OOPS Concepts definitions:

=====================

.

1.CLASS:

Class is the collection of objects and methods.

In our project We are using lots of predefined classes like

1.ChromeDriver

2.FireFoxDriver

3.InternetExplorerDriver

4.Actions

5.Robot

6.Select

7.RemoteWebDriver

OBJECT:

Object is an instance of a class.

Using object,we can able to access the methods in a class.

It is a runtime memory allocation.

Object creation syntax:

Classname objectname=new Classname();

METHODS:

Set of actions to be performed.

Reusable lines of code can be kept inside a method.

We can access the method using object whenever needed.

How to access class properities using object:

object.variable;

object.method();

for eg:-

----

get()

getTitle()

getCurrentUrl()

getText()

getAttribute()

2.ENCAPSULATION:

-->Wrapping up of data and code acting on data together in a single unit.

-->Encapsulation, is to make sure that "sensitive" data is hidden from users.

-->To achieve this, you must:declare class variables/attributes as private

-->Provide public getters and setters methods to access and update the value of a private variable.

I m using encapsulation in real time in POJO class where i create all my locator variables as private.Because private variables cannot be accessed outside the class.If we want to access private variables we have to use getters.

3.POLYMORPHISM:

===============

Performing single action in different ways.

Method Overloading/Static Binding/CompileTime Polymorphism:

--> In same class,there are multiple functions with same name but different parameters then these functions are said to be overloaded.

--> Same method overloaded again and again using different arguments type,count and order.

Program:

```
package org.test;


public class Student {
	public void stuDetails() {
		System.out.println("Student details");


	}


	public void stuDetails(int stuId) {
		System.out.println("Student id: "+stuId);
	}

	public void stuDetails(String stuName, int stuId) {
		System.out.println("Student name: "+stuName);
		System.out.println("Student id: "+stuId);

	}

	public void stuDetails(int StuId, String stuName) {
		System.out.println("Student id: " + StuId);
		System.out.println("Student name: "+stuName);

	}
	public static void main(String[] args) {
		Student s=new Student();
		s.stuDetails();
		s.stuDetails(101);
		s.stuDetails(102, "Java");
		s.stuDetails("Java", 103);

	}
}
```

Output:

```
---------
Student details
Student id: 101
Student id: 102
Student name: Java
Student name: Java
Student id: 103
```

For example:In realtime i m overloading println,sendkeys and frame methods

```
System.out.println("Renu");
System.out.println(123);
System.out.println(12.2);
```

webelement.senkeys("Renu");   //Here Sendkeys accepts only one argument

Actions a=new actions(driver);

a.sendkeys(webelement,"Renu");   //using actions class also we can use sendkeys,here i am passing two arguments driver.switchTo.frame(String id);

driver.switchTo.frame(String name);

driver.switchTo.frame(int index);

driver.switchTo.frame(Webelement); //For same methods i m passing String type,int type and Webelement type

Method overriding/Dynamic Binding/RunTime Polymorphism:

-->Method Overriding is when one of the methods in the super class is redefined in the sub-class.

(or)When i am not satisfied with the business logic of my super class i am overriding that method in my subclass.

In realtime i m overriding many methods such as retry() method from IRetryAnalyser,transform() method from IAnnotationTransformer and getMessage() method from Exception.

Notes:

-->A constructor cannot be overridden.

-->Final - declared methods cannot be overridden

-->Any method that is static cannot be used to override.

Child class:

```
--------------
public class Child extends Parent{
	public void print() {
		System.out.println("Child class method");
	}

	public static void main(String[] args) {
		Child c=new Child();
		c.print();
	}

}
```

Parent class:

```
---------------
public class Parent {
	public void print() {
        System.out.println("Parent class method");
//overrided method


}


}
```

4.INHERITANCE:

Using inheritance we can access one class properties in another class without multiple object creation.so we can save memory.Also we can acheive code reusability.

In real time, I m using inheritance concept in base class where i will maintain all reusable methods.so,by extending base class i can call reusable methods wherever i want.

4a)SINGLE INHERITANCE:

One child class extends one parent class.

4b)MULTILEVEL INHERITANCE:

One child class extends more than one parent class in tree level structure.

4c)MULTIPLE INHERITANCE:

More than one parent class parallely supporting into one child class.We can't acheive multiple inheritance in java due to

1.Syntax error/compilation error(After extends keyword we cannot specify more than one class)

2.Priority problem(When parent classes having same method name with same arguments)

extending into same child class,

4d)HYBRID INHERITANCE:

It is the combination of single and multiple inheritance.

4e)HIERARCHIAL INHERITACE:

More than one child class inheriting the same parent class.

5.DATA ABSTRACTION:

==================

Hiding the implementation of the program.

(or)

Process of hiding certain details and showing only essential information to the user.

Abstraction can be achieved with either abstract classes or interfaces.

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

1.Abstract class:

Contains abstract as well as non-abstract methods.

We can't create objects for abstract class(to access the methods in abstract class, it must be inherited in another class).

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

Non-abstract method(Concrete method):normal method with business logic.

In real time,i m using 'By' in my project which is an abstract class.

```
public abstract class Sample {

        public abstract void clientId();  //abstract method which
has no logic

        public void clientName() {
            System.out.println("Client name is CTS");      //non-
abstract method with business logic
        }

}
```

Why And When To Use Abstract Classes and Methods?

To achieve security - hide certain details and only show the important details of an object.

2.Java Interface:

Another way to achieve abstraction in Java, is with interfaces.

-->An interface can have only abstract methods.

-->To access the interface methods, the interface must be "implemented" (kind of "inherited") by another class with the implements keyword (instead of "extends").

-->The body of the interface method is provided in the class which implements the Interface.

```
public interface Test{

void add();   //All the methods inside interface will have
'public abstract' by default,so no need to mention.


}
```

I m using many interfaces in my project such as Webdriver,webelement,Alert,Wait,Javascript executor,Takesscreenshot,List,Set,Map etc

Notes on Interfaces:

-->Like abstract classes, interfaces cannot be used to create objects

-->Interface methods do not have a body - the body is provided by the "implement" class

-->On implementation of an interface, you must override all of its methods

-->Interface methods are by default abstract and public

-->Interface attributes(variables) are by
default public, static and final

-->An interface cannot contain a constructor (as it cannot be used to create objects)

Why And When To Use Interfaces?

----

1) To achieve security - hide certain details and only show the important details of an object (interface).

2) Java does not support "multiple inheritance" (a class can only inherit from one superclass). However, it can be achieved with interfaces, class can implement multiple interfaces.

Advantages of Abstraction:

It reduces the complexity of viewing the things.

Avoids code duplication and increases reusability.

Acheiving Multiple inheritance using interface:

1.we will not get syntax error because after "implements" keyword we can specify any number of interfaces.

2.There will be no priority problem(Though all the interfaces have same method,the definition for the particular method will be present only in the class which implements all the interfaces--->so there will be one method definition)

```
package org.cts.test;

public class Company implements Int1,Int2{

	@Override
	public void intId() {
```

```
                System.out.println(123);
        }

        public void intName() {

                System.out.println("Multiple Inheritance acheived
through interface");
        }

        public static void main(String[] args) {
                Company c=new Company();
                c.intName();
                c.intId();

        }

}
```

Difference between Abstraction and Encapsulation:

Encapsulation is data hiding(information hiding) while Abstraction is detail hiding(implementation hiding)

While encapsulation groups together data and methods that act upon the data, data abstraction deals with exposing the interface to the user and hiding the details of implementation.

STRING:

======

Strings are collection of characters enclosed within " " . In Java, String is a Class,strings are treated as objects.

String Literal:

String str = “Java”;

Literal string shares the same memory incase of duplicates.

It is stored in String pool constants(String pool constants present inside heap memory)

When String declared like this, intern() method on String is called. This method references internal pool of string objects. If there already exists a string value “Java”, then str will get reference of that string and no new String object will be created.

String Object(Non-Literal String)

Non-literal string will not share the same memory even if it is a duplicate.

Since we use "new" operator everytime it creates new memory.

Non-literal strings stored in heap memory.

String str = new String(“Java”);

In this,JVM is forced to create a new string reference, even if “Java” is in the reference pool.

Performance Comparison between String literal and String constant:

If we compare performance of string literal and string object, string object will always take more time to execute than string literal because it will construct a new string every time it is executed.

Program:

```
package org.test;

public class Variable {

        public static void main(String[] args) {

                String s = "Java";
                String s1 = "Java";

                System.out.println("Literal String 's': " + s);
                System.out.println("Literal String 's' address: " +
System.identityHashCode(s));

                System.out.println("Literal String 's1': " + s1);
                System.out.println("Literal String 's1' address: " +
System.identityHashCode(s1));
```

```
                System.out.println();

                String s2 = new String("Java");
                String s3 = new String("Java");

                System.out.println("Non-literal String 's2': " + s2);
                System.out.println("Non-literal String 's2' address: "
+ System.identityHashCode(s2));

                System.out.println("Non-literal String 's3': " + s3);
                System.out.println("Non-literal String 's3' address: "
+ System.identityHashCode(s3));

                }
}
```

Output:

```
---------
Literal String 's': Java
Literal String 's' address: 2018699554
Literal String 's1': Java
Literal String 's1' address: 2018699554

Non-literal String 's2': Java
Non-literal String 's2' address: 1311053135
Non-literal String 's3': Java
Non-literal String 's3' address: 118352462
```

IMMUTABLE STRING:

Immutable string once string created,value can't be changed in reference.

Eg:String literals

MUTABLE STRING:

For mutable string,we can change the value in reference.

Eg:StringBuffer,StringBuilder

StringBuffer---->Synchronised(One thread can only access StringBuffer class at a time),Thread safe

StringBuilder---->Asynchronised(Multiple threads can access StringBuilder class at a time),Not thread safe.

Program:

```
package org.test;

public class Variable {

        public static void main(String[] args) {

                String s = "Hello";
                String s1 = "world";
                System.out.println("IMMUTABLE STRING");
                System.out.println("Immutable String 's' address: "
+ System.identityHashCode(s));
                System.out.println("Before concatenation 's': "+s);
                System.out.println("Immutable String 's1' address:
" + System.identityHashCode(s1));
                System.out.println("Before concatenation 's1':
"+s1);
                String co = s.concat(s1);
                System.out.println("Immutable String 'co' address:
" + System.identityHashCode(co));
                System.out.println("After concatenation 's': "+s);
```

```
		System.out.println("After concatenation third 
reference variable 'co': "+co);

		StringBuffer s2 = new StringBuffer("Hello");
		StringBuffer s3 = new StringBuffer("world");
		System.out.println("MUTABLE STRING");
		System.out.println("Mutable String 's2' address: " + 
System.identityHashCode(s2));
		System.out.println("Before appending s2: "+s2);
		System.out.println("Mutable String 's3' address: " + 
System.identityHashCode(s3));
		System.out.println("Before appending s3: "+s3);
		s2.append(s3);
		System.out.println("After appending s2: "+s2);
		System.out.println("Mutable String 's2' address: " + 
System.identityHashCode(s2));
	}
}
```

OUTPUT:

```
IMMUTABLE STRING
Immutable String 's' address: 2018699554
```

```
Before concatenation 's': Hello
Immutable String 's1' address: 1311053135
Before concatenation 's1': world
Immutable String 'co' address: 118352462
After concatenation 's': Hello
After concatenation third reference variable 'co': Helloworld

MUTABLE STRING
Mutable String 's2' address: 1550089733
Before appending s2: Hello
Mutable String 's3' address: 865113938
Before appending s2: world
After appending s2: Hello world
Mutable String 's2' address: 1550089733
```

Note:

System-class

identityHashCode() -method to find the address of the variable.

TYPES OF VARIABLES:

LOCAL VARIABLE:

Local variable scope is only inside the method/block.

We cant use access specifiers for local variable.

We need to initialise local variable.

INSTANCE/GLOBAL VARIABLE:

Global variable declared inside class and outside methods.

We can use access specifiers for global variables.

Need not initialise value for global variable.It will have default value.

STATIC VARIABLE:

Static variable retains value between the objects(Since static variable is shared by all the objects)

2 ways of accessing static variables:

----

classname.variablename

variablename

Static method:

If u specify a method as static, we need not create object for calling the methods.

2 ways of accessing static methods:

----

classname.methodname();

methodname();

Program to show the difference between global variable and static variable:

```
}


public class Variable {


static int a=20;

int b=10;


public void print() {

        System.out.println(a);

        System.out.println(b);

        a++;

        b++;


}

public static void main(String[] args) {

        Variable a=new Variable();

        a.print();

        Variable b=new Variable();

        b.print();

        Variable c=new Variable();

        c.print();

}


}
```

OUTPUT:

```
20
10
21       //static variable retains value between objects
10
```

FINAL VARIABLE:

1.If a variable declared as "final"---->we can't change the value assigned.

2.If a method as declared as "final"----->We can't override that method.

3.If a class as declared as "final"-------->We can't inherit that class.

Access Modifiers:

============

It is used to set the access level for classes, attributes, methods and constructors.

We divide modifiers into two groups:

Access Modifiers - controls the access level

Non-Access Modifiers - do not control access level, but provides other functionality

Access Modifiers:

----

private

default or no modifier

protected

public

A)Public(Project level access/Global level access)

In same package as well as different package,we can access the private variables and methods of one class in another class using extends keyword and creating separate object.

B)Protected(Global level access)

In same package,

Using extends keyword and by creating separate object also we can access the properities of another class.

If it is in different package,

using extends keyword only we can access,We cannot create object and access the protected attributes of a class.

C)Default(Package level access)

In same package only we can access(In different package we cannot access)

using extends keyword and by creating object.

D)Private(Class level access)

Outside class,we cannot access private variables.If we want to access private variables outside the class we use getters and setters.

Non Access Modifiers:

----

static

final

abstract

synchronised---->When Methods given with synchronised keyword,it can only be accessed by one thread at a time

transient--->At the time of serialization, if we don’t want to save value of a particular variable in a file, then we use transient keyword. When JVM comes across transient keyword, it ignores original value of the variable and save default value of that variable data type.

volatile--->volatile variable in Java is a special variable which is used to signal threads, a compiler that this particular variables value are going to be updated by multiple threads inside Java application. By making a variable volatile using the volatile keyword in Java, application programmer ensures that its value should always be read from main memory and thread should not use cached value of that variable from their own stack.

strictfp-->Java strictfp keyword ensures that you will get the same result on every platform if you perform operations in the floating-point variable.

STORAGE IN JAVA:

What is Stack Memory?

Stack in java is a section of memory which contains methods, local variables, and reference variables. Stack memory is always referenced in Last-In-First-Out order. Local variables are created in the stack.

Stack memory structure is mostly implemented for providing static memory allocation.

What is Heap Memory?

Heap is a section of memory which contains Objects and may also contain reference variables. Instance variables are created in the heap

Program Counter Register: Programs are a set of instructions.Program counter register holds the address of the upcoming instructions to be executed.

CONSTRUCTOR:

=============

Whenever you create object for a class,default costructor will automatically invoked since class name and constructor name are same.

Purpose of constructor:

-->A constructor in Java is a special method that is used to initialize objects.

-->The constructor is called when an object of a class is created.

-->All classes have constructors by default: if you do not create a class constructor yourself, Java creates one

Rules:

-->constructor name must match the class name.

-->It cannot have a return type (like void)

Types:

Default constructor:(without arguments)

```
public class MyClass {
    int x;
    public MyClass() {
     x = 5;
    }
 public static void main(String[] args)
{
 MyClass myObj = new MyClass();
 System.out.println(myObj.x);
}
}
// Outputs 5
```

Parameterized constructor:(With arguments)

```
public class MyClass {
 int x;
public MyClass(int y) {
 x = y;
}
 public static void main(String[] args) {
MyClass myObj = new MyClass(5);
System.out.println(myObj.x);
}
}
// Outputs 5
```

Exception:

========

Exception results in abnormal termination of the program (or)Whenever exception occurs,program terminates at the particular line itself.

We can avoid the abnormal termination of the program by handling the exception.

```
  Object
     |
Throwable
     |
Exception
```

Exception handling:

----

try block:

The code which is expected to throw exceptions is placed in the try block.

Catch block:

When an exception occurs, that exception occurred is handled by catch block associated with it.

Every try block should be immediately followed either by a catch block or finally block.

finally block:

The finally block follows a try block or a catch block.

A finally block of code will always be executed whether exception occurs or not.

we can have,

```
try{                         try{                          try{
       try{


}              }                      }                 }

catch(Exception e)       catch(ArithmeticException e)
       catch(Exception e)         finally

{              {                      {                 {

}              }                      }                 }

                             catch(Exception e)                    finally

                {                     {

                }                     }
```

Impossible combination:

-->Throwable class is the super class of all exception.

-->This will handle all the exceptions.so all the child catch blocks will not be used/reached by any code.

-->so while writing the program itself,it will show error.

```
try

{

}
```

```
Catch(Throwable e){
}
Catch(Exception e){
}
Catch(ArithmeticException e){
}
```

possible combination:

-->we can specify multiple catch blocks

```
try
{
}
Catch(ArithmeticException e){
}
Catch(IndexOutOfBoundException e){
}
Catch(Throwable e){
}
```

throw:

It is used to explicitly throw an exception.

You cannot throw multiple exceptions.Only one exception can be thrown at a time.

Throw is used within the method.

Checked exception cannot be handled using throw.

Throw is followed by an instance.

throws:

----

throws keyword is used to declare an exception at method level.

Can handle checked as well as unchecked exception.

Throws is used with the method signature.

You can handle multiple exceptions using throws.

Errors :

=====

These are not exceptions at all, but problems that arise beyond the control of the user or the programmer. Errors are typically ignored in your code because we cannot handle the error.

For example, JVM crash, stack overflow,insufficient memory.They are also ignored at the time of compilation.

Note:

-->A catch block cannot exist without a try statement.

-->It is not compulsory to have finally block whenever a try/catch block is present.

-->The try block cannot be present without either catch block or finally block.

-->No code can be present in between the try, catch, finally blocks.

-->We can have 'n' number of exception throwing statements in try block,once a exception caught control will be moved to catch block,all the remaining lines wont be executed.