**Java:**

**Final🡺** final is modifier applicable for classes,methods and variables,if a class is declared as final we can’t extend that class we caanot create child class for that class

2if method declared as final we cannot override it

3.if a variable declared as final then it will become constant and we can’t perform re-assignment for the variable.

**Finally**=>finally is a block always associated with try{}catch(error){}finally{}

Try🡺maintains risky code

Catch🡺maintain error handling code which raised in try

Finally🡺maintains cleanup code like resource de allocation

**Finalize🡺it is a method present in object class and it is always invoked by garbage collector just before destroying an object to perform cleanup activities**

**Note:**

Finally meant for cleanup activities related to try block(if is there any resources open in try block those closed in finally is an error raises) where as finalize meant for cleanup activities related to object

**Difference b/n string and stringBuffer and stringBuilder:**

**String object are immutable**

**stringBuffer is mutable**

**example:**

String a=new String(‘durga’); stringBuffer a=new stringBuffer (‘durga’);

Line 2//s.concat(‘software’); s.concat(‘software’);

sopn(s) sopn(s)

output:durga durga software

once we create a string object we cannot perform any operation on sring object once we change means with new change a new object is created in line 2 we are changing a new object will return in 2nd line but that value is not storing in any variable and the main object will remains same

**stringBuilder:**

🡺string builder is same as string buffer but in string buffer every method itself synchronized because of this advantage is at a time only one thread is allowed to operate on object but disadvantage is until completion of first thread second thread has to wait and is until completion of second thread third thread has to wait etc because of this performance is low and hear thread safety will be present and it is introduced in 1.0v

🡺but in stringbuilder no method is synchronized so at a time any number of threads can operate symaltaneously because of non-synchronization method and no hear thread safety will be present and performance wise good compared to string buffer and it is introduced in 1.5v

**== vs .equals**

== 🡺it is for reference and address comparision

.equals()🡺it is method present in object class and also it is meant for content comparision

String s1 =new String(“durga”);

String s2= new String(“durga”);

Sopln(s1==s2);//false

Sopln(s1.equals(s2))//true

**Modifiers available in java(14):**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **modifier** | **Outer**  **class** | **Inner class** | **method** | **Vari**  **ables** | **Outer**  **interface** | **Inner**  **interface** | **Outer**  **enum** | **Inner**  **enum** | **blocks** | **constructor** |
| **public** | yes | **yes** | **yes** | **yes** | **yes** | **yes** | **yes** | **yes** |  | **yes** |
| **private** |  | **yes** | **yes** | **yes** |  | **yes** |  | **yes** |  | **yes** |
| **protected** |  | **yes** | **yes** | **yes** |  | **yes** |  | **yes** |  | **yes** |
| **default** | **yes** | **yes** | **yes** | **yes** | **yes** | **yes** | **yes** | **yes** |  | **yes** |
| **final** | **yes** | **yes** | **yes** | **yes** |  |  |  |  |  |  |
| **abstract** | **yes** | **yes** | **yes** |  | **yes** | **yes** |  |  |  |  |
| **static** |  | **yes** | **yes** | **yes** |  | **yes** |  | **yes** | **yes** |  |
| **synchronize** |  |  | **yes** |  |  |  |  |  | **yes** |  |
| **native** |  |  | **yes** |  |  |  |  |  |  |  |
| **strictfp** | **yes** | **yes** | **yes** |  | **yes** | **yes** | **yes** | **yes** |  |  |
| **transient** |  |  |  | **yes** |  |  |  |  |  |  |
| **volatile** |  |  |  | **yes** |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |

**Illegal combinations:**

**Method level**

if a method is declared as public it cannot declared with protected or private like public protect method name(){}

🡺for abstract🡺final, static, synchronized, native, private, strictfp

**Variable level**

Public🡺private,protected

Final🡺volatile

**Class level:**

Final🡺abstract

Public🡺private,protected

**Interfaces Vs Concrete Vs Abstract:**

|  |  |  |
| --- | --- | --- |
| **Interfaces** | **Abstract** | **Class** |
| If we don’t know any thing about implementation just we have requirement specification (100% abstraction)then we should go for interfaces | If we taking about implementation but not completely(partial implementation)we go for abstract | If we talking about complete implantation and ready to provide services then we should go for classes |
| **Example: Servlet** | **Generic Servlet And http Servlet** | **Own servlet** |
| Every method present in interface is by default public and abstract whether we are declaring or not | Every method present in abstract need not be public and abstract .in addition we can take concrete method also |  |
| We can’t declare interface methods with the following modifiers  Public🡺private, protected  abstract🡺final, static, synchronized, native, , strictfp | There is no restriction on abstract class method modifiers |  |
| Using interface keyword we can create interface | Using abstract keyword we can create |  |
|  |  |  |
| Every variable present in interface is always public ,static and final whether we are declaring or not | Every variables present in abstract class is need not be public ,static,final. |  |
| We cannot declare interface variable with the following modifiers private, protected, transient, volatile | There is no restriction on abstract class variables modifiers |  |
| For interface variables compulsory initialization is manadatory at the time of declaration otherwise we will get compile time | For abstract class variable it is not required to perform initialization at the time of declaration |  |
| Inside interface we cannot declare instance and static blocks otherwise we will get compile time error | Inside abstract we can declare instance and static blocks |  |
| Inside interface we cannot declare Constructor | Inside abstract we can declare constructor |  |
| Object creation not possible | Object creation not possible |  |

**Access Specifiers Vs Access Modifiers**

In old languages C++ public,protect,private,default considered as access specifiers.remaining ststic etc are considered as Access Modifiers

🡺in java there is no concept of access specifiers here all considered as access modifiers

**System.out.println():**

**System present in java.lang package**

**Class System{  
static PrintSteam out;**

**}**

**System🡺System is a class present in java.lang package**

**Out🡺**Out is static variable system class of type PrintStream as it static variable we can access it with classname

**Prinln()🡺is a method present in** PrintStream class

**Difference b/n overloading and overriding:**

|  |  |
| --- | --- |
| **overloading** | **override** |
| Two methods are said to be overloaded if and only if both methods having same name but different argument types | **What ever** parent class has by default available to the child class through inheritance some times child may not satisfy with parent method implementation .then child allow to redefine that method based on its requirement.this process is called overidding |
|  | The parent class method is overridden method, child class method is called overriding method |
| Method signature different | Method signature same |
| Methods may have different return type | Methods may have same return type  Or it may have co-varient return type |
| For private,static,final methods overloading is applicable | For private,static,final methods overloading is not applicable |
| There is no restriction for access modifiers in overloading | There is restriction |
| It is also known as compile time polymorphism or static polymorphism or early binding | It is also known as runtime polymorphism or dynamic polymorphism or late binding |

**Exception and error:**

Throwable class acts as root for java exception hierarchy

Throwable classes contain two child class they are is exception and error

|  |  |
| --- | --- |
| **Exception** | **Error** |
| Most of the cases exceptions are caused by our program only and this are recoverable | Most of the time errors are not caused by program these are due to lack of system resources(like memory ) and errors are non recovarable |
| **Example:**  For example if our program requirement is to read data from a remote file locating at some particular location if that file is not available in that location then we will get an error  And if file not found exception occurs then we can provide a local file and rest of the program be continue normally  Complete this process is done using try,catch  Try{ }  Catch{} | For example outofmemory error occurs being a programmer we cann’t do anything and program will be terminated automatically |

**Checked exception and unchecked exception:**

|  |  |
| --- | --- |
| **checked** | **unchecked** |
| The exceptions which are checked by compiler for smooth execution of the program at runtime are called checked exception | The exceptions which are checked by compiler called **unchecked Exception** |
| **Example:FileNotFound Exception** | **Example: Arithmetic,NullPointer** |
| In the case of checked exceptions compiler will check whether programmers are handling exception or not if the programmer not handling then we will get compile time error | In the case of unchecked exceptions compiler will check whether programmers are handling exception or not |
| Runtime exception and its child classes, error and its child classes are unchecked except this remaining( InteruptedException ,IO exception (filenotfound,eof) are checked exceptions |  |

**Note:** whether exception checked or unchecked compulsory it will occur only at run time. There is no chance of occurring any exception at compile time

**Partially checked Vs Full checked(these are types of checked exception):**

A checked exception is said to be fully checked exception if and only if all its child classes are also checked.

Example:

IOException,InteruptedException

A checked exception is said to be **Partially** checked exception if and only if all its child classes are also Unchecked.

Note:

The only partially checked exceptions in java are Exception,throwable9

**Example: Exception**

**Fibonacci:**

**a=0**

**b=1**

**for i in range(10):**

**if(i<2):**

**print(i)**

**else:**

**c=a+b**

**print(c)**

**a=b**

**b=c**

**Input reading in java:**

In Java, there are four different ways for reading input from the user in the command line environment(console).

**1.Using Buffered Reader Class**

This is the Java classical method to take input, Introduced in JDK1.0. This method is used by wrapping the System.in (standard input stream) in an InputStreamReader which is wrapped in a BufferedReader, we can read input from the user in the command line.

* The input is buffered for efficient reading.
* The wrapping code is hard to remember.

**Implementation:**

* Java

|  |
| --- |
| // Java program to demonstrate BufferedReader  **import** java.io.BufferedReader;  **import** java.io.IOException;  **import** java.io.InputStreamReader;  **public** **class** Test {  **public** **static** **void** main(String[] args)  **throws** IOException      {          // Enter data using BufferReader          BufferedReader reader = **new** BufferedReader(  **new** InputStreamReader(System.in));            // Reading data using readLine          String name = reader.readLine();            // Printing the read line          System.out.println(name);      }  } |

**Input:**

Geek

**Output:**

Auxiliary Space : O(1)

Geek

**Note:**

To read other types, we use functions like Integer.parseInt(), Double.parseDouble(). To read multiple values, we use split().

**2. Using Scanner Class**

This is probably the most preferred method to take input. The main purpose of the Scanner class is to parse primitive types and strings using regular expressions, however, it is also can be used to read input from the user in the command line.

* Convenient methods for parsing primitives (nextInt(), nextFloat(), …) from the tokenized input.
* Regular expressions can be used to find tokens.
* The reading methods are not synchronized

To see more differences, please see [this](https://www.geeksforgeeks.org/difference-between-scanner-and-bufferreader-class-in-java/) article.

* Java

|  |
| --- |
| // Java program to demonstrate working of Scanner in Java  **import** java.util.Scanner;    **class** GetInputFromUser {  **public** **static** **void** main(String args[])      {          // Using Scanner for Getting Input from User          Scanner in = **new** Scanner(System.in);            String s = in.nextLine();          System.out.println("You entered string " + s);    **int** a = in.nextInt();          System.out.println("You entered integer " + a);    **float** b = in.nextFloat();          System.out.println("You entered float " + b);      }  } |

**Input:**

GeeksforGeeks

12

3.4

**Output:**

You entered string GeeksforGeeks

You entered integer 12

You entered float 3.4

**3. Using Console Class**

It has been becoming a preferred way for reading user’s input from the command line. In addition, it can be used for reading password-like input without echoing the characters entered by the user; the format string syntax can also be used (like System.out.printf()).

**Advantages:**

* Reading password without echoing the entered characters.
* Reading methods are synchronized.
* Format string syntax can be used.
* Does not work in non-interactive environment (such as in an IDE).

* Java

|  |
| --- |
| // Java program to demonstrate working of System.console()  // Note that this program does not work on IDEs as  // System.console() may require console  **public** **class** Sample {  **public** **static** **void** main(String[] args)      {          // Using Console to input data from user          String name = System.console().readLine();            System.out.println("You entered string " + name);      }  } |

**Input:**

GeeksforGeeks

**Output:**

Most used user input for competitive coding. The command-line arguments are stored in the String format. The parseInt method of the Integer class converts string argument into Integer. Similarly, for float and others during execution. The usage of args[] comes into existence in this input form. The passing of information takes place during the program run. The command line is

given to args[]. These programs have to be run on cmd.

You entered string GeeksforGeeks

**4.**[**Using Command line argument**](https://www.geeksforgeeks.org/command-line-arguments-in-java/)

**Code:** 

* Java

|  |
| --- |
| // Program to check for command line arguments  **class** Hello {  **public** **static** **void** main(String[] args)      {          // check if length of args array is          // greater than 0  **if** (args.length > 0) {              System.out.println(                  "The command line arguments are:");                // iterating the args array and printing              // the command line arguments  **for** (String val : args)                  System.out.println(val);          }  **else**              System.out.println("No command line "                                 + "arguments found.");      }  } |

**Command Line Arguments:**

javac GFG1.java

java Main Hello World

**Output:**

The command line arguments are:

Hello

World

Please refer [this](https://www.geeksforgeeks.org/fast-io-in-java-in-competitive-programming/) for more faster ways of reading input.

# Difference Between Scanner and BufferedReader Class in Java

 [Read](javascript:void(0))

 [Discuss](javascript:void(0))

 [Courses](javascript:void(0))

 [Practice](javascript:void(0))

In Java, Scanner and BufferedReader class are sources that serve as ways of reading inputs. [Scanner class](https://www.geeksforgeeks.org/scanner-class-in-java/) is a simple text scanner that can parse primitive types and strings. It internally uses regular expressions to read different types while on the other hand [BufferedReader class](https://www.geeksforgeeks.org/java-io-bufferedreader-class-java/) reads text from a character-input stream, buffering characters so as to provide for the efficient reading of the sequence of characters

The eccentric difference lies in reading different ways of taking input via the next() method that is justified in the below programs over a similar input set.

**Example 1:**

* Java

|  |
| --- |
| // Java Program to Illustrate Scanner Class    // Importing Scanner class from  // java.util package  **import** java.util.Scanner;    // Main class  **class** GFG {        // Main driver method  **public** **static** **void** main(String args[])      {            // Creating object of Scanner class to          // read input from keyboard          Scanner scn = **new** Scanner(System.in);            System.out.println("Enter an integer & a String");            // Using nextInt() to parse integer values  **int** a = scn.nextInt();            // Using nextLine() to parse string values          String b = scn.nextLine();            // Display name and age entered above          System.out.printf("You have entered:- " + a + " "                            + "and name as " + b);      }  } |

**Output:**

Enter an integer & a String

10 John

You have entered:- 10 and name as John

Let us try the same using Buffer class and the same Input below as follows:

**Example 2:**

* Java

|  |
| --- |
| // Java Program to Illustrate BufferedReader Class    // Importing required class  **import** java.io.\*;    // Main class  **class** GFG {        // Main driver method  **public** **static** **void** main(String args[])  **throws** IOException      {            // Creating object of class inside main() method          BufferedReader br = **new** BufferedReader(  **new** InputStreamReader(System.in));            System.out.println("Enter an integer");            // Taking integer input  **int** a = Integer.parseInt(br.readLine());            System.out.println("Enter a String");            String b = br.readLine();            // Printing input entities above          System.out.printf("You have entered:- " + a                            + " and name as " + b);      }  } |

**Output:**

**Outputs explanation:**

In Scanner class if we call n*extLine() method* after any one of the seven *nextXXX() method* then the nextLine() does not read values from console and cursor will not come into console it will skip that step. The nextXXX() methods are nextInt(), nextFloat(), nextByte(), nextShort(), nextDouble(), nextLong(), next().

In BufferReader class there is no such type of problem. This problem occurs only for the Scanner class, due to nextXXX() methods ignoring newline character and nextLine() only reads till the first newline character. If we use one more call of nextLine() method between nextXXX() and nextLine(), then this problem will not occur because nextLine() will consume the newline character.

***Tip:****See*[*this*](https://ide.geeksforgeeks.org/CErAhD)*for the corrected program. This problem is same as*[*scanf() followed by gets() in C/C++*](https://www.geeksforgeeks.org/problem-with-scanf-when-there-is-fgetsgetsscanf-after-it/)*. This problem can also be solved by using next() instead of nextLine() for taking input of strings as shown*[*here*](https://ide.geeksforgeeks.org/QBYyY67IDj)*.*

**Following are the Major Differences between Scanner and BufferedReader Class in Java**

* BufferedReader is synchronous while Scanner is not. BufferedReader should be used if we are working with multiple threads.
* BufferedReader has a significantly larger buffer memory than Scanner.
* The Scanner has a little buffer (1KB char buffer) as opposed to the BufferedReader (8KB byte buffer), but it’s more than enough.
* BufferedReader is a bit faster as compared to Scanner because the Scanner does the parsing of input data and BufferedReader simply reads a sequence of characters.