Inheritance in OOPS | Selenium

Table of content

Inheritance in Java

Inheritance in Java

Inheritance is the process where the characteristics are inherited from ancestors (superclasses). Inheritance in java can be defined as a mechanism where a new class (subclass) is derived from an existing class(superclass) properties.

It is possible for classes to inherit or acquire the properties and methods of other classes, just like a son can acquire some traits and behavior from his father.

In Java inheritance is declared using the extends keyword. You can declare that one class extends another class by using the keyword extends in the class definition

extends keyword used for Class to Class relationship.
extends keyword used for interface to interface relationship
implements keyword used with Interface to the class interface

Child Class / Sub Class : The class that extends the features of another class is known as child class, subclass, or derived class.

Parent Class / Superclass : The class whose properties and functionalities are used(inherited) by another class is known as a parent class, superclass, or Base class.

Important things in Inheritance

The most important benefit of inheritance is code reuse because subclasses inherit the variables and methods of the superclass.

Private members of the superclass are not directly accessible to subclass, as private members' scope is only the particular class where they are declared.

Superclass members with default access are accessible to subclass only if they are in the same package. Default access is nothing but the state where you do not specify Public, Private, Protected.

Constructors are not inherited by subclasses.

We can override the method of Superclass in the Subclass, this most important when you inherit an interface

We can override the method of Superclass in the Subclass, this most important when you inherit an interface

If a class is Final, then we cannot inherit that class.

Types of inheritance :

There are four types of inheritance in java; those are :

Single Inheritance

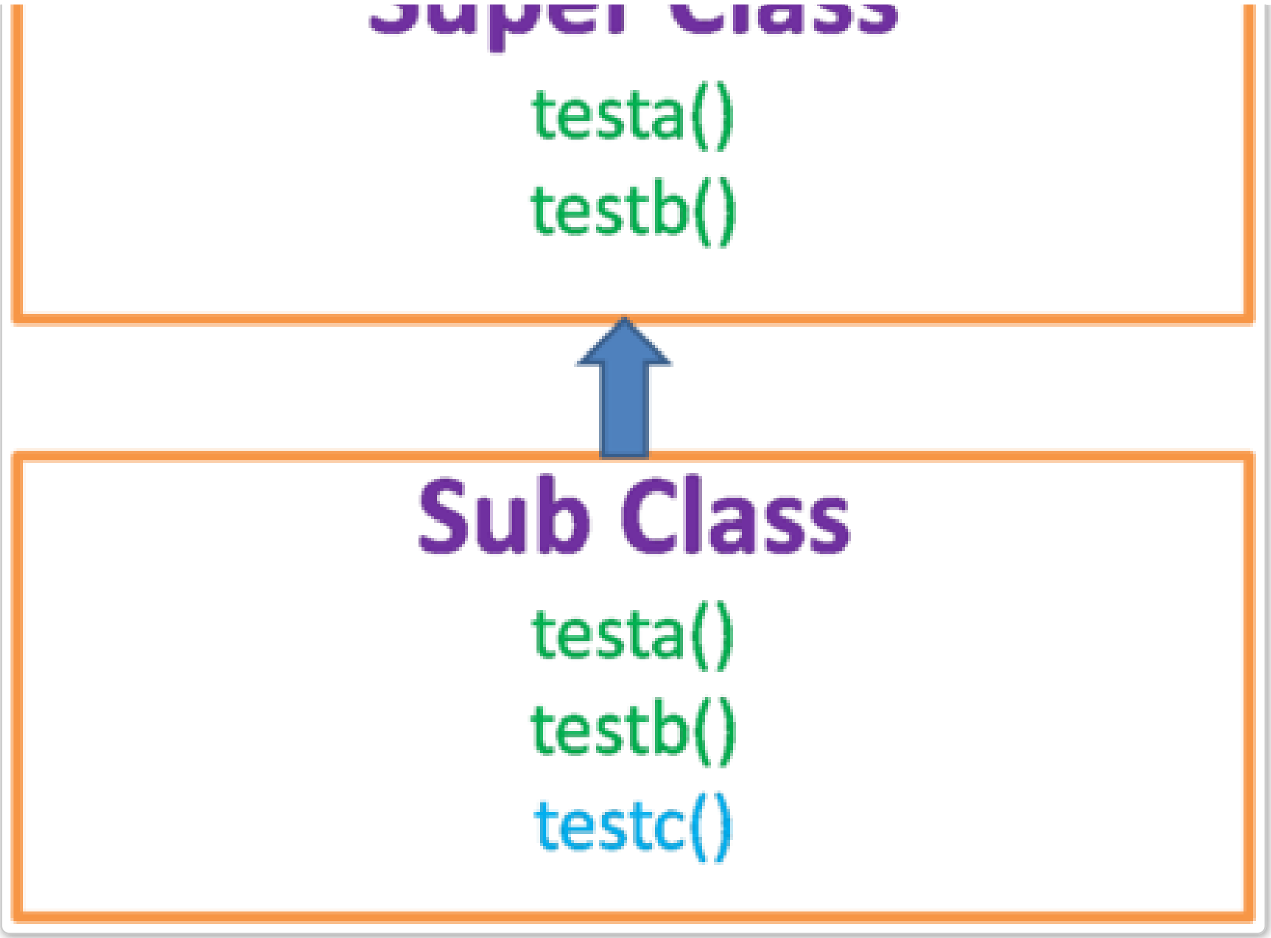
Multi-level Inheritance

Multiple Inheritance

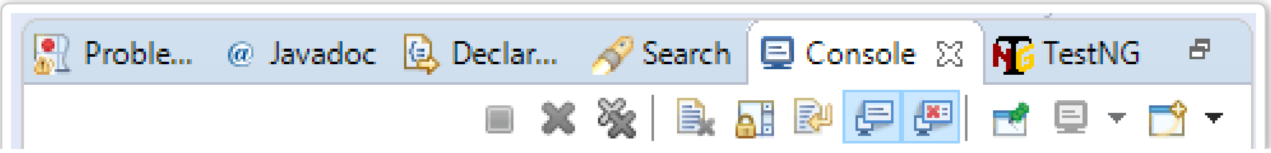
Hybrid Inheritance

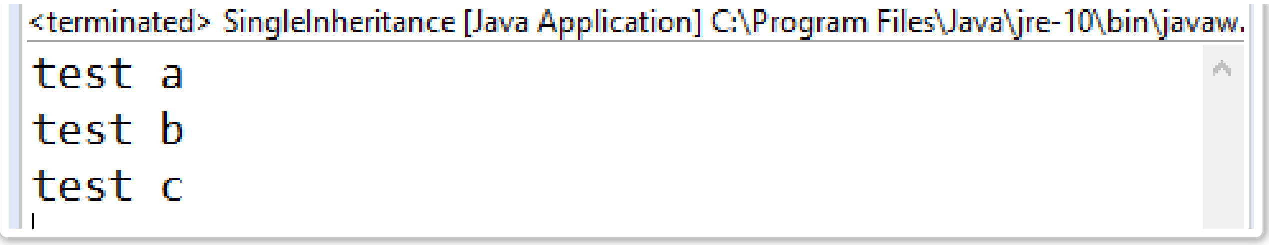
Single Inheritance :

When a single child class inherits the properties of a parent class, it is known as single inheritance. There are no other child classes present for that parent class.



```
package cherchertech;
class BaseClass {
   public void testa() {
      System.out.println("test a");
   }
   public void testb() {
      System.out.println("test b");
   }
}
class ChildClass extends BaseClass{
   public void testc() {
      System.out.println("test c");
   }
}
public class SingleInheritance {
   public static void main(String[] args) {
      ChildClass cc = new ChildClass();
      // methods present in base class
      cc.testa();
      cc.testb();
      // method present in child class
      cc.testc();
   }
}
```





```
<terminated> SingleInheritance [Java Application] C:\Program Files\Java\jre-10\bin\javaw.
test a
test b
test c
```

Single inheritance in Selenium :

There are few classes which are having a single inheritance in **selenium webdriver**

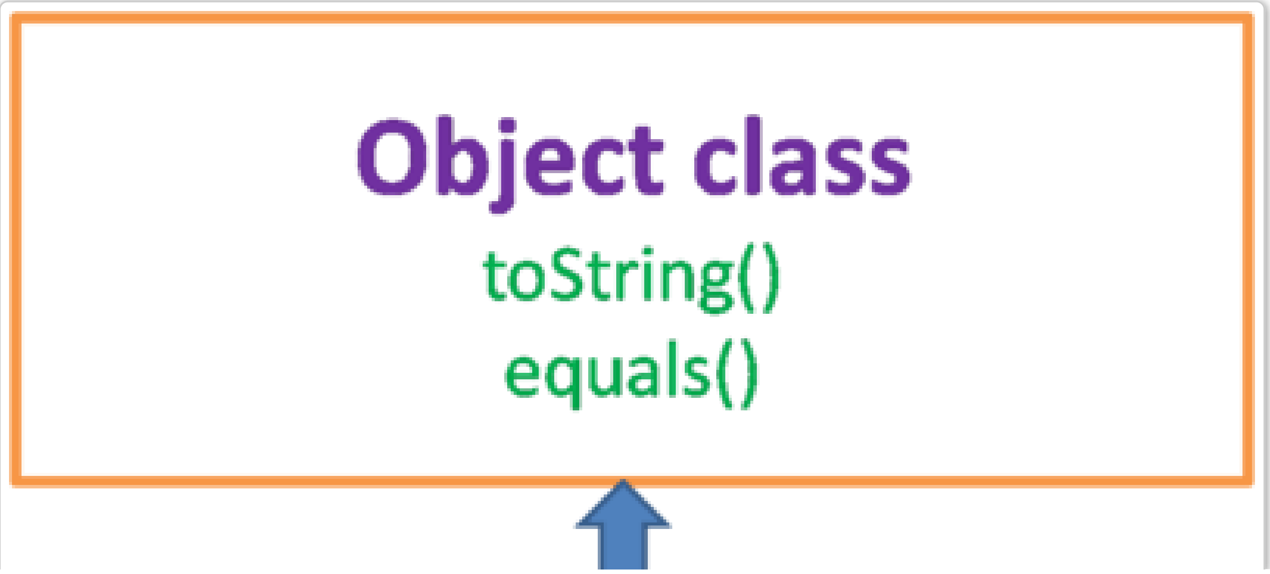
I have considered the Object class while writing these classes, for example for single inheritance in selenium.

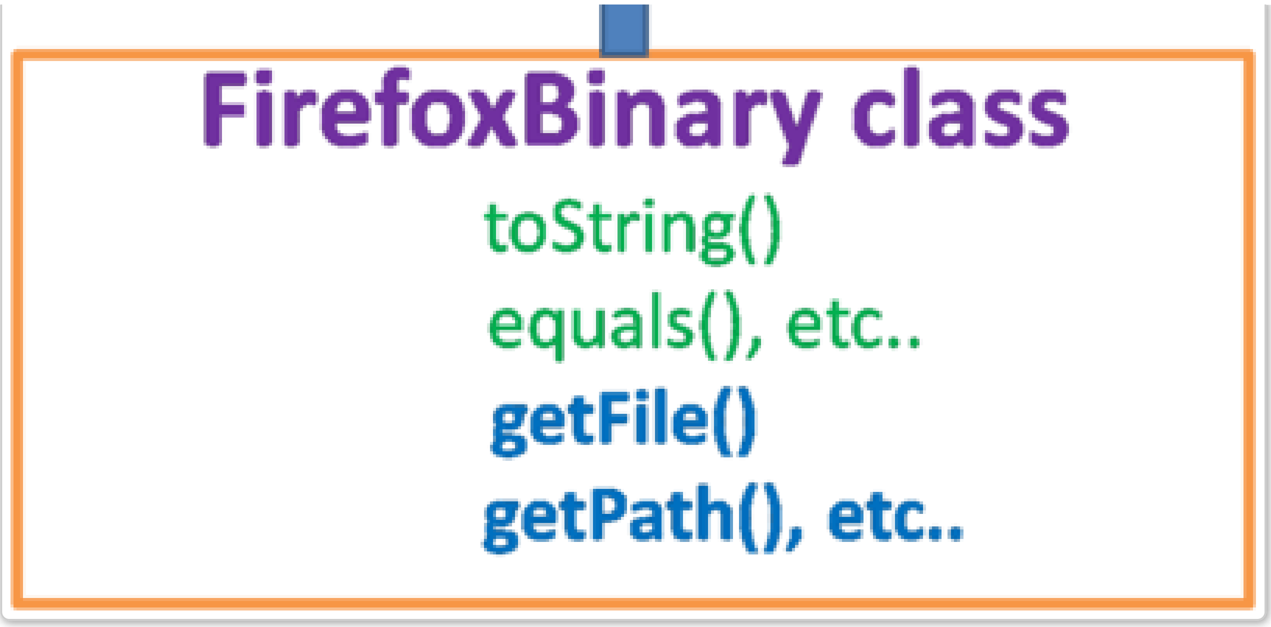
Few examples for single inheritance in selenium webdriver :

1. *Point Class*
2. *Dimension class*
3. *FirefoxBinary class*
4. *FirefoxProfile class*
5. *Rectangle class*
6. *JSON class*

From the below image, you can understand that FirefoxBinary class extends Object class and FirefoxBinary class will have all its own methods along with inherited Object class methods.

By default, if you write any class in java without writing any extends, still the single inheritance is applicable for it because all the classes in java are inherited from Object class.





Multi-level Inheritance :

When a class extends a class, which extends another class, then it is called multilevel inheritance.

Multilevel inheritance is nothing but a combination of more than one single inheritance.

In multiple inheritance, you have one path only to reach the highest class in the inheritance.

> *Minimum three classes are required to achieve the Multi-level inheritance in java.*

The last subclass will have all the properties present in its superclasses.

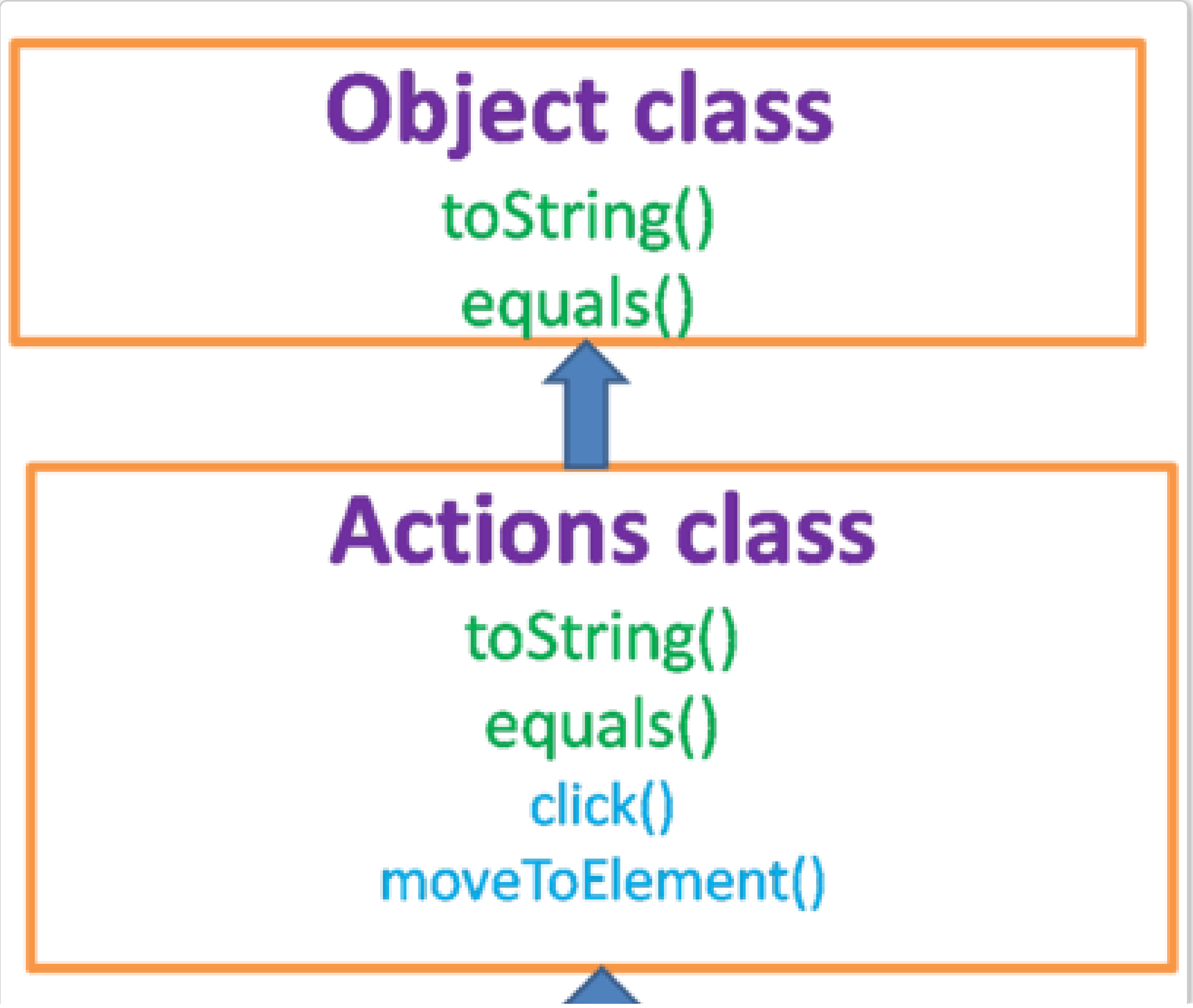
In the below example, you have only one path to reach from the last subclass to topmost class, **Grand child class -> Child Class -> Base class.**

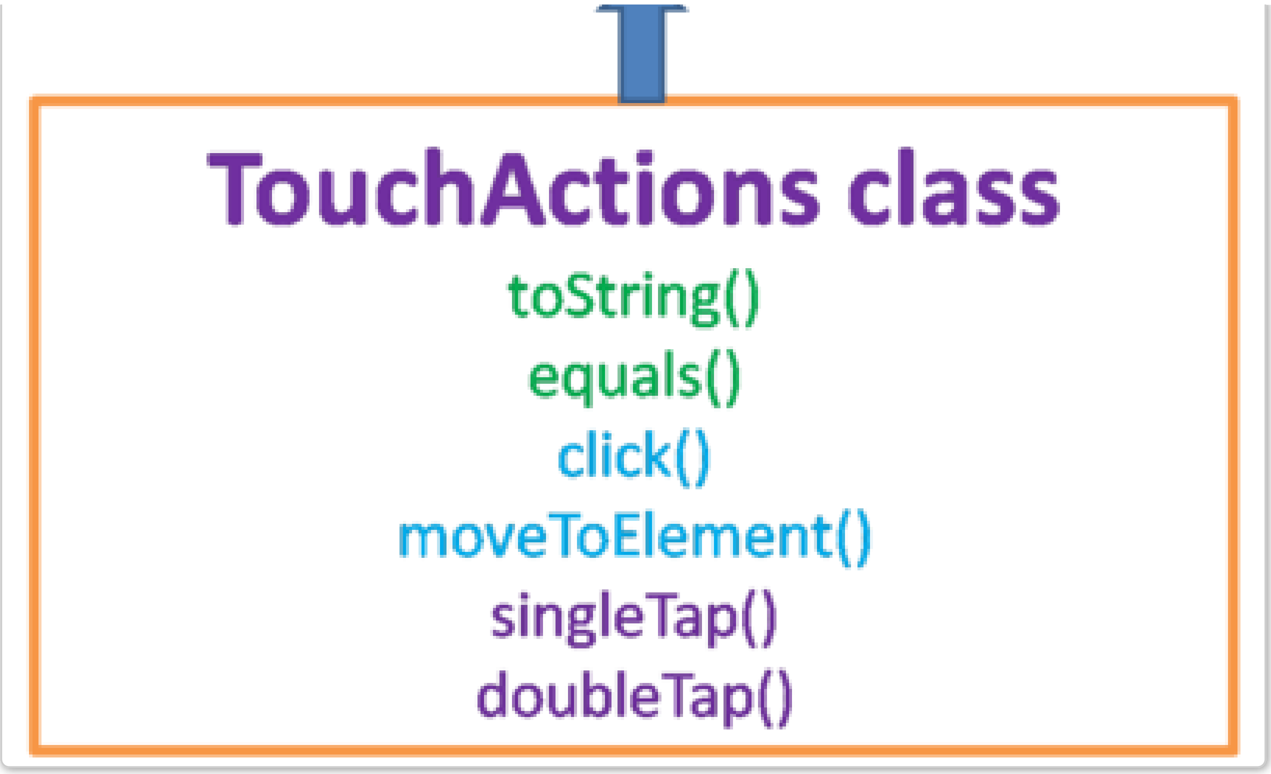
```
package cherchertech;
class BaseClass {
   public void testa() {
      System.out.println("Base class test");
   }
}
class ChildClass extends BaseClass{
   public void testb() {
      System.out.println("Child class test");
   }
}
class GrandChildClass extends ChildClass{
   public void testc() {
    public void testc() {
        System.out.println("Grand child class");
    }
}
public class MultiLevelInheritance {
    public static void main(String[] args) {
        GrandChildClass gc = new GrandChildClass();
        // method present in base class
        gc.testa();
        // method present in child class
        gc.testb();
        // method present in grand child class
        gc.testc();
    }
}
```

Multi-level Inheritance in Selenium

TouchActions in selenium works on the principle of Multi-level Inheritance, TouchActions class extends the **Actions class**, and Actions class extends the Object class in selenium.

Touch Actions class will have all the methods present in the Object class and **Actions class**, I have mentioned only a few methods from each class because of space constraints.





Multiple Inheritance :

Multiple inheritance is nothing but a class/interface is inheriting the properties of more than one class or interface.

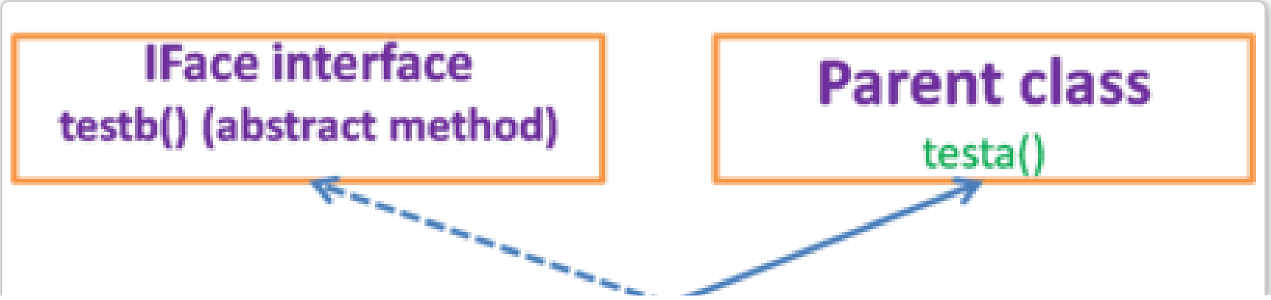
A class cannot inherit more than one class because it forms multiple paths to reach the topmost class from the same last subclass.

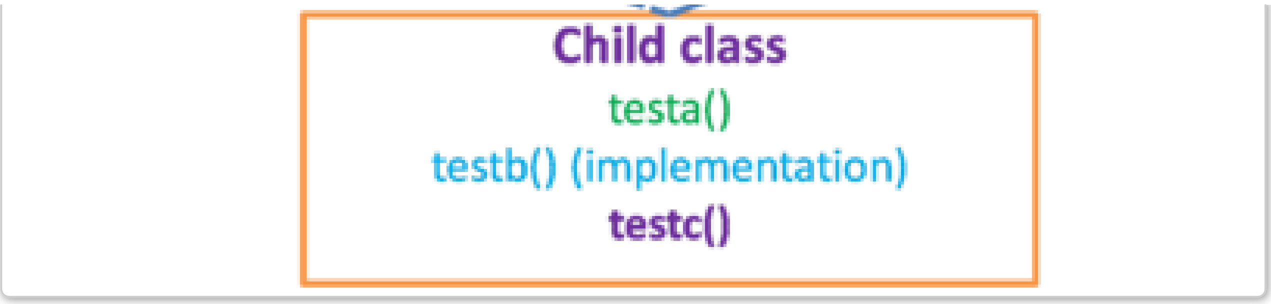
This is where you need to remember that all the classes in java are subclasses to Object class.

So Multiple inheritance is not possible when a class tries to inherit more than one class, if a class tries so, then it creates a diamond problem.

> *But a class can inherit one class and implement multiple interfaces and this is also called as multiple inheritance.*

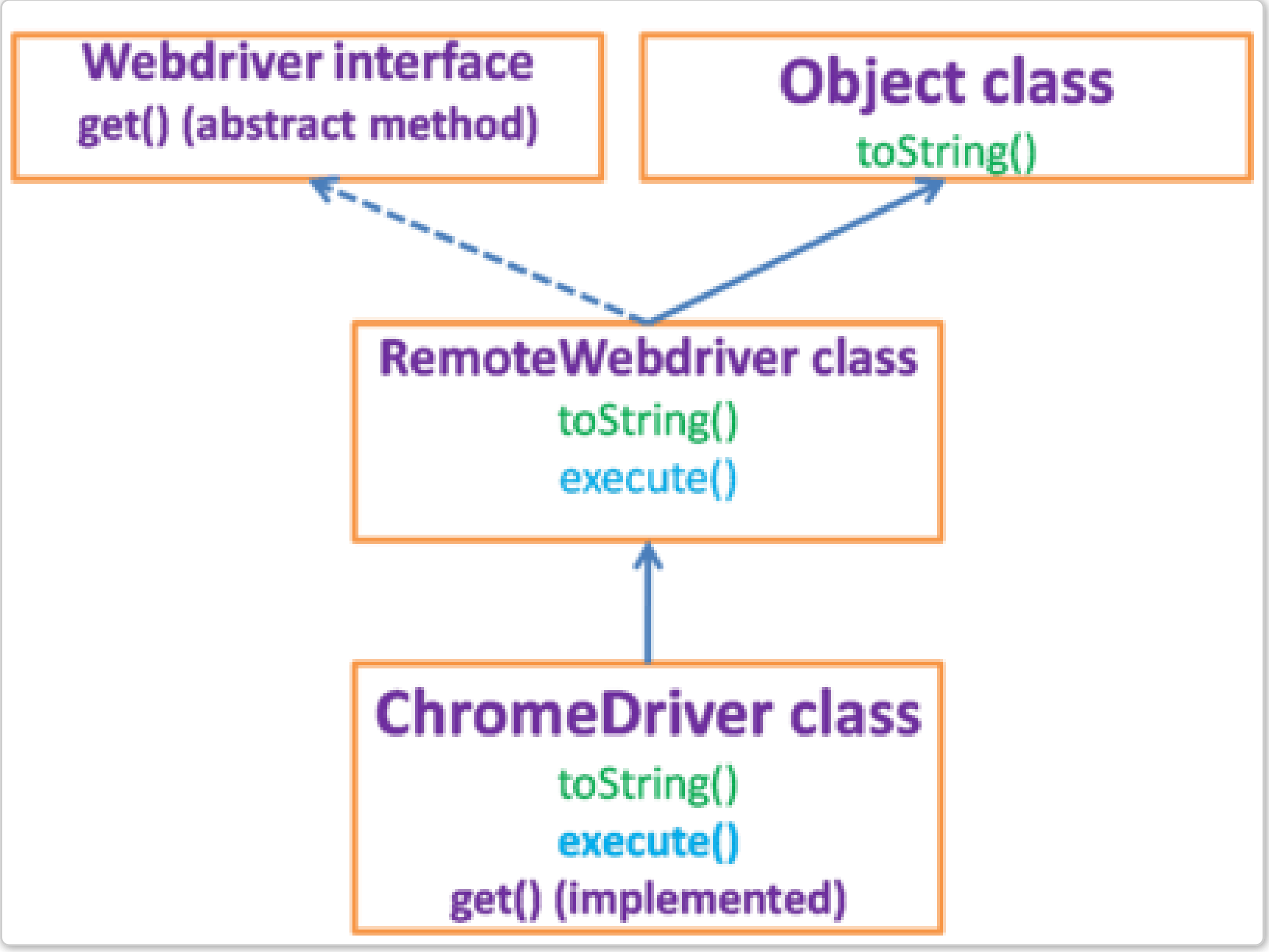
Similar to these Interfaces also can inherit from more than one interface, java allows multiple inheritance only because interfaces will not have any implementations.





Multiple inheritance in Selenium :

In selenium, ChromeDriver extends the RemoteWebdriver, RemoteWebdriver implements Webdriver and extends Object class. RemoteWebdriver follows Multiple-inheritance as it uses methods from Object class and also overrides the methods present in the Webdriver by implementing them.



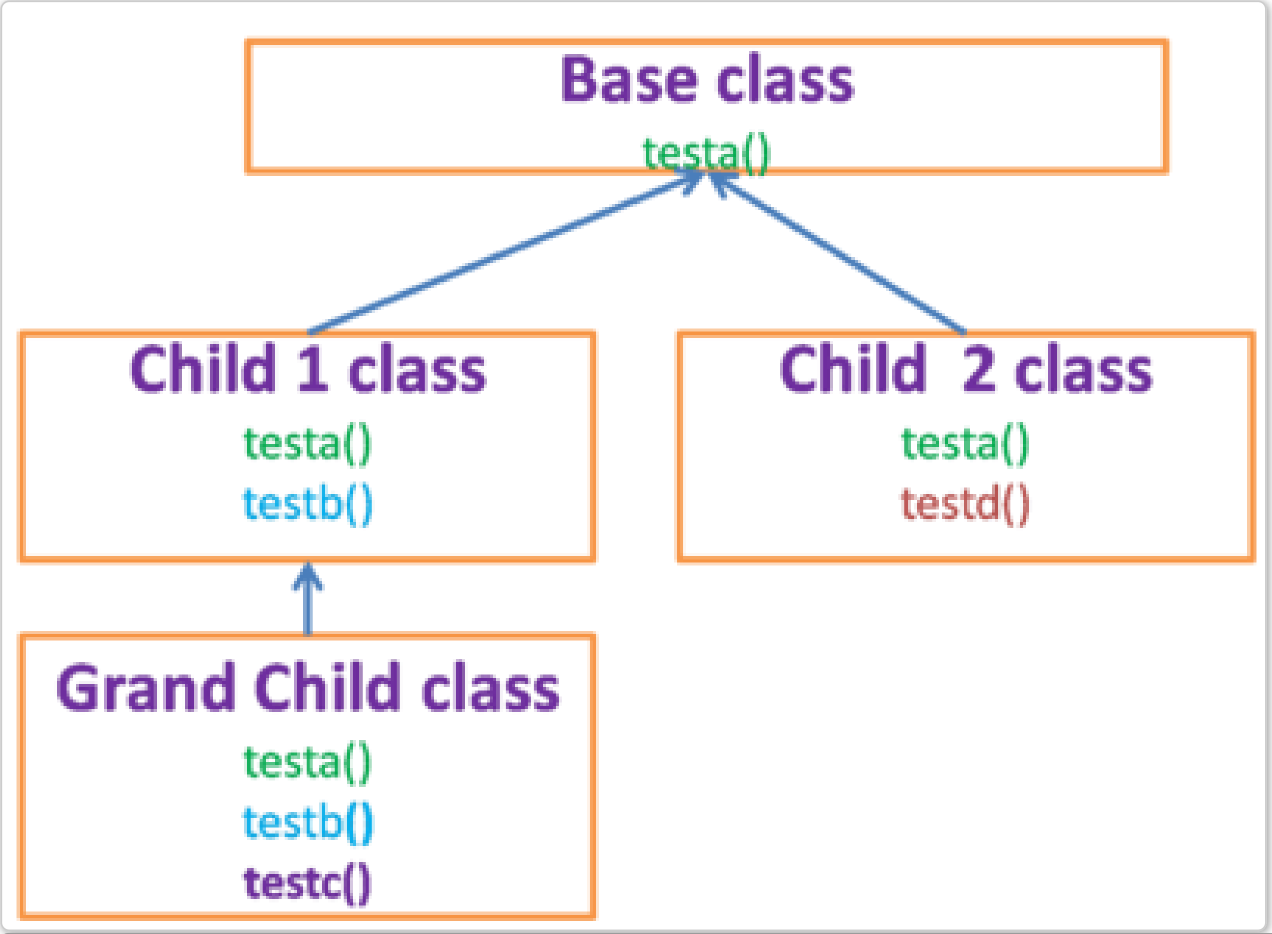
Hybrid Inheritance:

Hybrid inheritance is the combination of every type of inheritance that exists. As java doesn't support multiple inheritance, hybrid inheritance also can't be implemented.

If you consider the above definition, then Hybrid inheritance is not possible

For most of the java people, the Hybrid inheritance definition is : Combination of Single level and Multilevel inheritance.

Another way of saying : When two classes inherit properties from a single class, then it is called Hybrid inheritance.

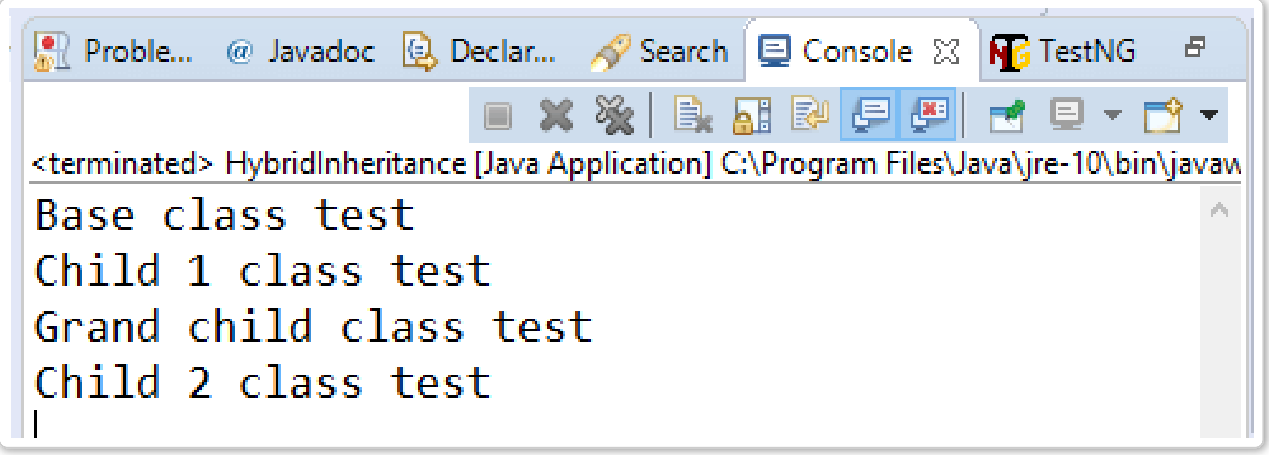


```
package cherchertech;
class BaseClass {
   public void testa() {
      System.out.println("Base class test");
   }
}
class Child_1_Class extends BaseClass{
   public void testb() {
      System.out.println("Child 1 class test");
   }

}
class Child_2_Class extends BaseClass{
   public void testd() {
      System.out.println("Child 2 class test");
   }
}
class GrandChildClass extends Child_1_Class{
   public void testc() {
      System.out.println("Grand child class test");
   }
}
public class HybridInheritance {
   public static void main(String[] args) {
      GrandChildClass gc = new GrandChildClass();
```

```
        // method present in base class
        gc.testa();
        // method present in child 1 class
        gc.testb();
        // method present in grand child class
        gc.testc();

        // create object child class 2
        Child_2_Class c2c = new Child_2_Class();
        // method present in child 2 class
        c2c.testd();
    }
}
```



```
<terminated> HybridInheritance [Java Application] C:\Program Files\Java\jre-10\bin\javaw
Base class test
Child 1 class test
Grand child class test
Child 2 class test
```

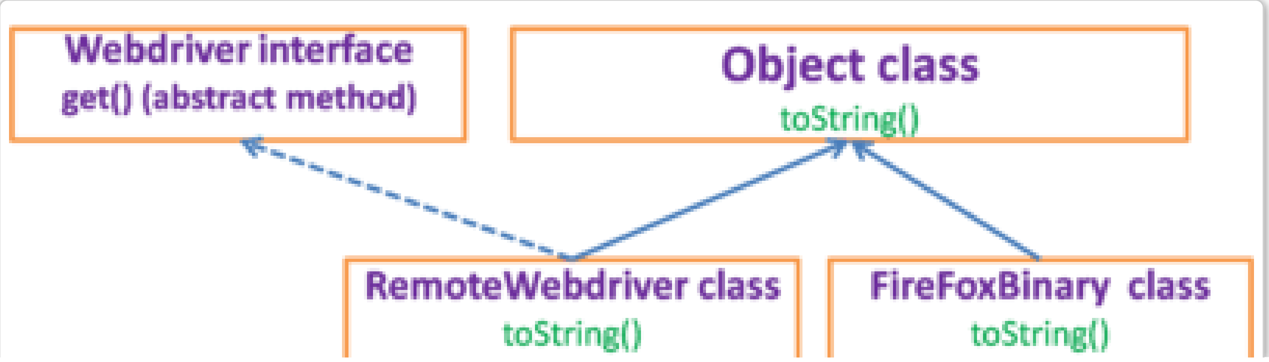
Hybrid Inheritance in Selenium :

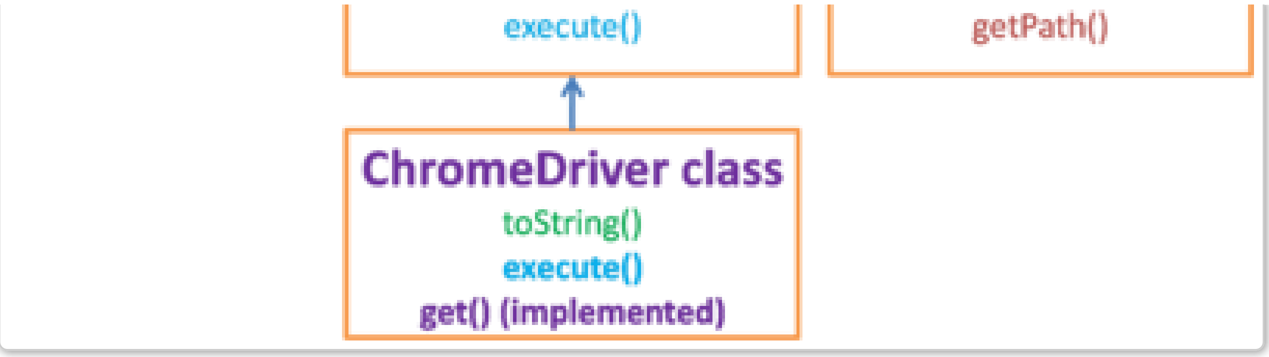
ChromeDriver class is the example for the Hybrid inheritance, ChromeDriver is inherited from the RemoteWedriver, and RemoteWebdriver is inherited from the Object class; this is one path, and this is nothing but Multi-level Inheritance.

In another path, ChromeDiver extends RemoteWebdriver, and RemoteWebdriver also implements the Webdriver interface.

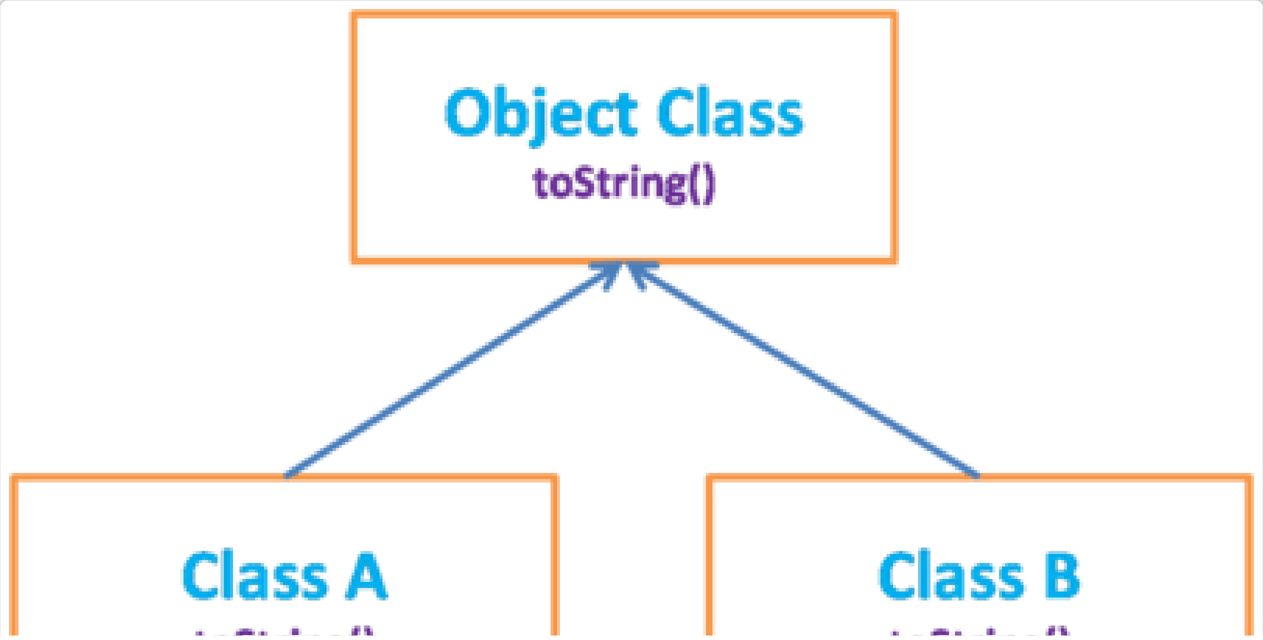
FirefoxBinary class extends the Object class this is third path, also called Single-Inheritance.

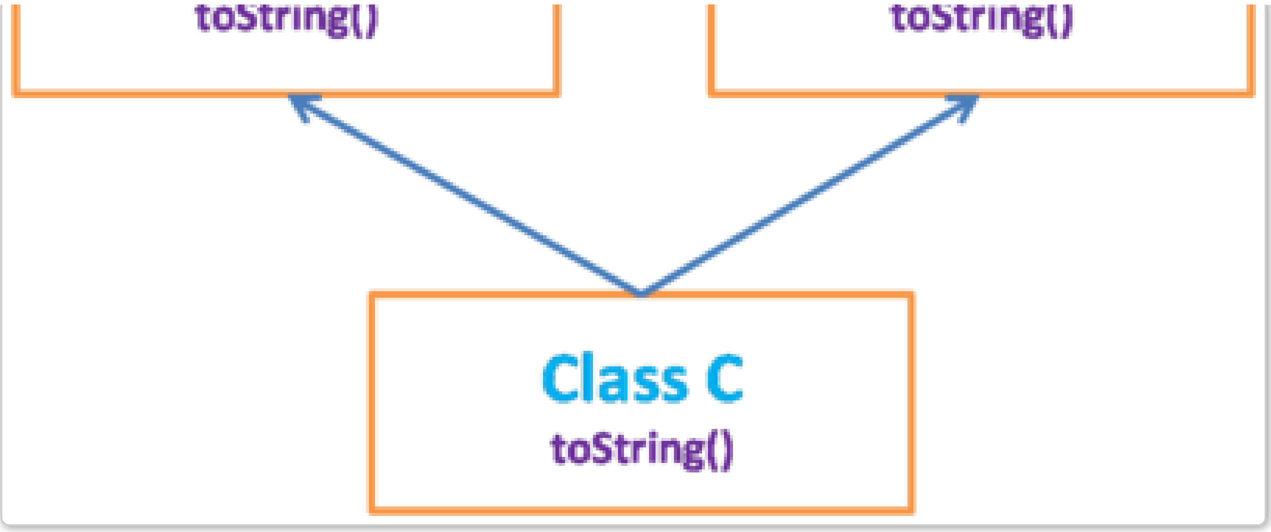
RemoteWebdriver follows multiple inheritance by implementing Webdriver and by extending Object class.





Diamond problem and Why Java Does not have it :





The above image shows the diamond problem; let me explain the problem.

1. Class A extends Object class
2. Class B extends Object class
3. Class C extends Class A and class B.
4. Object class has methods called toString().
5. The relation between class C and classes A, B is Multiple inheritance
5. Classes A, B, C have method toString(), because they inherit the Object class

Let's create an object for class C and call the toString() method, which indirectly gives a call to the toString() method present in the Object class.

Now take a moment and think, In how many ways we can reach the Object class from class C, yes You are right, in two ways we can reach it.

1. Class C -> Class A -> Object class

2. Class C -> Class B -> Object class

Take a moment and think which is the way you are going to use, Path 1 or Path 2 (consider you don't know the consequences of the path, good or bad).

Are you able to conclude? No, okay now the same thing happens with the Programming language as well; it gets confusing which path to choose like Mr.Nobody movie; the Programming language will throw an exception of ambiguity.

To avoid this java Programming language author made a decision not to include the multiple inheritance. So we don't have Multiple inheritance between the classes.

In My language : There are two section of school students, among them each section has a student with roll number 1224, If principal asks an attender to call a guy (one guy) with roll number 1224, who does that attender will call ?, he cannot call anybody as there are two guys with same roll number, unless he is stupid like me.

Upcasting :

Upcasting is casting a subtype to a supertype, upward to the inheritance tree. In other words, Making a child or grandchild as the grandfather.

While creating an Object to a child class, we will write Parent class type as the type for the object.

We can mention the type to which '(A)' we are going to Upcast the object.

```
A a = (A) new B(); // where B is sub class of A
```

Once we upcast the subclass, then only methods from the Superclass are available for that object.

But if a subclass has any method which is the same as the Superclass method, then the superclass method will be overridden by the subclass, and sub-class method implementation only will work.

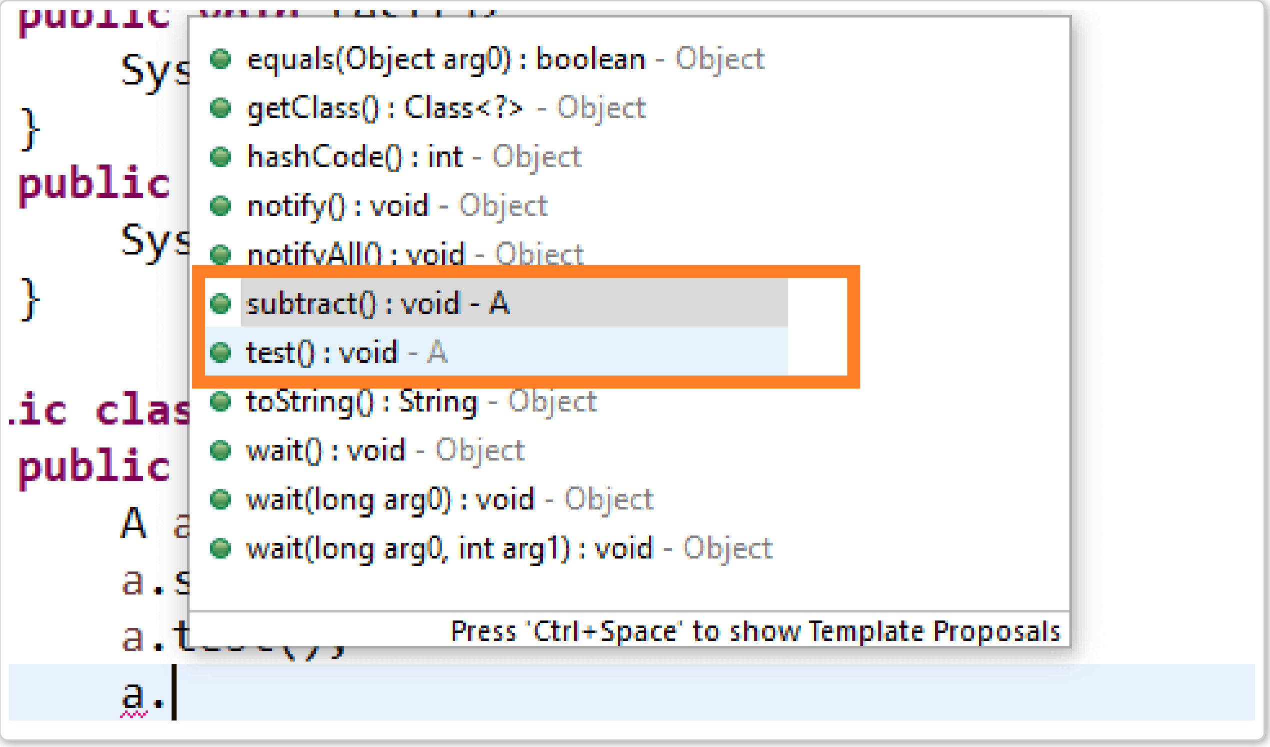
```
class A {
   public void test(){
      System.out.println("test A");
   }
   public void subtract(){
      System.out.println("subtract A");
   }
}
class B extends A{
   public void test(){
      System.out.println("test B");
   }
   public void add(){
      System.out.println("add B");
   }
}
public class C {
   public static void main(String[] args) {
      A a =(A) new B();
      a.subtract();
      a.test();
   }
}
```

The output of the up-casting program

```
subtract A
test B
```

We don't have a subclass method in suggestions as well

We don't have a subclass method in suggestions as well.



```
public void test() {
        Sys
}
public
        Sys
}

.ic clas
public
        A a
        a.s
        a.test();
        a.
```

- equals(Object arg0) : boolean - Object
- getClass() : Class<?> - Object
- hashCode() : int - Object
- notify() : void - Object
- notifyAll() : void - Object
- subtract() : void - A
- test() : void - A
- toString() : String - Object
- wait() : void - Object
- wait(long arg0) : void - Object
- wait(long arg0, int arg1) : void - Object

Press 'Ctrl+Space' to show Template Proposals

Auto-Upcasting :

Auto-upcasting is nothing but the upcasting occurs automatically by the compiler, we can write the above example's object creation like below

```
A a = new B();
```

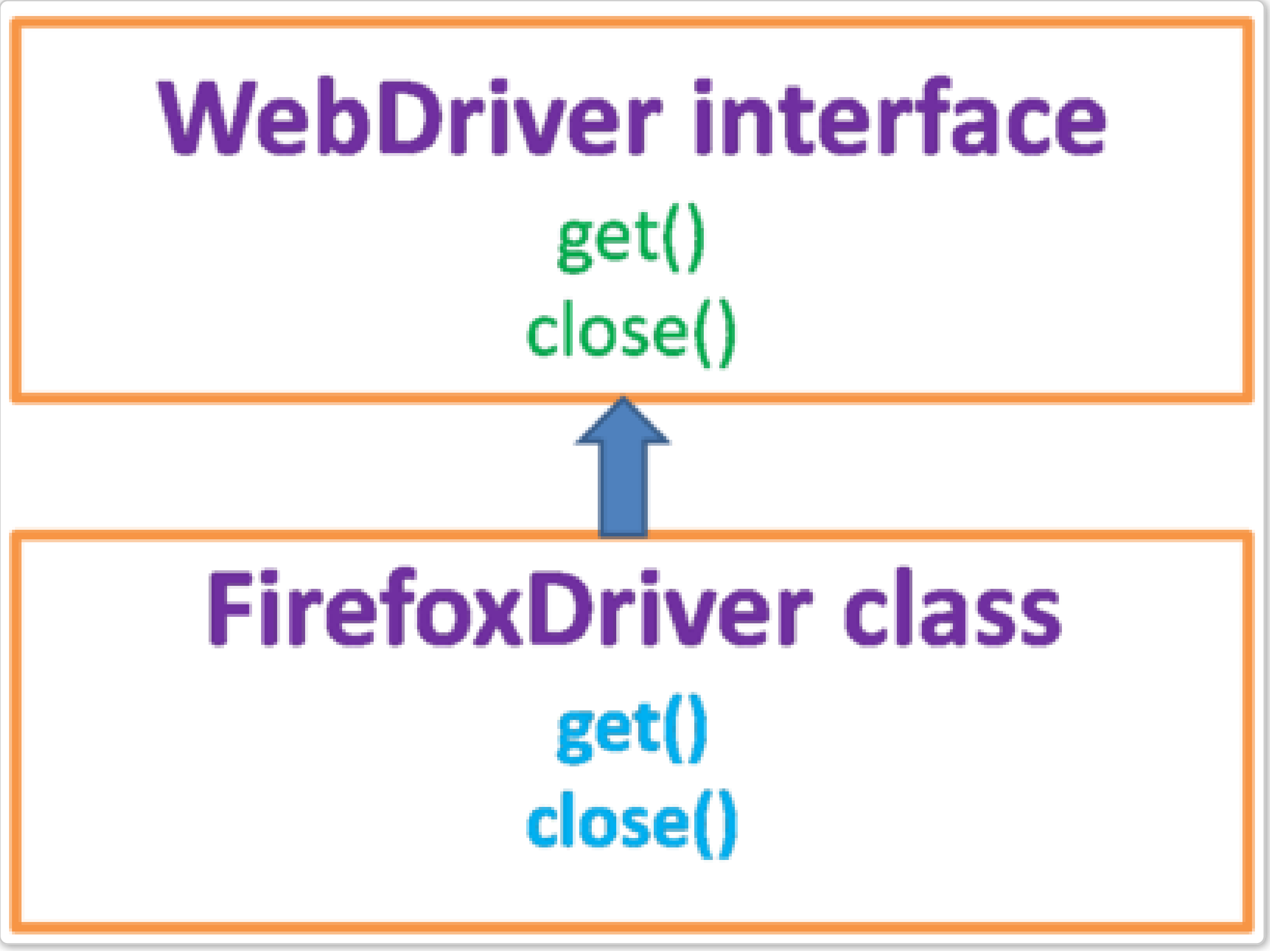
Auto - Upcasting in Selenium :

I hope you know where auto-upcasting occurs in selenium webdriver; if you are not aware of it, then auto-upcasting occurs in the **first line of the webdriver code**.

```
WebDriver babyDriver = new FirefoxDriver();
```

In the above example, we are assigning the lower level class object to the upper-level class/interface type.

WebDriver is Interface, and FirefoxDriver class extends the Webdriver interface, so in the above example, we are assigning the FirefoxDriver Object to Webdriver type.



Downcasting

Downcasting is nothing but converting the subclass type which is upcasted to Superclass into subclass type; instead of performing the downcasting, we can directly create the object to subclass.

For creating a downcasting process, Upcasting is mandatory.

For above Upcasting example:

```
A a = new B(); // auto-upcasting
B b =(B) new B(); // downcasting
"Donot you think instead of writing above two steps, we can write like below"
B b = new B();
```

Recommended Readings

- Polymorphism in OOPS | Selenium
- Abstraction in OOPS | Selenium
- Encapsulation in OOPS | Selenium
- Wrapper Classes in Java
- Benefits of Java
- Sprint
- Core Java Interview Questions Set 1
- selenium framework interview questions