OOP:

1. Encapsulation : hiding the data, getter & setter

2. Inheritance : super&sub relationship, sub class gets rich

3. Abstraction : hiding the implementation, abstract class & interface

4. Polymorphism : objects to behave in multiple forms

WarmUp:

Create a class called Animal (Do not use abstraction for now):

Actions: Talk()

create sub classes of Animal class:

1. Tiger:

Actions: Hunts()

2. Octopus:

Actions: Swim()

override super class' instance method Talk()

create a class called Zoo:

Create Arrays of Tiger

Create Arrays of Octopus

Is A relation: is inherited relationship between classes

Ex: class Dog extends Animal

class Husky extends Dog

Dog is An animal

Husky is A Dog

Husky is an Animal

Animal obj = new Husky();

Has A relation: instance of the class (object) is used in another classes

class Car{

Engine obj = new Engine();

}

Car Has An engine

class Zoo{

Tiger obj1 = new Tiger();

Octopus obj2 = new Octopus();

Animal obj3 = new Tiger();

}

Zoo Has A Tiger

Zoo Has An Octopus

Zoo Has an Animal

Tiger Is An animal

classA obj = new classA();

referencetype object

object MUST be created from Non-Abstract class

Object MUST be concrete.

Polymorphism:

occurs when the super class/interface is reference type, and object is created from the non-abstract classes that have IS A relation with the super class/interface

Ex:

WebDriver driver = new ChromeDriver();

driver.get("google.com") // overridden method will be executed

opens the chrome browser

WebDriver driver1 = new FireFoxDriver();

driver1.get("google.com");

opens the firefox browser

2 IMPORTANT RULES:

1) reference type decides what is accessible and what can be used

2) reference type can be parent class or interface, and objects can be any sub class

if we make the abstract class as reference type, the object MUST be created from it's sub class (Non-Abstract)

if we make the interface as reference type, object MUST be created from the classes (non abstract) that are implementing the interface

Method overriding: ONLY THE INSTANCE METHOD CAN BE OVERRIDEN

cannot be static, final, private

MUST take place at sub class

extends & implements:

class extends class

interface extends interface

class implements interface

1. create an abstract class called ScrumTeam

Data: Name

JobTitile

Salary

Actions: DailyStandUp();

Demo();

2. Create the following sub classes of ScrumTeam:

1. Testers:

Actions: FindBug

2. Developers:

Actions: FixBug

3. Create a class called BOA

create Array of Testers named sdets, at leats store two objects

create array of Developers named dev, at least store three objects

create ArrayList of ScrumTeam:

store all the developer' and testers' objects

extends: one sub class can only have one super class. one super class can have multiple sub class

class extends extends

interface extends interface

implements: a class can implement multiple interfaces

Class implements interface

Object:

ClassName objectName = new ExistingConstructor

reference-Type ref-name

Polymorphism: behaviors of the object in multiple ways

occurs when parent class/Interface is reference type, and object is created from sub class (non abstract)

if abstract class is reference type, object needs to be created from non abstract sub class

if interface is reference type, object needs to be created from the classes that's implementing the interface

WebDriver driver = new CrhomeDriver();

driver.get

driver.nevigate

IS A: is inherited relationship between classes

Has A: if one class' object is used in another class

we cannot create objects from interface and abstract class, because abstraction is not concrete

Polymorphism: occurs when a super class/interface is reference and object from is created from child class

A extends B

sub super

B obj = new A()

ref-type ref-name object

reference type decides what can be accessible

child class cannot be the reference to parent class' object.

if we make the abstract class as reference, object MUST be created from sub class(non abstract)

if we make the interface as reference, object MUST be created from the classes(Non Abstract) that's implementing the interface

A obj11 = new B();

obj11.methodA(); // from B class

// if a method is exist in both reference type and object, the overridden one gets executed

obj11.methodC();

// if the method is not being overridden, then the reference type' method gets executed

WebDriver driver = new ChromeDriver();

driver.get("URL") // opens the chrome browser, overridden

// ChromeDriver driver = new FireFoxDriver();

// there is no IS A relation between Chrome Driver and FireFoxDriver

interface: WebDriver, WebElemnet, JavaScriptExecuter, TakeScreenShot...

IS A: inherited/implemented relation between class

Dog extends Animal

PitBull extends Dog

Dog IS An Animal

Pitbull IS A Dog

PitBull IS An Animal

Has A:

class Car{

Engine obj = new Engine();

}

Car HAS An Engine

class Zoo {

Lion li = new Lion();

Tiger ti = new Tiger();

....

}

Zoo Has A lion

Zoo Has A tiger

method Overriding : same method name , same parameter

only the instance method can be overridden

HAS to be overridden in the sub class

Cannot be private, static, final

access modifier needs to be same or more visible

return-type HAS to be same

@Override has to be applicable

there are TWO types of castings between class:

Upcasting and Downcasting

RemoteDriver implements WebDriver, JaveScriptExecuter, TakeScreenSH

ChromeDriver extends RemoteDriver

1. UpCasting: converting sub type to super type

Always allowed to use

WebDriver driver = new ChromeDriver();

// JavaScriptExecuter js = new ChromeDriver(); // not the same browser we are automating the test case with. new window

JavaScriptExecuter js = (JavaScriptExecuter)driver;

js.ExecuteScript("window.open()")

TakeScreenShot ts = new ChromeDriver();

TakeScreenShot ts = (TakeScreenShot) driver;

ts.TakeScreenShotAs("");

2. Downcasting: converting super type to sub type

not allowed. throw ClassCastException

multiple reference name, refer to the same object ( same memory)

OOP Summary:

OOP: Object Oriented Programming

1. Encapsulation: data hiding by giving private access modifier to the variable, and provide public getter/setter

benefits: control the data condition. ( by using setter)

Data Security

if the private data is final, we can ONLY generate getter

2. Inheritance: sub & super relationship. methods & variables of one class can be inherited to another

super --> sub

benefits: reusable

easy to maintain

organized

3. Abstraction: hiding the implementation. concentrating on the importance/essentials without worrying about the details

2 ways to achieve abstraction:

1. Abstract class

2. Interface

benefits: reusability

able to focus on what to do instead of how to do

4. Polymorphism: object behavior in many forms.

super class/interface can be reference type to any child class' objects

benefits: flexible

- one single variable with reference type of parent class is able to any objects of child class