|  |  |  |
| --- | --- | --- |
| throw | throws | throwable |
| Keyword | keyword | class |
| Used to throw a custom exception manually | Used to declare an exception to be thrown to the caller | Throwable class implements Serializable interface and extends Object class |
| We can throw checked and unchecked exceptions | If the caller of the method doesn’t handle the exception, we will get compile time error | Superclass of Error and Exception classes. Root class of Exception Hierarchy |
| int a=5 int b=10  if(a>b)  {throw new RuntimeException(“e”); | Public void mtd() **throws** IOException{} | Throwable itself is a checked exception  **e.**[**printStackTrace()**](https://www.geeksforgeeks.org/throwable-printstacktrace-method-in-java-with-examples/?ref=rp)Prints the current throwable and its backtrace to the standard error stream.  **e.getMessage ()**  Returns: the detailed message string of current Throwable instance( may also return null) |

|  |  |  |
| --- | --- | --- |
| final | finally | finalize() |
| Modifier | Block | method |
| Used to declare class, methods and variables  final class – cannot be inherited  final method – cannot be overridden  final variable – cannot be reinitialized | Follows the catch block of try and catch block of try and catch.  finally block is executed irrespective of whether the exception is handled or not |  |
| final class demo{}  final void mtd()  final int a=10;  public void m1()  { int a=5;} ======🡺compile time error | try{} catch{} **finally{}** |  |

|  |  |
| --- | --- |
| Constructor | Method |
| Block of code | Collection of statements |
| To initialize an object | To provide functionality or perform operations, return a value |
| Invoked once object is created | Needs to called using object after object creation |
| NO Return type | HAS a return type |
| Name – same as class name | Name – could be any |
| Cannot be inherited by subclasses | Can be inherited by subclasses |

|  |  |
| --- | --- |
| **Static** | **Final** |
| static variable can be reinitialized | final variable cannot be reinitialized |
| Static method cannot be overridden | final method cannot be overridden |
| Static class can be inherited | Final class cannot be inherited |
| Not mandatory to initialize variable while declaring | mandatory to initialize variable while declaring |

|  |  |
| --- | --- |
| **Exceptions** | **Errors** |
| Unexpected events that hinder the flow of execution  Can be handled | Abnormal conditions or serious problems that a reasonable application should not try to catch |
| Types: Checked and Unchecked  Examples: IOException, NullPointerException, SQLException,etc. | Types: Syntax Error, Runtime Error, Logical Error.  Examples: OutOfMemoryError, StackOverflowError ,VirtualMachineError,IOError,AssertionError,AWTError,etc |

|  |  |
| --- | --- |
| **Checked Exceptions** | **Unchecked Exceptions** |
| Occurs at Compile Time | Occurs at Runtime |
| Checked by compiler | Unchecked by compiler |
| JVM requires the exception to be caught or handled | JVM does not require the exception to be handled |
| IOException =>FileNotFoundException  =>InterruptedOfException  =>EOFException  InterruptedException  SQLException  AWTException | ArithmeticException  NullPointerException  ClassCastException  IndexOutOfBoundException => ArrayIndexOutOfBoundsException  =>StringIndexOutOfBoundsException |

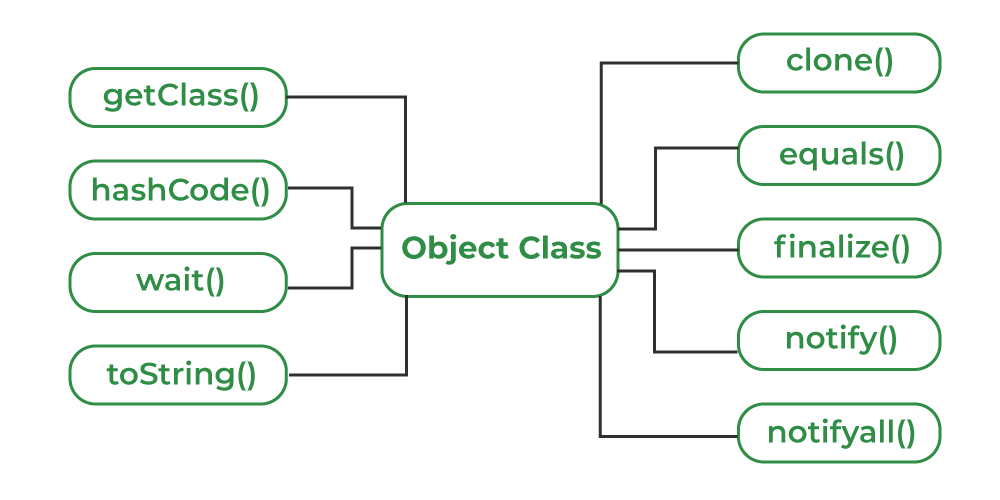
|  |  |  |  |
| --- | --- | --- | --- |
| **Public** | **Protected** | **Default** | **Private** |
| Accessed from anywhere | Accessed within package | accessed outside package using child class | Accessed only within package | Accessed only within class |

|  |  |  |
| --- | --- | --- |
| **Selenium IDE** | **Selenium RC** | **Selenium WebDriver** |
| Record and playback feature | Requires server to start the execution | IDE for script development |handles headless |
| Scripts created in UI Interface | Standalone jars to execute test in browser | Contains API supported by multiple languages |
| Testing Only in firefox | Testing in multiple browsers | Testing in multiple browsers |
| No | No | Elements can be identified |
| No | No | Can handle dropdown |
| No | No | Have Listeners |
| No | No | Handles navigation |

**Exceptions points**

* Exception occurs because of a statement
* If an exception occurs, Throwable Object is created
* All built-in classes of the Throwable type are overridden toString() method
* **Exception Handling** is a mechanism to accommodate continuous flow of execution
* Handle Exception using: try {} catch{} blocks
* If super class method doesn’t declare an exception =>subclass overridden method should not declare checked exception but can declare unchecked exception
* If super class method declares an exception =>subclass overridden method should declare the same exception or no exception

**Object class**



**THREADS:**

Thread is class

**java.lang.illegalthreadstateexception**  : Thrown to indicate that a thread is not in an appropriate state for the requested operation

**Try and catch blocks**

Statements responsible for exceptions should be written within **try{}**

**Catch{}** – to catch the Throwable Object reference thrown by try{}

**Multiple catch{}** :

**try{}**

**catch(IOException ioe)**

**{ }**

**catch(Exception e)**

**{ }**

Catch{} are executed from top to bottom | Parent exception should be at the bottom

**Wrapper classes**

To wrap primitive datatypes into Objects

Why should we use Wrapper classes? Encapsulation , type conversion, autoboxing, unboxing,thread-safe,immutable.

Since J2SE 5.0 wrapping is done implicitly.

For each primitive datatype=>Wrapper class is available .

**Eight** wrapper classes

1. Boolean boolean
2. Character char
3. Integer int
4. Byte byte
5. Long long
6. Short short
7. Float float
8. Double double

To compare two Wrapper Objects: **equals() returns boolean**

ValueOf() **Integer x =Integer.valueOf(9);**

To convert num as String to Integer **: Integer.parseInt(String)**

**Autoboxing :** compiler implicitly converts primitive to wrapper objects

**Public static void demo(Integer x)**

**{ Syso(x)} | main() { int x=20;** demo(x); }

**AutoUnboxing :** compiler implicitly converts wrapper objects to primitive

**Public static void demo(int x) | main(){ Integer x=new Integer(10); demo(x);}**

**What is the difference between Java SE and J2SE?**

Java SE (formerly J2SE) stands for Java standard edition and is normally used for developing desktop applications, forms the core/base API. Java EE (formerly J2EE) stands for Java enterprise edition for applications which run on servers, for example web sites.

**What is Stale Element Reference Exception and how to handle it?**

Unable to interact with an element => Reasons: page refresh | DOM update | location of the web element being changed

findElement() – stores the reference ID of the element. Used whenever required. Uses same reference instead of finding the element again.

**HANDLING:**

**1.Explicit wait : wait.until(ExpectedConditions.presenceOfElementLocated(Web element locator")))**

**2.try{} catch{} :**

**try{**

**WebElement ele = wait.until(ExpectedConditions.presenceOfElementLocated(By.cssSelector("textarea[name='q']")));**

**}**

**catch(StaleElementReferenceException e)**

**{**

**Driver.navigate().refresh(); //refresh**

**WebElement e1 = wait.until(ExpectedConditions.presenceOfElementLocated(By.cssSelector("textarea[name='q']")));   //re-locate**

**1. What are the 3 members of class?**

**Variables**

**Methods**

**Initializers**

**2. What are the 3 types of tokens?**

**Keyword**

**Identifier**

**Literals**

**3.Role of Class Loader in JRE**

Class loaders are responsible for loading Java classes dynamically to the JVM (Java Virtual Machine) during runtime. They're also part of the JRE (Java Runtime Environment). Therefore, the JVM doesn't need to know about the underlying files or file systems in order to run Java programs thanks to class loaders.

4.**Marker Interfaces**

* Serializable
* Cloneable
* Remote

**5. Rules of Method overloading**

* method signature should be the same
* Should have is a relationship
* return type should be same
* access modifier should be the same or higher than parent
* method should be non static

5.**What type of methods are present in Listener class?**

Default methods : enable you to add new functionality to existing interfaces

<https://www.javadoc.io/doc/org.testng/testng/7.0.0/org/testng/ITestListener.html> - Refer

**ABSTRACTION – by Sumanth Sir**

Process of hiding the method implementation and providing the most important features is called abstraction.

* Abstract class : class has at least one abstract method | abstract class doesn’t have to necessarily possess abstract method
* Concrete class : consists of concrete methods | no abstract keyword
* Abstract Method : method prefixed with **keyword abstract** | has **no implementation** |method declaration ends with **;**
* Concrete Method: method has implementation | not prefixed with abstract keyword

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Abstract class can have main method, constructor or anything.. no restrictions\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Cannot create object for abstract class\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*static methods cannot be abstract : abstract methodscannot be overridden\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*base class – no object created but inherited------base class can be abstract in nature\*\*\*\*\*\*\*\*\*\***

**@override : to give the compiler an information but is not mandatory**

**ABSTRACTION USING INTERFACE**

**What can be available within an Interface?**

* **Static final variable**
* **Static method**
* **Public abstract non static method**
* **Default method**
* **By default interface is abstract in nature. Compiler adds abstract by default**
* **Interface used for 100% abstraction(only with non static methods) and multiple inheritance**
* **All non static methods are public and abstract in nature in Interface**
* **Have restrictions : no constructors , initializers, non static methods**
* **Create static final variable – compiler adds static and final before the variable inside Interface. We cannot reinitialize the value**
* **Create a static methods – we can create main method**

**Why interface cannot inherit a class?**

Interface – super most . So cannot inherit class

Class can contain non static methods. So we cannot have non static methods in Interface

**Where have we used abstraction in java?**

Collections (Interfaces contain abstract methods)

**Where have you used Multiple inheritance?**

Listerner implementation is an example of multiple inheritance

**Why should java main method have String[] args?**

to receive any command-line arguments passed to the program when it is executed.

**What happens if the main() method is written without String args[]?**

The program will compile, but not run, because JVM will not recognize the main() method. Remember JVM always looks for the main() method with a string type array as a parameter.

**JAVA METHODS**

Collections.sort(colRefVar) 🡺 for sorting a collection

Collections.reverse(colRefVar) 🡺 for reversing a collection

Collections.shuffle(colRefVar) 🡺shuffles the collection randomly

Arrays.sort(arrayRefVar) 🡺 for sorting an Array

String1.compareTo(String2) 🡺 To compare two strings | a>b =1 a<b =-1 a=b-0

\* Comparable --->class implements comparable type

\* String, Integer, Boolean, etc of Wrapper Class are comparable type

\* To use sort method – class should implement Comparable Interface and override compareTo()

\* To do sorting - elements should be of homogenous and implement comparable interface

\* Convert HashSet to TreeSet =====> to sort a set

\* Convert HashMap into TreeMap==>to sort a map based on key

WHENEVER THERE ARE STRING TYPE OBJECTS COMPARABLE INTERFACE IS IMPLEMENTED AND COMPARETO() IS OVERRIDDEN BY DEFAULT. WE USE COMPARABLE AND COMPARATOR ONLY FOR CUSTOM OBJECTS

**TREESET**

1. Homogenous
2. Maintains Natural Order (Priorty is given to upper case while sorting)
3. For Reversing the Order
4. we can use
5. pass the ref as argument to TreeSet declaration

Comparator<Type> compRef = Comparator.reverseOrder() ;

TreeSet<Type> ts = new TreeSet<>(compRef);

For Interger [ Comparator<Integer> ref = (a,b)-> b – a ; ]

1. Duplicates NOT allowed

ts.add(ele) 🡺 add a particular element

ts.addAll(Collection) 🡺 add a collection to the existing treeset

ts.addFirst(1) 🡺 Throws UnsupportedOperationException (Cannot be used Explicitly)

ts.addLast(60) 🡺 Throws UnsupportedOperationException (Cannot be used Explicitly)

ts.removeFirst(); 🡺 Removes the first element

ts.removeLast(); 🡺 Removes the last element