAT file from the directory :
- [https://drive.google.com/drive/folders/12R1Ozkh_-VX8CnmBOSsLL6WHYeJkCrxV
?usp=sharing](https://drive.google.com/drive/folders/12R1Ozkh_-VX8CnmBOSsLL6WHYeJkCrxV?usp=sharing)
- (Note : Focus on data transfer from DAT file to CSV, rest of things are for experienced developers)

Reference Links

- <https://www.codesdope.com/java-file-io/>

Day - 9

Threading Concepts

Threading Concepts

Thread Lifecycle

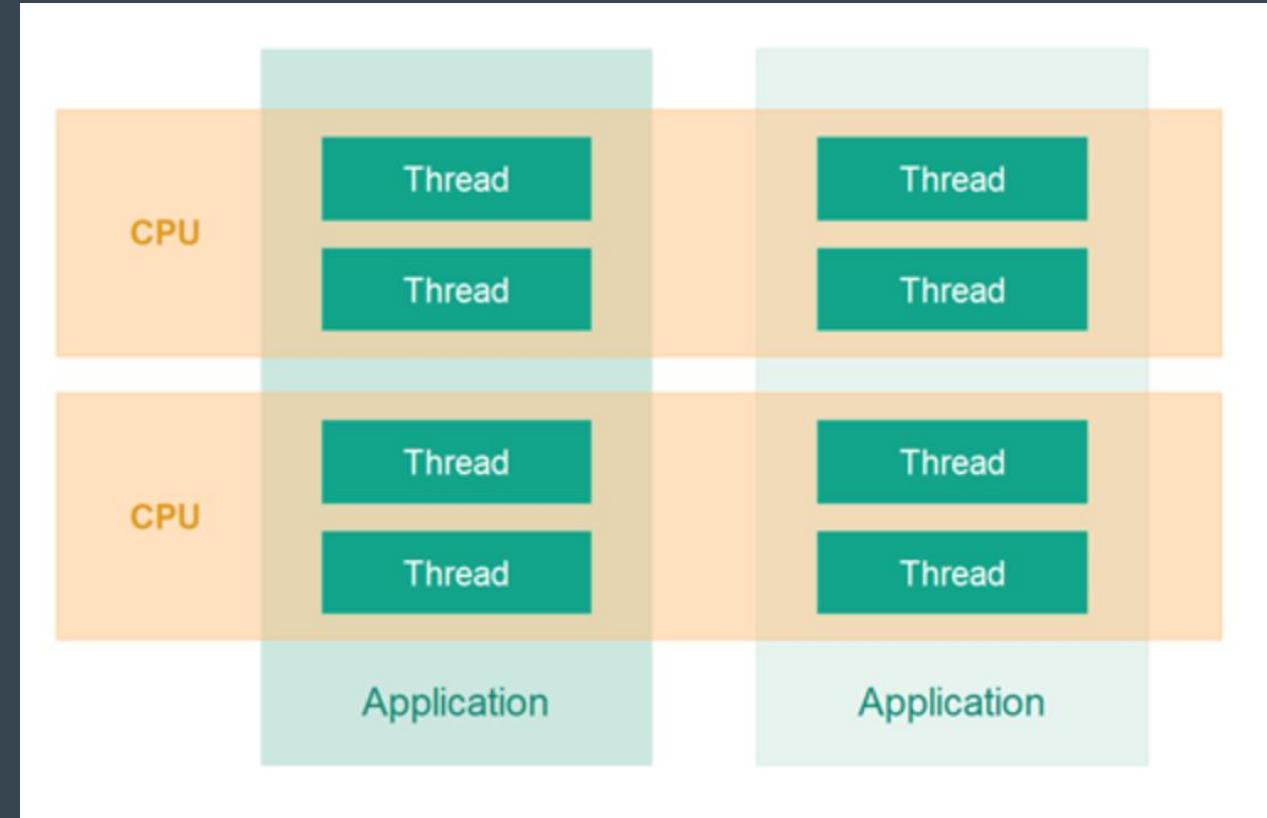
Ways to create threads

Synchronised keyword

Inter-thread communication - Wait, Join, notify, notifyAll methods

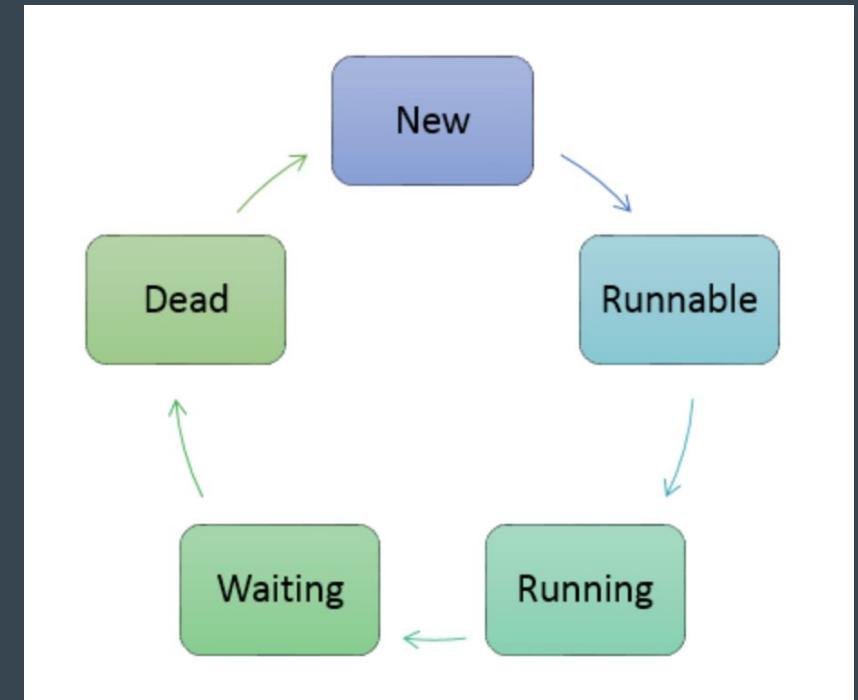
Threading Concepts

- What is multithreading?
 - a process of executing two or more threads simultaneously
- Advantages:
 - Better Utilization of a Single CPU
 - Better Utilization of Multiple CPUs or CPU Cores
 - Better User Experience with Regards to Responsiveness
 - Better User Experience with Regards to Fairness



Thread Life Cycle

- **New:** The thread is created using class "Thread". It remains in this state till the program **starts** the thread.
- **Runnable:** The instance of the thread is invoked with a start method. The thread control is given to scheduler to finish the execution, but the thread is not running.
- **Running:** When the scheduler starts executing a thread, then the state is changed to "running" state.
- **Waiting:** When a thread has to wait.
- **Dead:** When the thread completes the execution.



Ways to Create Thread

Thread Subclass

```
class MyThread extends Thread {  
  
    @Override  
    public void run() {  
        System.out.println("MyThread running");  
    }  
}  
  
public class ThreadSubclass {  
    Run | Debug  
    public static void main(String[] args) {  
        MyThread myThread = new MyThread();  
        myThread.start();  
    }  
}
```

Thread class provide constructors and methods to create and perform operations on a thread. Thread class extends Object class and implements Runnable interface.

Commonly used Constructors of Thread class:

- Thread()
- Thread(String name)
- Thread(Runnable r)
- Thread(Runnable r, String name)

Ways to Create Thread

Runnable Interface

```
class MyRunnableThread implements Runnable {  
  
    @Override  
    public void run() {  
        System.out.println("My thread from Runnable interface");  
    }  
  
}  
  
public class ThreadRunnable {  
    Run | Debug  
    public static void main(String[] args) {  
        MyRunnableThread runnableThread = new MyRunnableThread();  
        Thread thread = new Thread(runnableThread);  
        thread.start();  
    }  
}
```

The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. Runnable interface have only one method named run().

1. **public void run():** is used to perform action for a thread.

Ways to Create Thread

Using the Thread Class: Thread(Runnable r, String name)

```
public class MyThread2 implements Runnable {  
    public void run() {  
        System.out.println("Now the thread is running ...");  
    }  
    // main method  
    public static void main(String args[]) {  
        Runnable r1 = new MyThread2(); // creating an object of the class MyThread2  
        // creating an object of the class Thread using Thread(Runnable r, String name)  
        Thread th1 = new Thread(r1, "My new thread");  
        th1.start(); // the start() method moves the thread to the active state  
        String str = th1.getName(); // getting the thread name by invoking the getName() method  
        System.out.println(str);  
    }  
}
```

Race Condition

- It occurs in a multi-threaded environment when more than one thread try to access a shared resource (modify, write) at the same time
- It is safe if multiple threads are trying to read a shared resource as long as they are not trying to change it.
- The problem arises when these threads try to access the same resource.
- Examples of shared resources are class variables, DB record in a table, writing in a file.

Race Condition Example

```
RaceConditionDemo.java    RaceConditionDemo

class Counter implements Runnable {
    private int c = 0;

    public void increment() {
        try {
            Thread.sleep(10);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        c++;
    }

    public void decrement() { c--; }

    public int getValue() { return c; }

    @Override
    public void run() {
        // incrementing
        this.increment();
        System.out.println("Value for Thread after increment "
            + Thread.currentThread().getName() + " " + this.getValue());

        // decrementing
        this.decrement();
        System.out.println("Value for Thread at last "
            + Thread.currentThread().getName() + " " + this.getValue());
    }
}
```

```
public class RaceConditionDemo {

    Run | Debug
    public static void main(String[] args) {
        Counter counter = new Counter();
        Thread t1 = new Thread(counter, "Thread-1");
        Thread t2 = new Thread(counter, "Thread-2");
        Thread t3 = new Thread(counter, "Thread-3");
        t1.start();
        t2.start();
        t3.start();
    }
}
```

Avoiding Race Conditions

- Ensure only one thread can access the shared resource at a time
- The above process is called synchronization
- The concept of monitor is used to achieve synchronization
- Every object created in Java has one associated monitor
- Java provides two synchronization idioms

```
synchronized <returntype> method_name(parameter_list){  
    ..  
    ..  
}
```

```
synchronized(object_reference){  
    // code block  
}
```

Synchronized Examples

Synchronized method example

```
class Message {  
    public synchronized void displayMsg(String msg) {  
        System.out.println("Inside displayMsg method " + Thread.currentThread().getName());  
        System.out.print("**" + msg);  
        try {  
            Thread.sleep(3);  
        } catch (InterruptedException e) {  
            e.printStackTrace();  
        }  
        System.out.println("*");  
        System.out.println("Exit thread " + Thread.currentThread().getName());  
    }  
}
```

Synchronized Examples

Synchronized block example

```
class Message2 {  
    public void displayMsg(String msg) {  
        synchronized (this) {  
            System.out.println("Inside displayMsg method " + Thread.currentThread().getName());  
            System.out.print("**" + msg);  
            try {  
                Thread.sleep(3);  
            } catch (InterruptedException e) {  
                e.printStackTrace();  
            }  
            System.out.println("*");  
            System.out.println("Exit thread " + Thread.currentThread().getName());  
        }  
    }  
}
```

Inter-thread communication

- Producer-Consumer problem demo
- `wait()`: method tells the current thread to give up monitor and go to sleep, until another thread invokes the `notify()` or `notifyAll()` method for this object.
- `notify()`: wakes up a single thread that is waiting on this object's monitor
- `notifyAll()`: wakes up all the threads that called `wait()` on the same object.
- `wait()`, `notify()` and `notifyAll()` methods can only be called from a synchronized context, as these methods are about releasing the monitor and acquiring it.
- Producer-Consumer solution demo

Other thread methods

- `isAlive()`: It tests if the thread it is called upon is alive or not. A thread is alive if it has been started and not yet terminated.
- `join()`: It waits until the thread on which it is called terminates. There are three overloaded `join` methods.
 - `join()` - Waits indefinitely for this thread to die.
 - `join(long millis)` - Waits at most millis milliseconds for this thread to die. A timeout of 0 means to wait forever.
 - `join(long millis, int nanos)` - Waits at most millis milliseconds plus nanos nanoseconds for this thread to die.

Handson topics/details

- Create thread through different ways
- Read two separate files from individual threads
- Create three threads and each should print current date 10 times with 100 ms interval
- Try examples for synchronised method, block and understand its' effect
- Fix the race condition example
- Print chat like conversation which maintains order of questions and answers. One thread should print only questions and the second thread should print only answers

Java 8+ Features

Lambda Expression

Functional Interface

Stream API

Date/Time new features

Optional

To touch base on the new features

Java 9 Features

Reference Links

<https://www.javatpoint.com/multithreading-in-java>

<http://tutorials.jenkov.com/java-concurrency/index.html>

<https://www.youtube.com/watch?v=TCd8QIS-2KI>

<https://www.netjstech.com/2015/06/race-condition-in-java-multi-threading.html>

<https://www.netjstech.com/2015/06/synchronization-in-java-multithreading-synchronizing-thread.html>

<https://www.netjstech.com/2015/07/inter-thread-communiction-wait-notify-java-multi-thread.html>

<https://www.netjstech.com/2015/07/why-wait-notify-and-notifyall-methods-in-object-class-java-multi-threading.html>

Day - 10

Java 8+ Features

Lambda Expression : Java8

It provides a clear and concise way to represent one method interface using an expression. It is very useful in collection library. It helps to iterate, filter and extract data from collection.

The Lambda expression is used to provide the implementation of an interface which has functional interface. It saves a lot of code. In case of lambda expression, we don't need to define the method again for providing the implementation. Here, we just write the implementation code.

Java lambda expression is treated as a function, so compiler does not create .class file.

Main Usage is : To provide the implementation of Functional interface and Less coding.

Java Lambda Expression Syntax

`(argument-list) -> {body}`

1) Argument-list: It can be empty or non-empty as well.

2) Arrow-token: It is used to link arguments-list and body of expression.

3) Body: It contains expressions and statements for lambda expression.

Without Lambda Expression

```
interface Drawable{  
  
    public void draw();  
  
}  
  
public class LambdaExpressionExample {  
  
    public static void main(String[] args) {  
  
        int width=10;  
  
        //without lambda, Drawable implementation using anonymous class  
  
        Drawable d=new Drawable(){  
  
            public void draw(){System.out.println("Drawing "+width);}  
  
        };  
  
        d.draw();  
  
    }  
}
```

Java Lambda Expression Example

```
@FunctionalInterface //It is optional  
  
interface Drawable{  
  
    public void draw();  
  
}  
  
public class LambdaExpressionExample2 {  
  
    public static void main(String[] args) {  
  
        int width=10;  
  
        //with lambda  
  
        Drawable d2=()->{  
  
            System.out.println("Drawing "+width);  
  
        };  
  
        d2.draw();  
  
    }  
}
```

Java Lambda Expression Example: No Parameter

```
interface Sayable{  
  
    public String say();  
  
}  
  
public class LambdaExpressionExample3{  
  
    public static void main(String[] args) {  
  
        Sayable s=()->{  
  
            return "I have nothing to say";  
  
        };  
  
        System.out.println(s.say());  
  
    }  
}
```

Java Lambda Expression Example: Single Parameter

```
interface Sayable{  
  
    public String say(String name);  
  
}  
  
public class LambdaExpressionExample4{  
  
    public static void main(String[] args) {  
  
        // Lambda expression with single parameter.  
        Sayable s1=(name)->{  
  
            return "Hello, "+name;  
  
        };  
  
        System.out.println(s1.say("Sonoo"));  
  
        // You can omit function parentheses  
        Sayable s2= name ->{  
  
            return "Hello, "+name;  
  
        };  
  
        System.out.println(s2.say("Sonoo"));  
  
    }  
}
```

Java Lambda Expression Example: Multiple Parameters

```
interface Addable{  
    int add(int a,int b);  
}  
  
public class LambdaExpressionExample5{  
    public static void main(String[] args) {  
        // Multiple parameters in lambda expression  
        Addable ad1=(a,b)->(a+b);  
        System.out.println(ad1.add(10,20));  
        // Multiple parameters with data type in lambda expression  
        Addable ad2=(int a,int b)->(a+b);  
        System.out.println(ad2.add(100,200));  
    }  
}
```

Java Lambda Expression Example: Creating Thread

Implementing run method by using lambda expression.

```
public class LambdaExpressionExample9{
```

```
    public static void main(String[] args) {
```

```
        //Thread Example without lambda
```

```
        Runnable r1=new Runnable(){
```

```
            public void run(){
```

```
                System.out.println("Thread1 is running...");
```

```
}
```

```
};
```

```
        Thread t1=new Thread(r1);
```

```
        t1.start();
```

```
        //Thread Example with lambda
```

```
        Runnable r2=()->{
```

```
            System.out.println("Thread2 is running...");
```

```
};
```

```
        Thread t2=new Thread(r2);
```

```
        t2.start();
```

```
}
```

```
}
```

Functional Interface

Lambda expression provides implementation of functional interface. An interface which has only one abstract method is called functional interface. Java provides an annotation `@FunctionalInterface`, which is used to declare an interface as functional interface.

Functional Interface is also known as Single Abstract Method Interfaces or SAM Interfaces. It is a new feature in Java, which helps to achieve functional programming approach.

A functional interface can have methods of object class. See in the following example.

```
@FunctionalInterface  
interface sayable{  
    void say(String msg); // abstract method  
    // It can contain any number of Object class methods.  
    int hashCode();  
    String toString();  
    boolean equals(Object obj);  
}  
  
public class FunctionalInterfaceExample2 implements sayable{  
    public void say(String msg){  
        System.out.println(msg);  
    }  
    public static void main(String[] args) {  
        FunctionalInterfaceExample2 fie = new FunctionalInterfaceExample2();  
        fie.say("Hello there");  
    }  
}
```

Stream API

- `java.util.stream.`
- Consists of classes, interfaces and enum
- Allows functional-style operations on the elements.

Stream Features:

- Stream does not store elements. It simply conveys elements from a source such as a data structure, an array, or an I/O channel, through a pipeline of computational operations.
- Stream is functional in nature. Operations performed on a stream does not modify it's source. For example, filtering a Stream obtained from a collection produces a new Stream without the filtered elements, rather than removing elements from the source collection.
- Stream is lazy and evaluates code only when required.
- The elements of a stream are only visited once during the life of a stream. Like an Iterator, a new stream must be generated to revisit the same elements of the source.

```

import java.util.*;
class Product{
    int id;
    String name;
    float price;
    public Product(int id, String name, float price) {
        this.id = id;
        this.name = name;
        this.price = price;
    }
}
public class JavaStreamExample {
    public static void main(String[] args) {
        List<Product> productsList = new ArrayList<Product>();
        //Adding Products
        productsList.add(new Product(1,"HP Laptop",25000f));
        productsList.add(new Product(2,"Dell Laptop",30000f));
        productsList.add(new Product(3,"Lenevo Laptop",28000f));
        productsList.add(new Product(4,"Sony Laptop",28000f));
        productsList.add(new Product(5,"Apple Laptop",90000f));
        List<Float> productPriceList = new ArrayList<Float>();
        for(Product product: productsList){
            // filtering data of list
            if(product.price<30000){
                productPriceList.add(product.price); // adding price to a productPriceList
            }
        }
        System.out.println(productPriceList); // displaying data
    }
}

```

```

List<Float> productPriceList2 =productsList.stream()
    .filter(p -> p.price > 30000)// filtering data
    .map(p->p.price) // fetching price
    .collect(Collectors.toList()); // collecting as list
System.out.println(productPriceList2);

```

Optional class

public final class from java.util package

used to deal with NullPointerException in Java application.

It provides methods which are used to check the presence of value for particular variable.

Example: Java Program without using Optional

This program terminates abnormally and throws a nullPointerException.

```
public class OptionalExample {  
    public static void main(String[] args) {  
        String[] str = new String[10];  
        String lowercaseString = str[5].toLowerCase();  
        System.out.print(lowercaseString);  
    }  
}
```

Java Optional Example: If Value is not Present

```
import java.util.Optional;  
  
public class OptionalExample {  
    public static void main(String[] args) {  
        String[] str = new String[10];  
        Optional<String> checkNull = Optional.ofNullable(str[5]);  
        if(checkNull.isPresent()){ // check for value is present or not  
            String lowercaseString = str[5].toLowerCase();  
            System.out.print(lowercaseString);  
        }else  
            System.out.println("string value is not present");  
    }  
}
```

Output:

```
string value is not present
```

Java Default Methods

Create default methods inside the interface.

Default Methods

- Defined inside the interface
- Tagged with default
- non-abstract methods.

You can also define static methods inside the interface. Static methods are used to define utility methods.

Abstract Class vs Java 8 Interface

From Java 8, interface and an abstract class is almost similar

The key difference is that we can create constructor in the abstract class.

```
interface Sayable{
    // Default method
    default void say(){
        System.out.println("Hello, this is default method");
    }
    // Abstract method
    void sayMore(String msg);
}

public class DefaultMethods implements Sayable{
    public void sayMore(String msg){           // implementing abstract method
        System.out.println(msg);
    }
    public static void main(String[] args) {
        DefaultMethods dm = new DefaultMethods();
        dm.say();    // calling default method
        dm.sayMore("Work is worship"); // calling abstract method
    }
}
```

To touch base on the new features

Method References

Stream Filter

Base64 Encode Decode

Default Methods

forEach() method

Collectors class

StringJoiner class

Optional class

JavaScript Nashorn

Parallel Array Sort

Type Inference

Parameter Reflection

Type Annotations

JDBC Improvements

Java 7 Features

Java 9 Features

Platform Module System (Project Jigsaw)

Interface Private Methods

Try-With Resources

Anonymous Classes

@SafeVarargs Annotation

Collection Factory Methods

Process API Improvement

New Version-String Scheme

JShell: The Java Shell (REPL)

Control Panel

Stream API Improvement

Java 8+ Features

Major Features	Java 8 (LTS)	Java 9	Java 10	Java 11 (LTS)
APIs Features	Lambdas <pre>s -> do(s) Eg. arr.forEach((String s) -> System.out.print(s)); arr.forEach(s-> System.out.print(s)); arr.forEach(System.out::print);</pre>	Modularity <pre>module my.module { requires module.name; exports my.moduel.api; }</pre>	Var improved inferencing <pre>var a = new ArrayList<String>() var stream = list.stream();</pre>	Var on Lambdas <pre>var s -> do(s)</pre>
JVM/Lang.	Interface: default method and static	JShell: command line interpreter Interface: private methods GC: G1 is the default GC Module: java --describe-module \${module} Logging: unified logging with -Xlog <pre>java -Xlog:all=debug,file=application.log -version</pre>	JIT: new experimental javah removed Improvements of repository and thread-local	GC <ul style="list-style-type: none"> Epsilon GC : the GC no-opt JAVA EE: deprecated java HelloWorld.java (without compilation) and shebang Interface: nested class

Handson topics/details

learn examples and test various features of all Java 8+ versions

Reference Links

- <https://www.baeldung.com/java-8-functional-interfaces>
- <https://www.geeksforgeeks.org/functional-interfaces-java/>
- <https://4comprehension.com/busy-developers-guide-to-java-9-10-11-12-13-and-above/>
- <https://dev.to/therajsaxena/java-9-to-java-13-top-features-362l>
- <https://www.javatpoint.com/java-9-features>
- <https://www.journaldev.com/13121/java-9-features-with-examples>

Day - 11

Coding Standards

Coding Standards

Introduction

- Declarations
- Statements
- White Space
- Naming Conventions
- Programming Practices
- Code Examples

File Names

File Organization

Indentation

Comments

Introduction

Why Have Code Conventions?

- Code conventions are important to programmers for a number of reasons:
 - 80% of the lifetime cost of a piece of software goes to maintenance.
 - Hardly any software is maintained for its whole life by the original author.
 - Code conventions improve the readability of the software, allowing engineers to understand new code more quickly and thoroughly.
 - If you ship your source code as a product, you need to make sure it is as well packaged and clean as any other product you create.

File Names

- File Suffixes
 - Java Software uses the following file suffixes:
 - Java source= .java
 - Java bytecode= .class
- Common File Names
 - Frequently used file names include:
 - **GNUmakefile**: The preferred name for makefiles. We use gnumake to build our software.
 - **README**: The preferred name for the file that summarizes the contents of a particular directory.

File Organization

- A file consists of sections that should be separated by blank lines and an optional comment identifying each section.
- Files longer than 2000 lines are cumbersome and should be avoided.
- Reference for [example](#) java program

Java Source Files

- Each Java source file contains a single public class or interface. When private classes and interfaces are associated with a public class, you can put them in the same source file as the public class. The public class should be the first class or interface in the file.
- Java source files have the following ordering:
 - Beginning comments
 - Package and Import statements
 - Class and interface declarations

Beginning comments

All source files should begin with a c-style comment that lists the class name, version information, date, and copyright notice:

```
/*
 * Classname
 *
 * Version information
 *
 * Date
 *
 * Copyright notice
 */
```

Package and Import Statements

The first non-comment line of most Java source files is a package statement. After that, import statements can follow. For example:

```
package java.awt;  
  
import java.awt.peer.CanvasPeer;
```

Indentation

- Four spaces should be used as the unit of indentation. The exact construction of the indentation (spaces vs. tabs) is unspecified. Tabs must be set exactly every 8 spaces (not 4).
- Line Length
 - Avoid lines longer than 80 characters, since they're not handled well by many terminals and tools.
- Wrapping Lines
 - When an expression will not fit on a single line, break it according to these general principles:
 - Break after a comma.
 - Break before an operator.
 - Prefer higher-level breaks to lower-level breaks.
 - Align the new line with the beginning of the expression at the same level on the previous line.
 - If the above rules lead to confusing code or to code that's squished up against the right margin, just indent 8 spaces instead.

Comments

- Java programs can have two kinds of comments: implementation comments and documentation comments. Implementation comments are those found in C++, which are delimited by /*...*/, and //. Documentation comments (known as "doc comments") are Java-only, and are delimited by /**...*/. Doc comments can be extracted to HTML files using the javadoc tool.
- Implementation comments are meant for commenting out code or for comments about the particular implementation. Doc comments are meant to describe the specification of the code, from an implementation-free perspective, to be read by developers who might not necessarily have the source code at hand.
- Comments should be used to give overviews of code and provide additional information that is not readily available in the code itself. Comments should contain only information that is relevant to reading and understanding the program. For example, information about how the corresponding package is built or in what directory it resides should not be included as a comment.
- Discussion of nontrivial or nonobvious design decisions is appropriate, but avoid duplicating information that is present in (and clear from) the code. It is too easy for redundant comments to get out of date. In general, avoid any comments that are likely to get out of date as the code evolves.
- Comments should not be enclosed in large boxes drawn with asterisks or other characters.
- Comments should never include special characters such as form-feed and backspace.
- **Note:** The frequency of comments sometimes reflects poor quality of code. When you feel compelled to add a comment, consider rewriting the code to make it clearer.

Block Comments

Block comments are used to provide descriptions of files, methods, data structures and algorithms. Block comments may be used at the beginning of each file and before each method. They can also be used in other places, such as within methods. Block comments inside a function or method should be indented to the same level as the code they describe.

```
/*
 * Here is a block comment.
 */
```

Block comment with Indentation

```
/*-
 * Here is a block comment with some very special
 * formatting that I want indent(1) to ignore.
 *
```

```
128
*    one
```

Class and Interface Declarations

Part of Class/Interface Declaration	Notes
Class/interface documentation comment (<code>/** ... */</code>)	
class or interface statement	
Class/interface implementation comment (<code>/*...*/</code>), if necessary	This comment should contain any class-wide or interface-wide information that wasn't appropriate for the class/interface documentation comment.
Class (static) variables	First the public class variables, then the protected, then package level (no access modifier), and then the private.
Instance variables	First public, then protected, then package level (no access modifier), and then private.
Constructors	
Methods	These methods should be grouped by functionality rather than by scope or accessibility. For example, a private class method can be in between two public instance methods. The goal is to make reading and understanding the code easier.

Single-Line Comments

Short comments can appear on a single line indented to the level of the code that follows. If a comment can't be written in a single line, it should follow the block comment format. A single-line comment should be preceded by a blank line.

```
if (condition) {  
    /* Handle the condition. */  
    ...  
}
```

Trailing Comments

Very short comments can appear on the same line as the code they describe, but should be shifted far enough to separate them from the statements. If more than one short comment appears in a chunk of code, they should all be indented to the same tab setting.

```
if (a == 2) {  
    return TRUE;      /* special case */  
}  
else {  
    return isPrime(a); /* works only for odd a */  
}
```

End-Of-Line Comments

The // comment delimiter can comment out a complete line or only a partial line. It shouldn't be used on consecutive multiple lines for text comments; however, it can be used in consecutive multiple lines for commenting out sections of code

```
if (foo > 1){  
  
    // Do a double-flip.  
  
    ...  
  
}  
  
else {  
  
    return false;      // Explain why here.  
  
}  
  
//if (bar > 1){  
  
//  
  
//    // Do a triple-flip.  
  
//    ...  
  
//}  
  
//else {  
  
//    return false;  
  
//}
```

Documentation Comments

Doc comments describe Java classes, interfaces, constructors, methods, and fields. Each doc comment is set inside the comment delimiters `/**...*/`, with one comment per class, interface, or member. This comment should appear just before the declaration:

```
/**  
 * The Example class provides ...  
 */  
  
public class Example { ...}
```

Declarations

- Number Per Line
 - One declaration per line is recommended since it encourages commenting. In other words,
 - int level; // indentation level
 - int size; // size of table
 - Object currentEntry; // currently selected table entry
- Initialization
 - Try to initialize local variables where they're declared. The only reason not to initialize a variable where it's declared is if the initial value depends on some computation occurring first.
- Placement
 - Put declarations only at the beginning of blocks. (A block is any code surrounded by curly braces "{" and "}).) Don't wait to declare variables until their first use; it can confuse the unwary programmer and hamper code portability within the scope.
 - void myMethod() {
 - int int1 = 0; // beginning of method block
 -
 - if (condition) {
 - int int2 = 0; // beginning of "if" block
 - ...
 - }
 - }
 - </blockquote>

Class and Interface Declarations

When coding Java classes and interfaces, the following formatting rules should be followed:

- No space between a method name and the parenthesis "(" starting its parameter list
- Open brace "{" appears at the end of the same line as the declaration statement
- Closing brace "}" starts a line by itself indented to match its corresponding opening statement, except when it is a null statement the ")" should appear immediately after the "{"

```
class Sample extends Object {
```

```
    int ivar1;
```

```
    int ivar2;
```

```
    Sample(int i, int j) {
```

```
        ivar1 = i;
```

```
        ivar2 = j;
```

```
}
```

```
    int emptyMethod() {}
```

```
    ...
```

```
}
```

Statements

- Simple Statements
 - Each line should contain at most one statement.

```
argv++;      // Correct
argc--;      // Correct
argv++; argc--; // AVOID!
```

- Compound Statements
 - Compound statements are statements that contain lists of statements enclosed in braces "{ statements }".
 - The enclosed statements should be indented one more level than the compound statement.
 - The opening brace should be at the end of the line that begins the compound statement; the closing brace should begin a line and be indented to the beginning of the compound statement.
 - Braces are used around all statements, even single statements, when they are part of a control structure, such as an if-else or for statement. This makes it easier to add statements without accidentally introducing bugs due to forgetting to add braces.

Statements

- return Statements
 - A return statement with a value should not use parentheses unless they make the return value more obvious in some way.

```
return;  
  
return myDisk.size();  
  
return (size ? size : defaultSize);
```

Statements

- if, if-else, if else-if else Statements
 - The if-else class of statements should have the following form:

```
if (condition) {  
    statements;  
}  
  
if (condition) {  
    statements;  
} else {  
    statements;  
}  
  
if (condition) {  
    statements;  
} else if (condition) {  
    statements;  
} else {  
    statements;  
}
```

- Note: if statements always use braces, {}. Avoid the following error-prone form:

```
if (condition) //AVOID! THIS OMITS THE BRACES {}!  
    statement;
```

Statements

- for Statements
 - A for statement should have the following form:

```
for (initialization; condition; update) {  
    statements;  
}
```

- An empty for statement (one in which all the work is done in the initialization, condition, and update clauses) should have the following form:

```
for (initialization; condition; update);
```

- When using the comma operator in the initialization or update clause of a for statement, avoid the complexity of using more than three variables. If needed, use separate statements before the for loop (for the initialization clause) or at the end of the loop (for the update clause).

Statements

- while Statements
 - A while statement should have the following form:

```
while (condition) {  
    statements;  
}
```

- An empty while statement should have the following form:

```
while (condition);
```

Statements

- do-while Statements
 - A do-while statement should have the following form:

```
do {  
    statements;  
} while (condition);
```

- switch Statements

```
switch (condition) {  
case ABC:  
    statements;  
    /* falls through */  
case DEF:  
    statements;  
    break;  
case XYZ:  
    statements;  
    break;  
default:  
    statements;  
    break;  
}
```

Statements

- try-catch Statements
 - A try-catch statement should have the following format:

```
try {  
    statements;  
} catch (ExceptionClass e) {  
    statements;  
}
```

- A try-catch statement may also be followed by finally, which executes regardless of whether or not the try block has completed successfully.

```
try {  
    statements;  
} catch (ExceptionClass e) {  
    statements;  
} finally {  
    statements;  
}
```

White Space

- Blank Lines
 - Blank lines improve readability by setting off sections of code that are logically related.
 - Two blank lines should always be used in the following circumstances:
 - Between sections of a source file
 - Between class and interface definitions
 - One blank line should always be used in the following circumstances:
 - Between methods
 - Between the local variables in a method and its first statement
 - Before a block or single-line comment
 - Between logical sections inside a method to improve readability
- Blank Spaces
 - Blank spaces should be used in the following circumstances:
 - A keyword followed by a parenthesis should be separated by a space.

```
while (true) {  
    ...  
}
```

White Space

- Note that a blank space should not be used between a method name and its opening parenthesis. This helps to distinguish keywords from method calls.
 - A blank space should appear after commas in argument lists.
 - All binary operators except . should be separated from their operands by spaces. Blank spaces should never separate unary operators such as unary minus, increment ("++"), and decrement ("--") from their operands. Example:

```
a += c + d;  
a = (a + b) / (c * d);  
  
while (d++ = s++) {  
    n++;  
}  
printSize("size is " + foo + "\n");
```

- The expressions in a for statement should be separated by blank spaces. Example:
 - for (expr1; expr2; expr3)
 - Casts should be followed by a blank space. Examples:

```
myMethod((byte) aNum, (Object) x);  
myMethod((int) (cp + 5), ((int) (i + 3))  
        + 1);
```

Naming Conventions

Naming conventions make programs more understandable by making them easier to read. They can also give information about the function of the identifier-for example, whether it's a constant, package, or class-which can be helpful in understanding the code.

Identifier Type	Rules for Naming	Examples
Packages	<p>The prefix of a unique package name is always written in all-lowercase ASCII letters and should be one of the top-level domain names, currently com, edu, gov, mil, net, org, or one of the English two-letter codes identifying countries as specified in ISO Standard 3166, 1981.</p> <p>Subsequent components of the package name vary according to an organization's own internal naming conventions. Such conventions might specify that certain directory name components be division, department, project, machine, or login names.</p>	com.sun.eng com.apple.quicktime.v2 edu.cmu.cs.bovik.cheese
Classes	<p>Class names should be nouns, in mixed case with the first letter of each internal word capitalized. Try to keep your class names simple and descriptive.</p> <p>Use whole words-avoid acronyms and abbreviations (unless the abbreviation is much more widely used than the long form, such as URL or HTML).</p>	class Raster; class ImageSprite;
Interfaces	Interface names should be capitalized like class names.	interface RasterDelegate; interface Storing;
Methods	Methods should be verbs, in mixed case with the first letter lowercase, with the first letter of each internal word capitalized.	run(); runFast(); getBackground();
Variables	<p>Except for variables, all instance, class, and class constants are in mixed case with a lowercase first letter. Internal words start with capital letters.</p> <p>Variable names should not start with underscore _ or dollar sign \$ characters, even though both are allowed.</p> <p>Variable names should be short yet meaningful. The choice of a variable name should be mnemonic- that is, designed to indicate to the casual observer the intent of its use. One-character variable names should be avoided except for temporary "throwaway" variables. Common names for temporary variables are i, j, k, m, and n for integers; c, d, and e for characters.</p>	Copy Copied to Clipboard Error: Could not Copy int i; char c; float myWidth;
Constants	The names of variables declared class constants and of ANSI constants should be all uppercase with words separated by underscores ("_"). (ANSI constants should be avoided, for ease of debugging.)	static final int MIN_WIDTH = 4; static final int MAX_WIDTH = 999; static final int GET_THE_CPU = 1;

Programming Practices

- Providing Access to Instance and Class Variables
 - Don't make any instance or class variable public without good reason. Often, instance variables don't need to be explicitly set or gotten-often that happens as a side effect of method calls.
 - One example of appropriate public instance variables is the case where the class is essentially a data structure, with no behavior. In other words, if you would have used a struct instead of a class (if Java supported struct), then it's appropriate to make the class's instance variables public.
- Referring to Class Variables and Methods
 - Avoid using an object to access a class (static) variable or method. Use a class name instead.

```
classMethod();           //OK
AClass.classMethod();    //OK
anObject.classMethod(); //AVOID!
```

Programming Practices

- Constants
 - Numerical constants (literals) should not be coded directly, except for -1, 0, and 1, which can appear in a for loop as counter values.
- Variable Assignments
 - Avoid assigning several variables to the same value in a single statement. It is hard to read. Example:
 - `fooBar.fChar = barFoo.lchar = 'c'; // AVOID!`
 - Do not use the assignment operator in a place where it can be easily confused with the equality operator

```
if ((c++ = d++) != 0) {  
    ...  
}
```

- should be written as

```
if (c++ = d++) {           // AVOID! (Java disallows)  
    ...  
}
```

- Do not use embedded assignments in an attempt to improve run-time performance. This is the job of the compiler. Example:
- **`d = (a = b + c) + r; // AVOID!`**
- should be written as

```
if (a == b && c == d)      // AVOID!  
if ((a == b) && (c == d)) // RIGHT
```

Programming Practices

- Expressions before `?' in the Conditional Operator
 - If an expression containing a binary operator appears before the ? in the ternary ?: operator, it should be parenthesized

```
(x >= 0) ? x : -x;
```

- Special Comments
 - Use **XXX** in a comment to flag something that is bogus but works. Use **FIXME** to flag something that is bogus and broken.

Code Examples

- Java Source File Example
 - Here is the [example](#) shows how to format a Java source file containing a single public class. Interfaces are formatted similarly.

Handson topics/details

10 Java Core Best Practices Every Java Programmer Should Know

Create the calculator program with 2 operations: Addition and Subtraction and follow the java best practices and coding standards.

Reference Links

- Summery best code practice
- Clean code characteristics
- Bug free java-code
- Name, Comment etc. rule book

Junit

Day - 13

Unit Testing and Junit

Unit Testing and Junit

What is Unit testing

Purpose of the unit tests

Testing terminology

Junit annotations

Assert class

Jar Files for Junit framework

Junit Coding standards

What is Unit testing

A unit test is a way of testing a unit - the smallest piece of code that can be logically isolated in a system.

A unit test is a piece of code written by a developer that executes a specific functionality in the code to be tested and asserts a certain behavior or state.

In most programming languages, that is a function, a subroutine, a method or property.

Generally programmers implement the unit tests

Purpose of the unit tests

To identify software regressions introduced by changes in the source code
It helps the developer to avoid breaking existing code during development activities

Testing terminology

The code which is tested is typically called the code under test. If you are testing an application, this is called the application under test.

Junit annotations

@Test	Identifies a method as a test method.
@Before	Executed before each test. It is used to prepare the test environment (e.g., read input data, initialize the class).
@After	Executed after each test. It is used to cleanup the test environment (e.g., delete temporary data, restore defaults). It can also save memory by cleaning up expensive memory structures.
@BeforeClass	Executed once, before the start of all tests. It is used to perform time intensive activities, for example, to connect to a database. Methods marked with this annotation need to be defined as static to work with JUnit.
@AfterClass	Executed once, after all tests have been finished. It is used to perform clean-up activities, for example, to disconnect from a database. Methods annotated with this annotation need to be defined as static to work with JUnit.
@Ignore or @Ignore("Why disabled")	Marks that the test should be disabled. This is useful when the underlying code has been changed and the test case has not yet been adapted. Or if the execution time of this test is too long to be included. It is best practice to provide the optional description, why the test is disabled.
@Test (expected = Exception.class)	Fails if the method does not throw the named exception.
@Test(timeout=100)	Fails if the method takes longer than 100 milliseconds.

Assert class

The org.junit.Assert class provides methods to assert the program logic.

The common methods of Assert class are as follows:

void assertEquals(boolean expected,boolean actual): checks that two primitives/objects are equal. It is overloaded.

void assertTrue(boolean condition): checks that a condition is true.

void assertFalse(boolean condition): checks that a condition is false.

void assertNull(Object obj): checks that object is null.

void assertNotNull(Object obj): checks that object is not null.

Jar Files for Junit framework

You need to load junit4.jar and hamcrest-core.jar files in your IDE in order to use the annotations and assert class methods. Click [here](#)/[here](#) to download the required jar files.

Junit Coding standards

General rules	Names rules	Use as explanation of intent Use as clarification of code Use as warning of consequences	Small number of instance variables Base class should know nothing about their derivatives Better to have many functions than to pass some code into a function to select a behavior Prefer non-static methods to static methods
Follow standard conventions Keep it simple stupid. Simpler is always better Reduce complexity as much as possible Boy scout rule. Leave the campground cleaner than you found it Always find root cause Always look for the root cause of a problem	Choose descriptive and unambiguous names Make meaningful distinction Use pronounceable names Use searchable names	Source code structure Separate concepts vertically Related code should appear vertically dense Declare variables close to their usage	Tests One assert per test Readable Fast Independent Repeatable
Design rules	Replace magic numbers with named constants Avoid encodings. Don't append prefixes or type information	Dependent functions should be close Similar functions should be close Place functions in the downward direction Keep lines short Don't use horizontal alignment Use white space to associate related things and disassociate weakly related	Code smells Rigidity. The software is difficult to change. A small change causes a cascade of subsequent changes Fragility. The software breaks in many places due to a single change Immobility. You cannot reuse parts of the code in other projects because of involved risks and high effort Needless. Complexity Needless. Repetition Opacity. The code is hard to understand
Understandability tips	Don't use flag arguments. Split method into several independent methods that can be called from the client without the flag	Don't break indentation	'Clean code' by Robert C. Martin summary by Wojtek Lukaszuk
Be consistent If you do something a certain way, do all similar things in the same way Use explanatory variables Encapsulate boundary conditions. Boundary conditions are hard to keep track of Put the processing for them in one place Prefer dedicated value objects to primitive type Avoid logical dependency. Don't write methods which works correctly depending on something else in the same class Avoid negative conditionals	Comments rules Always try to explain yourself in code Don't be redundant Don't add obvious noise Don't use closing brace comments Don't comment out code Just remove	Objects and data structures Hide internal structure Prefer data structures Avoid hybrids structures (half object and half data) Should be small Do one thing	

Handson topics/details

- <https://10.0.1.212:8443/download/attachments/168337085/Junit.zip?version=1&modificationDate=1637564912631&api=v2>
- Import the above attached Junit.zip project into the eclipse.
- Add Testcases (positive and negative) for all the methods present in Helper.java
- Cover all the methods and conditions as part of code coverage

Reference Links

- <https://www.tutorialspoint.com/junit/index.htm>
- <https://www.youtube.com/watch?v=o5k9NOR9lrl>
- <https://www.javatpoint.com/junit-tutorial>
- <https://dzone.com/articles/junit-tutorial-beginners>
- [JUnit Best Practices](#)

Day - 14

Mockito Framework

Mockito Framework

What is Mockito Framework

Adding Mockito to Project

Mockito Important Methods

Limitation of Mockito Framework

What is Mockito Framework

Java based Mocking framework

Mockito is used to mock interfaces/Classes so that a dummy functionality can be added to a mock interface/class that can be used in unit testing

Mocking: Mocking is a way to test the functionality of a class in isolation

Mock objects do the mocking of the real service.

Mock object is a dummy implementation for an interface or a class

Adding Mockito to a project

```
<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-core</artifactId>
    <version>3.7.7</version>
</dependency>
```

Mockito Important Methods

mock() , @mock

when()

thenReturn()

thenThrow()

verify()

spy()

Limitation of Mockito Framework

In Mockito Framework, we can not mock static, final and private methods.

PowerMockito is an extension API to support Mockito.

Power Mockito provides capabilities to work with the Java Reflection API in a simple way to overcome the problems.

Avoid PowerMockito as much as possible.

Handson topics/details

- Import the attached [mockitosample.zip](#) project into the eclipse.
- Write testcases for AuthenticatorApplication class in the handson package.

Reference Links

- <https://www.vogella.com/tutorials/Mockito/article.html>
- <https://www.tutorialspoint.com/mockito/index.htm>
- <https://www.baeldung.com/mockito-series>

Java Automation

Day - 15

Introduction to Automation

Introduction to Automation

What is Automation?

Why Automation?

What to Automate?

What not to automate?

What is Selenium?

What is Automation?

Automation Testing or Test Automation is a software testing technique that is performed using special automated testing software tools to execute a test case suite.

Phases of Automation Testing

Test Tool Selection

Define scope of Automation

Planning, Design and Development

Test Execution

Maintenance

Why Automation?

Registration Form

Sign up

Username

Password

Name

Address

Country

Email

Sex

Male

Female

Language

English

Non English

Submit

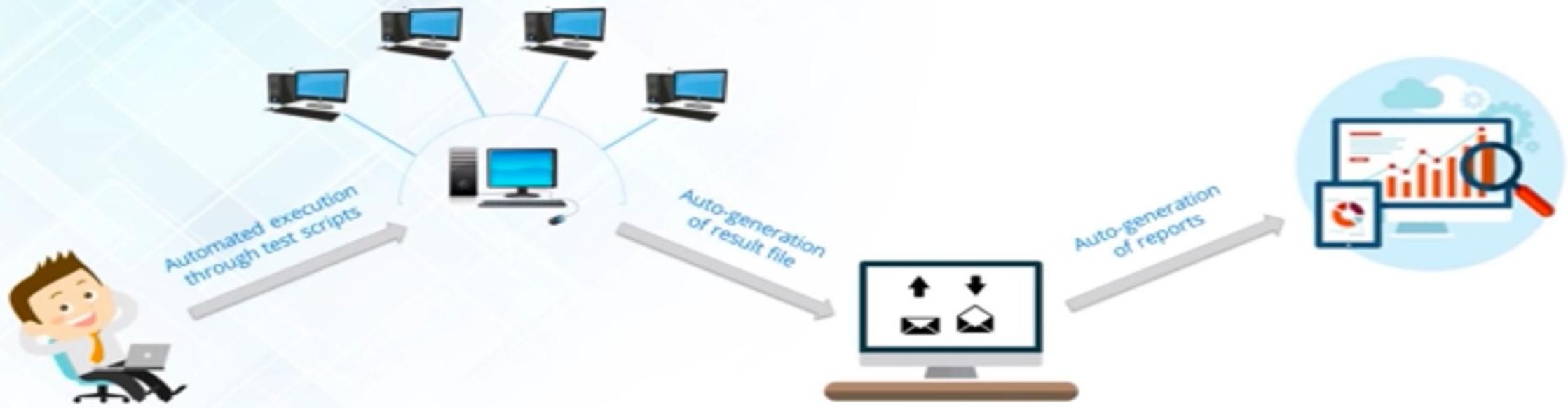
Testing web apps manually involves:

- Loading all transactions
- Downloading those transactions
- Creating pass/ fail reports for each
- Validating the form
- Taking screenshots for each validation



WEB APPLICATION

Why Automation?



Why Automation?

Faster Execution



More Accurate

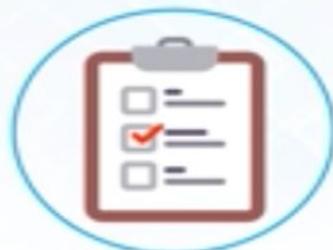


Lesser Investment In Human Resource



Features of Automation Testing

Supports Regression Testing



Frequent Executions



Supports Lights Out Execution



What to Automate?

Ones to be run on every build/release (Sanity/Smoke)

Having same workflow different data

Load / Stress testing

Ones taking long time to run

Involve large data as inputs

Require to be tested on multiple set of environments

Need to capture images/reports on each level of execution

What is Selenium?

Selenium is an open-source tool that is used for automating the tests carried out on web browsers.

Pros:

Open Source

Supports Parallel Execution

Supports different frameworks(TestNG, JUNIT, Cucumber)



- Supports different PL → [Java](#), [Python](#), [C#](#), [PHP](#), [Ruby](#), [Perl](#), [JavaScript](#)
- Supports different OS → [Windows](#), [Mac](#), [Linux](#), [iOS](#), [Android](#)
- Supports different Browsers → [IE](#), [Firefox](#), [Chrome](#), [Safari](#), [Opera](#)

What not to Automate?

User experience tests for usability

Tests that you will only run one-time.

Test that need to run ASAP

Tests without predictable results

Test that cannot be 100% automated should not be automated at all — unless doing so will save a considerable amount of time.

Test Cases for which the requirements are frequently changing

What is Selenium?

Cons:

We can use Selenium only to test web applications.

It is not possible to perform testing on images.

There is no native reporting facility.

Only User Community Support is available

Reference Links

- <https://www.perfecto.io/blog/what-is-test-automation>
- <https://blog.vsoftconsulting.com/blog/open-source-vs-commercial-test-automation-tools>
- <https://www.indiumsoftware.com/blog/selenium-test-automation/#:~:text=Selenium%20has%20a%20large%20and,choice%20for%20many%20testing%20projects.>
- <https://www.javatpoint.com/selenium-webdriver>
- <https://www.javatpoint.com/selenium-webdriver-installation>
- <https://www.javatpoint.com/selenium-grid>
- <https://www.javatpoint.com/selenium-ide>

Day - 16

Selenium Basics

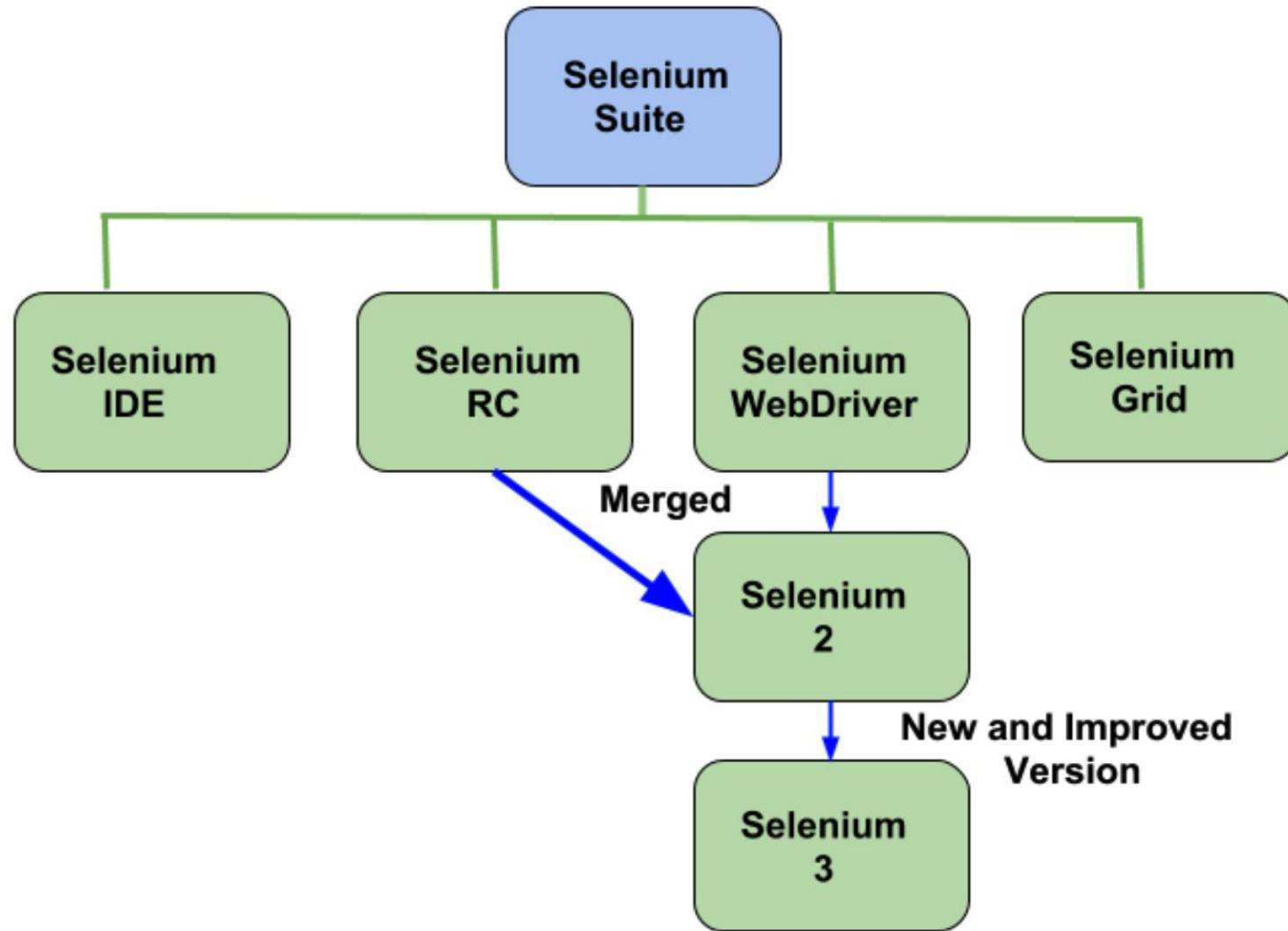
Selenium Basics

Introduction to Selenium
Selenium IDE
Limitations of Selenium
Selenium Grid
Selenium Webdriver
Setup and Installation

Handson topics/details

- Setup environment for selenium WebDriver

Introduction to Selenium



Selenium IDE

Selenium IDE (Integrated Development Environment) is the major tool in the Selenium Suite.

It is a complete integrated development environment (IDE) for Selenium tests.

It is implemented as a Firefox Add-On and as a Chrome Extension.

It allows for recording, editing and debugging of functional tests.

It was previously known as Selenium Recorder.

Below is the link to add IDE to chrome

<https://chrome.google.com/webstore/detail/selenium-ide/mooikfkahbdckldjndioackbalphokd?hl=en>

Limitations of Selenium IDE

Not suitable for testing huge volume of data

Incapable of handling multiple windows

Connections with the database can not be tested

Does not support capturing of screenshots on test failures

No feature available for generating result reports

Selenium Grid

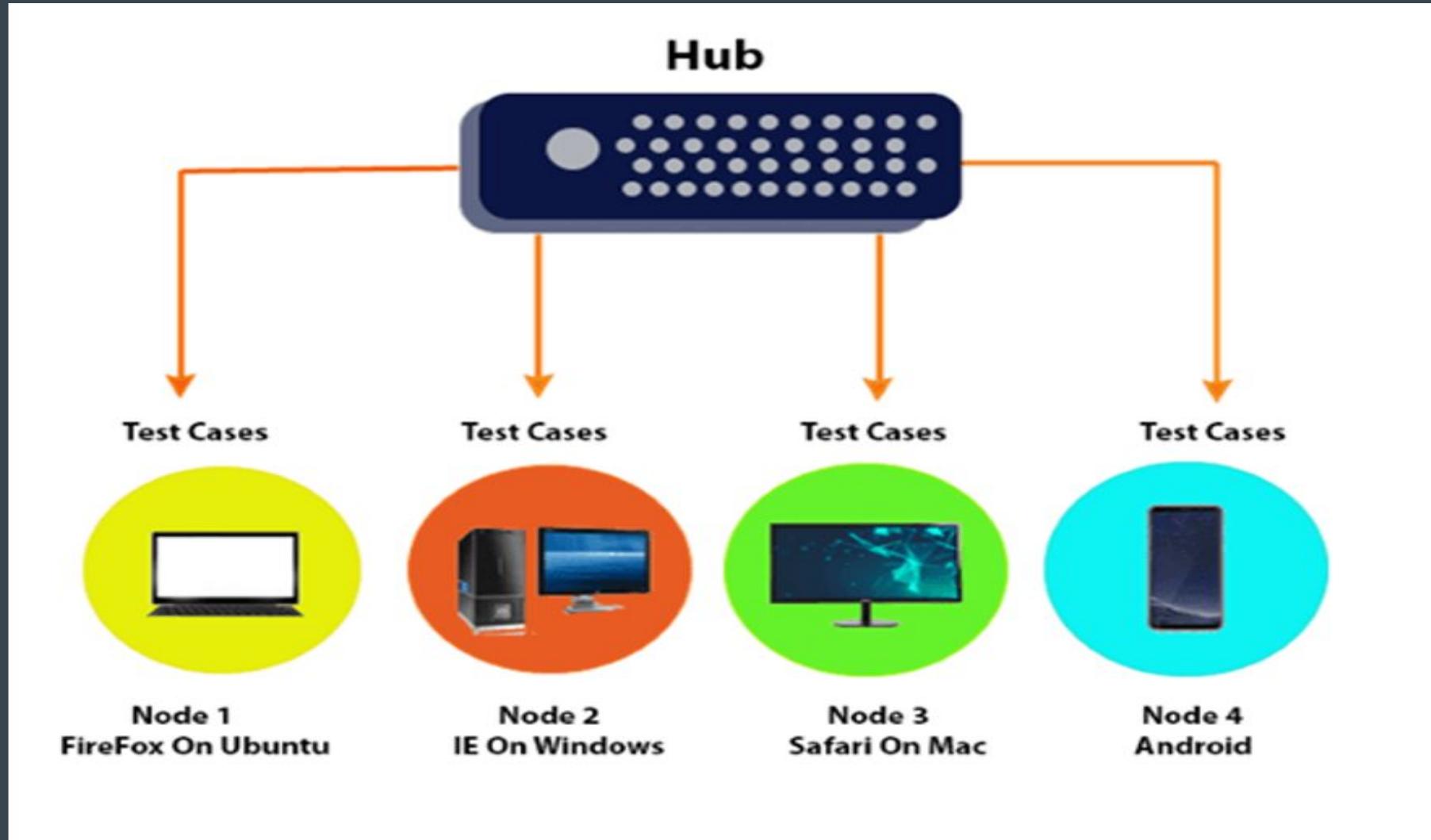
Selenium Grid is a component in Selenium that allows us to run test cases in different machines across different platforms.

It supports distributed test execution.

When we have multiple test cases, and it's not possible to run all the test cases in the same machine.

We have multiple servers, so local machine distributes the test cases across different machines/servers.

Selenium Grid



Selenium Grid

Hub

A Hub is a central point or a local machine that receives all the test requests and distributes them to the right nodes. The machine which actually triggers the test case known as Hub
There can be only one hub in a selenium grid.

Node

Nodes are the selenium instances which will execute the test cases that you loaded on the hub.
Nodes can be launched on multiple machines with different platforms and browsers.

Selenium Webdriver

Selenium Webdriver is an open-source used to automate functional and regression testing of web applications.

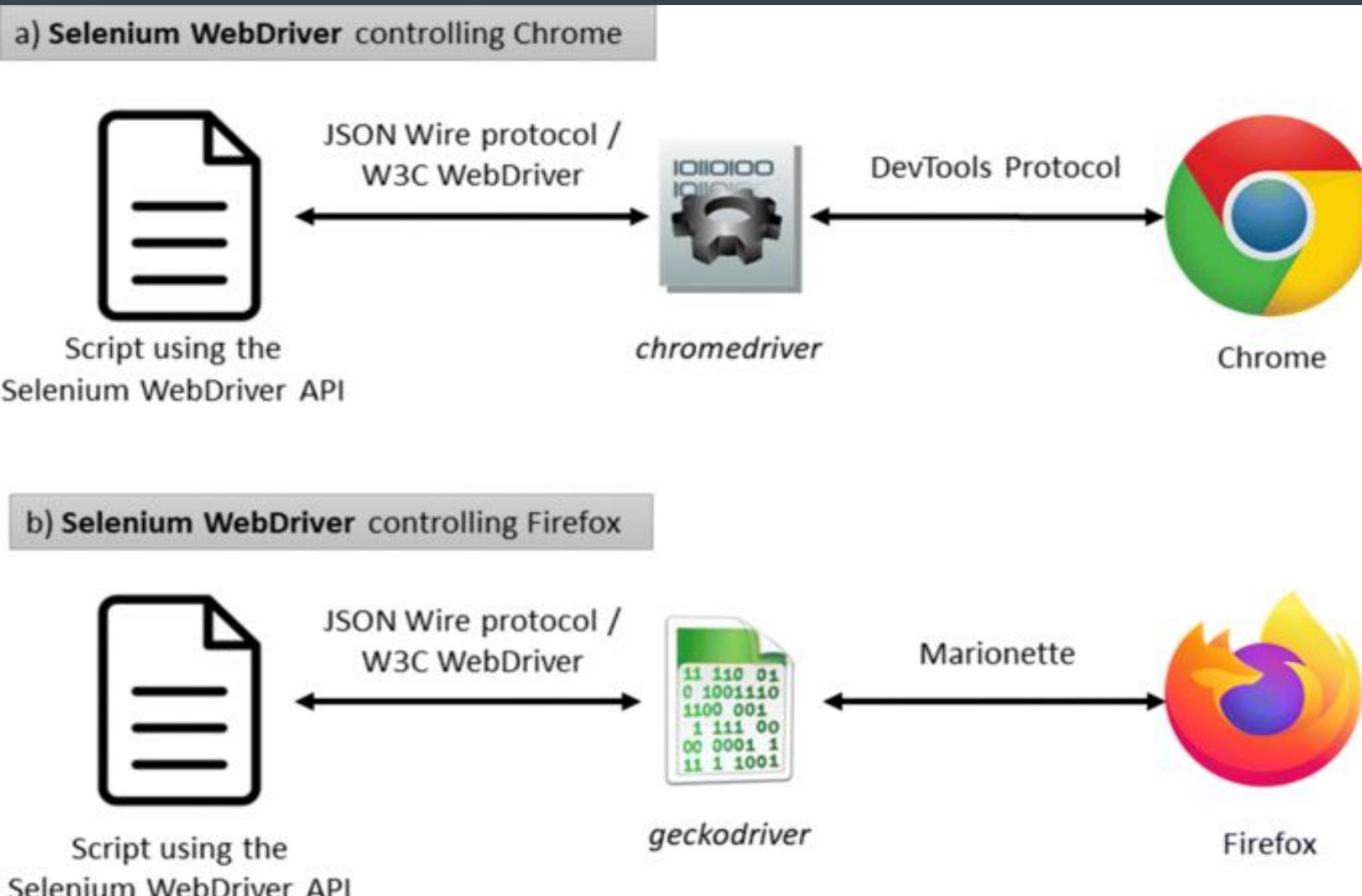
The Selenium Webdriver tool is used for automating web application testing to verify that it works as expected or not

It mainly supports browsers like Firefox, Chrome, Safari and Internet Explorer.

In WebDriver, test scripts can be developed using any of the supported programming languages and can be run directly in most modern web browsers. Languages supported by WebDriver include C#, Java, Perl, PHP, Python and Ruby.

WebDriver supports multiple browsers to run test. Browsers supported by WebDriver include Chrome, IE, firefox and safari.

Selenium Webdriver



Setup and Installation

Install Java

Download and Install IntelliJ

<https://www.jetbrains.com/idea/download/#section=Windows>

Download selenium drivers (chrome driver, firefox driver, IE driver)

<https://chromedriver.chromium.org/downloads> (chrome driver)

<https://github.com/mozilla/geckodriver/releases> (firefox driver)

Download selenium JARs

<https://www.selenium.dev/downloads/>

In above link go to downloads > selenium client > java > click on download link

Basic Example

Open IntelliJ

Create new project

Add Selenium JARs in the project (file > Project structure, Module< Dependencies > Add Jars)

Add chrome driver into the project

Open amazon.in site in chrome driver

```
System.setProperty("webdriver.chrome.driver", filePath);
```

```
WebDriver driver = new ChromeDriver();
```

```
driver.get("https://www.amazon.in/");
```

Close browser

```
driver.close();
```

Handson topics/details

Download and Install locators plugin into your browser and play around with the plugin to find different types of web elements

Write a program that open the browser with google search box. Enter your city and state (Ex. “Ahmedabad, Gujarat”) and search the entered text. Perform this steps in Chrome as well as Firefox browser.

Reference Links

- [How Selenium IDE works?](#)
- <https://www.toolsqa.com/selenium-webdriver/selenium-locators/>
- [Introduction to Firebug, Downloading and installing of Firebug](#)
- Downloading and installing of Xpath
(<https://www.toolsqa.com/selenium-webdriver/xpath-in-selenium/>)
(<https://www.toolsqa.com/selenium-webdriver/xpath-firebug-firepath/>)

Day - 17

Locators

Locators

Understanding of HTML DOM

What are Locators?

Locating by DOM (Document Object Model)

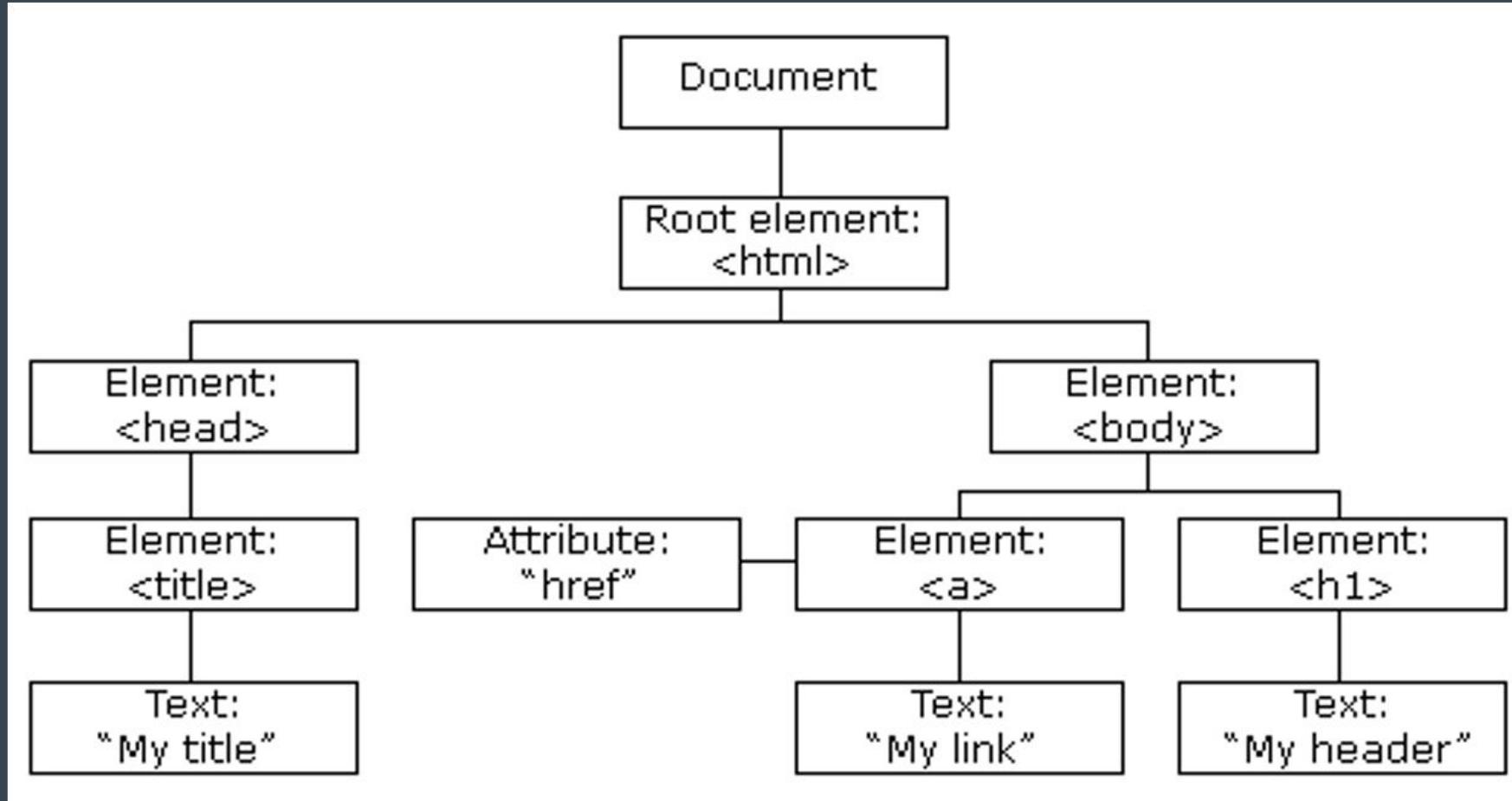
Locating by Webdriver

Syntax for Locating Elements

CSS Selector

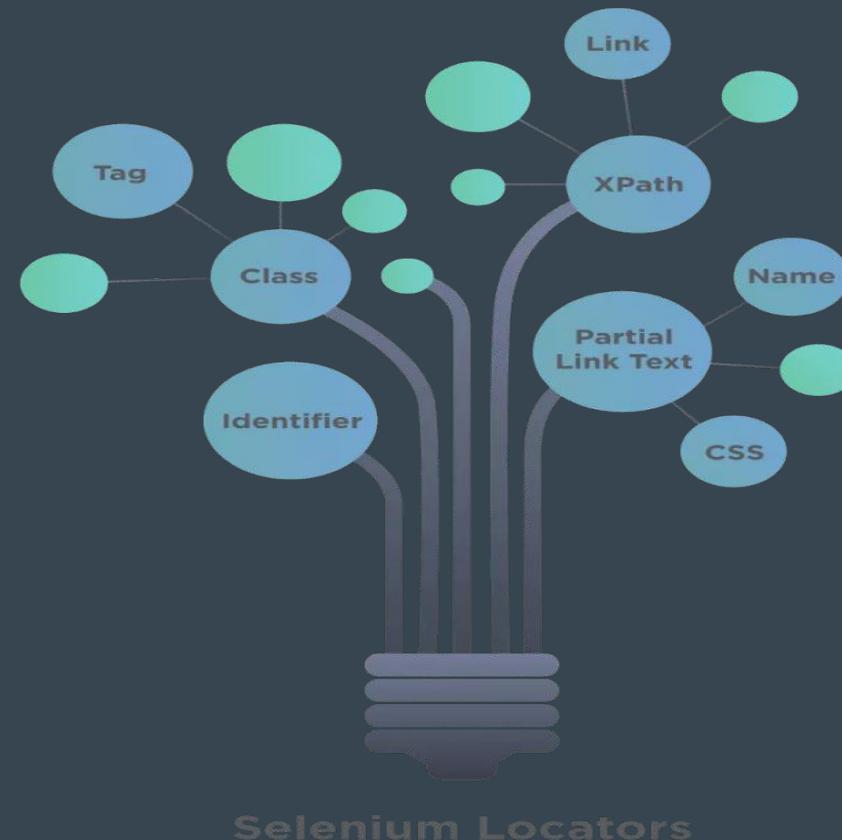
Understanding of HTML DOM

When a web page is loaded, the browser creates a Document Object Model of the page. The HTML DOM model is constructed as a tree of Objects



What are Locators? Types of Selenium Locators

Locators are one of the essential components of Selenium infrastructure, which help Selenium scripts in uniquely identifying the WebElements(such as text box, button, etc.) present of the web page



Locating by DOM (Document Object Model)

Method	Syntax	Usage
getElementById	document.getElementById("id of the element")	id of the element = this is the value of the ID attribute of the element to be accessed. This value should always be enclosed in a pair of parentheses ("").
getElementsByName	document.getElementsByName("name")[index]	name = name of the element as defined by its 'name' attribute index = an integer that indicates which element within getElementsByName's array will be used.
dom:name (applies only to elements within a named form)	document.forms["name of the form"].elements["name of the element"]	name of the form = the value of the name attribute of the form tag that contains the element you want to access name of the element = the value of the name attribute of the element you wish to access
dom:index	document.forms[index of the form].elements[index of the element]	index of the form = the index number (starting at 0) of the form with respect to the whole page index of the element = the index number (starting at 0) of the element with respect to the whole form that contains it

Locating by Webdriver

Locating elements in Selenium WebDriver is performed with the help of **findElement()** and **findElements()** methods provided by WebDriver and WebElement class.

findElement() returns a WebElement object based on a specified search criteria or ends up throwing an exception if it does not find any element matching the search criteria.

findElements() returns a list of WebElements matching the search criteria. If no elements are found, it returns an empty list.

Syntax for Locating Elements

By ID	driver.findElement(By.id (<element ID>))	Locates an element using the ID attribute
By name	driver.findElement(By.name (<element name>))	Locates an element using the Name attribute
By class name	driver.findElement(By.className (<element class>))	Locates an element using the Class attribute
By tag name	driver.findElement(By.tagName (<htmntagname>))	Locates an element using the HTML tag
By link text	driver.findElement(By.linkText (<linktext>))	Locates a link using link text
By partial link text	driver.findElement(By.partialLinkText (<linktext>))	Locates a link using the link's partial text
By CSS	driver.findElement(By.cssSelector (<css selector>))	Locates an element using the CSS selector
By XPath	driver.findElement(By.xpath (<xpath>))	Locates an element using XPath query

CSS Selector

Tag and ID	css=tag#id	tag = the HTML tag of the element being accessed # = the hash sign. This should always be present when using a Selenium CSS Selector with ID id = the ID of the element being accessed
Tag and class	css=tag.class	tag = the HTML tag of the element being accessed . = the dot sign. This should always be present when using a CSS Selector with class class = the class of the element being accessed
Tag and attribute	css=tag[attribute=value]	tag = the HTML tag of the element being accessed [and] = square brackets within which a specific attribute and its corresponding value will be placed attribute = the attribute to be used. It is advisable to use an attribute that is unique to the element such as a name or ID. value = the corresponding value of the chosen attribute.
Tag, class, and attribute	css=tag.class[attribute=value]	tag = the HTML tag of the element being accessed . = the dot sign. This should always be present when using a CSS Selector with class class = the class of the element being accessed [and] = square brackets within which a specific attribute and its corresponding value will be placed attribute = the attribute to be used. It is advisable to use an attribute that is unique to the element such as a name or ID. value = the corresponding value of the chosen attribute.

Handson topics/details

- Use the below webpages and play around to locate web elements.
- <http://demo.guru99.com/test/selenium-xpath.html>
- Identifying objects using ID, Name, Class, Tag Name, Link Text, Partial Link Text, XPath and CSS from <https://www.jquery.org/>
- Find all possible locators from <https://demoqa.com/automation-practice-form>

Reference Links

- <https://www.guru99.com/locators-in-selenium-ide.html>
- <https://www.toolsqa.com/selenium-webdriver/xpath-helper/>
- <https://medium.com/@lgusers123456789/pom-builder-v0-9-2-guideline-459d8ff4cd3>

Day - 18

Xpath

Xpath

Introduction to Xpath

XPath Types

Example of Absolute

Xpath

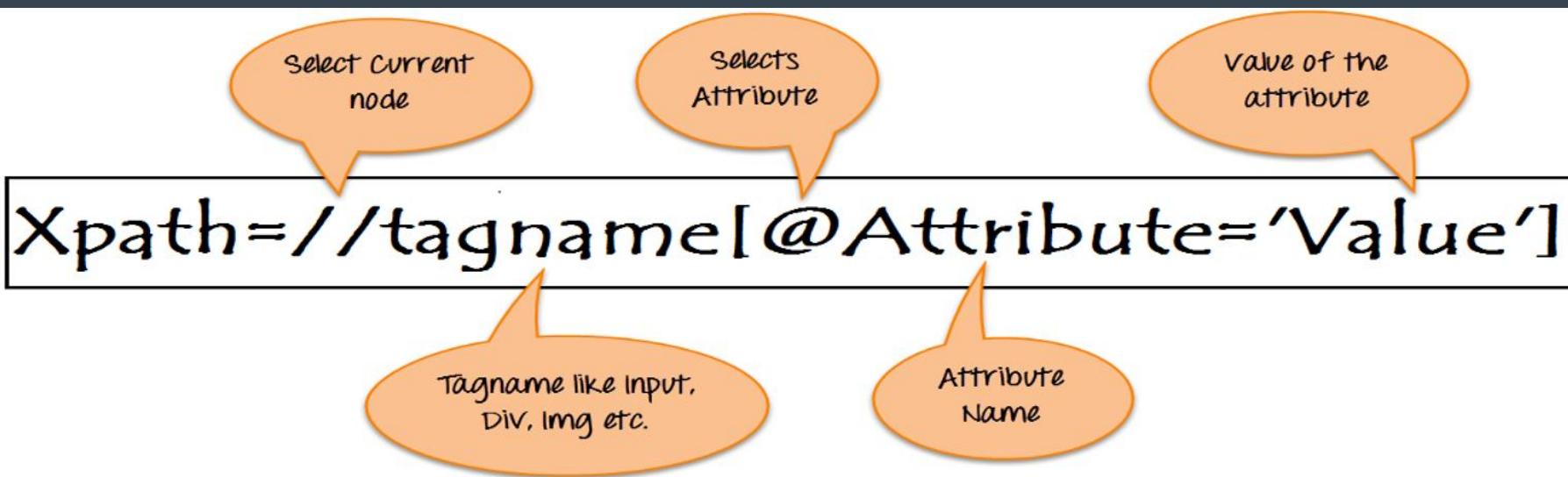
Example of Relative Xpath

Basic XPath

- Xpath Functions
- AND & OR
- Starts-with
- Text()
- Complex & Dynamic elements
- Best Practices For Selenium Locators

Introduction to Xpath

- XPath in Selenium is an XML path used for navigation through the HTML structure of the page. It is a syntax or language for finding any element on a web page using XML path expression. XPath can be used for both HTML and XML documents to find the location of any element on a webpage using HTML DOM structure.
- **Syntax:** Xpath=//tagname[@attribute='value']
 - // : Select current node.
 - **Tagname:** Tagname of the particular node.
 - @: Select attribute.
 - **Attribute:** Attribute name of the node.
 - **Value:** Value of the attribute.



XPath Types

- There are two types of XPath
- Absolute XPath
 - It is the direct way to find the element, but the disadvantage of the absolute XPath is that if there are any changes made in the path of the element then that XPath gets failed.
 - The key characteristic of XPath is that it begins with the single forward slash(/), which means you can select the element from the root node.
- Relative XPath
 - Relative Xpath starts from the middle of HTML DOM structure. It starts with double forward slash (//). It can search elements anywhere on the webpage, means no need to write a long xpath and you can start from the middle of HTML DOM structure. Relative Xpath is always preferred as it is not a complete path from the root element.

Example of Absolute Xpath

- Example of Absolute Xpath
 - /html/body/div[2]/div[1]/div/h4[1]/b/html[1]/body[1]/div[2]/div[1]/div[1]/h4[1]/b[1]

The screenshot shows a web page with three main sections: TESTING, SAP, and another partially visible section. The TESTING section contains links: Learn Software Testing, QTP (Quick Test Professional), Learn Selenium, and Learn Mobile App Testing. The SAP section contains links: Learn SAP Beginner, Learn SAP ABAP, Learn SAP HR/HCM, and Learn SAP FICO. Below the sections is a navigation bar with icons for Home, Back, Forward, Stop, Refresh, Console, HTML, CSS, Script, DOM, Net, Cookies, and FirePath. The FirePath tab is selected. The FirePath interface shows the DOM tree with the following absolute XPath path highlighted in red: html/body/div[1]/section/div[1]/div/div/div[1]/div/div/div/div[3]/div[1]/div/h4[1]/b. The path points to the bolded word 'Testing' in the first h4 element of the third div under the first section. A callout bubble labeled 'Element' points to the TESTING section header. Another callout bubble labeled 'Absolute Path' points to the highlighted XPath path in the FirePath tool. At the bottom left, a red box highlights the text '1 matching node'.

```
html/body/div[1]/section/div[1]/div/div/div[1]/div/div/div/div[3]/div[1]/div/h4[1]/b
```

1 matching node

Example of Relative Xpath

- Example of Relative Xpath
 - //div[@class='featured-box cloumnsize1']//h4[1]//b[1]

The screenshot illustrates the use of relative XPath in a web browser's developer tools. The page displays three columns of course offerings: TESTING, SAP, and Live Tests. The TESTING column has a heading 'TESTING' with sub-links for Software Testing, QTP, Selenium, and Mobile App Testing. The SAP column has links for SAP Beginner, ABAP, HR/HCM, and FICO. The Live Tests column has links for Test Selection, Selection, Ecc, and UFT.

The FirePath toolbar is open, showing various tabs like Console, HTML, CSS, Script, DOM, Net, Cookies, and FirePath. The FirePath tab is active, and the XPath input field contains the relative XPath expression: `//*[@class='featured-box']//*[text()='Testing']`. This expression selects the **Testing** heading under the TESTING column.

The resulting DOM tree on the right shows the selected node (**Testing**) highlighted in blue. The tree structure includes the row of featured boxes, the col-md-3 column, and the specific featured box for 'Testing'. The XPath expression is also highlighted in red in the input field.

A red box at the bottom left indicates "1 matching node".

Handling complex & Dynamic elements

- Examples:

- Xpath=//input[@type='text']
- Xpath=//label[@id='message23']
- Xpath=//input[@value='RESET']
- Xpath=//*[@class='barone']
- Xpath=//a[@href='http://demo.guru99.com/']
- Xpath= //img[@src='//cdn.guru99.com/images/home/java.png']

Basic XPath

- Basic XPath:
 - XPath expression select nodes or list of nodes on the basis of attributes like ID , Name, Classname, etc. from the XML document as illustrated below. e.g. Xpath=//input[@name='uid']

The screenshot shows a web browser window with a login form. The form includes fields for 'UserID' and 'Password', and buttons for 'LOGIN' and 'RESET'. A validation message 'User-ID must not be blank' is displayed next to the UserID field. An orange callout bubble labeled 'Element' points to the UserID input field.

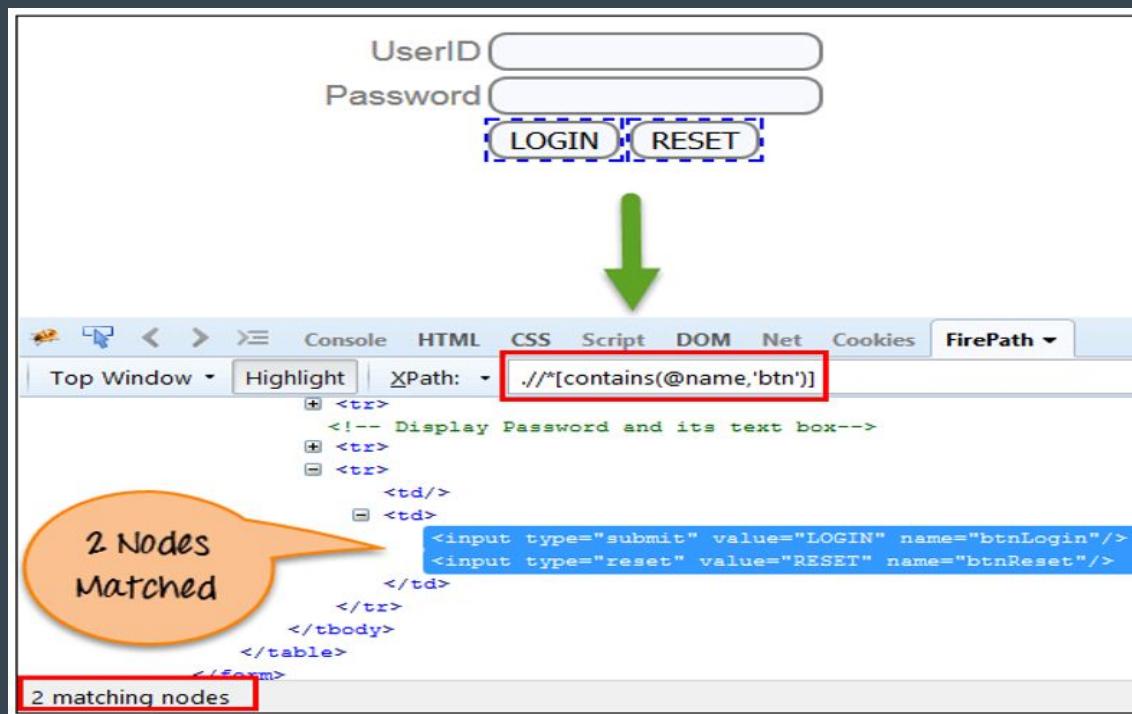
Below the browser is the FirePath developer tool interface. The 'XPath' dropdown shows the expression `//input[@name='uid']`. The tool's tree view highlights the corresponding HTML code in the DOM tree:

```
<!-- Display User ID label and its text box-->
<tbody>
  <tr>
    <td align="right">UserID</td>
    <td>
      <input type="text" onblur="validateuserid(); onkeyup="validateuserid(); maxlength="10" name="uid"/>
      <label id="message23" style="visibility: visible;">User-ID must not be blank</label>
    </td>
  </tr>
  <!-- Display Password and its text box-->
<tr>
<tr>
</tbody>
```

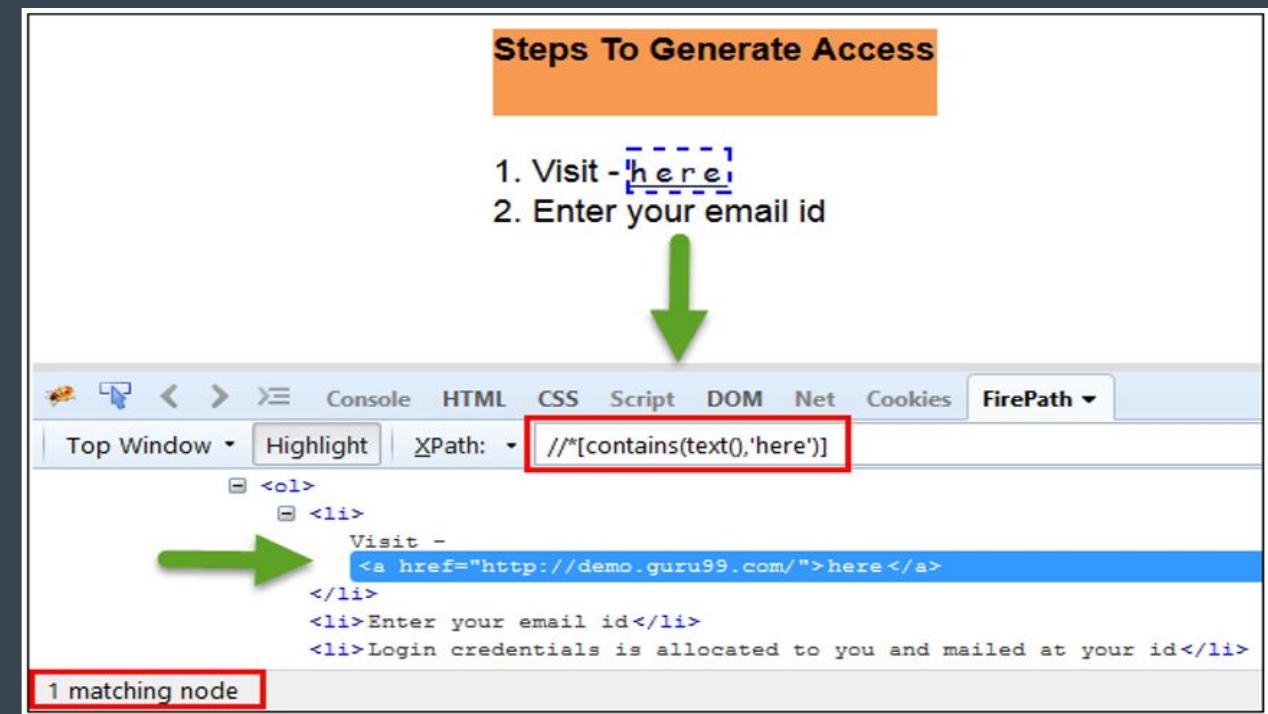
A large orange callout bubble labeled 'Basic xpath' points to the highlighted XPath expression in the FirePath interface. In the bottom left corner of the FirePath window, there is a red-bordered box containing the text '1 matching node'.

Xpath Functions

- Contains():
 - It is a method used in XPath expression. It is used when the value of any attribute changes dynamically, for example, login information. The contain feature has an ability to find the element with partial text



A screenshot of the FirePath tool interface. At the top, there are tabs: Console, HTML, CSS, Script, DOM, Net, Cookies, and FirePath. The FirePath tab is selected. Below the tabs, the URL is "Top Window". The "Highlight" tab is selected. The "XPath" field contains the expression `//*[contains(@name,'btn')]`. The DOM tree shows a table with two rows. The second row contains two td elements, each with an input type="submit" or type="reset" attribute. A green arrow points from the login form above to the DOM tree. A callout bubble says "2 Nodes Matched". In the bottom left corner of the tool window, it says "2 matching nodes".



A screenshot of the FirePath tool interface. At the top, there are tabs: Console, HTML, CSS, Script, DOM, Net, Cookies, and FirePath. The FirePath tab is selected. Below the tabs, the URL is "Top Window". The "Highlight" tab is selected. The "XPath" field contains the expression `//*[contains(text(),'here')]`. The DOM tree shows an ol with three li items. The first li item contains a link with the href "http://demo.guru99.com/" and the text "here". A green arrow points from the "Steps To Generate Access" text above to the DOM tree. A green arrow also points from the DOM tree to the "1 matching node" message at the bottom. A callout bubble says "Visit - here".

AND & OR

Gtpl Bank

User ID(*)

Password(*)

* mandatory field

LOGIN **RESET**

Elements Console Sources Network Performance Memory Application Security Audits

```
<br>
<form name="frmLogin" method="POST" action="index.php">
  <table border="0" width="50%" align="center">
    <!-- Display User ID label and its text box-->
    <tr>
      <td>
        <input type="submit" name="btnLogin" value="LOGIN">
        <input type="reset" name="btnReset" value="RESET">
      </td>
    </tr>
  </table>
</form>
```

tml.gr_demo_guru99_com body

//*[@type='submit' or @name='btnReset']

1 of 2 Cancel

User ID

Password

LOGIN **RESET**

DOM

XPath: //input[@type='submit' AND @name='btnLogin']

```
+ <tr>
  <td>
    <input type="submit" value="LOGIN" name="btnLogin"/>
    <input type="reset" value="RESET" name="btnReset"/>
  </td>
</tr>
</tbody>
```

1 matching node

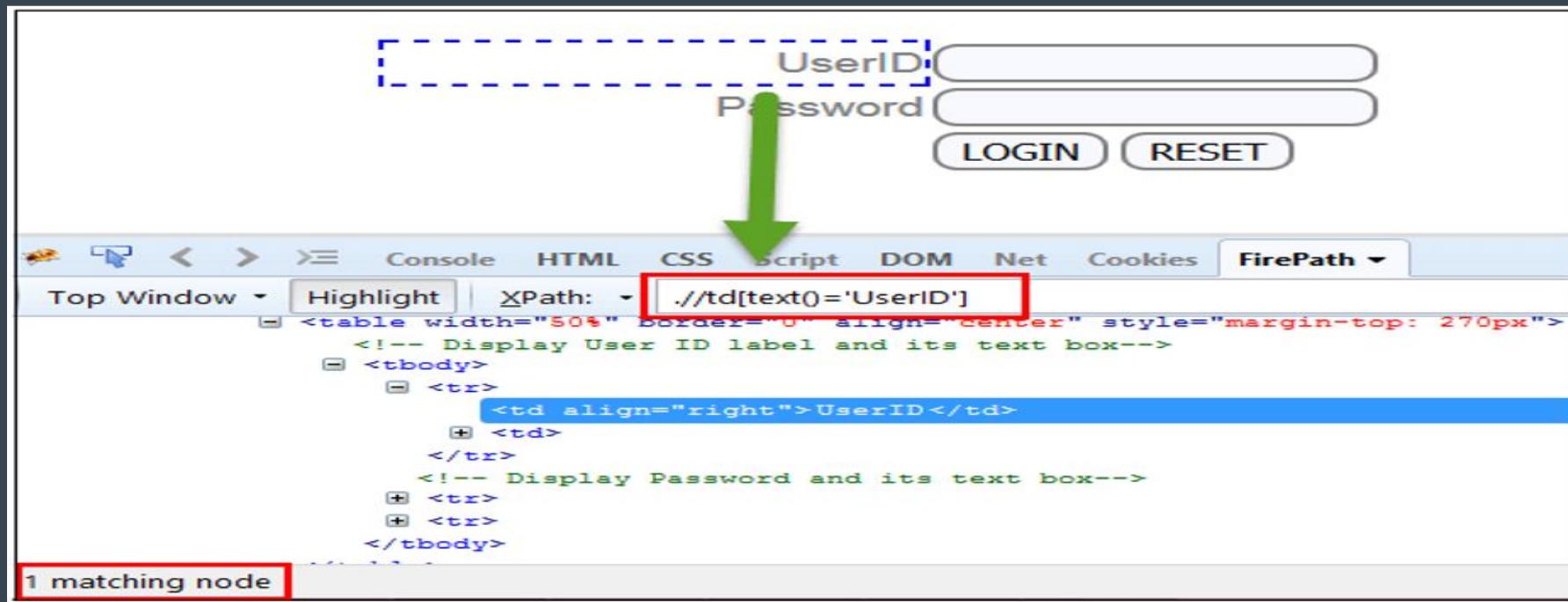
Starts-with

- Starts-with:
 - XPath starts-with() is a function used for finding the web element whose attribute value gets changed on refresh or by other dynamic operations on the webpage.



Text()

- Text():
 - The XPath text() function is a built-in function of selenium webdriver which is used to locate elements based on text of a web element. It helps to find the exact text elements and it locates the elements within the set of text nodes. The elements to be located should be in string form.



Complex & Dynamic elements

Following	<ul style="list-style-type: none"> • Xpath=//*[@type='text']//following::input • Xpath=//*[@type='text']//following::input[1] 	Selects all elements in the document following the current node()
Ancestor	<ul style="list-style-type: none"> • Xpath=//*[@text()='Enterprise Testing']//ancestor::div • Xpath=//*[@text()='Enterprise Testing']//ancestor::div[1] 	selects all ancestors element (grandparent, parent, etc.) of the current node
Child	<ul style="list-style-type: none"> • Xpath=//*[@id='java_technologies']//child::li • Xpath=//*[@id='java_technologies']//child::li[1] 	Selects all children elements of the current node
Preceding	<ul style="list-style-type: none"> • Xpath=//*[@type='submit']//preceding::input • Xpath=//*[@type='submit']//preceding::input[1] 	Select all nodes that come before the current node
Following-sibling	<ul style="list-style-type: none"> • xpath=//*[@type='submit']//following-sibling::input 	Select the following siblings of the context node.
Parent	<ul style="list-style-type: none"> • Xpath=//*[@id='rt-feature']//parent::div • Xpath=//*[@id='rt-feature']//parent::div[1] 	Selects the parent of the current node
Self	<ul style="list-style-type: none"> • Xpath =//*[@type='password']//self::input 	Selects the current node or 'self' means it indicates the node itself
Descendant	<ul style="list-style-type: none"> • Xpath=//*[@id='rt-feature']//descendant::a 	Selects the descendants of the current node(child node , grandchild node, etc.)

Following-sibling

The screenshot shows a web browser interface with developer tools open, specifically the FirePath plugin for Firefox. The FirePath toolbar includes icons for selection, navigation, and highlighting, followed by tabs for Console, CSS, Script, DOM, Net, Cookies, and FirePath (which is currently selected). Below the toolbar is a dropdown menu set to 'Top Window' and a 'Highlight' button. The main area displays an HTML tree on the left and the FirePath Xpath expression on the right.

Xpath using following-sibling

UserID
Password

LOGIN **RESET**

I Nodes matched

XPath: `//*[@type='submit']//following-sibling::input`

`<tr>
 <td>
 <input type="submit" value="LOGIN" name="btnLogin"/>
 <input type="reset" value="RESET" name="btnReset"/>
 </td>
</tr>`

1 matching node

Following

Guru99 Bank

The screenshot shows the Guru99 Bank login page with a FirePath tool overlay. The page features a login form with fields for UserID and Password, and buttons for LOGIN and RESET. A callout bubble says "Showing 3 Nodes". The FirePath toolbar at the bottom has tabs for Console, HTML, CSS, Script, DOM, Net, Cookies, and FirePath. The FirePath tab is active, showing the XPath expression `//*[@type='text']//following::input`. The FirePath panel displays the DOM structure with the password input node highlighted.

The screenshot shows the Guru99 Bank login page with a FirePath tool overlay. The page features a login form with fields for UserID and Password, and buttons for LOGIN and RESET. A callout bubble says "Showing particular Node". The FirePath toolbar at the bottom has tabs for Console, HTML, CSS, Script, DOM, Net, Cookies, and FirePath. The FirePath tab is active, showing the XPath expression `//*[@type='text']//following::input[1]`. The FirePath panel displays the DOM structure with the password input node highlighted. The browser's developer tools are visible at the bottom, showing the selected element.

Ancestor

The screenshot shows a web interface with four main sections:

- TESTING**:
 - Learn Software Testing
 - QTP (Quick Test Professional)
 - Learn Selenium
 - Learn Mobile App Testing
 - Learn Cucumber Testing
 - Learn SoapUI
 - Learn Agile Testing
- SAP**:
 - Learn SAP Beginner
 - Learn SAP ABAP
 - Learn SAP ESR
 - Learn SAP SD
 - Learn SAP CRM
- LIVE PROJECTS**:
 - Live Testing Project
 - Live Selenium Project
 - Live Ecommerce Project
 - Live UFT Testing
 - Live HP ALM Exercise
 - Live Mobile Testing
 - Live Security Testing
- MUST LEARN!**:
 - Learn Excel Tutorials
 - Learn Accounting
 - Learn Ethical Hacking
 - Cloud Computing for Beginners
 - Learn Photoshop CC
 - Learn BigData
 - Learn Digital Marketing

An orange callout bubble contains the text "13 Nodes matched" and points to the SAP section. Another orange callout bubble contains the text "xpath using ancestor" and points to the FirePath toolbar, which displays the XPath expression `//*[text()='Enterprise Testing']//ancestor::div`.

```
<document>
  <html lang="en-gb" xml:lang="en-gb" slick-uniqueid="3">
    <head>
```

Child

The screenshot shows a web page with four main categories: TESTING, SAP, LIVE PROJECTS, and MUST LEARN!. Each category contains a list of items. A speech bubble points to the TESTING category with the text "xpath using child".

TESTING

- Learn Software Testing
- QTP (Quick Test Professional)
- Learn Selenium
- Learn Mobile App Testing
- Learn Cucumber Testing
- Learn SoapUI
- Learn Agile Testing

TEST MANAGEMENT

- Learn HP Quality Center/ALM
- Learn Test Management

SAP

- Learn SAP Beginner
- Learn SAP ABAP
- Learn SAP HR/HCM
- Learn SAP FICO
- Learn SAP Basis
- Learn SAP SD
- Learn SAP CRM
- Learn SAP MM
- Learn SAP CO
- Learn SAP Payroll

LIVE PROJECTS

- Live Testing Project
- Live Selenium Project
- Live Ecommerce Project
- Live UFT Testing
- Live IIP ALM Exercise
- Live Mobile Testing
- Live Security Testing
- Live PHP Project
- Live Scrum(Agile) Testing
- Live Insurance testing

MUST LEARN!

- Learn Excel Tutorials
- Learn Accounting
- Learn Ethical Hacking
- Cloud Computing for Beginners
- Learn Photoshop CC
- Learn BigData
- Learn Digital Marketing
- Learn Business Analyst
- Learn Informatica
- Learn Project Management

Xpath using child

Console HTML CSS Script DOM Net Cookies FirePath ▾

Top Window ▾ Highlight XPath: //*[@id='java_technologies']/child::li

```
+ <h4>
  <ul id="java_technologies" class="menu">
    + <li>
    + <li>
  </ul>
```

71 matching nodes

Preceding

The screenshot shows a login interface with fields for UserID and Password, and buttons for LOGIN and RESET. The FirePath tool is open, displaying the DOM tree. A red box highlights the XPath expression `//*[@type='submit']//preceding::input`. An orange callout bubble points to this expression with the text "Xpath using preceding". Another orange callout bubble points to the highlighted input fields with the text "Showing 2 Nodes". At the bottom left, a red box highlights the message "2 matching nodes".

UserID
Password
LOGIN RESET

Showing 2 Nodes

Xpath using preceding

2 matching nodes

```
Top Window | Highlight | XPath: //*[@type='submit']//preceding::input
<!-- Display User ID label and its text box--&gt;
<tr>
 UserID |  |

<!-- Display Password and its text box--&gt;
<tr>
 Password |  |
```

Parent

A few of our most popular courses

- SELENIUM
- JAVA
- QTP
- SAP Beginners
- Linux
- Test Management

FirePath

Top Window ▾ Highlight XPath: //*[@id='rt-feature']//parent::div

65 matching nodes

Xpath using parent

65 Nodes matched

```
<div id="rt-page-surround">
  <header id="rt-top-surround">
    <section id="rt-section">
      <div id="rt-showcase-surround">
        <div id="rt-transition">
          <div id="rt-mainbody-surround">
            <div id="rt-feature">
              <div class="rt-container">
```

Self

Guru99 Bank

User ID

Password

Showing 1 Node

Xpath using self

FirePath

XPath: `//*[@type='password']//self::input`

```
<tr>
  <td align="right"> Password </td>
<td>
  <input type="password" onblur="validatepassword(); onkeyup="validatepassword(); name="password"/>
  <label id="message18"/>
</td>
```

Descendant

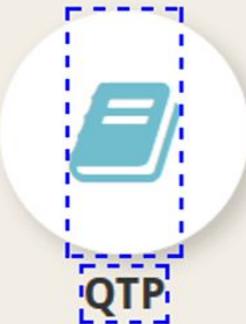
A few of our most popular courses



SELENIUM



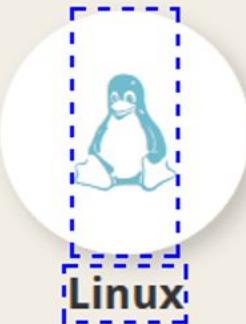
JAVA



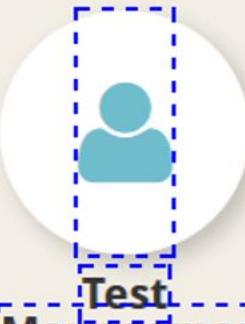
QTP



SAP
Beginners



Linux



Test
Management

Xpath using descendant

12 matching nodes

Top Window ▾ Highlight XPath: //*[@id='rt-feature']//descendant::a

12 Nodes matched

```
<canvas height="130px" width="125px"/>
[+] <div class="canvas-middle" style="width: 125px; height: 125px; margin-top: -95px;">
[+]   <a href="/selenium-tutorial.html">
[+]     <h4>
[+]       <a style="color:#343434;" href="/selenium-tutorial.html">SELENIUM </a>
[-]     </h4>
```

Best Practices For Selenium Locators

- ID and name attributes take precedence over other locators if your web page contains unique ID and name, then it's always advisable to use them instead of XPath as they are faster and more efficient.
- Do not use dynamic attribute values to locate elements, as they may change frequently and result in breakage of locator script. e.g. td1, td2
- While using locators, make sure that your locator points precisely to the required element. If the needed scenario needs to perform some operation on a single element, then make sure that your locator exactly matches to only one element. If the locator points to several different elements, then it may cause breakage in your script.
- While working with XPath or CSS locators, one should avoid directly using the one generated by the Chrome Dev Tools. It may seem one of the easiest ways to generate XPath, but in the long run, it induces reliability issues, code breakage, maintainability issues, etc. It may look tempting to use these, but you would be better off creating your customized XPath in the longer run.

Handson topics/details

Identify web elements only via x-path using all of the function that was taught during training

<http://demo.guru99.com/test/selenium-xpath.html>

Reference Links

- <https://www.guru99.com/xpath-selenium.html>
- [XPath and CSS selectors notes](#)

Day - 19

Selenium WebDriver (Part - 1)

Selenium WebDriver (Part - 1)

- Drivers and Configurations
- Webdriver APIs, Classes and Methods
- Synchronization -Explicit Wait and Implicit Wait
- Expected Conditions
- Iframes

WebDriver

WebDriver drives a browser natively, as a user would.

Selenium WebDriver refers to both the language bindings and the implementations of the individual browser controlling code. This is commonly referred to as just WebDriver.

WebDriver is a W3C Recommendation

WebDriver is designed as a simple and more concise programming interface.

WebDriver is a compact object-oriented API.

It drives the browser effectively.

Terminology

API: Application Programming Interface. This is the set of “commands” you use to manipulate WebDriver.

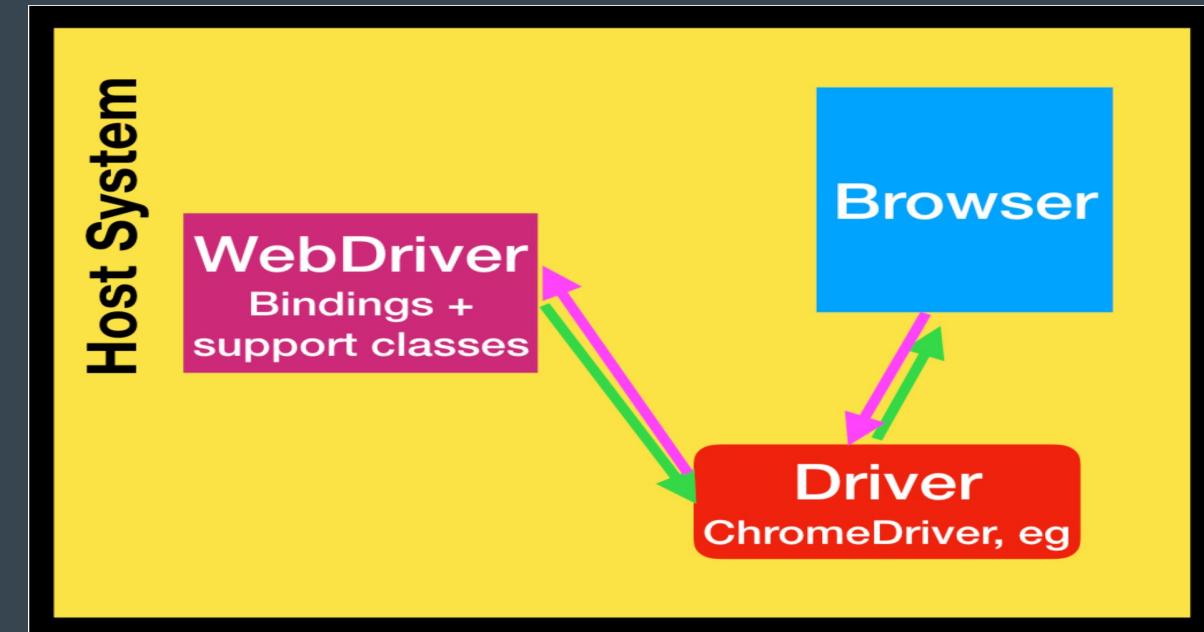
Library: A code module which contains the APIs and the code necessary to implement them. Libraries are specific to each language binding, eg .jar files for Java, .dll files for .NET, etc.

Driver: Responsible for controlling the actual browser. Most drivers are created by the browser vendors themselves.

Framework: An additional library used as a support for WebDriver suites. These frameworks may be test frameworks such as JUnit or NUnit.

Parts and Pieces

The driver is specific to the browser, such as ChromeDriver for Google's Chrome
WebDriver has one job and one job only: communicate with the browser via Driver.
WebDriver does not know a thing about testing
WebDriver passes commands to the browser through the driver, and receives
information back via the same route.



WebDriver Methods

Modifier and Type	Method and Description
void	close() Close the current window, quitting the browser if it's the last window currently open.
WebElement	findElement(By by) Find the first WebElement using the given method.
java.util.List<WebElement>	findElements(By by) Find all elements within the current page using the given mechanism.
void	get(java.lang.String url) Load a new web page in the current browser window.
java.lang.String	getCurrentUrl() Get a string representing the current URL that the browser is looking at.
java.lang.String	getPageSource() Get the source of the last loaded page.
java.lang.String	getTitle() The title of the current page.
java.lang.String	getWindowHandle() Return an opaque handle to this window that uniquely identifies it within this driver instance.
java.util.Set<java.lang.String>	getWindowHandles() Return a set of window handles which can be used to iterate over all open windows of this WebDriver instance by passing them to switchTo().WebDriver.Options.window()
WebDriver.Options	manage() Gets the Option interface
WebDriver.Navigation	navigate() An abstraction allowing the driver to access the browser's history and to navigate to a given URL.
void	quit() Quits this driver, closing every associated window.
WebDriver.TargetLocator	switchTo() Send future commands to a different frame or window.

API Commands and Operations

Open a page

Get driver object

Navigate to another page

Get current URL

Go back

Go forward

Refresh

Get title

Quit browser

Waits

What are wait commands ?

- Waits are commands in Selenium that are very important for executing automated test scripts.
- During automated testing of websites, issues may occur due to variations in time lag for loading web elements. Wait commands help observe and troubleshoot these issues.

Why use wait commands ?

- When a page loads on a browser, various web elements on it may load at different time intervals.
- Wait commands direct a test script to pause for a certain time before throwing an `ElementNotVisibleException`.

Types of wait commands

- Implicit Wait
- Explicit Wait
- Fluent Wait

Implicit Wait

Implicit Wait directs the Selenium WebDriver to wait for a certain measure of time before throwing an exception.

Once this time is set, WebDriver will wait for the element before the exception occurs.

Specifies the amount of time the driver should wait when searching for an element if it is not immediately present

It works only for searching an element

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

Explicit Wait

Using Explicit Wait command, the WebDriver is directed to wait until a certain condition occurs before proceeding with executing the code.

Setting Explicit Wait is important in cases where there are certain elements that naturally take more time to load.

Explicit wait is more intelligent, but can only be applied for specified elements.

In order to declare explicit wait, one has to use “ExpectedConditions”.

It allows to wait until element can be interacted with.

Explicit Wait

```
@BeforeMethod

public void setup() throws InterruptedException {

    // initializing driver variable using FirefoxDriver
    driver=new FirefoxDriver();

    // launching gmail.com on the browser
    driver.get("https://gmail.com");

    // maximized the browser window
    driver.manage().window().maximize();

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

}

@Test

public void test() throws InterruptedException {
    // saving the GUI element reference into a "element" variable of WebElement type
}
```

Fluent Wait

The Fluent Wait command defines the maximum amount of time for Selenium WebDriver to wait for a certain condition to appear.

It also defines the frequency with which WebDriver will check if the condition appears before throwing the “ElementNotVisibleException”.

It looks for a web element repeatedly at regular intervals until timeout happens or until the object is found.

The user can configure the wait to ignore any exceptions during the polling period.

Fluent Wait

Syntax

```
Wait wait = new FluentWait(WebDriver reference)
    .withTimeout(timeout, SECONDS)
    .pollingEvery(timeout, SECONDS)
    .ignoring(Exception.class);

WebElement foo=wait.until(new Function<WebDriver, WebElement>() {
    public WebElement apply(WebDriver driver) {
        return driver.findElement(By.id("foo"));
    }
});
```

Example of Fluent Wait Command

```
//Declare and initialise a fluent wait
FluentWait wait = new FluentWait(driver);
//Specify the timeout of the wait
wait.withTimeout(5000, TimeUnit.MILLISECONDS);
//Specify polling time
wait.pollingEvery(250, TimeUnit.MILLISECONDS);
//Specify what exceptions to ignore
wait.ignoring(NoSuchElementException.class)

//This is how we specify the condition to wait on.
//This is what we will explore more in this chapter
wait.until(ExpectedConditions.alertIsPresent());
```

Expected Conditions

- alertIsPresent()
- elementSelectionStateToBe()
- elementToBeClickable()
- elementToBeSelected()
- frameToBeAvailableAndSwitchToIt()
- invisibilityOfTheElementLocated()
- invisibilityOfElementWithText()
- presenceOfAllElementsLocatedBy()
- presenceOfElementLocated()
- textToBePresentInElement()
- textToBePresentInElementLocated()
- textToBePresentInElementValue()
- titleIs()
- titleContains()
- visibilityOf()
- visibilityOfAllElements()
- visibilityOfAllElementsLocatedBy()
- visibilityOfElementLocated()

HTML iframe tags

```
<html>
<body>
  <div class="box">
    <iframe name="iframe1" id="IF1" height="50%" width="50%">
      <div align="left" style="border: 1px solid black; padding: 5px; margin-bottom: 10px;>
        <h3>BrowserStack</h3>
        <p>This is the first frame. It contains a sub-frame below it.</p>
      </div>
      <div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;>
        <h3>BrowserStack</h3>
        <p>This is the second frame. It contains a sub-frame below it.</p>
      </div>
    </div>
  </div>
</body>
</html>
```

```
List<WebElements> iframes = driver.findElements(By.tagName("iframe"));
```

²⁴⁷ The elements in the iframes cannot be found until switching to the specific  QTEST DATA SYSTEMS

Handling Iframes

Methods to switch back and forth from iframes.

`switchTo().frame(int frameNumber)`

`switchTo().frame(string frameName)`

`switchTo().frame(WebElement frameElement)`

`switchTo().defaultContent()`

Hands-on

Get the laptop name and price from 2 pages on the product listing on amazon.in, flipkart.com or croma.com

Check out any tea by entering dummy details on <http://www.practiceselenium.com/>

Go to <https://www.saucedemo.com/>

Create a automation project and perform the following operation

Login with the given user

Checkout any product

Reference Links

https://www.selenium.dev/documentation/en/webdriver/understanding_the_components/

<https://hackr.io/blog/what-is-selenium-webdriver>

<https://dzone.com/articles/30-apis-every-tester-must-know-about-in-selenium>

<https://www.javadoc.io/doc/org.seleniumhq.selenium/selenium-api/3.141.59/org/openqa/selenium/WebDriver.html>

<https://www.selenium.dev/selenium/docs/api/java/org/openqa/selenium/package-summary.html>

<https://www.browserstack.com/guide/wait-commands-in-selenium-webdriver>

<https://www.selenium.dev/selenium/docs/api/java/org/openqa/selenium/support/ui/ExpectedConditions.html>

<https://www.browserstack.com/guide/expectedconditions-in-selenium>

<https://www.browserstack.com/guide/handling-frames-in-selenium>

<https://www.softwaretestinghelp.com/handling-iframes-using-selenium/>

Selenium WebDriver (Part - 2)

- The Select support class
- Actions Class
- Javascript Executors
- Alerts and Window Handling
- Headless browser Testing -Ghost Driver

Select Class

The Select Class in Selenium is a method used to implement the HTML SELECT tag.

The html select tag provides helper methods to select and deselect the elements. Before handling dropdown in Selenium and controlling drop-down boxes, we must do following two things:

Import the package `org.openqa.selenium.support.ui.Select`

Instantiate the drop-down box as an object, Select in Selenium WebDriver

Select Methods

Modifier and Type	Method	Description
void	<code>deselectAll()</code>	Clear all selected entries.
void	<code>deselectByIndex(int index)</code>	Deselect the option at the given index.
void	<code>deselectByValue(java.lang.String value)</code>	Deselect all options that have a value matching the argument.
void	<code>deselectByVisibleText(java.lang.String text)</code>	Deselect all options that display text matching the argument.
boolean	<code>equals(java.lang.Object o)</code>	
<code>java.util.List<WebElement></code>	<code>getAllSelectedOptions()</code>	
<code>WebElement</code>	<code>getFirstSelectedOption()</code>	
<code>java.util.List<WebElement></code>	<code>getOptions()</code>	
<code>WebElement</code>	<code>getWrappedElement()</code>	
int	<code>hashCode()</code>	
boolean	<code>isMultiple()</code>	
void	<code>selectByIndex(int index)</code>	Select the option at the given index.
void	<code>selectByValue(java.lang.String value)</code>	Select all options that have a value matching the argument.
void	<code>selectByVisibleText(java.lang.String text)</code>	Select all options that display text matching the argument.

Examples

```
import org.openqa.selenium.support.ui.Select;

public class SelectClass {
    @Test
    public static void main(String[] args) throws InterruptedException {
        System.setProperty("webdriver.chrome.driver", "C:\\Users\\Vaishnavi\\Desktop\\chromedriver.exe");
        WebDriver driver = new ChromeDriver();
        driver.get("<a href=\"http://www.facebook.com\">http://www.facebook.com</a>");
        driver.manage().window().maximize();
        //js.executeScript("window.scrollBy(0,300)");
        WebElement month_dropdown = driver.findElement(By.id("day"));
        Select oSelect = new Select(month_dropdown);
        oSelect.selectByIndex(3);
        Thread.sleep(3000);
        WebElement year_yy = driver.findElement(By.id("year"));
        Select year_y = new Select(year_yy);
        year_y.selectByValue("2000");
        Thread.sleep(3000);
        WebElement month_m = driver.findElement(By.id("month"));
        Select month_d1 = new Select(month_m);
        month_d1.selectByVisibleText("Jul");
        driver.quit();
    }
}
```

Action Class

Actions Class in Selenium is a built-in feature provided by the selenium for handling keyboard and mouse events.

It includes various operations such as multiple events clicking by control key, drag and drop events and many more.

These operations from the action class are performed using the advanced user interaction API in Selenium Webdriver.

Import the package `org.openqa.selenium.interactions.Actions`

Action Class Methods

- **Mouse Actions:**
 - doubleClick(): Performs double click on the element
 - clickAndHold(): Performs long click on the mouse without releasing it
 - dragAndDrop(): Drags the element from one point and drops to another
 - moveToElement(): Shifts the mouse pointer to the center of the element
 - contextClick(): Performs right-click on the mouse
- **Keyboard Actions:**
 - sendKeys(): Sends a series of keys to the element
 - keyUp(): Performs key release
 - keyDown(): Performs keypress without release

Example

```
WebDriver driver = new FirefoxDriver();

String URL = "https://demoqa.com/droppable/";

driver.get(URL);

// It is always advisable to Maximize the window before performing DragNDrop action
driver.manage().window().maximize();

driver.manage().timeouts().implicitlyWait(10000, TimeUnit.MILLISECONDS);

//Actions class method to drag and drop
Actions builder = new Actions(driver);

WebElement from = driver.findElement(By.id("draggable"));

WebElement to = driver.findElement(By.id("droppable"));
//Perform drag and drop
builder.dragAndDrop(from, to).perform();

//verify text changed in to 'Drop here' box
String textTo = to.getText();

if(textTo.equals("Dropped!")) {
System.out.println("PASS: Source is dropped to target as expected");
} else {
System.out.println("FAIL: Source couldn't be dropped to target as expected");
}

driver.close();

}
```

Javascript Executor

JavaScriptExecutor is used to perform operations on a web page.

To use JavaScriptExecutor in Selenium scripts there is no need to install an addon or plugin.

```
import org.openqa.selenium.JavascriptExecutor;
```

```
public class JavaScriptExecutorClassDummy {
```

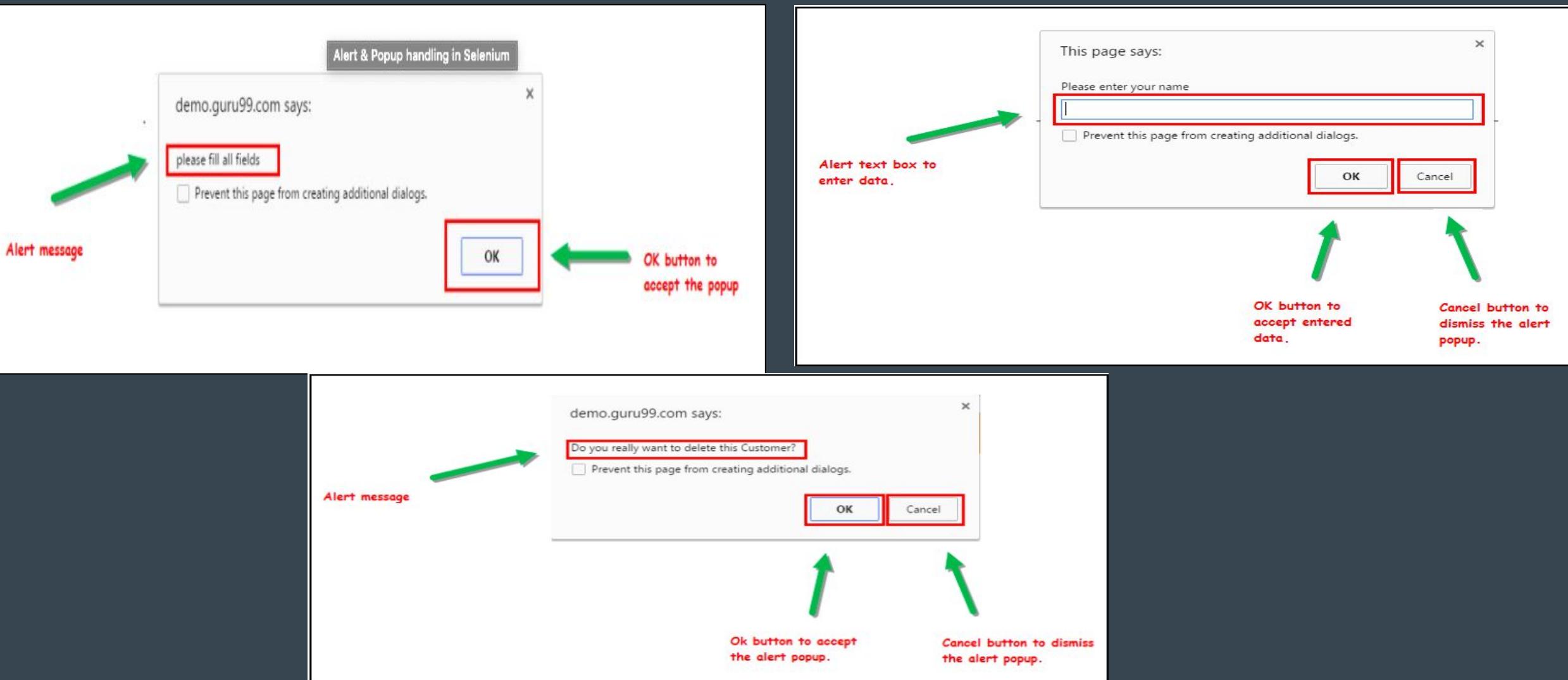
```
    static WebDriver driver;
```

```
@Test
```

```
    public static void javaScriptExeMethod(){
```

```
        System.setProperty("webdriver.gecko.driver", "D://Selenium Environment//Drivers//geckodriver.exe");
```

Alerts in Selenium



Alert Class Methods

- **void dismiss() // To click on the 'Cancel' button of the alert.**
 - `driver.switchTo().alert().dismiss();`
- **void accept() // To click on the 'OK' button of the alert.**
 - `driver.switchTo().alert().accept();`
- **String getText() // To capture the alert message.**
 - `driver.switchTo().alert().getText();`
- **void sendKeys(String stringToSend) // To send some data to alert box.**
 - `driver.switchTo().alert().sendKeys("Text");`

Handle popup in selenium

- In automation, when we have multiple windows in any web application, the activity may need to switch control among several windows from one to other in order to complete the operation
- **Driver.getWindowHandles();**
 - To handle all opened windows by web driver, we can use "Driver.getWindowHandles()" and then we can switch window from one window to another in a web application. Its return type is Iterator<String>.
- **Driver.getWindowHandle();**
 - When the site opens, we need to handle the main window by driver.getWindowHandle(). This will handle the current window that uniquely identifies it within this driver instance. Its return type is String.

Example

```
//Get handles of the windows
String mainWindowHandle = driver.getWindowHandle();
Set<String> allWindowHandles = driver.getWindowHandles();
Iterator<String> iterator = allWindowHandles.iterator();

// Here we will check if child window has other child windows and will fetch the
while (iterator.hasNext()) {
    String ChildWindow = iterator.next();
    if (!mainWindowHandle.equalsIgnoreCase(ChildWindow)) {
        driver.switchTo().window(ChildWindow);
```

Headless Browser

A headless browser is a browser which doesn't have a GUI.

Headless browser is used to simulate programs even though there is no browser installed on your local system.

You couldn't see any browser in your system but you will get the same result in your console.

ChromeOptions options = new ChromeOptions();options.

addArguments("--headless");

WebDriver driver = new ChromeDriver(options)

Hands-on

- Go to flipkart.com click on “Careers”. Search “Engineering” on next page
- Automate gmail composing a mail with a picture attachment
 - Practice drag and drop for all 3 tabs in <https://www.globalsqa.com/demo-site/draganddrop/>

Reference Links

- <https://www.selenium.dev/selenium/docs/api/java/index.html?org/openqa/selenium/support/ui>Select.html>
- <https://www.selenium.dev/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html>
- <https://www.browserstack.com/guide/action-class-in-selenium>
- <https://www.softwaretestingmaterial.com/javascriptexecutor-selenium-webdriver/>
- <https://www.softwaretestinghelp.com/handle-alerts-popups-selenium-webdriver-selenium-tutorial-16/>
- <https://www.softwaretestingmaterial.com/headless-browser-testing-using-selenium-webdriver/>
- <https://github.com/detro/ghostdriver>
- <https://www.softwaretestingmaterial.com/headless-browser-testing-using-phantomjsdriver/>

Day - 21

Automation Frameworks

Automation Frameworks

Data Driven Test Framework

Keyword Driven Test Framework

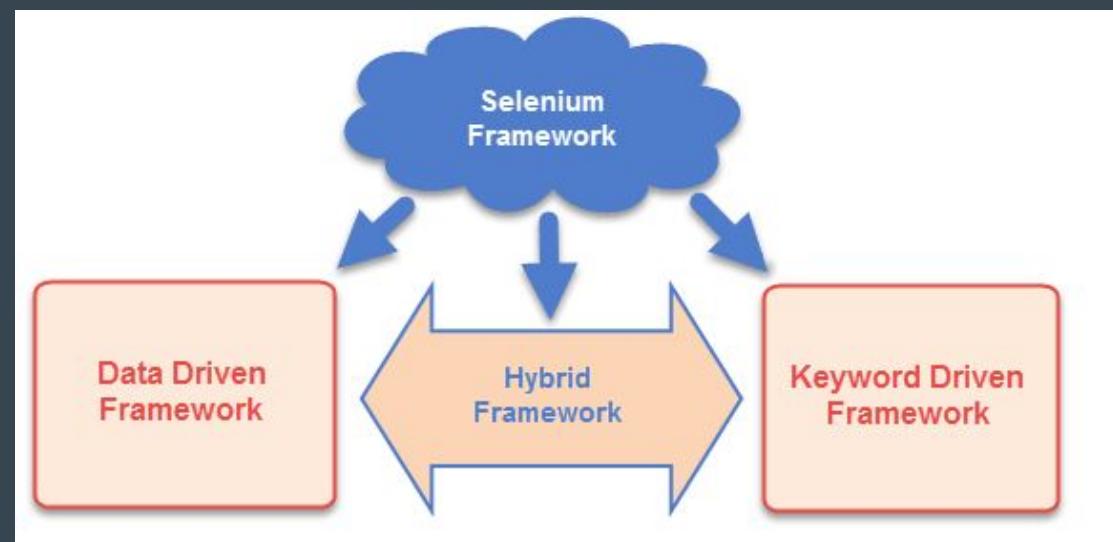
Hybrid Test Framework

What is a TEST Automation Framework?

A set of guidelines like coding standards , test-data handling , object repository treatment etc... which when followed during automation scripting produce beneficial outcomes like increased code re-usage , higher portability , reduced script maintenance cost etc.

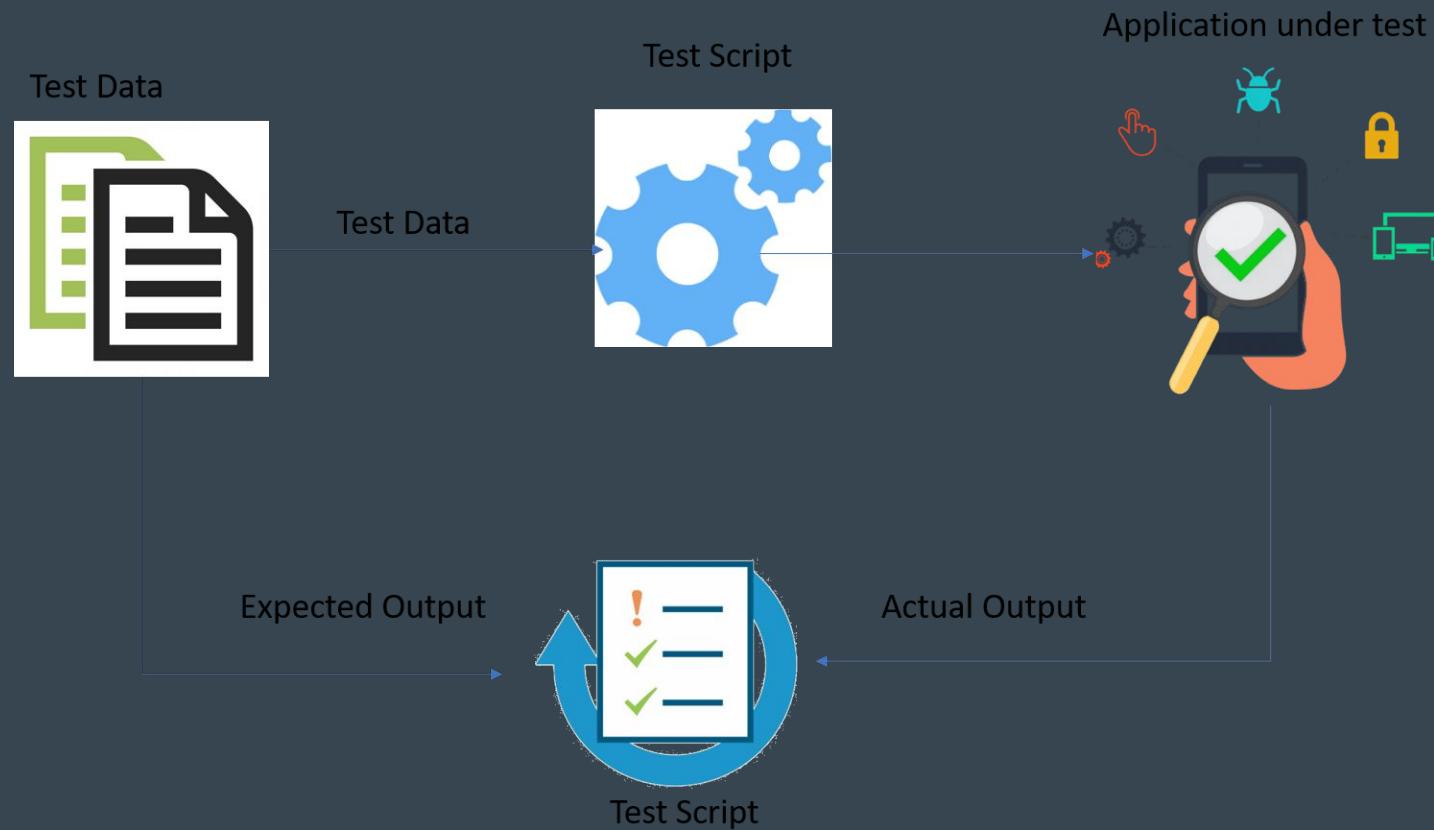
There are mainly three type of frameworks created by Selenium WebDriver to automate manual test cases

- Data Driven Test Framework
- Keyword Driven Test Framework
- Hybrid Test Framework



Data Driven Framework

Data Driven Framework in Selenium is a method of separating data sets from the test case. Once the data sets are separated from the test case, it can be easily modified for a specific functionality without changing the code. It is used to fetch test cases and suites from external files like Excel, .csv, .xml or some database tables.

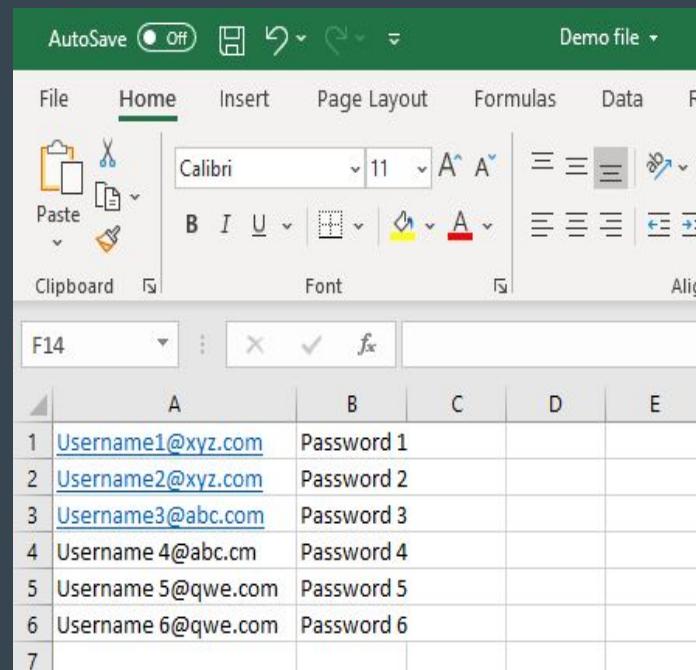


Data Driven Framework

Ex: Developing the Flight Reservation Login script using this method will involve two steps.

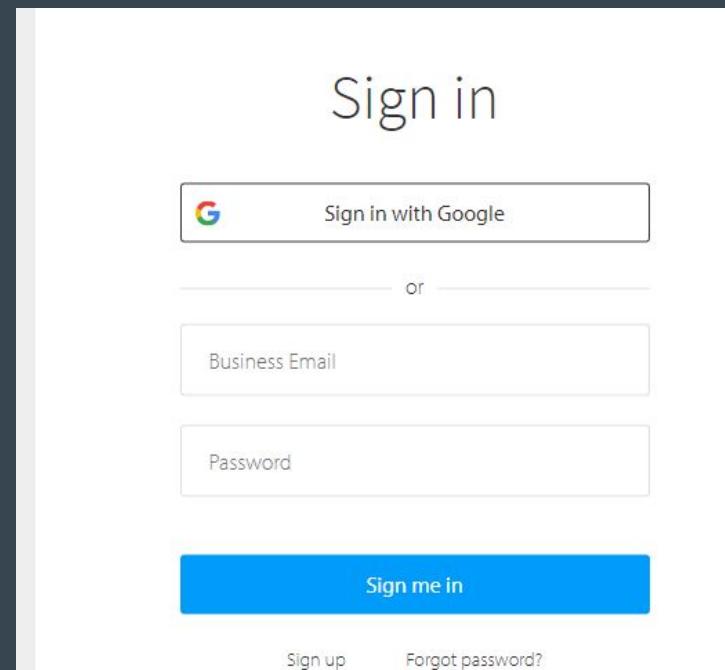
Step 1) Create a Test - Data file which could be Excel , CSV , or any other database source.

Step 2) Develop Test Script and make references to your Test- Data source.



A screenshot of Microsoft Excel showing a spreadsheet titled "Demo file". The spreadsheet contains a single sheet with data from row 1 to 7. The columns are labeled A through E. The data is as follows:

	A	B	C	D
1	Username1@xyz.com	Password 1		
2	Username2@xyz.com	Password 2		
3	Username3@abc.com	Password 3		
4	Username 4@abc.cm	Password 4		
5	Username 5@qwe.com	Password 5		
6	Username 6@qwe.com	Password 6		
7				



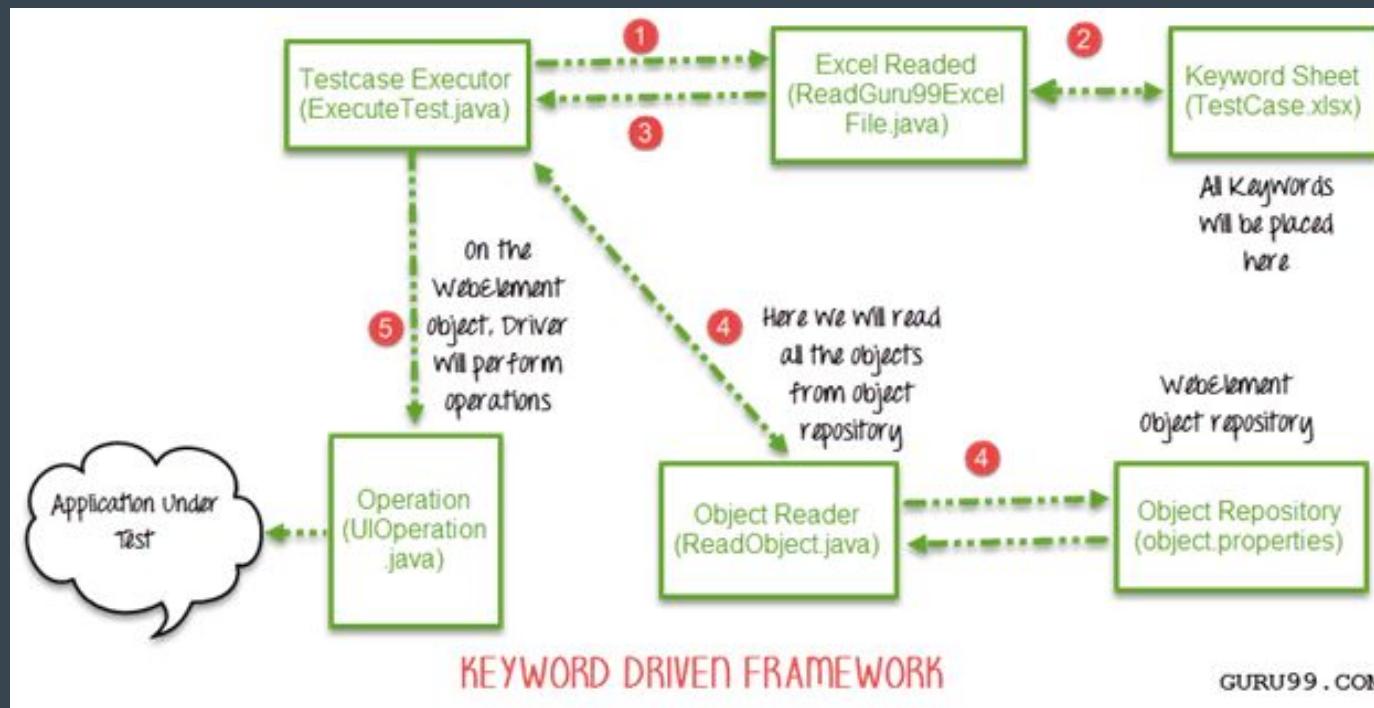
A screenshot of a "Sign in" page. At the top, it says "Sign in". Below that is a "Sign in with Google" button with the Google logo. Underneath the button is the word "or". There are two input fields: one for "Business Email" and one for "Password". At the bottom is a large blue "Sign me in" button. Below the button are links for "Sign up" and "Forgot password?".

Data Driven Framework

- **Advantages**
 - Changes to the Test Scripts do not affect the Test Data
 - Test Cases can be executed with multiple Sets of Data
 - A Variety of Test Scenarios can be executed by just varying the Test Data in the External Data File
- **Disadvantages**
 - More time is needed to plan and prepare both Test Scripts and Test Data

Keyword Driven Framework

Keyword Driven Framework in Selenium is a method used for speeding up automated testing by separating keywords for common set of functions and instructions. All the operations and instructions to be performed are written in some external file like an Excel sheet. Users can easily control and specify the functionalities

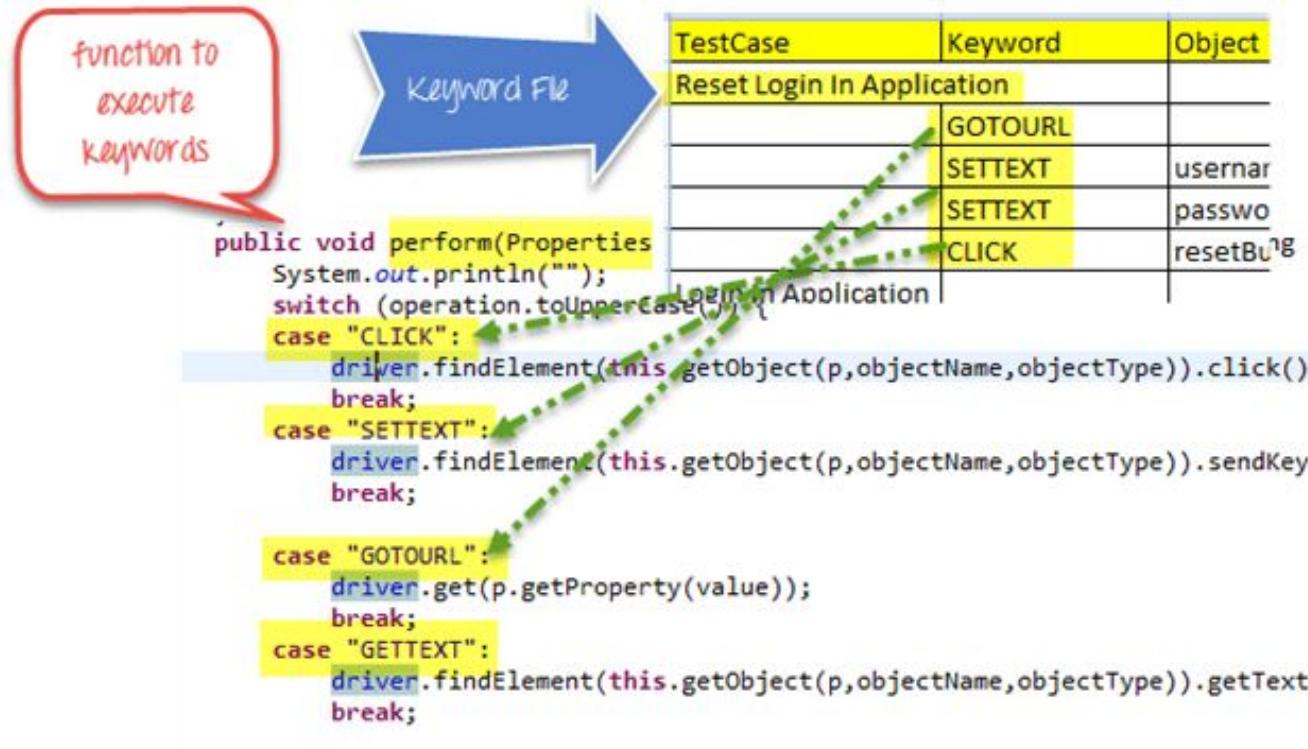


Keyword Driven Framework

Testcase Name	Keywords	Object Name	Object Type can be xpath , name,css etc.	Value for textbox area, url etc
TestCase	Keyword	Object	ObjectType	value
Reset Login In Application	GOTOURL SETTEXT SETTEXT CLICK	username password resetButton	name name name	url Demo testPassword
Login In Application	GOTOURL SETTEXT SETTEXT CLICK	username password loginButton	name name name	url Demo testPassword

Excel Sheet For Keyword Driven Test

Keyword Driven Framework



Keyword class

object.prop... UIOperation....

```
url=http://demo.guru99.com/V4/
username=uid
password=password
title=barone
loginButton=btnLogin
resetButton=btnReset|
```

This will be our object repository

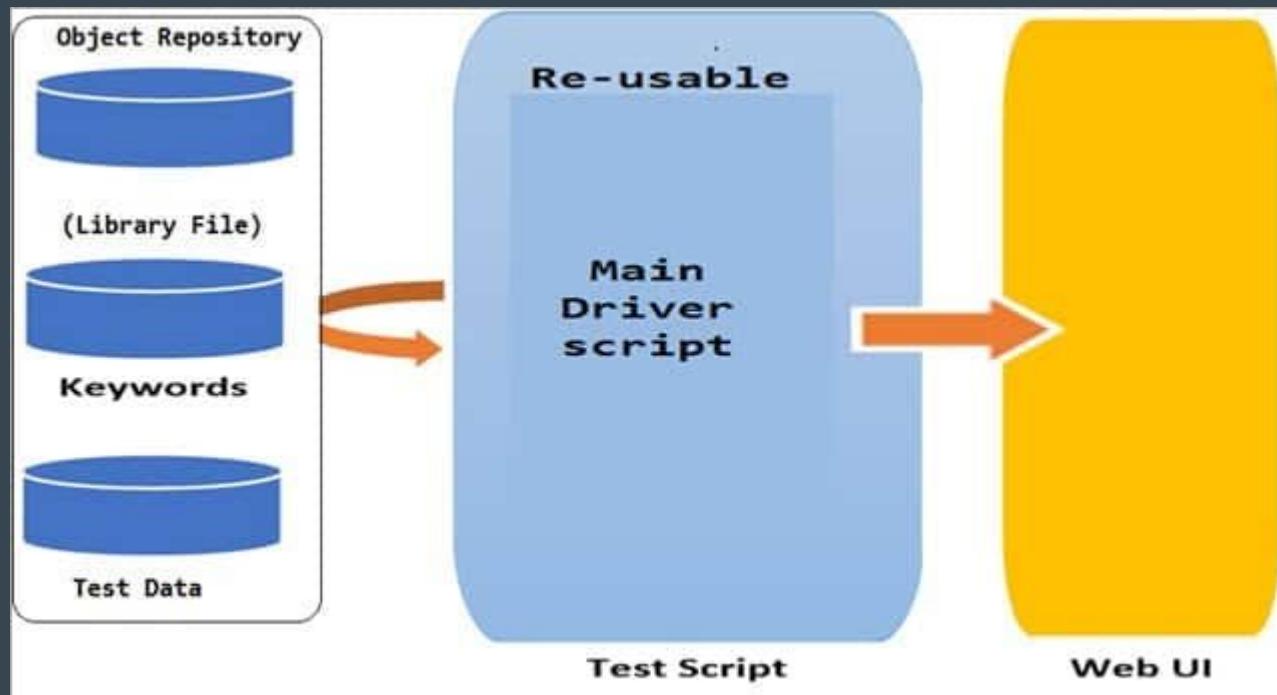
Object repository

Keyword Driven Framework

- **Advantages**
 - Provides high code reusability
 - Test Tool Independent
 - Independent of Application Under Test, the same script works for AUT (with some limitations)
- **Disadvantages**
 - Initial investment being pretty high, the benefits of this can only be realized if the application is considerably big and the test scripts are to be maintained for quite a few years.
 - High Automation expertise is required to create the Keyword Driven Framework.

Hybrid Framework

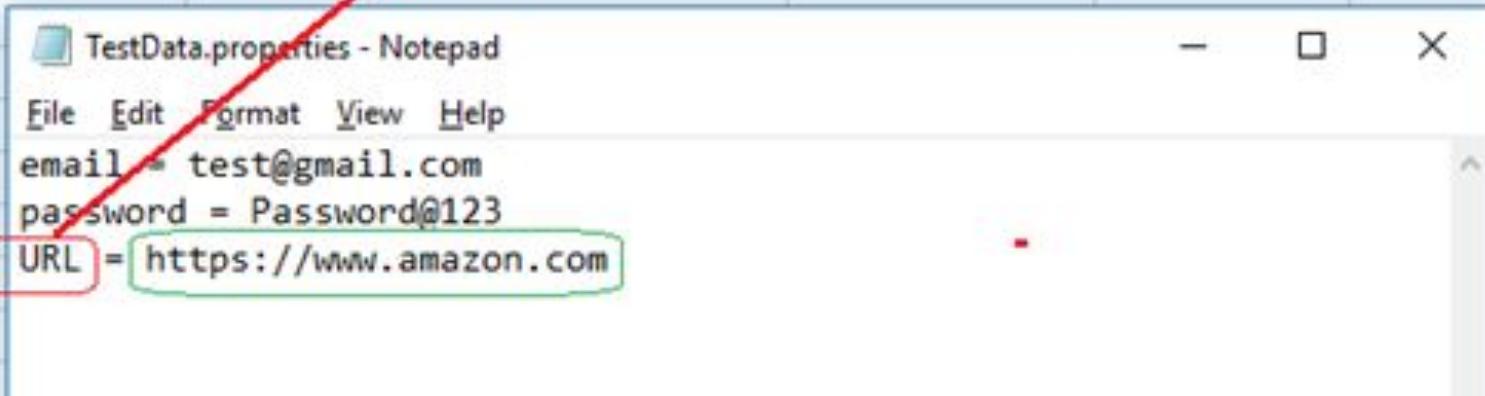
Hybrid Framework in Selenium is a concept where we are using the advantage of both Keyword driven framework as well as Data driven framework. It is an easy to use framework which allows manual testers to create test cases by just looking at the keywords, test data and object repository without coding in the framework.



Hybrid Framework

Externalizing Test Data from Test case Template

Test Steps	Locator Type	Locator Name	TestData	AssertionType	ExpectedValue
enter_URL			URL		
Click	xpath	LoginLink			
typeIn	xpath	EmailTextbox	email		
Click	xpath	ContinueButton			
typeIn	id	PasswordTextbox	password		
Click	id	SignInButton			
AssertElement	xpath	UserIcon		Displayed	



```
 TestData.properties - Notepad
File Edit Format View Help
email = test@gmail.com
password = Password@123
URL = https://www.amazon.com
```

Handson topics/details

- To Learn about the various types of framework and to present an idea to the tutor that when to use which type of framework with any software example

Reference Links

<https://www.browserstack.com/guide/hybrid-framework-in-selenium>

<https://www.browserstack.com/guide/keyword-driven-framework-in-selenium>

<https://www.browserstack.com/guide/data-driven-framework-in-selenium>

<https://www.guru99.com/creating-keyword-hybrid-frameworks-with-selenium.htm>

Day - 22

Cucumber Testing Tool

Cucumber Testing Tool

Introduction

BDD in Cucumber

Advantages

Gherkin Language

Why Gherkin?

Gherkin Syntax

Important Terms

Examples

Best practices

Introduction

- What is Cucumber?
 - Cucumber is a testing tool that supports Behavior Driven Development (BDD). It offers a way to write tests that anybody can understand, regardless of their technical knowledge. In BDD, users (business analysts, product owners) first write scenarios or acceptance tests that describe the behavior of the system from the customer's perspective, for review and sign-off by the product owners before developers write their codes. Cucumber framework uses Ruby programming language.

BDD in Cucumber

- How BDD works in Cucumber Automation?
 - Consider you are assigned to create Funds Transfer module in a Net Banking application.
 - There are multiple ways to test it in Cucumber Testing framework
 - Fund Transfer should take place if there is enough balance in source account
 - Fund Transfer should take place if the destination a/c details are correct
 - Fund Transfer should take place if transaction password / rsa code / security authentication for the transaction entered by user is correct
 - Fund Transfer should take place even if it's a Bank Holiday
 - Fund Transfer should take place on a future date as set by the account holder
 - The Test Scenario become more elaborate and complex as we consider additional features like transfer amount X for an interval Y days/months , stop schedule transfer when the total amount reaches Z , and so on
 - The general tendency of developers is to develop features and write test code later. As, evident in above case, Test Case development for this case is complex and developer will put off Testing till release , at which point he will do quick but ineffective testing.
 - To overcome this issue, Cucumber BDD (Behavior Driven Development), was conceived. It makes the entire testing process easy for a developer

BDD in Cucumber

In Cucumber BDD, whatever you write must go into Given-When-Then steps. Let's consider the same example above in BDD

```
Given that a fund transfer module in net banking application has been developed  
And I am accessing it with proper authentication
```

```
WhenI shall transfer with enough balance in my source account  
Or I shall transfer on a Bank Holiday  
Or I shall transfer on a future date  
And destination a/c details are correct  
And transaction password/RSA code/security authentication for the transaction is correct  
And press or click send button
```

```
Then amount must be transferred  
And the event will be logged in log file
```

Advantages

- It is helpful to involve business stakeholders who can't easily read code
- Cucumber Testing tool focuses on end-user experience
- Style of writing tests allow for easier reuse of code in the tests
- Quick and easy set up and execution
- Cucumber test tool is an efficient tool for testing

Gherkin Language

- What is Gherkin Language?
 - Gherkin is a business readable language which helps you to describe business behavior without going into details of implementation. It is a domain specific language for defining tests in Cucumber format for specifications. It uses plain language to describe use cases and allows users to remove logic details from behavior tests.
 - The text in Gherkin language acts as documentation and skeleton of your automated tests. Gherkin format is based on TreeTop Grammar which exists in 37+ languages. Therefore you can write your gherkin in 37+ spoken languages.
 - This script serves two primary purposes:
 - Documents user scenarios
 - Writing an automated test (BDD)

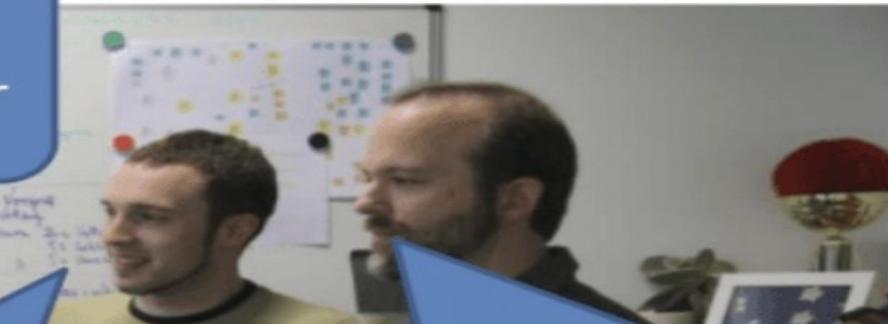
Why Gherkin?

Before Gherkin

We would like to encourage new users to buy in our shop.
Therefore we offer 10% discount for their first order.



```
public void CalculateDiscount(Order order)
{
    if (order.Customer.IsNew)
        order.FinalAmount =
            Math.Round(order.Total * 9/10);
}
```

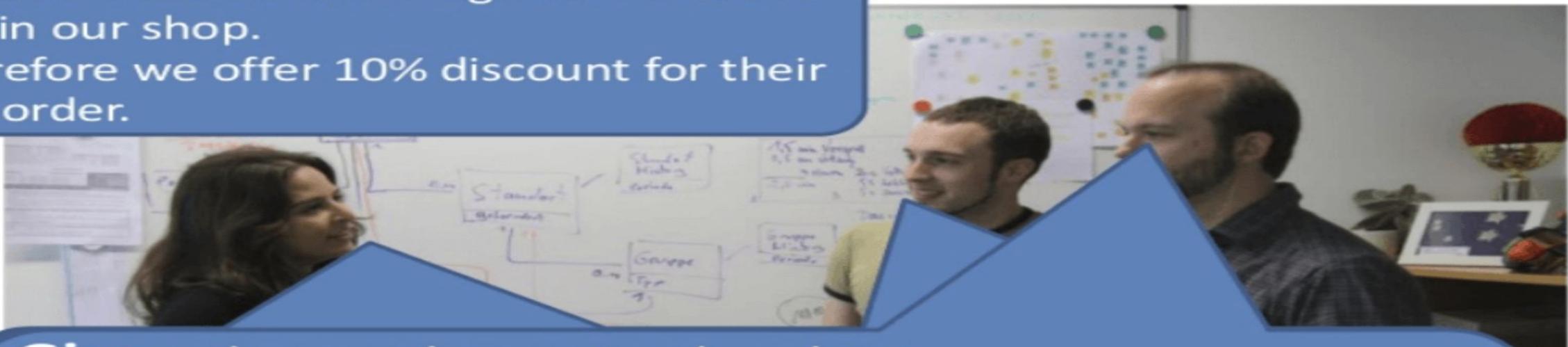


Register as “bart_bookworm”
Go to “/catalog/search”
Enter “ISBN-0955683610”
Click “Search”
Click “Add to Cart”
Click “View Cart”
Verify “Subtotal” is “\$33.75”

Why Gherkin?

After Gherkin

We would like to encourage new users to buy in our shop.
Therefore we offer 10% discount for their first order.



Given the user has not ordered yet

When the user adds a book with the price of EUR 37.5 into the shopping cart

Then the shopping cart sub-total is EUR 33.75.

Gherkin Syntax

- Gherkin is line-oriented language just like YAML and Python. Each line called step and starts with keyword and end of the terminals with a stop. Tab or space are used for the indentation.
- In this script, a comment can be added anywhere you want, but it should start with a # sign. It reads each line after removing Gherkin's keywords as given, when, then, etc.
- Gherkin Scripts: connects the human concept of cause and effect to the software concept of input/process/output.

```
Feature: Title of the Scenario
```

```
Given [Preconditions or Initial Context]
```

```
When [Event or Trigger]
```

```
Then [Expected output]
```

Important Terms

A Gherkin document has an extension .feature and simply just a test file with a fancy extension. Cucumber reads Gherkin document and executes a test to validate that the software behaves as per the Gherkin syntax.

- Feature
- Background
- Scenario
- Given
- When
- Then
- And
- But
- Scenario Outline Examples

Important Terms

- Feature:
 - The file should have extension .feature and each feature file should have only one feature. The feature keyword being with the Feature: and after that add, a space and name of the feature will be written.
- Scenario:
 - Each feature file may have multiple scenarios, and each scenario starts with Scenario: followed by scenario name.
- Background:
 - Background keyword helps you to add some context to the scenario. It can contain some steps of the scenario, but the only difference is that it should be run before each scenario.

Important Terms

- Given:
 - The use of Given keyword is to put the system in a familiar state before the user starts interacting with the system. However, you can omit writing user interactions in Given steps if Given in the “Precondition” step.

Syntax:

```
Given
```

```
Given - a test step that defines the 'context'  
Given I am on "/"
```

Important Terms

- When:
 - When the step is to define action performed by the user.

Syntax:

When

A When - a test step that defines the 'action' performed
When I perform "Sign In."

Important Terms

- Then:
 - The use of 'then' keyword is to see the outcome after the action in when step. However, you can only

Syntax:

Then

Then - test step that defines the 'outcome.'

Then I should see "Welcome Tom."

Important Terms

- And & But
 - You may have multiple given when or Then.

But

A But - additional test step which defines the 'action' 'outcome.'

But I should see "Welcome Tom."

And - additional test step that defines the 'action' performed

And I write "EmailAddress" with "Tomjohn@gmail.com."

Important Terms

- Given, When, Then, and, but are test steps. You can use them interchangeably. The interpreter doesn't display any error. However, they will surely not make any 'sense' when read.



```
Given The login page is opening
When I input username, password and click the Login button
Then I am on the Homepage
```

Examples

Example 1:

```
Feature: Login functionality of social networking site Facebook.
```

```
Given: I am a facebook user.
```

```
When: I enter username as username.
```

```
And I enter the password as the password
```

```
Then I should be redirected to the home page of facebook
```

The scenario mentioned above is of a feature called user login.

All the words written in bold are Gherkin keywords.

Gherkin will analyze each step written in the step definition file. Therefore, the steps are given in the feature file and the step definition file should match.

Examples

Example 2:

Feature: User Authentication Background:

Given the user is already registered to the website Scenario:

Given the user is on the login page

When the user inputs the correct email address

And the user inputs the correct password

And the user clicks the Login button

Then the user should be authenticated

And the user should be redirected to their dashboard

And the user should be presented with a success message

Best practices

- Each scenario should execute separately
- Every feature should able to be executed along
- Steps information should be shown independently
- Connect your Scenarios with your requirements
- Keep a complete track of what scenarios should be included in a requirement document
- Create modular and easy to understand steps
- Try to combine all your common scenarios

Handson topics/details

Create a first cucumber script, following link can be followed for the reference:

<https://www.guru99.com/your-first-cucumber-script.html>

Reference Links

- <https://www.guru99.com/cucumber-tutorials.html>
- <https://www.jetbrains.com/help/idea/enabling-cucumber-support-in-project.html>
- <https://www.axelerant.com/resources/team-blog/setup-for-selenium-with-cucumber-using-maven>
- <https://www.toolsqa.com/cucumber/cucumber-jvm-feature-file/>
- <https://www.guru99.com/using-cucumber-selenium.html>
- <https://www.toolsqa.com/selenium-cucumber-framework/cucumber-reports/>
- <https://www.jetbrains.com/help/idea/enabling-cucumber-support-in-project.html>
- <https://www.axelerant.com/resources/team-blog/setup-for-selenium-with-cucumber-using-maven>
- <https://www.toolsqa.com/cucumber/cucumber-jvm-feature-file/>
- <https://www.guru99.com/using-cucumber-selenium.html>

THANK YOU!

- ✉ info@crestdatasys.com
- 🌐 <http://www.crestdatasys.com/>

