prepSmart Quest

Quest for Knowledge, Achieve with PrepSmart

SDET: Java-Interface and Abstract Class Concepts

INTERFACE AND ABSTRACT CLASS

Before we start with the topic INTERFACE AND ABSTRACT CLASS, let's go with layman doubts

1. Why we need Interface class at all…. When we can write separate class with concrete methods with logics insides.
2. Ok, we started using interface class for some reason, then why we need same kind of class called Abstract class
3. Now Interface has methods which are declared in the class , why there is no whole logic in it. Why we need such class when there is no methods which have logics.
4. When both class (Interface and Abstract) is similar.. why we need two types of classes
5. What is the real life use of these classes (Interface and Abstract) in testing.
6. why interface does not have constructor

Before diving deep into knowing the classes, let's get aware of few terms which will help us knowing the Interface and Abstract classes.

1. **Implementation**
2. **Extends**
3. **Static**
4. **Inner class**
5. **Final**
6. **Anonymous Class**
7. **Instantiated**
8. **Public**
9. **loose coupling**
10. **Tight coupling**

Stack Memory	Stack Memory in Java is used for static memory allocation and the execution of a thread. It contains primitive values that are specific to a method and references to objects that are in a heap, referred from the method. Access to this memory is in Last-In-First-Out (LIFO) order.
Analogy	Stack Literal meaning is pile of object , typical neatly arranged.When there is stack of books, the last book that is placed on the top , will be first to get removed. Thats what stack name is coined here
Comment	The Stack stores primitive values like int a; bolean b; or the variable created to refer object by new keyword. Abc obj = new Abc(); here obj (the reference variable is stored).
Heap Memory	The heap memory is for dynamic allocation. Unlike the stack, there's no enforced pattern to the allocation and deallocation of blocks from the heap; you can allocate a block at any time and free it at any time. This makes it much more complex to keep track of which parts of the heap are allocated or free at any given time
Analogy	Heap Literal meaning is disordered collection of object or particle like mountain shape.So heap has no proper order of placement. So allocating and deallocating of memory takes little more time and seems to complex.
Object	In Java an object is allocated memory dynamically in heap. A call to the default constructor creates an object of the class in heap memory.
Reference Variable	A Reference Variable simply means a variable allocated memory statically in the stack at compile time.
Implement	A Java keyword included in the class declaration to specify any interfaces that are implemented by the current class
Analogy	Implementation is the noun form of the verb implement, or "to carry out or accomplish," and you'll often see it used in reference to a government plan or act. Use this word to describe the process of turning formal plans — often very detailed conceptual plans that will affect many — into reality.
Comment	In a gist , it’s a plan on paper (without concrete substance) or high level overview which is to be implemented and comes into life only when implemented. In real world scenarios, the rule are defined in books, but does not have shape or form, which can be seen, felt or analyzed how it works in the real world. It is felt or takes shape when it is once implemented in the real world. For suppose building a concrete structure, in civil engineering, there may certain rules , but it acquires a form or body only when its implemented in the real world. Like wise Interface is like rule book, where methods are just declared , but its acquires the logic and body when its implemented in a derived class and we can see it, sense how it is working on our application or behaving after its implementation.
Extends	The extends keyword extends a class (indicates that a class is inherited from another class). In Java, it is possible to inherit attributes and methods from one class to another. We group the "inheritance concept" into two categories: • subclass (child) - the class that inherits from another class • superclass (parent) - the class being inherited from

Analogy	The verb extend can have several related meanings, including thrust out, continue, broaden, expand, unfold, span, or increase in scope. Extend, as a verb, is used in many ways. .For example, If someone extends a monetary help, its depends upon other person to what extends he/she can take the money as per requirement.
static	In terms of Java, static keyword is used when a resource(variable, method) is in shareable mode i.e that resource is used by many objects. If we make a resource static then only single copy of that resource is created for all the objects. We can make a variable ,method ,block and inner class as static. Here object(memory allocation in heap area) remains same, but the values can be changed
Analogy / Literal Meaning	pertaining to or characterized by a fixed or stationary condition. showing little or no change: a static concept; a static relationship. lacking movement, development, or vitality
Final	final keyword is used when we don’t want to change the value of that resource(variable ,method , class) .If we declare a resource final then we can’t manipulate the value.
Analogy	In a gist, which is cannot be changed anymore.
Inner Class	Java inner class or nested class is a class which is declared inside the class or interface. We use inner classes to logically group classes and interfaces in one place so that it can be more readable and maintainable
Instantiated	Instantiation means defining or creating new object for class to access all properties like methods, fields, etc. from class.
Initialization	Initialization means assigning initial value to variables after declaring or while declaring
Anonymous Class	In Java, a class can contain another class known as nested class. It's possible to create a nested class without giving any name. A nested class that doesn't have any name is known as an anonymous class. An anonymous class must be defined inside another class. Hence, it is also known as an anonymous inner class.
public	public is a Java keyword which declares a member's access as public. Public members are visible to all other classes. This means that any other class can access a public field or method.
Loose Coupling	When an object gets the object to be used from the outside, then it is a loose coupling situation. As the main object is merely using the object, this object can be changed from the outside world easily marked it as loosely coupled objects.
Tight Coupling	It is when a group of classes are highly dependent on one another. This scenario arises when a class assumes too many responsibilities, or when one concern is spread over many classes rather than having its own class. The situation where an object creates another object for its usage, is termed as Tight Coupling.

Let’s answer the questions in the first page Step by Step .

To know about the Interface, we should know about the abstraction first:

Literal Meaning of Abstraction: **Abstract** for something that is not a material object or is general and not based on specific examples. **Abstract** is from a Latin word **meaning** "pulled away, detached," and the basic idea is of something detached from physical, or concrete, reality.

In a gist, which does not have pure physical form or body or which just gives us a basic idea only.

Now let’s see about Interface.

Interface:

An interface is a reference type in Java. It is similar to class. It is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface.

Along with abstract methods, an interface may also contain constants, default methods, static methods, and nested types. Method bodies exist only for default methods and static methods.

Writing an interface is similar to writing a class. But a class describes the attributes and behaviors of an object. And an interface contains behaviors that a class implements.

Unless the class that implements the interface is abstract, all the methods of the interface need to be defined in the class.

An interface is similar to a class in the following ways −

- An interface can contain any number of methods.
- An interface is written in a file with a .java extension, with the name of the interface matching the name of the file.
- The byte code of an interface appears in a .class file.
- Interfaces appear in packages, and their corresponding bytecode file must be in a directory structure that matches the package name.

However, an interface is different from a class in several ways, including −

- You cannot instantiate an interface.
- An interface does not contain any constructors.
- All of the methods in an interface are abstract.
- An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.
- An interface is not extended by a class; it is implemented by a class.
- An interface can extend multiple interfaces.

Let's see some practical example and answer the below questions :

- Why we need Interface class at all…. When we can write separate class with concrete methods with logics insides.
- Now Interface has methods which are implemented by the class , why there is no logic /body in it. Why we need such class when there is no methods which have logics.
- What is the real life use of these class in testing
- Why interface does not have constructor

You might have gone through definition of Interface, but its need practical example to understand .

Defined Methods: The method which has only name defined.

```
void get();
```

Concrete Methods: The method which has body also.

```
                public void abc(){
          //  statement or logic
}
```

So let's discuss why we need Interface.

As you know WebDriver is a Interface and its has set of the defined methods in it.

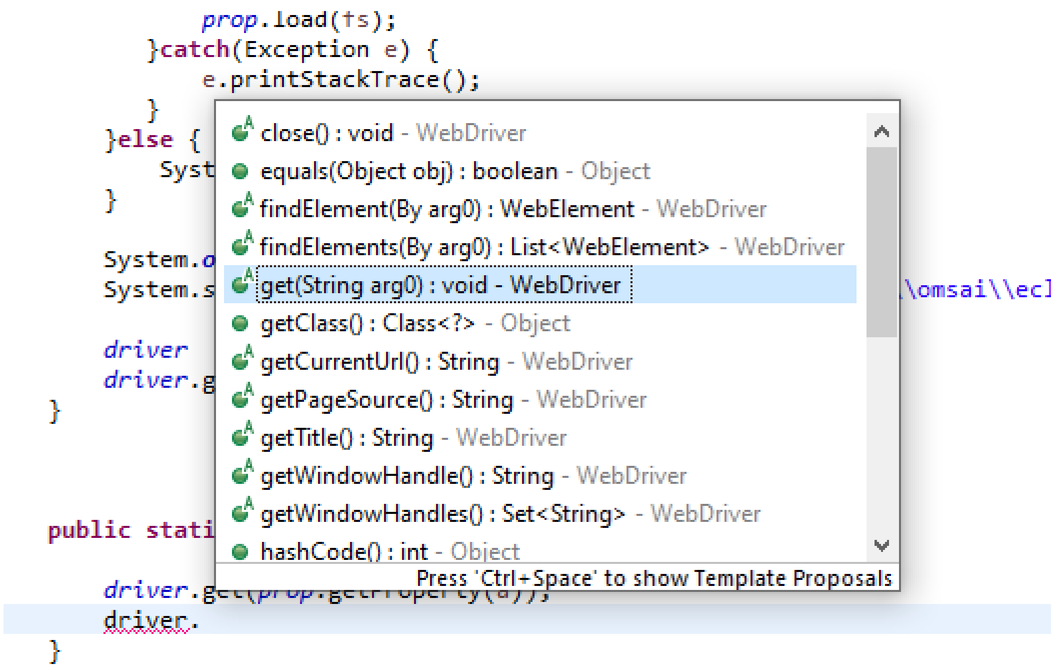
Those methods are

- close()
- findElement(By by)
- findElements(By by)
- get(java.lang.String url)
- getCurrentUrl()
- getPageSource()
- getTitle()
- getWindowHandle()
- getWindowHandles()
- manage()
- quit()
- switchTo()

Now as a end-user / automation Tester , we are aware of the methods for driver irrespective of browser.

Browser may be chrome , Firefox , Safari , Android ,IE

Now understand the difference here.



```
            prop.load(fs);
        }catch(Exception e) {
            e.printStackTrace();
        }
    }else {
        Syst
    }

    System.o
    System.s                                                         \omsai\\ecl

    driver
    driver.g
}


public stati

    driver.get(prop.getProperty(a));
    driver.
}
```

```
close() : void - WebDriver
equals(Object obj) : boolean - Object
findElement(By arg0) : WebElement - WebDriver
findElements(By arg0) : List<WebElement> - WebDriver
get(String arg0) : void - WebDriver
getClass() : Class<?> - Object
getCurrentUrl() : String - WebDriver
getPageSource() : String - WebDriver
getTitle() : String - WebDriver
getWindowHandle() : String - WebDriver
getWindowHandles() : Set<String> - WebDriver
hashCode() : int - Object
                     Press 'Ctrl+Space' to show Template Proposals
```

Chrome has its own way to launch URL, to fetch URL, close the browser, identify the elements present in it.

Now Safari has its own way of launching and closing and identifying the elements.

Again the Firefox has its own way to open, close and identify the elements.

Now coming to the example to understand

As automation tester, you may know with the common methods , that the driver can be handled by like launching, getting page title, finding the elements.

So Interface is nothing but set of declared common methods which can be used across similar classes.

With below example it will be more clear.

Now if you are scripting a code on chrome browser and Firefox browser, the code need to launch the URL, findElements, getTitle, need to switch between windows, and you may be expecting same common methods to be applicable for chrome as well as Firefox.

There cannot be different method names for chrome and different methods names for firefox and different method name for Android, Safari. But each browser class can have its own logic inside of method. Eg: get() method when implemented in chrome class , will be have its own logic but the method name remains same for it. Same as in the case of Firefox class and safari class. So Interface gives the independence to each class to have its own logic inside the method (overriding method) according to its own usage.

Now as automation tester, you can use get() method, without worrying about what browser you are working on but the logic for each browser class will be different.

And every class should   compulsorily implement those methods because it should not miss out or else it will affect the end user. For example, if chrome class implements get() and firefox class does not implement it, then end user wont be able to work on Firefox. So, similar class should have common method that are declared in the interface build for them. That is why interface forces the class implementing it to implement the declared methods.

If we create :

ChromeDriver driver=new ChromeDriver();

chrome driver methods will be executed.

And if we create :

WebDriver driver=new ChromeDriver();

again ChromeDriver methods are executed [as per method overriding].

Then why we write latter one only while executing?

Answer:

WebDriver driver = new ChromeDriver();

Through WebDriver driver = new ChromeDriver(); we are creating an instance of the WebDriver interface and casting it to ChromeDriver class. All the browser drivers like:

FirefoxDriver

ChromeDriver

InternetExplorerDriver

PhantomJSDriver

SafariDriver etc

implemented the WebDriver interface (actually the RemoteWebDriver class implements WebDriver Interface and the Browser Drivers extends RemoteWebDriver). So if we use WebDriver driver, then we can use the already initialized driver (as common object variable) for all browsers we want to automate e.g. Mozilla, Chrome, InternetExplorer, Edge, Opera, Safari as follows:

```
WebDriver driver = new FirefoxDriver();

// or
WebDriver driver = new ChromeDriver();
// or
WebDriver driver = new InternetExplorerDriver();
// or
WebDriver driver = new EdgeDriver();
// or
WebDriver driver = new OperaDriver();
// or
WebDriver driver = new SafariDriver();
```

So above example clears the doubt of these three questions.

- Why we need Interface class at all…. When we can write separate class with concrete methods with logics insides.
- Now Interface has methods which are implemented by the class , why there is no logic /body in it. Why we need such class when there is no methods which have logics.
- What is the real life use of these class in testing

Answer: The Interface is collections of declared method which should be implemented across similar class .The interface does not have concrete methods because the class implementing it should have its own logic according to its nature but the method name cannot be changed but only can add the logic /body to it .(Its called method overriding).

Eg: Chrome class can have its own logic for get(), Safari can have its own logic for get(). But the end user is not aware of the what's the logic the individual class is implementing it. These is called abstraction (not letting the end user to know whole logic but giving the method name to work with it)

- why objects cannot be created for the Interface class

Answer: Java has golden rule that if methods which are only defined and not concrete in a class cannot have constructor. So Interface which is purely abstract cannot have constructor and without constructor objects cannot be created.
And more over methods, variables are static , for which memory is already allocated in heap area, creating object will again and again will waste the memory.

- Why interface does not have constructor

- An Interface in Java doesn't have a constructor because all data members in interfaces are public static final by default, they are constants (assign the values at the time of declaration).
- There are no data members in an interface to initialize them through the constructor.
- In order to call a method, we need an object, since the methods in the interface don't have a body there is no need for calling the methods in an interface.
- Since we cannot call the methods in the interface, there is no need of creating an object for an interface and there is no need of having a constructor in it.

- Why we need interface when we have Abstract Class?

  Answer: As you know , In Java we can extend only one Class but can implement multiple interfaces.For example if you want to create a background Thread in your Class and your extending some Class for reusability, then how will you extend Thread Class? In this case Runnable comes to rescue which is an interface.
  The advantage is Interface solves the Multi-Inheritance problem/ Diamond problem. A class implement as many as Interface.

- Why the keyword **implements** is used for interface , why not **extends**?

  Answer: To understand this , we need to understand Interface and its nature. As we know Interface is set of defined method, which is similar to a country having rules and states implementing has to implement it fully. So Interface is like having set rules (defined method) which has to be implemented or no other go.

  And Extends keyword more fits in the case, where class has the liberty to use or inherit the methods of super class depending upon the requirement of it. Its like someone extending help and other person has liberty to take it or not depending upon the requirement.

- If its concrete methods in the super class , it can be called depending upon the usage ,
- If its abstract method it has to get override in the child class and it happens only when the super class is abstract class.

Abstract Class :

What is Abstraction?

Answer: **Abstraction** is a process of hiding the implementation details and showing only functionality to the user.

Now you might have understood about Abstraction and process of hiding from WebDriver Interface example in the Interface section.

What is Abstraction in Java ?

Answser: A class which is declared with the abstract keyword is known as an abstract class in Java. It can have abstract and non-abstract methods (method with the body).

Way to achieve Abstraction?

Answer: There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%)
2. Interface (100%)

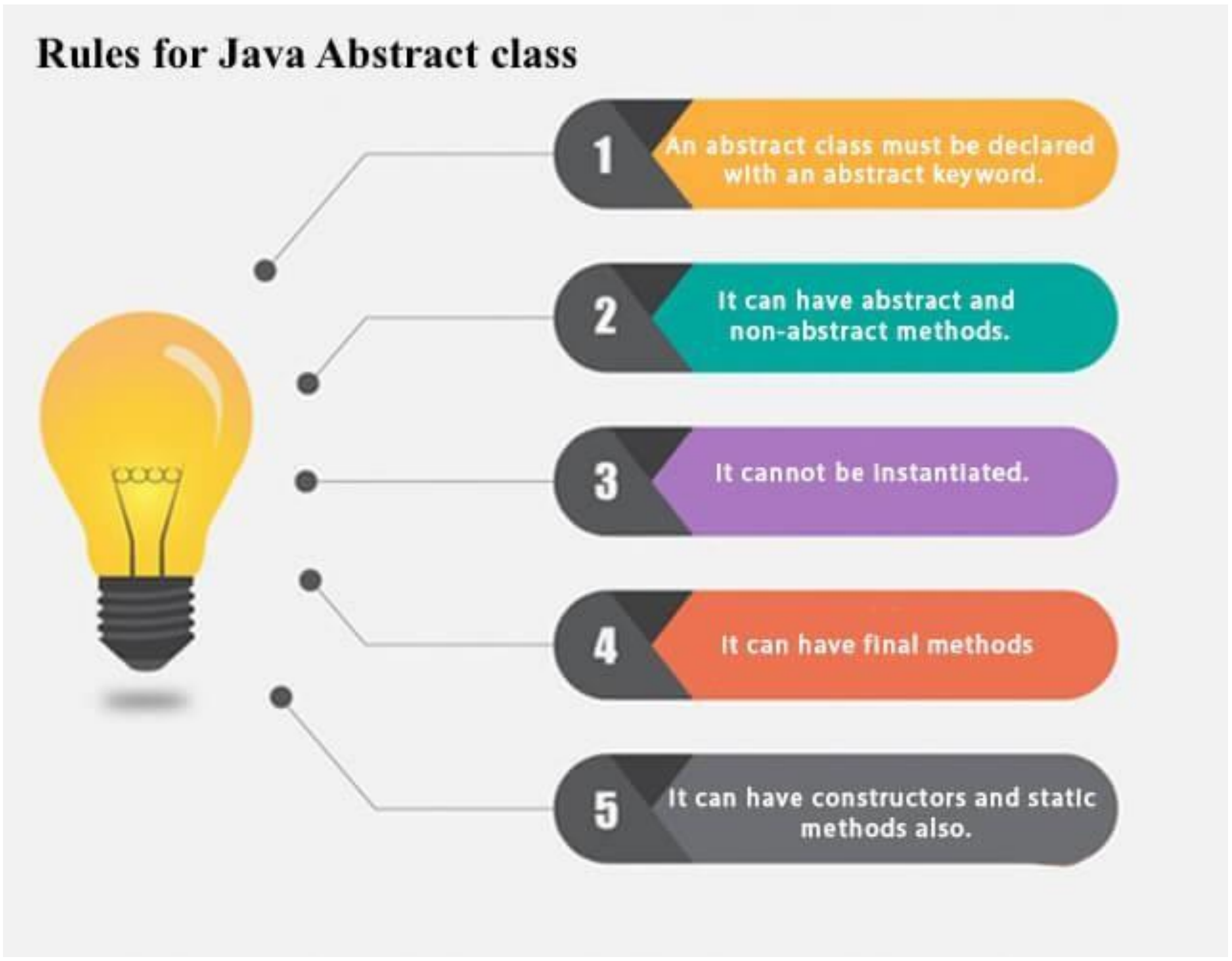
Abstract class in Java

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.

Points to Remember

- 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.


Rules for Java Abstract class
1 An abstract class must be declared with an abstract keyword.
2 It can have abstract and non-abstract methods.
3 It cannot be instantiated.
4 It can have final methods
5 It can have constructors and static methods also.

Code Snippet:

```
 abstract class Bike{
 abstract void run();
 }
class Honda4 extends Bike{
void run(){System.out.println("running safely");}
public static void main(String args[]){
Bike obj = new Honda4();
obj.run();
```

```
  }
  }
```

Code Snippet:

```
abstract class Shape{
abstract void draw();
}
//In real scenario, implementation is provided by others i.e. unknown by end user
class Rectangle extends Shape{
void draw(){System.out.println("drawing rectangle");}
}
class Circle1 extends Shape{
void draw(){System.out.println("drawing circle");}
}
//In real scenario, method is called by programmer or user
class TestAbstraction1{
public static void main(String args[]){
Shape s=new Circle1();//In a real scenario, object is provided through method, e.g., getSh
ape() method
s.draw();
}
}
```

```
OutPut: drawing circle
```

Why Abstract class cannot be instantiated ? / Why object cannot be created for Abstract class?/ When Abstract class can have constructor , why objects cannot be created?

Answer:

abstract class can have both concrete and abstract methods ; so having a **constructor in abstract class is just like having a concrete method** .Till the constructor is not tried to be invoked within that abstract class .

As , for invoking , we need to create object , which is a concept of instantiating , that is against the protocol of abstract class . Though, this constructor can be called after extending this abstract class into a concrete class and creating an object of concrete class .

Logically even, we try to create object in the child class using Abstract class constructor it will show error.

In hypothetical situation, if a object gets created by using Abstract class constructor , Abstract class anyhow will have at least one abstract method which not defined and the object created will not able to call the unimplemented / abstract method and which will throw compile time error.

Interface cannot have a constructor , because it is purely abstract . **It does not support concrete methods** . And , hence it does not have constructor.

- What happens if we try to create object using Abstract class constructor?

Answer: Its legal to have constructor in Abstract class as the Abstract class have both concrete methods and Abstract method. So constructor is as equal as concrete method.

But when we try to create object using Abstract class constructor, it will throw compile time error as all the methods are not implemented , at least one method remains Abstract, and which cannot be called by object created in the child class.

- **Why we need abstract class at all when we have Interface?**

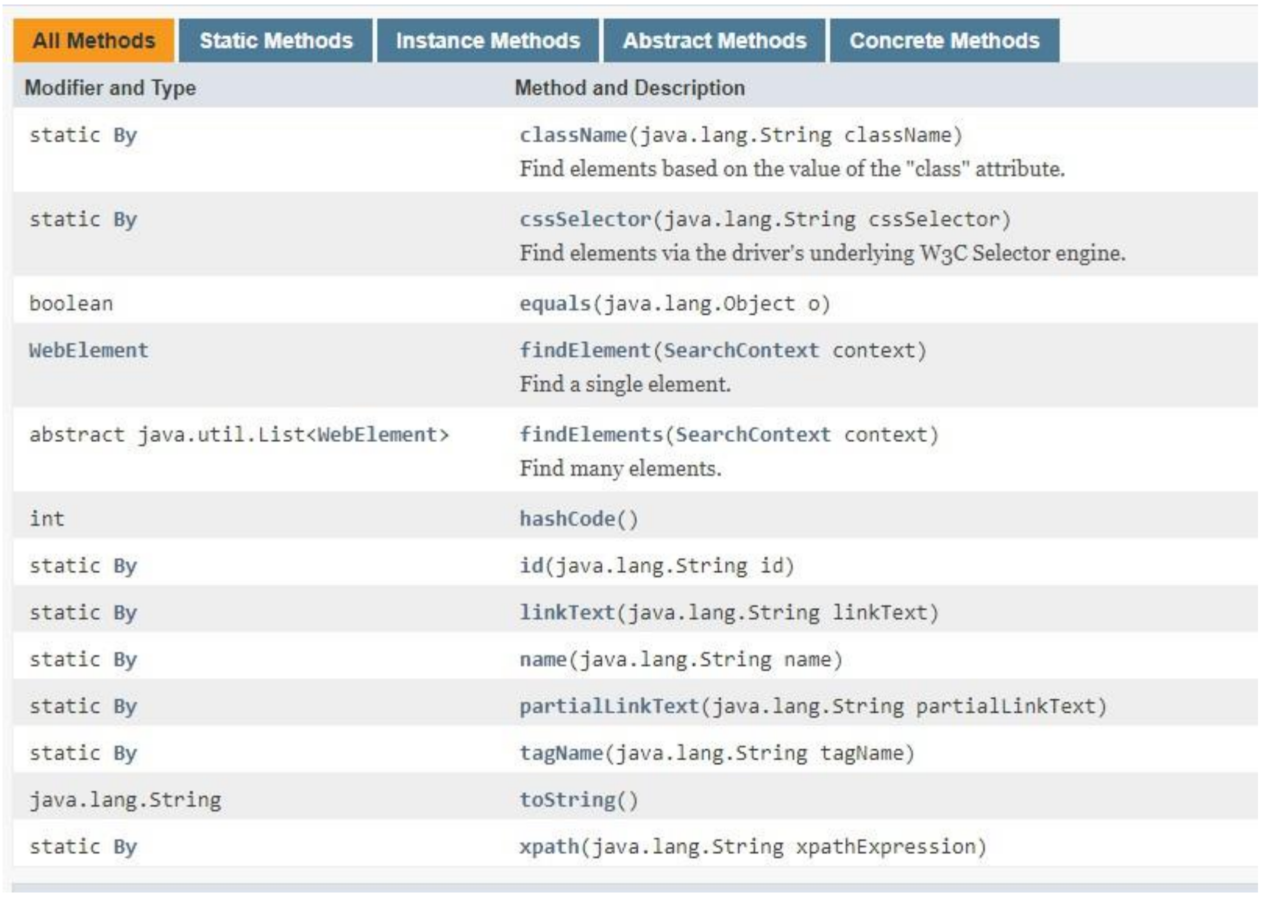
Answer: Abstract classes provide you the flexibility to have certain concrete methods and some other methods that the derived classes should implement. By contrast, if you use interfaces, you would need to implement all the methods in the class that extends the interface. An abstract class is a good choice if you have plans for future expansion – i.e. if a future expansion is likely in the class hierarchy. If you would like to provide support for future expansion when using interfaces, you'll need to extend the interface and create a new one.

For suppose, your derived class A extend a abstract Class, another derived Class B extends same abstract class. Now you get a idea to add a concrete method in the abstract class , and use it in Class A only and do not require in Class B. So Abstract classes provides flexibility to implement the method in required derived Class A and does not force to have it in Class B. This is known as Loosely Coupled

For Suppose, your derived class A implements a Interface, another derived Class B implements same Interface. Now you get a idea to declare method in the Interface, and implement it in Class A only and do not require in Class B. But Interface do not provide such flexibility. It forces all derived classes to implement newly declared method. For example, any changes in Webdriver Interface will affect the classes which is implementing the WebDriver Interface, and they are forced to implement the methods in their Own class. This is known as Tightly Coupled

Real Time Example of Abstract class:

Here By Class has all concrete methods, but one abstract Method (List<WebElement>)



Modifier and Type	Method and Description
static By	className(java.lang.String className) Find elements based on the value of the "class" attribute.
static By	cssSelector(java.lang.String cssSelector) Find elements via the driver's underlying W3C Selector engine.
boolean	equals(java.lang.Object o)
WebElement	findElement(SearchContext context) Find a single element.
abstract java.util.List<WebElement>	findElements(SearchContext context) Find many elements.
int	hashCode()
static By	id(java.lang.String id)
static By	linkText(java.lang.String linkText)
static By	name(java.lang.String name)
static By	partialLinkText(java.lang.String partialLinkText)
static By	tagName(java.lang.String tagName)
java.lang.String	toString()
static By	xpath(java.lang.String xpathExpression)

************* Push yourself, because no one else is going to do it for you.**********************