|  |  |
| --- | --- |
| **Abstract class** | **Interface** |
| 1) Abstract class can **have abstract and non-abstract** methods. | Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also. |
| 2) Abstract class **doesn't support multiple inheritance**. | Interface **supports multiple inheritance**. |
| 3) Abstract class **can have final, non-final, static and non-static variables**. | Interface has **only static and final variables**. |
| 4) Abstract class **can provide the implementation of interface**. | Interface **can't provide the implementation of abstract class**. |
| 5) The **abstract keyword** is used to declare abstract class. | The **interface keyword** is used to declare interface. |
| 6) An **abstract class**can extend another Java class and implement multiple Java interfaces. | An **interface** can extend another Java interface only. |
| 7) An **abstract class**can be extended using keyword ?extends?. | An **interface class**can be implemented using keyword ?implements?. |
| 8) A Java**abstract class**can have class members like private, protected, etc. | Members of a Java interface are public by default. |
| 9)**Example:**  public abstract class Shape{  public abstract void draw();  } | **Example:**  public interface Drawable{  void draw();  } |

Simply, abstract class achieves partial abstraction (0 to 100%) whereas interface achieves fully abstraction (100%).

**How final keyword is used in different ways ?**

*final* keyword is used in different contexts. First of all, *final* is a [non-access modifier](https://www.geeksforgeeks.org/access-and-non-access-modifiers-in-java/) applicable **only to a variable, a method or a class**. Following are different contexts where final is used.

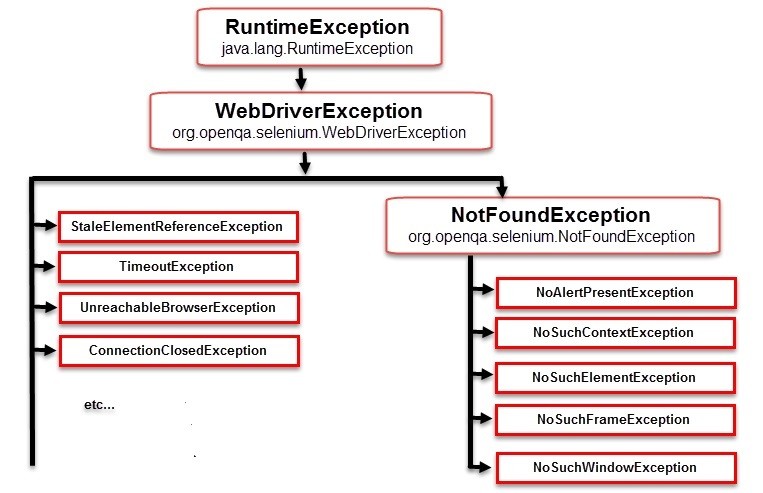
Final Variable >>>>>> To create constant variables

Final Methods >>>>> To Prevent Method overloading

Final Class >>>>>>>> To prevent Inheritance

**What is an Exception?**

“An Exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program’s instructions or in simple words, any issue which makes your test case stop in between the execution.”



When exception occurs, the normal flow of program halts & an exception object is created. The program then tries to find someone that can handle the raised exception. The exception object contains a lot of debugging information such as method hierarchy, line number where the exception occurred, type of exception etc. The process of creating the exception object and handing it over to run-time environment is called “throwing the exception”.

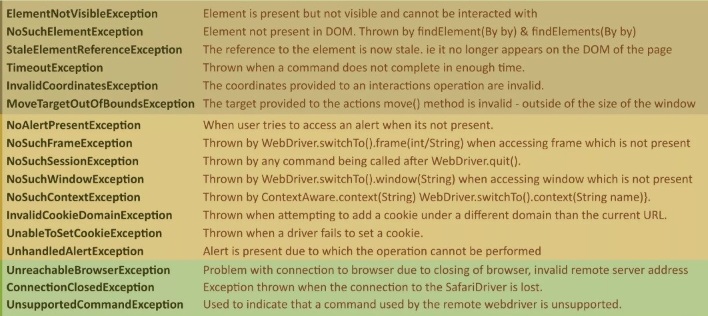
**Common Exceptions in Selenium WebDriver**

There is a complete list of ***Exceptions in Selenium WebDriver*** mentioned in the Selenium Doc which you may or may not encounter in course of your testing. Hence in this article we will focus on some *most common exceptions in Selenium WebDriver*,

1. **ElementNotVisibleException**: Although an element is present in the DOM, it is not visible (cannot be interacted with). E.g. Hidden Elements – defined in HTML using type=”hidden”.
2. **ElementNotSelectableException**: Although an element is present in the DOM, it may be disabled (cannot be clicked/selected).
3. **InvalidSelectorException**: Selector used to find an element does not return a WebElement. Say XPath expression is used which is either syntactically invalid or does not select WebElement.
4. **NoSuchElementException**: WebDriver is unable to identify the elements during run time, i.e. FindBy method can’t find the element.
5. **NoSuchFrameException**: WebDriver is switching to an invalid frame, which is not available.
6. **NoAlertPresentException**: WebDriver is switching to an invalid alert, which is not available.
7. **NoSuchWindowException**: WebDriver is switching to an invalid window, which is not available.
8. **StaleElementReferenceException**: The referenced element is no longer present on the DOM page (reference to an element is now Stale). E.g. The Element belongs to a different frame than the current one OR the user has navigated away to another page.
9. **SessionNotFoundException**: The WebDriver is performing the action immediately after ‘quitting’ the browser.
10. **TimeoutException**: The command did not complete in enough time. E.g. the element didn’t display in the specified time. Encountered when working with waits.
11. **WebDriverException**: The WebDriver is performing the action immediately after ‘closing’ browser.

**What is Exception Handling?**

Exception handling refers to the anticipation, detection, and resolution of exception, i.e. the block of code that processes the exception object.



**How to Handle Exceptions in Selenium WebDriver**

**Try/Catch:** A method catches an exception using a combination of the try and catch keywords. Try is the start of the block and Catch is at the end of try block to handle the exceptions. A try/catch block is placed around the code that might generate an exception. Code within a try/catch block is referred to as protected code, and the syntax for using try/catch looks like the following:

try{

// Some code

}

catch(Exception e){

// Code for Handling the exception

}

Below methods can be used to display Exception information:

* **printStackTrace()**: prints the stack trace , exception name and description.
* **toString()**: returns a text message describing the exception name and description.
* **getMessage()**: displays the description of exception