JDK JRE JVM

Project

**Packages**

**Java Packages & API**

A package in Java is used to group related classes. Think of it as a folder in a file directory. We use packages to avoid name conflicts, and to write a better maintainable code. Packages are divided into two categories:

Built-in Packages (packages from the Java API)

User-defined Packages (create your own packages)

**Class** -Collection of Methods, Objects and Variables

**Object** - An object can be defined as an instance of a class, and there can be multiple instances of a class in a program. An Object is one of the Java OOPs concepts which contains both the data and the function, which operates on the data

**Object** − **Objects** have states and behaviors. ... An **object** is an instance of a class. Class − A class can be defined as a template/blueprint that describes the behavior/state that the **object** of its type support.

**Access Specifiers**

Public

Private

Protected

Default

**Non Access Specifiers**

Final

Abstract

Oops

**Abstraction**

Data Abstraction is the property by virtue of which only the essential details are displayed to the user.The trivial or the non-essentials units are not displayed to the user. Ex: A car is viewed as a car rather than its individual components.

while driving a car, you do not have to be concerned with its internal working. Here you just need to concern about parts like steering wheel, Gears, accelerator, etc.

**Encapsulation**

Encapsulation is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates. Another way to think about encapsulation is, it is a protective shield that prevents the data from being accessed by the code outside this shield.

Technically in encapsulation, the variables or data of a class is hidden from any other class and can be accessed only through any member function of own class in which they are declared.

As in encapsulation, the data in a class is hidden from other classes, so it is also known as data-hiding.

Encapsulation can be achieved by Declaring all the variables in the class as private and writing public methods in the class to set and get the values of variables.

**Inheritance**

Inheritance is one of the Basic Concepts of OOPs in which one object acquires the properties and behaviors of the parent object. It’s creating a parent-child relationship between two classes. It offers robust and natural mechanism for organizing and structure of any software.

Inheritance is an important pillar of OOP(Object Oriented Programming). It is the mechanism in java by which one class is allow to inherit the features(fields and methods) of another class.

**Important terminology:**

**Super Class:** The class whose features are inherited is known as superclass(or a base class or a parent class).

**Sub Class:** The class that inherits the other class is known as subclass(or a derived class, extended class, or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.

Reusability: Inheritance supports the concept of “reusability”, i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

The keyword used for inheritance is extends

**Polymorphism**

Polymorphism refers to one of the OOPs concepts in Java which is the ability of a variable, object or function to take on multiple forms.

**Overloading**

**Overriding**

Frameworks

* POM
* TestNG
* Cucumber

**NAMING CONVENTIONS**

**Packages**

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

com.sun.eng

com.apple.quicktime.v2

edu.cmu.cs.bovik.cheese

**Classes**

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. Use whole words-avoid acronyms and abbreviations (unless the abbreviation is much more widely used than the long form, such as URL or HTML).

class Raster;

class ImageSprite;

A noun is a part of a speech that refers to a person, place, or thing. On the other hand, a verb is a part of a speech referring to some action, experience, or condition. Nouns may be the subject or object in a sentence whereas verbs form the main part of the predicate.

**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**

Except for variables, all instance, class, and class constants are in mixed case with a lowercase first letter. Internal words start with capital letters.

Variable names should not start with underscore \_ or dollar sign $ characters, even though both are allowed.

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.

int i;

char c;

float myWidth;

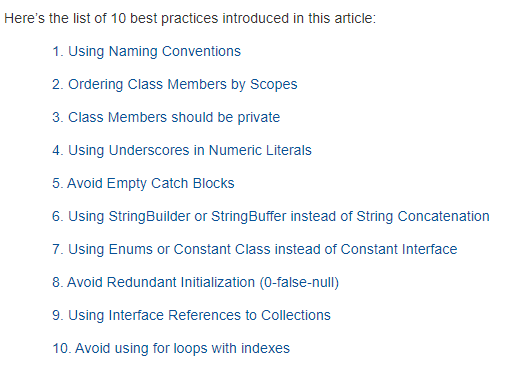
**Constants**

The names of variables declared class constants should be all uppercase with words separated by underscores ("\_").

static final int MIN\_WIDTH = 4;

static final int MAX\_WIDTH = 999;

static final int GET\_THE\_CPU = 1;



@BeforeSuite

@BeforeGroup

@BeforeClass

@BeforeMEthod

@Test(Priority=0)

@Test(Priority=1)

@Test

@Parameters("myName")

@AfterMethod

@AfterClass

@AfterGroup

@AfterSuite

As the name indicates, grouping in TestNG lets you group multiple test methods in named groups. Through groups in TestNG, you have the provision to run a particular set of tests that belong to a group or multiple groups.

<?xml version = "1.0" encoding = "UTF-8"?>

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >

<suite name = "Suite1">

<test name = "test1">

<parameter name = "myName" value="manisha"/>

<classes>

<class name = "ParameterizedTest1" />

</classes>

</test>

</suite>

Summary of TestNG Annotations

@BeforeSuite: The annotated method will be run before all tests in this suite have run.

@AfterSuite: The annotated method will be run after all tests in this suite have run.

@BeforeTest: The annotated method will be run before any test method belonging to the classes inside the tag is run.

@AfterTest: The annotated method will be run after all the test methods belonging to the classes inside the tag have run.

@BeforeGroups: The list of groups that this configuration method will run before. This method is guaranteed to run shortly before the first test method that belongs to any of these groups is invoked.

@AfterGroups: The list of groups that this configuration method will run after. This method is guaranteed to run shortly after the last test method that belongs to any of these groups is invoked.

@BeforeClass: The annotated method will be run before the first test method in the current class is invoked.

@AfterClass: The annotated method will be run after all the test methods in the current class have been run.

@BeforeMethod: The annotated method will be run before each test method.

@AfterMethod: The annotated method will be run after each test method.

@Test: The annotated method is a part of a test case

Parallel Execution

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">

<suite name="TestSuite" thread-count="3" parallel="methods" >

<test name="testGuru">

<classes>

<class name="TestGuru99MultipleSession">

</class>

</classes>

</test>

</suite>

Abstract Classes and Methods

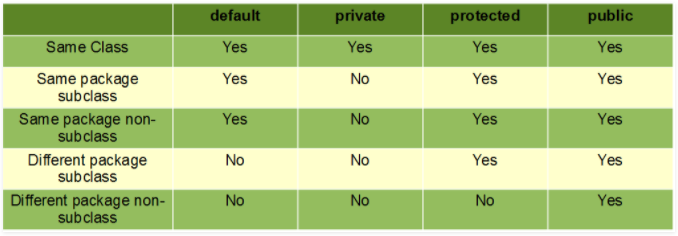
Data **abstraction** is the process of hiding certain details and showing only essential information to the user.  
Abstraction can be achieved with either **abstract classes** or [**interfaces**](https://www.w3schools.com/java/java_interface.asp) (which you will learn more about in the next chapter).

The abstract keyword is a non-access modifier, used for classes and methods:

* **Abstract class:** is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).

* **Abstract method:** can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).

An abstract class can have both abstract and regular methods:



### **ABSTRACTION**

In Page Object Model design pattern, we write locators (such as id, name, xpath etc.,) in a Page Class. We utilize these locators in tests but we can’t see these locators in the tests. Literally we hide the locators from the tests.

Abstraction is the methodology of hiding the implementation of internal details and showing the functionality to the users.

### **INTERFACE**

Basic statement we all know in Selenium is WebDriver driver = new FirefoxDriver();

WebDriver itself is an Interface. So based on the above statement WebDriver driver = new FirefoxDriver(); we are initializing Firefox browser using Selenium WebDriver. It means we are creating a reference variable (driver) of the interface (WebDriver) and creating an Object. Here WebDriver is an Interface as mentioned earlier and FirefoxDriver is a class.

An interface in Java looks similar to a class but both the interface and class are two different concepts. An interface can have methods and variables just like the class but the methods declared in interface are by default abstract. We can achieve 100% abstraction and multiple inheritance in Java with Interface.

### **INHERITANCE**

We create a Base Class in the Framework to initialize WebDriver interface, WebDriver waits, Property files, Excels, etc., in the Base Class.

We extend the Base Class in other classes such as Tests and Utility Class. Extending one class into other class is known as Inheritance.

### **POLYMORPHISM**

Combination of overloading and overriding is known as Polymorphism. We will see both overloading and overriding below.

Polymorphism allows us to perform a task in multiple ways.

### **METHOD OVERLOADING**

We use implicit wait in Selenium. Implicit wait is an example of overloading. In Implicit wait we use different time stamps such as SECONDS, MINUTES, HOURS etc.,

A class having multiple methods with same name but different parameters is called Method Overloading

### **METHOD OVERRIDING**

We use a method which was already implemented in another class by changing its parameters. To understand this you need to understand Overriding in Java.

Declaring a method in child class which is already present in the parent class is called Method Overriding. Examples are get and navigate methods of different drivers in Selenium .

### **ENCAPSULATION**

All the classes in a framework are an example of Encapsulation. In POM classes, we declare the data members using @FindBy and initialization of data members will be done using Constructor to utilize those in methods.

Encapsulation is a mechanism of binding code and data together in a single unit.

### **WEB ELEMENT:**

Web element is an interface used to identify the elements in a web page.

### **WEBDRIVER:**

WebDriver is an interface used to launch different browsers such as Firefox, Chrome, Internet Explorer, Safari etc.,

### **FIND BY:**

FindBy is an annotation used in Page Object Model design pattern to identify the elements.

### **FIND ELEMENT:**

Find Element is a method in POM to identify the elements in a web page.