Marker Interface

* An [interface](https://www.javatpoint.com/interface-in-java) that does not contain methods, fields, and constants is known as **marker interface**.
* In other words, an empty interface is known as **marker interface** or **tag interface.**
* It delivers the run-time type information about an object.
* It is the reason that the [JVM](https://www.javatpoint.com/jvm-java-virtual-machine) and compiler have additional information about an object.
* The **Serializable** and **Cloneable** interfaces are the example of marker interface.
* In short, it indicates a signal or command to the JVM.

Wrapper Class

* The **wrapper class in Java** provides the mechanism to convert primitive into object and object into primitive.
* Since J2SE 5.0, **autoboxing** and **unboxing** feature convert primitives into objects and objects into primitives automatically.
* The automatic conversion of primitive into an object is known as **autoboxing**
* and The automatic conversion of object into an primitive is known as **unboxing**.

Uses of Wrapper Class in java

1. Java is an object-oriented programming language, so we need to deal with objects many times like in Collections, Serialization, Synchronization, etc.
2. **Let us see the different scenarios, where we need to use the wrapper classes.**
   1. **Change the value in Method:** Java supports only call by value. So, if we pass a primitive value, it will not change the original value. But, if we convert the primitive value in an object, it will change the original value.
   2. **Serialization:** We need to convert the objects into streams to perform the serialization. If we have a primitive value, we can convert it in objects through the wrapper classes.
   3. **Synchronization:** Java synchronization works with objects in Multithreading.
   4. **java.util package:** The java.util package provides the utility classes to deal with objects.
   5. **Collection Framework:** Java collection framework works with objects only. All classes of the collection framework (ArrayList, LinkedList, Vector, HashSet, LinkedHashSet, TreeSet, PriorityQueue, ArrayDeque, etc.) deal with objects only.
3. The eight classes of the java.lang package are known as wrapper classes in Java. The list of eight wrapper classes are given below:

|  |  |
| --- | --- |
| **Primitive Type** | **Wrapper class** |
| Boolean | [Boolean](https://www.javatpoint.com/java-boolean) |
| Char | [Character](https://www.javatpoint.com/post/java-character) |
| Byte | [Byte](https://www.javatpoint.com/java-byte) |
| Short | [Short](https://www.javatpoint.com/java-short) |
| Int | [Integer](https://www.javatpoint.com/java-integer) |
| Long | [Long](https://www.javatpoint.com/java-long) |
| Float | [Float](https://www.javatpoint.com/java-float) |
| Double | [Double](https://www.javatpoint.com/java-double) |

1. **AutoBoxing :-**
   1. The automatic conversion of primitive data type into its corresponding wrapper class is known as **autoboxing**.
   2. for example, byte to Byte, char to Character, int to Integer, long to Long, float to Float, boolean to Boolean, double to Double, and short to Short.
   3. Since Java 5, we do not need to use the valueOf() method of wrapper classes to convert the primitive into objects.
2. **UnBoxing :-**
   1. The automatic conversion of wrapper type into its corresponding primitive type is known as unboxing.
   2. It is the reverse process of autoboxing.
   3. Since Java 5, we do not need to use the intValue() method of wrapper classes to convert the wrapper type into primitives.

**Recurssion**

* Recursion in java is a process in which a method calls itself continuously.
* A method in java that calls itself is called recursive method.

**Syntax:**

1. returntype methodname(){
2. //code to be executed
3. methodname();//calling same method
4. }

Custom Exception

Exception Handeling

* Exception Handeling in java is one of the powerful mechanism to Handle the runtime errors so that normal flow of execution of the program can not be disturbed and it maintained.
* If you want to execute your program as a normal flow in which no chance to disturb your program’s flow of execution then we can use exception handeling.

What is Exception?

* **Any Unexpected or unwanted event that disturbs the normal flow is known as Exception.**
* It is an **any Unexpected or Unwanted Event**, which occurs during the execution of a program i.e. at run time, that disturb the normal flow of the program is called as Exception.
* It is an Object which is thrown at runtime.
* **What is normal flow of the program:** the program which can execute line by line top to bottom.

What is Exception Handeling?

* Exception Handeling is a powerful mechanism that handle runtime error.
* Such as ClassNotFoundException, IOException ( Input Output Exception) etc.
* It will provide an alternate way to execute when the exception will occurs in the program.
* Example:
  + If we want to print no. from 1-10 then

statement 1;

statement 2;

statement 3;

statement 4;

statement 5;//exception occurs

statement 6;

statement 7;

statement 8;

statement 9;

statement 10;

* + if the exception occurs at statement 5 then after 5th statements other statements are not execute i.e. statement 6-10.
  + And the normal flow of the program gets disturbed.
  + However when we perform exception handeling, the rest of the code will executed.

Advantage Of Exception Handeling

* It main Advantage, It can not disturb normal flow of the program your software not get crashed execute normally.
* It maintain flow of the program even if the Exception will occur in the program it will execute normally.
* Exception Handling not removes the Exception or repair the Exception that occurs in the program, but it will find an **alternative way to handle this exception**.
* And it will execute that alternate way instead of Exception and maintain the flow of the program.

Hierarchy of java Exception Classes

* Super Class of All the classes in java is **Object Class**.
* And in exception Handeling Root Class is **Throwable Class.**
* and the there are two child of the throwable class which is Exception and Error
* In Most of the cases, Exception are occurred by our **Program.**
* Errors are occurred because of lack of System Resources; not by our program and thus, programmer cannot do anything that’s why it could not be handled.
  + Example our System Resources Should not Enough to execute the program like our system RAM should not enough, or not enough memory, or our system processor is slow etc.

Exception Class:

* In Most of the cases, Exception are occurred by our **Program.**
* We can say it will occur due to mistake of program will coding.
* It is Recoverable or we can handle it.
* And Programmer can handled it, on its level.
* It’s are of two types
  + **Checked or CompileTime Exception**
  + **UnChecked or RunTime Exception**

**Errors:**

* Errors are occurred because of **lack of System Resources**; not by our program and thus, programmer cannot do anything that’s why it could not be handled.
  + Example our System Resources Should not Enough to execute the program like our system RAM should not enough, or not enough memory, or our system processor is slow etc.
* Error Occur During **RunTime.**

**Difference Between Exception & Error**

| Exception | Error |
| --- | --- |
| Exception occurs because of our Programs. | 1. Error occurs because of Lack of System Resources. |
| 1. Exception are recoverable i.e. programmer can handle them using try-catch block. | Error are not recoverable i.e. programmer can not handle them to their level. |
| Exception are of two types:   * Compile Time Exception or Checked Exceptions * Runtime Exceptions or Unchecked Exceptions | 1. Error are only one type:  * Runtime Exceptions or Unchecked Exceptions |

( Super Class or Parent Class of the all the Classes in Java )

**Object**

**Throwable**

**Errors (Runtime)**

**(not handled)**

(Occur Lack of System Resources)

**Exception (handled)**

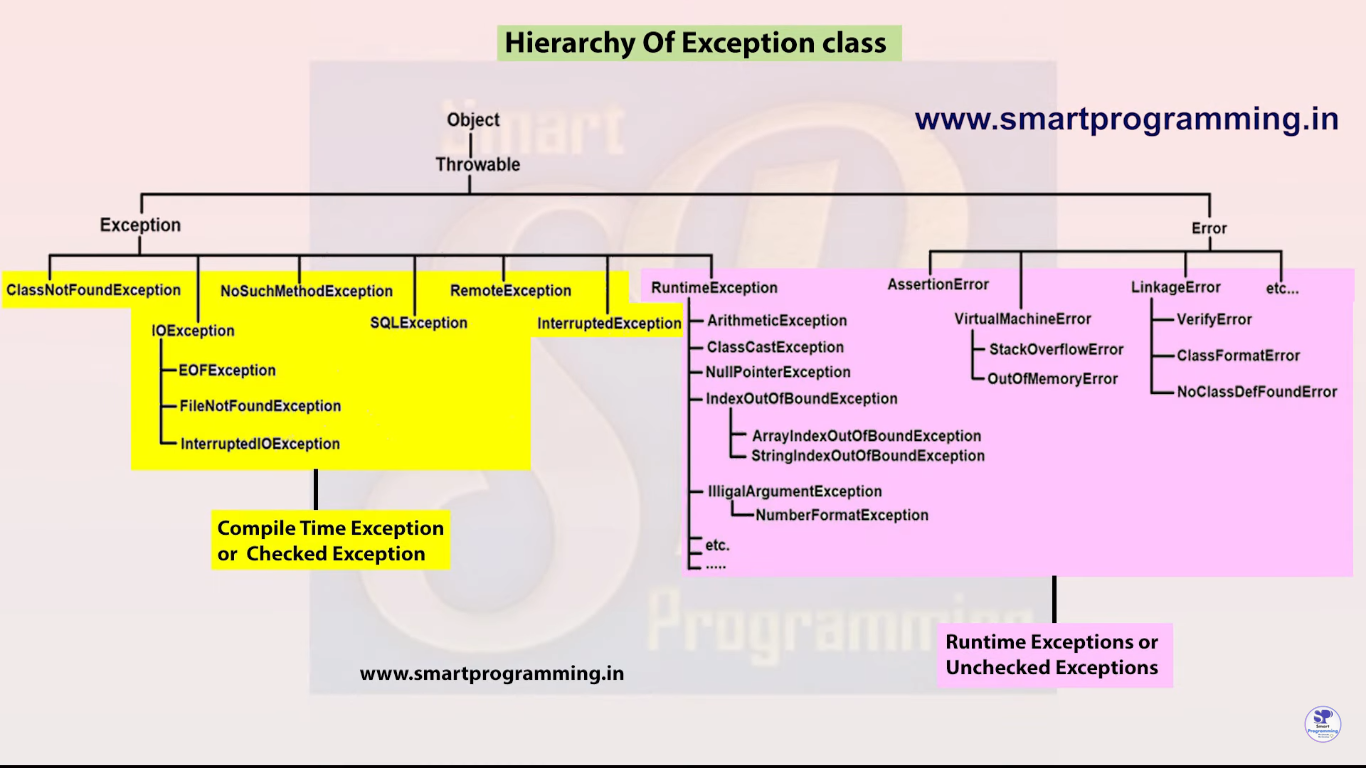
(Occur Programs)

* VirtualMachineError
  + StackOverFlowError
  + OutOfMemoryError
* LinkageError
  + VerifyError
  + ClassFormatError
  + NoClassDefFoundError
* AssertionError
* etc

**Checked or CompileTime Exception**

**UnChecked or RunTime Exception**

* ArithmeticException
* ClassCastException
* NullPointerException
* IndexOutOfBoundException
  + ArrayIndexOutOfBoundException
  + StringIndexOutOfBoundException
* IllegalArgumentException
  + NumberFormatException
* etc
* ClassNotFoundException
* NoSuchMethodException
* IOException(input Output Exception)
  + EOFException (End of File Exception)
  + FileNotFoundException
  + InterruptedIOException (interrupted input output Exception)
* SQLException
* RemoteException
* InterruptedException
* etc



**Difference Between Checked and Unchecked Exception**

**Note:**

* All Exceptions occur at runtime only.
* While compile time no exception occurs.

**Checked or CompileTime Exception:**

* It is Exception, that compiler have ability to check.
* That kind of exceptions that check by compiler while compile time is called Checked or CompileTime Exception.
* If we cannot handle that compile time exception or we cannot report that exception then compiler cannot compile that program.
* In **Intellij Idea** it has by default compile who compile when we write code.

**Unchecked or RunTime Exception:**

* It is Exception, that compiler cannot check
* That kind of Exception that cannot be check by compiler while compile time is known as Unchecked or Runtime Exception.
* Compiler ignore this exception, compiler haven’t ability to check those exceptions

|  |  |
| --- | --- |
| Checked / Compile Time Exception | Unchecked / Runtime Exception |
| 1. Checked Exception are the exceptions that are checked and handled at compile time. | 1. Unchecked Exception are the exceptions that are not checked at compiled time. |
| 1. The program gives a compilation error if a method throws a checked exception. | 1. The program compiles fine because the compiler is not able to check the exception. |
| 1. If some code within a method throws a checked exception, then the method must either handle the exception or it must specify the exception using throws keyword. | 1. A method is not forced by compiler to declare the unchecked exceptions thrown by its implementation. Generally, such methods almost always do not declare them, as well. |
| 1. A checked exceptions occurs when the chances of failure are too high. | 1. Unchecked exception occurs mostly due to programming mistakes. |
| 1. They are direct subclass of Exception class but do not inherit form RuntimeException.   ( The class that directly inherit throwable class except RuntimeException and Error is known as Checked or Compile time Exception). | 1. They are direct subclass of RuntimeException class.   ( The class that inherit the RuntimeException are known as Unchecked or Runtime Exception ). |

**Try-catch block**

* Java **try** block is used to enclose the code that might throw an exception. It must be used within the method.
* If an exception occurs at the particular statement in the try block, the rest of the block code will not execute. So, it is recommended not to keep the code in try block that will not throw an exception.

**Note:**

* Whenever there is exception, the method in which exception occurs will create an object and that object will store three things;

1. Exception Name ( class name )
2. Description ( which type of exception is )
3. Stack trace ( proper description in which line , which method is found an exception )

* and that created object will pass to the JVM.
* And JVM checks that exception is handling or not.
* If we cannot handle exception then JVM gives controller to the Default Exception handler and gives that object to the Default Exception Handler.
* And default exception handler prints that exception.
* Before going toward the default exception handler, JVM abnormally terminates the program first.
* If we not handled the exception then Default Exception Handler prints that Exception.
* If we want to handle that exception then it is customize the exception..
* We can handle the exception with the help of try-catch method.
* We can handle exception with the help of 5 keywords:

**1. try**

**2. catch**

**3. finally**

**4. throw**

**5. throws**

**Syntax of try-catch:**

try

{

// risky code

}

catch(ExceptionClassName ref.var.name)

{

//handling code

}

* When we use try then it is compulsory to give catch there…
* try-catch always comes together everywhere
* It is recommended that write as much as less line of code inside the try block.

**String**

* String is **non-primitive** data types because it reference a memory location where data is stored in the heap memory (or String Constant pool) i.e, it references to a memory where an object is actually placed. And thus the variable of a non-primitive data type is also called reference data types or object reference variable.
* This object reference variable lives on the stack memory and the object to which it points always lives on the heap memory.
* The stack holds a pointer to the object on the heap.
* Thus all non-primitive data types are simply called objects which are created by instantiating a class.
* String **size is not fixed** because its non-primitive data type
* String is the sequence of characters or say, String is an array of character.

Example: char[] c = {‘v’, ‘a’, ‘i’, ‘b’, ‘h’, ‘a’, ‘v’};

String s = new String(c);

It is same as String s = ‘vaibhav’;

**To represent sequence character java has one interface which is charSequence**

* String is a **class.**

Example:

Public final class String{

// code

};

Imp Interview Question

**\*\*\*Why String class is final?\*\*\*\* & Explain its methods?**

String class is final because, we didn’t have authority to inherit String class and then we can’t override the methods of the String Class.

* And Object is a parent class of the all the classes in java, hence the String class also inherit the object class.

All class in java by default extends object class

Ex. public final class String extends object{

// code

}

* And String class impliments charSequence because String is an array of character.

Ex.

public final class String extends object impliments charSequence, Serializable, Comparable{

// code

}

**Serializable :**

Seralizable is a interface is used when we need to store a copy of the object and send them to another process which runs on the same system or over the network.

**Comparable :**

Comparable interface is used to order the objects of the class. It has only one method i.e compareTo().

* String is an **object** which is **immutable.**

We can create object of the String, because it’s a class and we can create object of any class (if it not static).

We can create String class object by :-

String s = new String();

**It will create an Immutable object. \*\*\*\***

Normally, we create variable by using String:

String s = ‘name’; // it will also create an object…

Imp Interview Question

**What is difference between two types? \*\*\*\*\***

1. **String s = new String(‘name’);**
2. **String s = ‘name’; //** we use this mostly….

* There are total three main classes to create String:

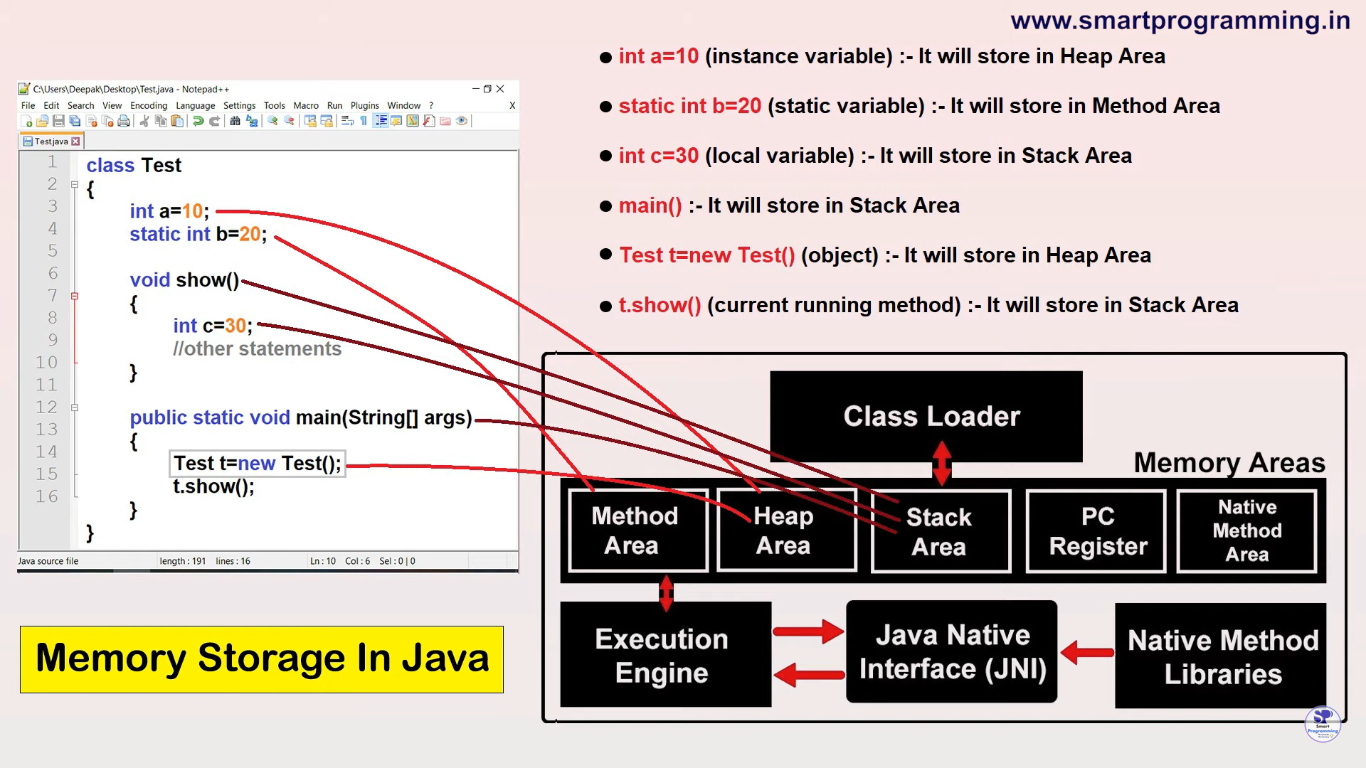
1. String class
2. StringBuffer class
3. StringBuilder class

* Actually, total 4 ways to create string, those three are correct but we can create string by directly creating object of String Class

Ex.

1. String s = new String(‘name’);
2. String s = ‘name’;
3. StringBuffer sb = new StringBuffer(‘name’);
4. StringBuilder sb = new StringBuilder(‘name’);

**String Constant Pool (SCP) or String Literal Pool concept:**

* String Constant Pool (or String Literal Pool ) is an area in heap memory where java stores String literal values.
* Till Java version 1.6 , string constant pool is in method area….

In this method area, String Constant Pool is constant.

* After Java version 1.7, String Constant Pool (SCP) is Shifted in the Heap Memory or Heap Area…

In heap Area, SCP will be Increases or Decreases

* It is Present in Heap Area:

**Heap Area**

**String Constant Pool or String Literal Pool**

Imp Interview Question

**What is difference between two types? \*\*\*\*\***

1. **String s = new String(‘name’);**
2. **String s = ‘name’; //** we use this mostly….
3. If we create object of string like

String str1= new String(‘name’);

JVM create 2 Objects:

1. new keyword will create a memory in heap area but outside the SCP.
2. And we wring string literal their also so it will create again memory inside heap memory but inside SCP…

It created this object for future use…

1. If we create object of string like:

String u = ‘hello’; // String Literal is 🡪 ‘hello’

Then JVM create only 1 Object: ( that’s why we use this)

Because of String Literal only directly inside SCP in Heap memory…

String Str1 = new String(‘name’)

**SCP (SLP)**

**Heap Memory**

JVM Internally manage

**Str1**

**name**

**name**

JVM Create this for future use

**hello**

This pointing if literally create this same literal object

String Str2 = “hello”

**Str2**

**name**

String Str4 = “hello”

**Str4**

String Str3 = new String(‘name’)

**Str3**

Note: The String objects present in SCP or SLP are not Applicable for Garbage Collection because a reference variable internally is maintained by JVM.

String Immutability

**Immutability:**

* Immutable means we can not change the original value.
* Immutability concept is used for “String Objects” i.e String objects are immutable.
* It means once String object is created; its data or state can’t be changed but a new string object is created.

Example:

1. If we create String object by using new keyword, then it creates 2 object in memory. 1st will create in heap memory but outside the SCP(String Constant Pool) and 2nd will created in SCP but that 2nd object of String not target by that created reference but JVM implicitly Refer a variable to the reference object hence, it can not goes to the Garbage Collector.

**Immutability Concept of java**

S

Vaibhav

**SCP**

**Heap Memory**

1. String s = new String(“Vaibhav”);

1

Vaibhav (internally refer By JVM)

**2) S.concat(“ java”)**

**System.out.println(S)** // Vaibhav

java

Vaibhav java

Python

**3) s = s.concat(“ python”)**

2

Vaibhav python

**System.out.println(s)** // Vaibhav python

**Why String are Immutable?**

* If we create String object by using string literal, then It will create object in SCP
* If there are many object created with the same value like:

String a = “Chandrapur”

String b = “Chandrapur”

String c = “Chandrapur”

* Then JVM create only one object in SCP and other object are point towards that single object because all value are same.
* If we can change the String c:

c = “Pune”

* If the object value is changed, then the all variable value also gets change.
* **To cure this problem Java String are Immutable.**
* In this case, JVM creates another object in SCP name as “Pune” and then variable c is pointing towards “Pune” and the other variables like variable a and b is point towards the “Chandrapur”.