**MODULE-1**

**PART-1**

**JAVA EVOLUTION & ENVIRONMENT**

## WHAT IS JAVA?

* Java is an object-oriented programming language originally developed by Sun Microsystems and released in 1995.
* Java was originally developed by **James Gosling** at **Sun Microsystems** (which has since merge into Oracle Corporation).
* Java programs are platform independent which means they can be run on any operating system with any type of processor as long as the Java interpreter is available on that system.
* Java code that runs on one platform does not need to be recompiled to run on another platform, it’s called “**write once, run anywhere”** (WORA).
* [Java Virtual Machine (JVM)](http://www.w3schools.in/java-tutorial/java-virtual-machine/) executes Java code, but is **written in platform specific languages such as**[**C**](http://www.w3schools.in/c/intro/)**/**[**C++**](http://www.w3schools.in/cplusplus/intro/)**/ASM etc**. JVM is not written in Java and hence **cannot be platform independent** and Java interpreter is actually a part of JVM.

## TYPES OF JAVA APPLICATIONS [W-S-E-M]

1. **Web Application:** Java is used to create server-side web applications. Currently, servlet, jsp, struts, jsf etc. technologies are used.
2. **Standalone Application:** It is also known as **desktop application** or **window-based application**. An application that we need to install on every machine or server such as media player, antivirus etc. AWT and Swing are used in java for creating standalone applications.
3. **Enterprise Application:** An application that is distributed in nature, such as banking applications etc. It has the advantage of high level security, load balancing and clustering. In java, **EJB** is used for creating enterprise applications.
4. **Mobile Application**: Java is used to create application softwares for mobile devices. Currently Java ME is used for creating applications for small devices, and also Java is programming language for Google Android application development.

**FEATURES OF JAVA [Simp-O-Secur-A-Port-R-M-I-H-Platform]**

* **Simple** − Java is designed to be easy to learn. If you understand the basic concept of OOP Java, it would be easy to master.
* **Object Oriented**- Almost everything in Java is a class, method or an object. Java provides Abstraction, Encapsulation, Inheritance, and Polymorphism.
* **Secure** − With Java's secure feature it enables to develop virus-free, tamper-free systems. Java contains built in security. Java never allows the programmer to memory manipulation (pointer) of the system.
* **Architecture-neutral** – Write once, run anywhere (WORA). With a JVM, you can write one program that will run on any platform.
* **Robust** − Java is a reliable language. It does not have permission to access all of your computer memory, so java program can’t cause a crash. Java programs are less prone to error. It also provides exception and error handling.
* **Multithreaded** − With Java's multithreaded feature it is possible to write programs that can perform many tasks simultaneously. This design feature allows the developers to construct interactive applications that can run smoothly.
* **Interpreted** – You need an interpreter to run JAVA programs. The programs are compiled into the Java Virtual Machine code called **byte code**. The byte code is machine independent and can run on any machine that has a Java interpreter, which is a part of JVM.
* **High Performance** − With the use of Just-In-Time compilers, Java enables high performance.
* **Platform Independent** − Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code. This byte code is distributed over the web and interpreted by the Virtual Machine (JVM) on whichever platform it is being run on.
* **Distributed** − Distributed computing involves several computers working together on a network. Java applications and applets can open and access objects across the web through URL’s as easily as they can access local file system.
* **Dynamic** − Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Maintaining different versions of application is easy in Java. Java supports dynamic memory allocation with automatic garbage collection.

**WHY IS JAVA SECURED?**

* No explicit pointer
* Java Programs run inside virtual machine sandbox

how java is secured

* **Classloader:** adds security by separating the package for the classes of the local file system from those that are imported from network sources.
* **Bytecode Verifier:** checks the code fragments for illegal code that can violate access right to objects.
* **Security Manager:** determines what resources a class can access such as reading and writing to the local disk.

These securities are provided by java language. Some security can also be provided by application developer through SSL, JAAS, Cryptography etc.

**DIFFERENCE BETWEEN C AND JAVA**

* Major difference is that C is a structure oriented language and Java is an object oriented language and has mechanism to define classes and objects.
* Java does not support an explicit pointer type
* Java does not have preprocessor, so we cant use #define, #include and #ifdef statements.
* Java does not include **structures**, **unions and enum data types**.
* Java does not include keywords like **goto, sizeof and typedef.**
* Java adds **labeled** break and continue statements.
* Java adds many features required for object oriented programming.

**DIFFERENCE BETWEEN C++ & JAVA**

**Features removed in java:**

* Java doesn’t support pointers to avoid unauthorized access of memory locations.
* Java does not include structures, unions and enum data types.
* Java does not support operator over loading.
* Preprocessor plays less important role in C++ and so eliminated entirely in java.
* Java does not perform automatic type conversions that result in loss of precision.
* Java does not support global variables. Every method and variable is declared within a class and forms part of that class.
* Java does not allow default arguments.
* Java does not support inheritance of multiple super classes by a sub class (i.e., multiple inheritance). This is accomplished by using ‘interface’ concept.
* It is not possible to declare unsigned integers in java.
* In java objects are passed by reference only. In C++ objects may be passed by value or reference.

**New features added in Java:**

* Multithreading that allows two or more pieces of the same program to execute concurrently.
* C++ has a set of library functions that use a common header file. But java replaces it with its own set of API classes.
* It adds packages and interfaces.
* Java supports **automatic garbage collection**.
* break and continue statements have been enhanced in java to accept labels as targets.
* The use of unicode characters ensures portability.

**Features that differ:**

* Though C++ and java supports Boolean data type, C++ takes any nonzero value as true and zero as false. True and false in java are predefined literals that are values for a boolean expression.
* Java has replacedthe destructor function with a finalize() function.
* C++ supports exception handling that is similar to java's. However, in C++ there is no requirement that a thrown exception be caught.

**PRINCIPLES OF OOPs**

In Object Oriented Programming, classes have attributes, represented by data member. The attributes distinguish an object of the class. Classes have behaviors, which are represented by methods. The methods define how an object acts or reacts.

**Feature of Object Oriented Programming :**

**Information Encapsulation (Hiding):-**

Object oriented programming allows you to encapsulate data that you do not want users of the object to access. Typically, attributes of a class are encapsulated.

**Abstraction:-**

Providing users with only what they need to know is known as abstraction. i.e. Abstraction lets us ignore the irrelevant details and concentrate on the essentials.

**Inheritance: -** Inheritance is the process by which objects of one class acquire the properties of objects of another class. Inheritance supports the concept of hierarchical classification. In OOP, the concept of inheritance provides the idea of reusability. This means that we can add additional features to an existing class without modifying it. This is possible by deriving a new class from the existing one. The new class will have the combined features of both the classes.

**Polymorphism: -** Polymorphism means “One Interface, multiple implementations.”

**PART-2**

**JAVA ENVIRONMENT**

**JAVA DEVELOPMENT KIT(JDK) [ J R E + tools for develop,debug,monitor]**

JDK (Java SE Development Kit) includes a complete JRE (Java Runtime Environment) plus tools for developing, debugging, and monitoring Java applications. **JDK is needed to develop Java applications and applets as well as run them.**

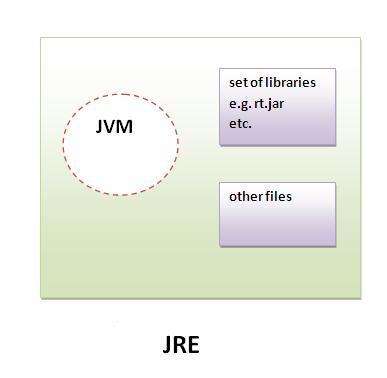
****

Java Development kit contains with the following **tools**

* **Appletviewer:** Enables us to run java applets(without actually using a java-compatible browser).
* **java:** java interpreter, which runs applets and applications by reading and interpreting bytecode files.
* **javac**: The java compiler, which translates java source code to bytecode files that the interpreter can understand.
* **javadoc :** Creates HTML format documentation from java source code files.
* **javah :** Produces header files for use with native methods
* **javap** : Java disassembler, which enables us to convert bytecode files into a program description.
* **jdb :** Java debugger, which helps us to find errors in our programs.

### JAVA RUNTIME ENVIRONMENT(JRE)

JRE is an acronym for Java Runtime Environment. It is used to provide runtime environment. It is the implementation of JVM. It physically exists. It contains set of libraries + other files that JVM uses at runtime.



**APPLICATION PROGRAMMING INTERFACE (API):**

Java supports several packages which contains **hundreds of classes & methods.These classes and methods are grouped together to form PACKAGES.**

**java.lang.\*** - language support package

A collection of classes and methods require for implementing basic features of java.

**java.util.\*** - utilities packages

A collection of classes to provide utility functions such as date & time functions.

**java.io.\*** - Input/Output package

A collection of classes required for input and output manipulations.

**java.net.\*** - networking package

A collection of classes for communicating with other computers via internet.

**java.awt.\*** - AWT(Abstract Window Toolkit) package

This package contains classes that implements platform independent GUI (Graphical User Interface)

**java.applet.\***- Applet package

This includes a set of classes that allow us to create JAVA applets.



**BYTECODE & JVM(Java Virtual Machine)**

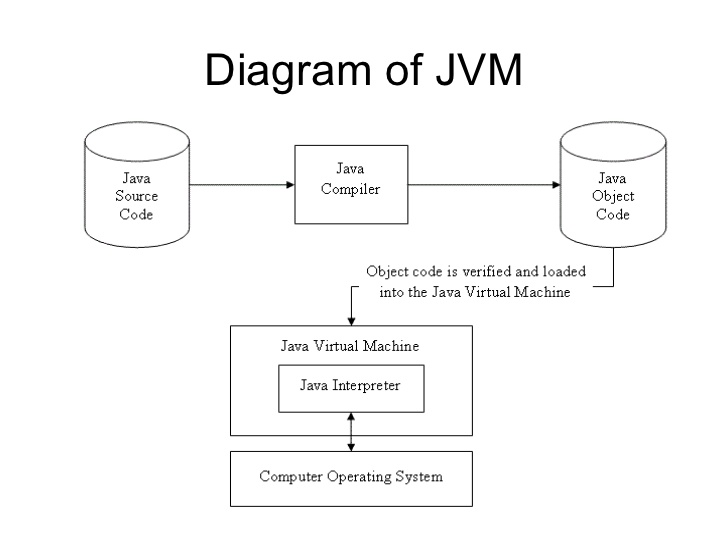
* Since platform-independence is a defining characteristic of Java, it is important to understand how it is achieved.
* Programs exist in two forms; source code and object code.
* **Source Code** is the textual version of the program that you write using a text editor.
* Executable form of a program is **object code**. The computer can execute object code. typically, object code is specific to a particular CPU. Therefore, it cannot be executed on a different platform.
* Like all computer languages, a java program begins with its source code.
* The difference is what happens when a Java program is compiled. Instead of producing executable code, the Java Compiler produces an object file that contains bytecode.
* **Bytecodes** are instructions that are not for any specific CPU. Instead, they are designed to be interpreted by a Java Virtual Machine (JVM).
* The key to Java’s platform-independence comes from the fact that the same bytecodes can be executed by any JVM on any platform.
* As long as there is a JVM implemented for a given environment, it can run any Java program. For example, Java programs can execute under Windows 98,Solaris,IRIX, or any other platform for which a JVM can be implemented for that platform. This would then allow any Java program to execute in that new environment.

**JVM (Java Virtual Machine)** is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.

JVMs are available for many hardware and software platforms. JVM, JRE and JDK are platform dependent because configuration of each OS differs. But, Java is platform independent.

The JVM performs following main tasks:

* Loads code
* Verifies code
* Executes code
* Provides runtime environment



## Just-in-time Compiler (JIT)

**JIT** is the part of the Java Virtual Machine (JVM) that is used to speed up the execution time. JIT interprets parts of the bytecode that have similar functionality at the same time, and hence reduces the amount of time needed for full interpretation.

**FIRST JAVA APPLICATION**

Create a file with name HelloWorld.java, because the compiler expects the file name to match the class identifier.

The HelloWorld application.

**public class HelloWorld**

**{**

**public static void main(String args[]) //** static tells, the main() is one to be executed first

**{**

**System.out.println("Hello World!!");**

**}**

**}**

**Explanation:**

**Class Declaration**

The first line public class HelloWorld declares a class, which is an Object-Oriented construct. As stated earlier Java is true Object-Oriented language and therefore, everything must be placed inside a class. Class is a keyword and declares that a new class definition follows.

**Opening Brace**

Every class definition in Java begins with an opening brace “{“ and ends with a matching closing brace “}”, appearing in the last line in the example.

## public static void main

* The word public means that it is accessible by any other classes.
* The word static means that it is unique.
* The word void means this main method has no return value.
* main is a method where the program starts.

**String args[]** declares a parameter named args, which contains an array of objects of the class type String.

**The Output Line**

The only executable Statement in the program is

**System.out.println("Hello World!!");**

The “println” method is a member of the out Object, which is **static data Member** of the “System” class. This line prints **Hello World!!** to the screen. The method println always appends a newline character to the end of the string.

## What happens at runtime?

|  |
| --- |
| At runtime, following steps are performed: |
| what happens at runtime when simple java program runs |

|  |
| --- |
| **Classloader:**is the subsystem of JVM that is used to load class files. |
| **Bytecode Verifier:**checks the code fragments for illegal code that can violate access right to objects. |
| **Interpreter:**read bytecode stream then execute the instructions. |

## DATA TYPES IN JAVA

1. Primary Data Type (Java supports 8 primitive data types): byte, short, int, long, float, double, char and boolean.
2. Non-Primitive Data Types: string, array etc.

#### Integer Types

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Type | Contains | Default | Size | Range |
| byte | Signed integer | 0 | 8 bit or 1 byte | -27 to 27-1 or -128 to 127 |
| short | Signed integer | 0 | 16 bit or 2 bytes | -215 to 215-1 or -32768 to 32767 |
| int | Signed integer | 0 | 32 bit or 4 bytes | -231 to 231-1 or -2147483648 to 2147483647 |
| long | Signed integer | 0 | 64 bit or 8 bytes | -263 to 263-1 or -9223372036854775808 to 9223372036854775807 |

#### Rational Numbers

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Type | Contains | Default | Size | Range |
| float | IEEE 754 floating point single-precision | 0.0f | 32 bit or 4 bytes | ±1.4E-45 to ±3.40282347E+38F |
| double | IEEE 754 floating point double-precision | 0.0 | 64 bit or 8 bytes | ±439E-324 to ±1.7976931348623157E+308 |

#### Characters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Type | Contains | Default | Size | Range |
| char | Unicode character unsigned | \u0000 | 16 bits or 2 bytes | 0 to 216-1 or \u0000 to \uFFFF |

#### Conditional

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Type | Contains | Default | Size | Range |
| boolean | true or false | False | 1 bit | true or false |

**JAVA Tokens**

* The smallest individual unit in a program is known as Token.
* Java language includes five types of tokens

1. Reserved Keywords

2. Identifiers

3. Literals

4. Operators

5. Separators

**KEYWORDS**

Keywords are essential to define language. Specific features of language are implemented using them. There are 50 Keywords in java language. They are,



## IDENTIFIERS

All Java components require names. **Names used for classes, variables, and methods** are called **identifiers**.

In Java, there are several points to remember about identifiers. They are as follows −

* All identifiers should begin with a letter (A to Z or a to z), currency character ($) or an underscore (\_).
* After the first character, identifiers can have any combination of characters.
* A key word cannot be used as an identifier.
* Most importantly, identifiers are case sensitive.
* Examples of legal identifiers: age, $salary, \_value, \_\_1\_value.
* Examples of illegal identifiers: 123abc, -salary.

**LITERALS**

Sequence of characters representing a constant value, to be stored in a variable.

Eg- int a=10; // 10 is an integer literal

boolean b=true; // true is a Boolean literal

**OPERATORS**

An operator is a symbol that takes one or more arguments and operates on them to produce a result.

Eg- 2+3, 5\*6 etc..

Java provides a rich set of operators to manipulate variables. We can divide all the Java operators into the following groups −

* Arithmetic Operators
* Relational Operators
* Bitwise Operators
* Logical Operators
* Assignment Operators
* Misc Operators

|  |  |
| --- | --- |
| **Operator Precedence** | |
| **Operators** | **Precedence** |
| postfix | *expr*++ *expr*-- |
| unary | ++*expr* --*expr* +*expr* -*expr* ~ ! |
| multiplicative | \* / % |
| additive | + - |
| shift | << >> >>> |
| relational | < > <= >= instanceof |
| equality | == != |
| bitwise AND | & |
| bitwise exclusive OR | ^ |
| bitwise inclusive OR | | |
| logical AND | && |
| logical OR | || |
| ternary | ? : |
| assignment | = += -= \*= /= %= &= ^= |= <<= >>= >>>= |

**SEPERATORS**

Symbols used to indicate where groups of code are divided and arranged.



**VARIABLE**

A variable is an Identifier that denotes a storage location used to store a data value. Unlike constants that

remain unchanged during the execution of program.

Examples of variables : average, height, total\_height.

Variable name may consist of alphabets, digits, the underscore(\_) and dollor characters.

**Rules to write Variable/Identifier in Java :**

* They must not begin with digit
* Upper and lowercase are distinct. This means that the variable Total is not the same as total or TOTAL.
* It should not be a keyword.
* White space is not allowed.
* Variable names can be of any length.

There are three kinds of variables in Java −

* 1. **Local variables**
* Local variables are declared in methods, constructors, or blocks.
* Local variables are created when the method, constructor or block is entered and the variable will be destroyed once it exits the method, constructor, or block.
  1. **Instance variables**
* Instance variables are declared in a class, but outside a method, constructor or any block.
* Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.

1. **Static variables**

* Class variables also known as static variables are declared with the static keyword in a class, but outside a method, constructor or a block.
* There would only be one copy of each class variable per class, regardless of how many objects are created from it.

**EXPRESSION**

An expression is a operators and operands. An expression may appear on the right side of an assignment statement. For example,

Int answer;

Answer=100\*31;

Java expression may contain Variables, constants or both. For example assuming that answer and count are variables, this expression is perfectly valid.

Answer=count-100;

**DECISION MAKING STATEMENTS**

Java supports following decision-making statements:

* [if statement](http://www.w3schools.in/java/decision-making/if/)
* [if-else statement](http://www.w3schools.in/java/decision-making/if-else/)
* [else-if statement](http://www.w3schools.in/java/decision-making/else-if/)
* [switch statement](http://www.w3schools.in/java/decision-making/switch/)
  1. **If Statement**

Java if statement is used to execute some code, if a condition is true otherwise if condition is false, execute nothing.

**Syntax:**

if(condition)

{

statement 1;

statement 2;

...

}

**Example:**

public class Sample{

public static void main(String args[]){

int a=20, b=30;

if(b>a)

System.out.println("b is greater");

}

}

**Output:**

b is greater

* 1. **If-else Statement**

Java if else statements is used to execute some code, if a condition is true otherwise if condition is false, execute nothing, or execute some other code.

**Syntax:**

if(condition)

{

//execute your code

}

else

{

//execute your code

}

**Example:**

public class Sample {

public static void main(String args[]) {

int a = 80, b = 30;

if (b > a) {

System.out.println("b is greater");

}

else {

System.out.println("a is greater");

}

}

}

**Output:**

a is greater

* 1. **Else if Statement**

Java elseif is a like doing another if condition for a true or false value.

**Syntax:**

if(condition)

{

//execute your code

}

else if(condition n)

{

//execute your code

}

else

{

//execute your code

}

**Example:**

public class Sample {

public static void main(String args[]) {

int a = 30, b = 30;

if (b > a) {

System.out.println("b is greater");

}

else if(a > b){

System.out.println("a is greater");

}

else {

System.out.println("Both are equal");

}

}

}

**Output:**

Both are equal

* 1. **Switch Statement**

Java switch statement is used when you have multiple possibilities for the if statement.

**Syntax:**

switch(variable)

{

case 1:

//execute your code

break;

case n:

//execute your code

break;

default:

//execute your code

break;

}

**Example:**

public class Sample {

public static void main(String args[]) {

int a = 5;

switch (a) {

case 1:

System.out.println("You chose One");

break;

case 2:

System.out.println("You chose Two");

break;

case 3:

System.out.println("You chose Three");

break;

case 4:

System.out.println("You chose Four");

break;

case 5:

System.out.println("You chose Five");

break;

default:

System.out.println("Invalid Choice. Enter a no between 1 and 5");

break;

}

}

}

**Output:**

You chose five

**LOOPS IN JAVA**

Sometimes we want a Java program to repeat something over and over again, loops are used to make a program do something more than one time.

Java loops execute a block of commands a specified number of times, until a condition is met.

Java supports following types of loops:

* [while loops](http://www.w3schools.in/java/loops/while/)
* [do while loops](http://www.w3schools.in/java/loops/do-while/)
* [for loops](http://www.w3schools.in/java/loops/for/)

1. **While loop**

Java while loops statement allows to repeatedly run the same block of code, until a condition is met.

**Syntax:**

While (condition)

{

statement(s);

Incrementation;

}

**Example:**

public class Sample {

public static void main(String args[]) {

/\* local variable Initialization \*/

int n = 1, times = 5;

/\* while loops execution \*/

while (n <= times) {

System.out.println("Java while loops:" + n);

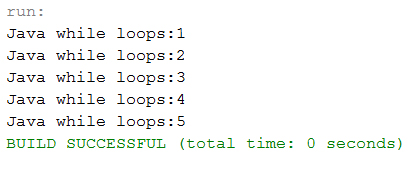
n++;

}

}

}

**Output:**



1. **do while loop**

Java do while loops is very similar to the java while loops, but it always executes the code block at least once and further more as long as the condition remains true.

**Syntax:**

do

{

statement(s);

} while( condition );

**Example:**

public class Sample {

public static void main(String args[]) {

/\* local variable Initialization \*/

int n = 1, times = 0;

/\* do-while loops execution \*/

do {

System.out.println("Java do while loops:" + n);

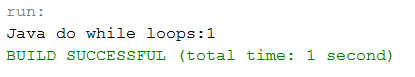
n++;

} while (n <= times);

}

}

**Output:**



1. **For loop**

Java for loops is very similar to Java while loops in that it continues to process a block of code until a statement becomes false, and everything is defined in a single line.

**Syntax:**

for ( init; condition; increment )

{

statement(s);

}

**Example:**

public class Sample {

public static void main(String args[]) {

/\* local variable Initialization \*/

int n = 1, times = 5;

/\* for loops execution \*/

for (n = 1; n <= times; n = n + 1) {

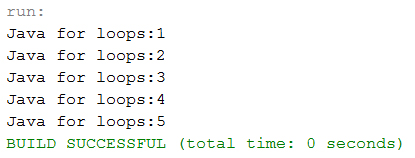
System.out.println("Java for loops:" + n);

}

}

}

**Output:**



## BRANCHING STATEMENTS IN JAVA

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

## break Statement

When a break statement is encountered inside a loop, the loop is terminated and program control resumes at the next statement following the loop.

public class BreakDemo

{

public static void main(String[] args)

{

for (int i = 1; i <= 10; i++)

{

if (i == 5)

{

break; // terminate loop if i is 5

}

System.out.print(i + " ");

}

System.out.println("Loop is over.");

}

}

**Output :**

1 2 3 4 Loop is over.

The break statement has two forms: labeled and unlabeled. You can also use an unlabeled break to terminate a for, while, or do-while loop

* 1. **continue Statement**

The continue statement skips the current iteration of a for, while, or do-while loop. When a continue statement is encountered inside the body of a loop, remaining statements are skipped and loop proceeds with the next iteration.

public class ContinueDemo

{

public static void main(String[] args)

{

for (int i = 1; i <= 10; i++)

{

if (i % 2 == 0)

{

continue; // skip next statement if i is even

}

System.out.println(i + " ");

}

}

}

**Output :**

1 3 5 7 9

The unlabeled form skips to the end of the innermost loop's body and evaluates the boolean expression that controls the loop.

## return Statement

The last of the branching statements is the return statement. The return statement exits from the current method, and control flow returns to where the method was invoked. The return statement has two forms: one that returns a value, and one that doesn't.

To return a value simply put the value (or an expression that calculates the value) after the return keyword.

return ++count;

The data type of the returned value must match the type of the method's declared return value. When a method is declared void, use the form of return that doesn't return a value.

return;

**STATEMENTS IN JAVA**

A statement specifies an action in a Java program, such as assigning the sum of x and y to z, printing a message to the standard output, writing data to a file, etc.

Statements in Java can be broadly classified into three categories:

* Declaration statement
* Expression statement
* Control flow statement

## Declaration Statement

A declaration statement is used **to declare a variable**. For example,

|  |
| --- |
| int num;  int num2 = 100;  String str; |

## Expression Statement

An **expression with a semicolon at the end** is called an expression statement. For example,

|  |
| --- |
| //**Increment and decrement expressions**  num++;  ++num;  num--;  --num;  **//Assignment expressions**  num = 100;  num \*= 10;  **//Method invocation expressions**  System.out.println("This is a statement");  someMethod(param1, param2); |

## Control Flow Statement

By default, all statements in a Java program are executed in the order they appear in the program. Sometimes you may want to execute a set of statements repeatedly for a number of times or as long as a particular condition is true.

All of these are possible in Java using control flow statements. If block, while loop and for loop statements are examples of control flow statements.

**MACHINE NEUTRAL**

Machine Neutral Platform Independent means that while writing the program in java, we don't need to care about on which Machine or platform the program will going to be executed. Like in C language if we want to run the program in Windows XP the we need to write the program in windows compatible C compiler and in windows platform to c program  in UNIX or Linux  we need to edit and compile the same program in Linux and UNIX compatible compiler and environment . Same in visual basic program

can only run in Windows platform.

Java breaks all these restriction, Java gives the freedom to write and compile the program in any platform or machine and the same program can run on different platform and machine. Example Java program written for Linux system can also run on windows and UNIX etc.

**How Java do this?**

The secret behind platform independent of java is JAVA VIRTUAL MACHINE.  
All language compilers translate source code into machine code for a specific computer. Java compiler also does the same thing. Then, how does java achieve architecture neutrality? The answer is that the Java compiler produces an itermedia code known as **bytecode** for a machine that does not exits. This machine is called the Java Virtual Machine and it exits only inside the computer memory. It is a simulated computer within the computer and does all major functions of a real computer.

**COMMAND LINE ARGUMENTS IN JAVA**

The parameters which are provided to the program at the command line are called command line arguments.

**Eg: d:\> java a 10 20 30**

Here, “java” is an interpreter, “a” is file name, 10, 20, 30 are arguments passed to the program while executing the program.

And the notation of String args[ ] in main method will be cleared with the command line arguments.

public static void main(String[ ] args)

The “String args[ ]” declares args as an array of strings. args by default holds the strings given to the program while executing it.

**EXAMPLE-1**

class Command

{

public static void main(String args[])

{

System.out.println(args[0]);//holds the first string

System.out.println(args[1]);//holds the second string

System.out.println(args[2]);//holds the third string

}

}

**EXAMPLE-2**

class applyCommand

{

public static void main(String[] args)

{

String name;

name=args[0];//first value given in command line

int Roll;

Roll=Integer.parseInt(args[1]);//converting string to integer

float marks;

marks=Float.parseFloat(args[2]);//converting string to float

System.out.println("Name is:"+name);

System.out.println("Roll no:"+Roll);

System.out.println("Marks are:"+marks);

}

}

**PART-3**

**ARRAYS AND STRINGS**

**ARRAY**

Normally, array is a collection of similar type of elements that have contiguous memory location.

**Java array** is an object that contains elements of similar data type. It is a data structure where we store similar elements. We can store only fixed set of elements in a java array.

Array in java is index based. First element of the array is stored at 0 index.



**Advantage of Java Array**

* **Code Optimization:** It makes the code optimized.We can retrieve or sort the data easily.
* **Random access:** We can get any data located at any index position.

**Disadvantage of Java Array**

* **Size Limit:** We can store only fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in java.

**SINGLE DIMENSIONAL ARRAY IN JAVA**

### Declaration of Array in java

dataType[] arr; (or)

dataType []arr; (or)

dataType arr[];

**Creating and Initializing Arrays**

One way to create an array is with the new operator.

arrayRefVar=**new** datatype[size];

The above statement does two things −

* It creates an array using new dataType[arraySize].
* It assigns the reference of the newly created array to the variable arrayRefVar.

// create an array of 10 integers

anArray = new int[10];

We can assign values to each element of the array:

anArray[0] = 100; // initialize first element

anArray[1] = 200; // initialize second element

anArray[2] = 300; // and so forth

Each array element is accessed by its numerical index:

System.out.println("Element 1 at index 0: " + anArray[0]);

System.out.println("Element 2 at index 1: " + anArray[1]);

System.out.println("Element 3 at index 2: " + anArray[2]);

Alternatively, you can use the shortcut syntax to create and initialize an array:

int[] anArray = {

100, 200, 300,

400, 500, 600,

700, 800, 900, 1000

};

Here the length of the array is determined by the number of values provided between braces and separated by commas.

### Example of single dimensional java array

class Testarray{

public static void main(String args[]){

int a[]=new int[5];//declaration and instantiation

a[0]=10;//initialization

a[1]=20;

a[2]=70;

a[3]=40;

a[4]=50;

  //printing array

for(int i=0;i<a.length;i++)//length is the property of array

System.out.println(a[i]);

}}

Output: 10

20

70

40

50

## Processing Arrays

When processing array elements, we often use either **for** loop or **foreach** loop because all of the elements in an array are of the same type and the size of the array is known.

### Example

Here is a complete example showing how to create, initialize, and process arrays −

public class TestArray {

public static void main(String[] args) {

double[] myList = {1.9, 2.9, 3.4, 3.5};

// Print all the array elements

for (int i = 0; i < myList.length; i++) {

System.out.println(myList[i] + " ");

}

// Summing all elements

double total = 0;

for (int i = 0; i < myList.length; i++) {

total += myList[i];

}

System.out.println("Total is " + total);

// Finding the largest element

double max = myList[0];

for (int i = 1; i < myList.length; i++) {

if (myList[i] > max) max = myList[i];

}

System.out.println("Max is " + max);

}

}

This will produce the following result −

### Output

1.9

2.9

3.4

3.5

Total is 11.7

Max is 3.5

## The foreach Loops

JDK 1.5 introduced a new for loop known as foreach loop or enhanced for loop, which enables you to traverse the complete array sequentially without using an index variable.

### Example

The following code displays all the elements in the array myList −

public class TestArray {

public static void main(String[] args) {

double[] myList = {1.9, 2.9, 3.4, 3.5};

// Print all the array elements

for (double element: myList) {

System.out.println(element);

}

}

}

This will produce the following result −

**Output**

1.9

2.9

3.4

3.5

**MULTIDIMENSIONAL ARRAY IN JAVA**

In such case, data is stored in row and column based index (also known as matrix form).

**Syntax to Declare Multidimensional Array in java**

dataType[][] arrayRefVar; (or)

dataType [][]arrayRefVar; (or)

dataType arrayRefVar[][]; (or)

dataType []arrayRefVar[];

Example to instantiate Multidimensional Array in java

**int**[][] arr=**new** **int**[3][3];//3 row and 3 column

Example to initialize Multidimensional Array in java

arr[0][0]=1;

arr[0][1]=2;

arr[0][2]=3;

arr[1][0]=4;

arr[1][1]=5;

arr[1][2]=6;

arr[2][0]=7;

arr[2][1]=8;

arr[2][2]=9;

**Example of Multidimensional java array**

Let's see the simple example to declare, instantiate, initialize and print the 2Dimensional array.

class Testarray3{

public static void main(String args[]){

//declaring and initializing 2D array

int arr[][]={{1,2,3},{2,4,5},{4,4,5}};

//printing 2D array

for(int i=0;i<3;i++){

 for(int j=0;j<3;j++){

   System.out.print(arr[i][j]+" ");

 }

 System.out.println();

}

}}

**Output:**

1 2 3

2 4 5

4 4 5

**STRINGS IN JAVA:**

* Generally, string is a sequence of characters.
* In java, string is an object that represents a sequence of characters.
* The **java.lang.String** class is used to create string object.
* String can be created in number of ways, here are a few ways of creating string object.

1) **Using a String literal**

String literal is a simple string enclosed in double quotes " ". A string literal is treated as a String object.

String str1=”Hello”;

2) **Using another String object**

String str2=new String(str1);

3) **Using new Keyword**

String str3=new String(“Java”);

4) **Using + operator (Concatenation)**

String str4=str1+str2;

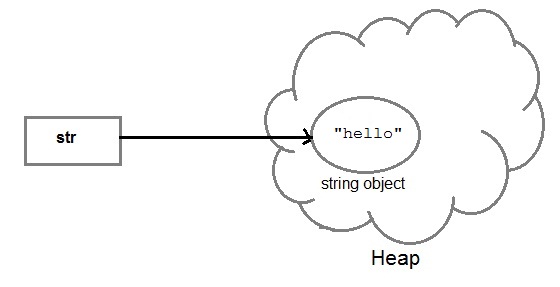
Or

String str4=”Hello”+”Java”;

**How String object are stored?**

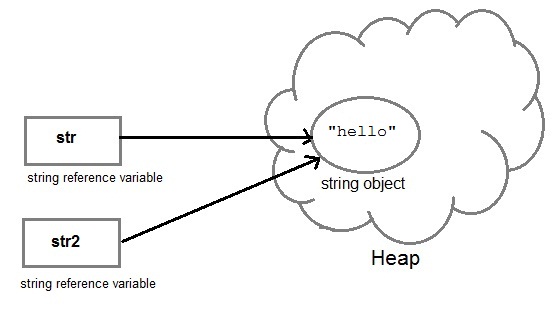
Each time you create a String literal, the JVM checks the string pool first. If the string literal already exists in the pool, a reference to the pool instance is returned. If string does not exist in the pool, a new string object is created, and is placed in the pool. String objects are stored in a special memory area known **as string constant pool inside the heap memory**.

**String str1=”Hello”;**



And, when we create another object with same string, then a reference of the string literal already present in string pool is returned.

**String str2=str;**



**Java String class methods**

The java.lang.String class provides many useful methods to perform operations on sequence of char values.The following methods are some of the most commonly used methods of String class.

1. **charAt()**

charAt() function returns the character located at the specified index.

String str = "studytonight";

System.out.println(str.charAt(2));

Output : u

1. **equalsIgnoreCase()**

equalsIgnoreCase() determines the equality of two Strings, ignoring thier case (upper or lower case doesn't matters with this fuction ).

String str = "java";

System.out.println(str.equalsIgnoreCase("JAVA");

Output : true

1. **length()**

length() function returns the number of characters in a String.

String str = "Count me";

System.out.println(str.length());

Output : 8

1. **replace()**

replace() method replaces all occurances of character with a specified new character.

String str = "Change me";

System.out.println(str.replace('m','M'));

Output : Change Me

1. **substring()**

substring() method returns a part of the string. substring() method has two forms,

**public String substring(int begin); ----- String stands for return type of the function**

**public String substring(int begin, int end); ---- end index is excluded**

The first argument represents the starting point of the subtring. If the substring() method is called with only one argument, the subtring returned, will contain characters from specified starting point to the end of original string.

But, if the call to substring() method has two arguments, the second argument specify the end point of substring.

String str = "0123456789";

System.out.println(str.substring(4));

Output : 456789

System.out.println(str.substring(4,7));

Output : 456

1. **toLowerCase()**

toLowerCase() method returns string with all uppercase characters converted to lowercase.

String str = "ABCDEF";

System.out.println(str.toLowerCase());

Output : abcdef

1. **valueOf() --- (primitive data types 🡪 String)**

Overloaded version of valueOf() method is present in “String” class for all primitive data types and for type Object.

NOTE : **valueOf() function is used to convert primitive data types into Strings.**

But for objects, valueOf() method calls toString() function.

1. **toString() --- (objects -🡪 strings)**

toString() method returns the **string representation of the “object”** used to invoke this method.

toString() is used to represent any Java Object into a meaningful string representation.

public class Car {

public static void main(String args[])

{

Car c=new Car();

System.out.println(c);

}

public String toString()

{

return "This is my car object";

}

}

Output : This is my car object

Whenever we will try to print any object of class Car, its toString() function will be called. toString() can also be used with normal string objects.

String str = "Hello World";

System.out.println(str.toString());

Output : Hello World

1. **toUpperCase()**

This method returns string with all lowercase character changed to uppercase.

String str = "abcdef";

System.out.println(str.toUpperCase());

Output : ABCDEF

1. **trim()**

This method returns a string from which any leading and trailing whitespaces has been removed.

String str = " hello ";

System.out.println(str.trim());

Output : hello

1. **equals() method----checks the content of the strings**

equals() method compares two strings for equality. Its general syntax is,

boolean equals (Object str)

It compares the **content of the strings**. It will return true if string matches, else returns false.

String s = "Hell";

String s1 = "Hello";

String s2 = "Hello";

s1.equals(s2); //true

s.equals(s1) ; //false

**Difference between == & equals()**

== operator **compares two object references** to check whether they refer to same instance. This also, will return true on successful match.

String s1 = "Java";

String s2 = "Java";

String s3 = new string ("Java");

test(sl == s2) //true

test(s1 == s3) //false

where as equals() compares the content of the strings.

**12.** **compareTo() method**

compareTo() method compares values and returns an int which tells if the string compared is less than, equal to or greater than the other string. Its general syntax is,

**int compareTo(String str)-----int denotes the return type of the compareTo()**

To use this function you must implement the Comparable Interface. compareTo() is the only function in Comparable Interface.

String s1 = "Abhi";

String s2 = "Viraaj";

String s3 = "Abhi";

s1.compareTo(s2); //return -1 because s1 < s2

s1.compareTo(s3); //return 0 because s1 == s3

s2.compareTo(s1); //return 1 because s2 > s1

1. **concat() method**

string s = "Hello";

string str = "Java";

string str2 = s.concat(str);

String str1 = "Hello".concat("Java"); //works with string literals too.

### StringBuffer class

* Create a **mutable** string object.
* It represents **growable and writable** character sequence.
* As we know that String objects are immutable, so if we do a lot of changes with **String** objects, we will end up with a lot of memory leak.
* So **StringBuffer** class is used when we have to make lot of modifications to our string.
* It is also thread safe i.e multiple threads cannot access it simultaneously.

StringBuffer defines some constructors. They are,

1. **StringBuffer** ( )- creates an empty string buffer and reserves room for 16 characters.
2. **StringBuffer** ( int size )- creates an empty string and takes an integer argument to set capacity of the buffer.

class Test {

public static void main(String args[])

{

String str = "Hello";

str.concat("Java");

System.out.println(str); // Output: Hello

StringBuffer strB = new StringBuffer("Hello");

strB.append("Java");

System.out.println(strB); // Output: HelloJava

}

}

#### Important methods of StringBuffer class [A-I-Rev-Rep-C-EC]

The following methods are some most commonly used methods of StringBuffer class.

#### append()

This method will concatenate the string representation of any type of data to the end of the invoking **StringBuffer** object. append() method has several overloaded forms.

**StringBuffer append(String str)**

**StringBuffer append(int n)**

**StringBuffer append(Object obj)**

The string representation of each parameter is appended to **StringBuffer** object.

StringBuffer str = new StringBuffer("test");

str.append(123);

System.out.println(str);

Output : test123

#### insert()

This method inserts one string into another. Here are few forms of insert() method.

**StringBuffer insert(int index, String str)**

**StringBuffer insert(int index, int num)**

**StringBuffer insert(int index, Object obj)**

Here the first parameter gives **the “index” at which position** the string will be inserted and **string** representation of second parameter is inserted into StringBuffer object.

StringBuffer str = new StringBuffer("test");

str.insert(4, 123);

System.out.println(str);

Output : test123

#### reverse()

This method reverses the characters within a **StringBuffer** object.

StringBuffer str = new StringBuffer("Hello");

str.reverse();

System.out.println(str);

Output : olleH

1. **replace()**

This method replaces the string from specified **start “index” to the end “index”**.

StringBuffer str = new StringBuffer("Hello World");

str.replace( 6, 11, "java");

System.out.println(str);

Output : Hello java

#### capacity()

This method returns the current capacity of **StringBuffer** object.

StringBuffer str = new StringBuffer();

System.out.println( str.capacity() );

Output : 16

1. **ensureCapacity()**

This method is used to ensure minimum capacity of StringBuffer object.

StringBuffer str = new StringBuffer("hello");

str.ensureCapacity(10);

**VECTORS**

* Vectors (the java.util.Vector class) are commonly used instead of arrays, because they expand automatically when new data is added to them.
* The Java 2 Collections API introduced the similar ArrayList data structure.
* ArrayLists are unsynchronized and therefore faster than Vectors, but less secure in a multithreaded environment
* A vector is a dynamic array, whose size can be increased, where as an array size cannot be changed.
* Reserve space can be given for vector, where as for arrays cannot.
* A vector is a class where as an array is not.
* Vectors can store any type of objects, where as an array can store only homogeneous values.
* **Create a Vector with default initial size**   
  Vector v = new Vector();
* **Create a Vector with an initial size**   
  Vector v = new Vector(300);
* **Advantages of Vector:**  
  - Size of the vector can be changed  
  - Multiple objects can be stored  
  - Elements can be deleted from a vector
* **Disadvantages of Vector:**  
  - A vector is an object, memory consumption is more.

## Important methods of Vector Class:

|  |  |
| --- | --- |
| **Method call** | **Task performed** |
| list.addElement(item) | Adds the item specified to the list at the end |
| list.elementAt(10) | Gives the name of the 10th object |
| list.size() | Gives the number of object present |
| list.removeElement(item) | Removes the specified item from the list |
| list.removeElementAt(n) | Remove the items stored in nth position of the list |
| list.removeAllElements() | Remove all the elements in the list |
| list.copyInto(array) | Copies all items from list to array |
| list.insertElementAt(item,n) | Inserts the item at nth position |

**Example:**

import java.util.\*;

public class VectorExample {

public static void main(String args[]) {

Vector list = new Vector ();

int length=args.length;

for(int i=0;i<length;i++)

{

list.addElement(args[i]);

}

list.insertElementAt(“COBOL”,2);

int size=list.size();

String listArray[]=new String[size];

list.copyInto(listArray);

System.out.println(“List of Languages”);

for(int i=0;i<size;i++)

{

System.out.println(listArray[i]);

}

}

}

**OUTPUT:**

D:\program\java VectorExample c c++ java python fortran

c

c++

cobol

java

python

fortran

**WRAPPER CLASSES3**

A wrapper class wraps (encloses) around a data type and gives it an object appearance. Wherever, the data type is required as an object, this object can be used (called as boxing). Wrapper classes include methods to unwrap the object and give back the data type (called as unboxing).

**Example 1:** Observe the following conversion.

int k = 100;  
Integer it1 = new Integer(k);

The int data type k is converted into an object, it1 using Integer class. The it1 object can be used in Java programming wherever k is required an object.

**Example 2:** The following code can be used to unwrap (getting back int from Integer object) the object it1.

int m = it1.intValue();//unwrapping  
System.out.println(m\*m); // prints 10000

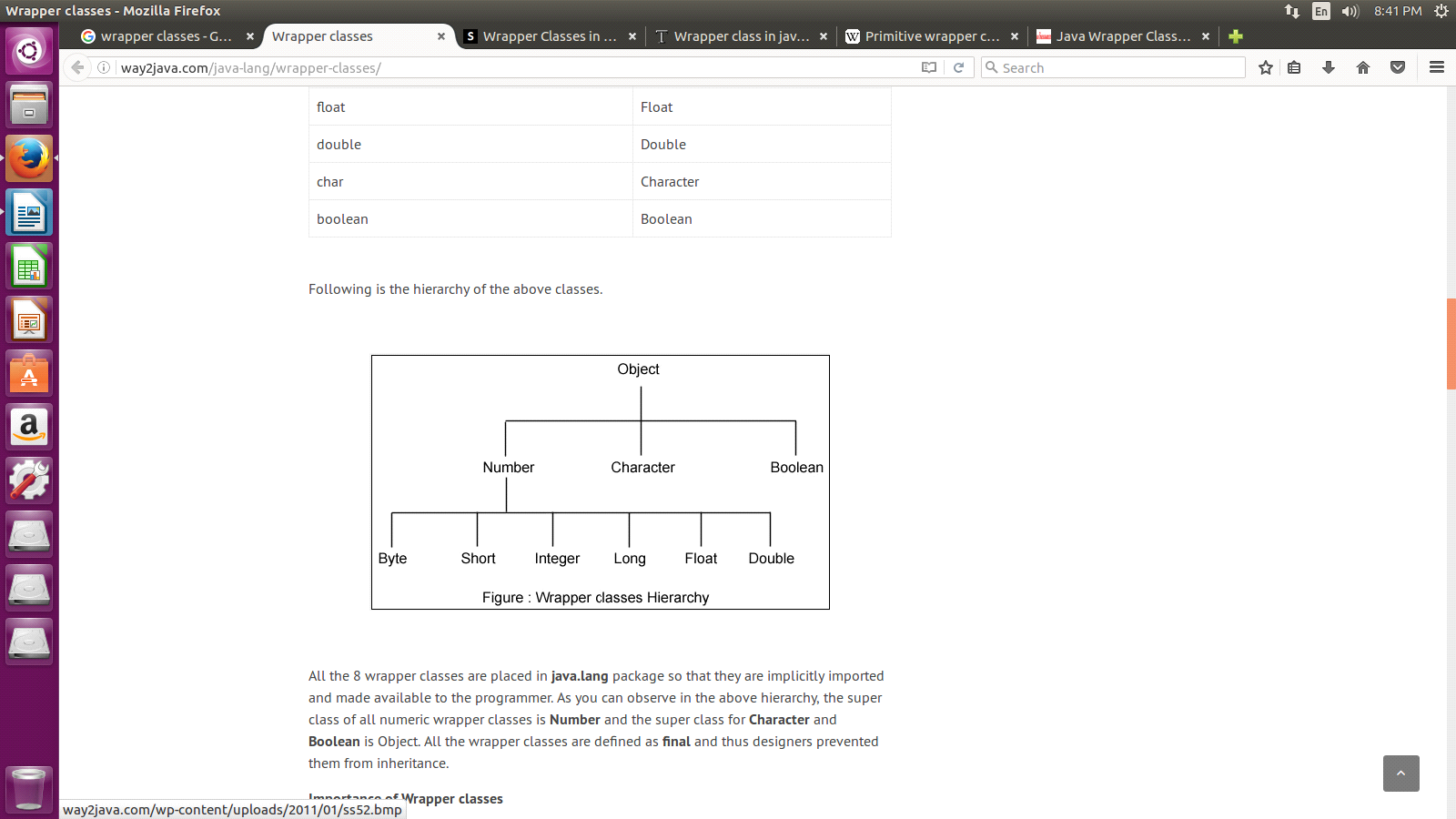
intValue() is a method of Integer class that returns an int data type.

**List of Wrapper classes**:

Eight wrapper classes exist in **java.lang** package that represent 8 data types. Following list gives.

|  |  |
| --- | --- |
| **Primitive data type** | **Wrapper class** |
| byte | Byte |
| short | Short |
| int | Integer |
| long | Long |
| float | Float |
| double | Double |
| char | Character |
| boolean | Boolean |

Following is the hierarchy of the above classes.



**Importance of Wrapper classes**

There are mainly two uses with wrapper classes.

* They convert primitive data types into objects. Objects are needed if we wish to modify the arguments passed into a method (because primitive types are passed by value).
* The classes in java.util package handles only objects and hence wrapper classes help in this case also.
* Data structures in the Collection framework, such as [ArrayList](https://www.geeksforgeeks.org/arraylist-in-java/) and [Vector](https://www.geeksforgeeks.org/vector-vs-arraylist-java/), store only objects (reference types) and not primitive types.
* An object is needed to support synchronization in multithreading.
* To convert **strings into data types** (known **as parsing operations**), here methods of type **parseXXX()** are used.

List of tables for conversions from one type to other type:

1) converting **Primitive Numbers to objects** numbers using constructor method

|  |  |
| --- | --- |
| **Constructor calling** | **Conversion action** |
| Interger IntVal=new Integer(i); | Primitive integer to integer object |
| Float FloatVal=new Float(f); | Primitive float to float object |
| Double DoubleVal=new Double(d); | Primitive double to double object |
| Long LongVal=new Long(l); | Primitive long to long object |

**IntVal, FloatVal, DoubleVal, LongVal----are the objects of the class**

**i,f,d,l---these are the values of respective data types that to converted into objects**

**before passing these variables as parameters, we need to assign some values to these variables.**

2)converting o**bject numbers to primitive numbers** using **typeValue()** method

|  |  |
| --- | --- |
| **Method calling** | **Conversion action** |
| int i=IntVal.intValue(); | Object to primitive integer |
| float f=FloatVal.floatValue(); | Object to primitive float |
| long l=LongVal.longValue(); | Object to primitive long |
| double d=DoubleVal.doubleValue(); | Object to primitive double |
|  |  |

3)converting **number to numeric strings** using **tostring()** method

|  |  |
| --- | --- |
| **Method calling** | **Conversion action** |
| str=Integer.toString(i); | Primityive intger to string |
| str=Float.toString(f); | Primitive float to string |
| str=Double.toString(d); | Primitive double to string |
| str=Long.toString(l); | Primitve long to string |

4)conversion from **strings to numeric object** using **valueOf()** method

|  |  |
| --- | --- |
| **Method calling** | **Conversion action** |
| DoubleVal=Double.valueOf(str); | Converts string to double object |
| FloatVal=Float.valueOf(str); | Conversts string to float object |
| Intval=Integer.valueOf(str); | Converts string to integer object |
| LongVal=Long.valueOf(str); | Converts string to long object |

5) converting **numeric strings to primitive numbers** using **parsing methods**

|  |  |
| --- | --- |
| **Method calling** | **Conversion action** |
| int i=Integer.parseInt(Str); | Converst string to primitive integer |
| long i=Long.parseLong(str); | Converts string to primitive long |

The following program expresses the style of converting data type into an object and at the same time retrieving the data type from the object.

**Program:**

import java.util.\*;

import java.io.\*;

public class WrappingUnwrapping

{

  public static void main(String args[])

  {                   //  data types

    byte grade = 2;

    int marks = 50;

    float price = 8.6f;

double rate = 50.5;

    Byte g1 = new Byte(grade);                    // wrapping

    Integer m1 = new Integer(marks);

    Float f1 = new Float(price);

    Double r1 = new Double(rate);

    System.out.println("Values of Wrapper objects (printing as objects)");

    System.out.println("Byte object g1:  " + g1);

    System.out.println("Integer object m1:  " + m1);

    System.out.println("Float object f1:  " + f1);

    System.out.println("Double object r1:  " + r1);

    byte bv = g1.byteValue();                 // unwrapping

    int iv = m1.intValue();

    float fv = f1.floatValue();

    double dv = r1.doubleValue();

    System.out.println("Unwrapped values (printing as data types)");

    System.out.println("byte value, bv: " + bv);

    System.out.println("int value, iv: " + iv);

    System.out.println("float value, fv: " + fv);

    System.out.println("double value, dv: " + dv);

  }

}

**BASIC I/O STREAMS**

**IO Stream**

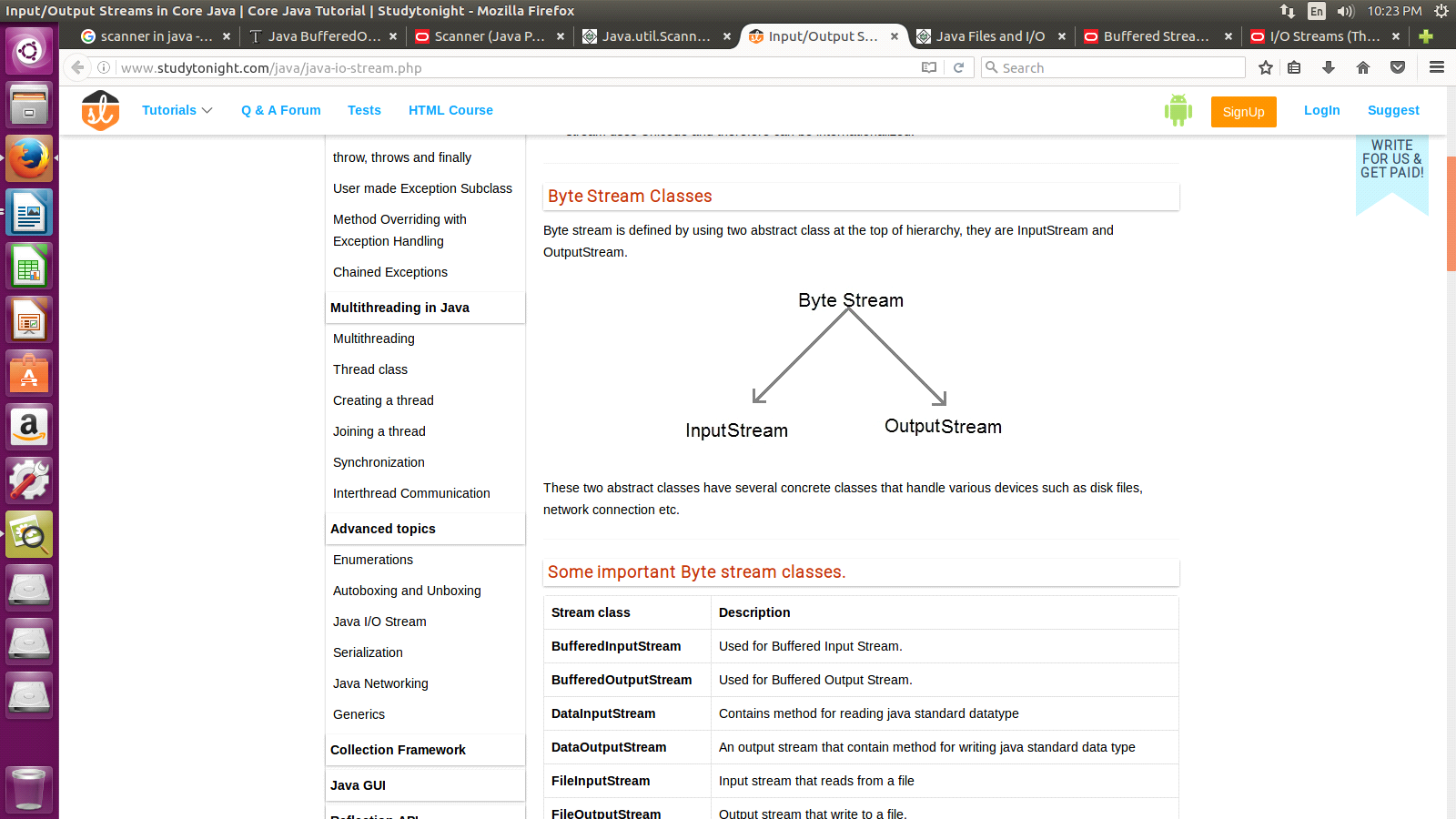
Java performs I/O through **Streams**. A Stream is linked to a physical layer by java I/O system to make input and output operation in java. In general, a stream means continuous flow of data. Streams are clean way to deal with input/output without having every part of your code understand the physical.

Java encapsulates Stream under **java.io** package. Java defines two types of streams. They are,

* **Byte Stream :** It provides a convenient means for handling input and output of byte.
* **Character Stream :** It provides a convenient means for handling input and output of characters. Character stream uses Unicode and therefore can be internationalized.

**Byte Stream Classes**

Byte stream is defined by using two abstract class at the top of hierarchy, they are InputStream and OutputStream.



These two abstract classes have several concrete classes that handle various devices such as disk files, network connection etc.

**Some important Byte stream classes.**

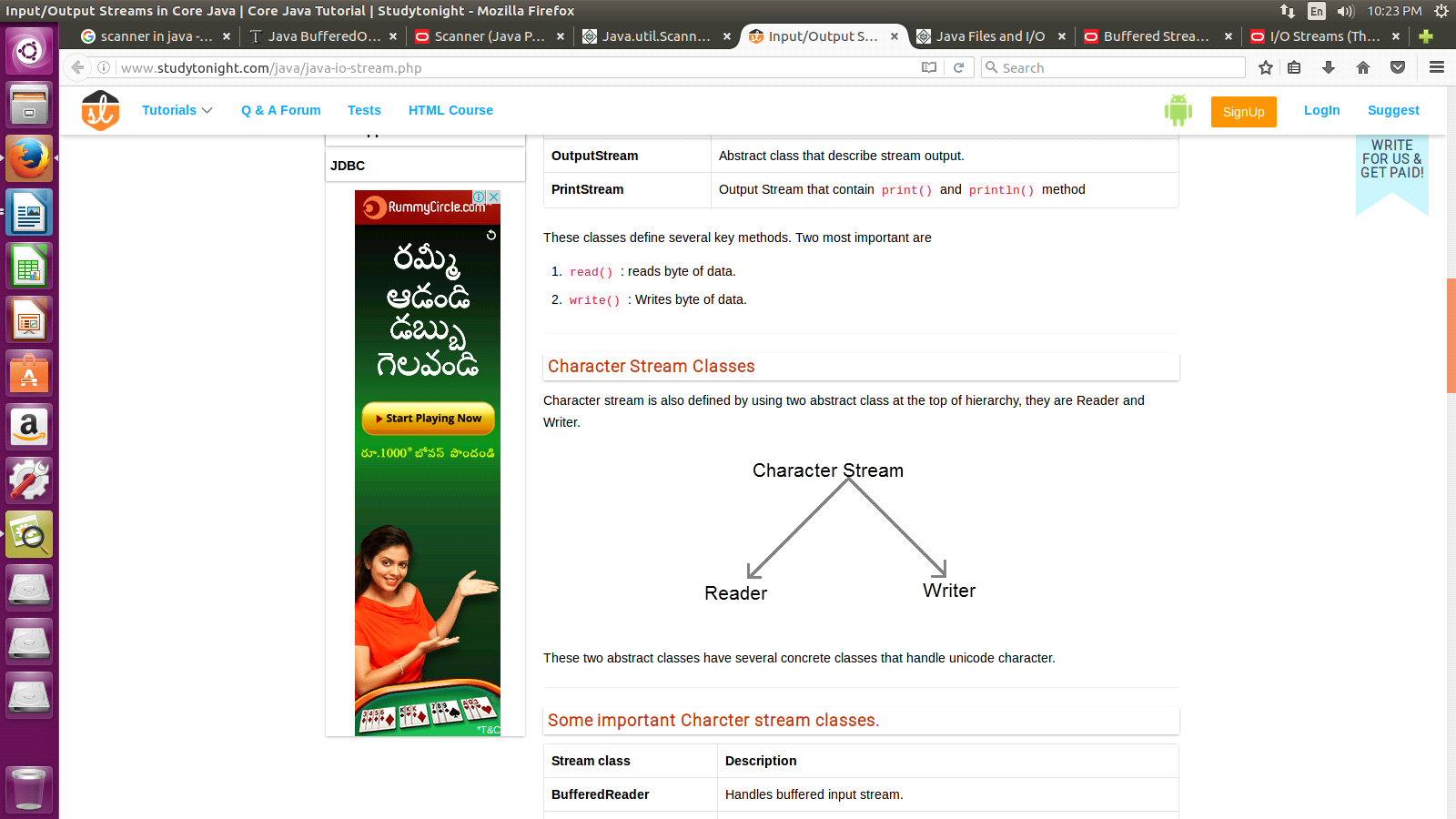
|  |  |
| --- | --- |
| **Stream class** | **Description** |
| **BufferedInputStream** | Used for Buffered Input Stream. |
| **BufferedOutputStream** | Used for Buffered Output Stream. |
| **DataInputStream** | Contains method for reading java standard datatype |
| **DataOutputStream** | An output stream that contain method for writing java standard data type |
| **FileInputStream** | Input stream that reads from a file |
| **FileOutputStream** | Output stream that write to a file. |
| **InputStream** | Abstract class that describe stream input. |
| **OutputStream** | Abstract class that describe stream output. |
| **PrintStream** | Output Stream that contain print()and println()method |

These classes define several key methods. Two most important are

* read() : reads byte of data.
* write() : Writes byte of data.

**Character Stream Classes**

Character stream is also defined by using two abstract class at the top of hierarchy, they are Reader and Writer.



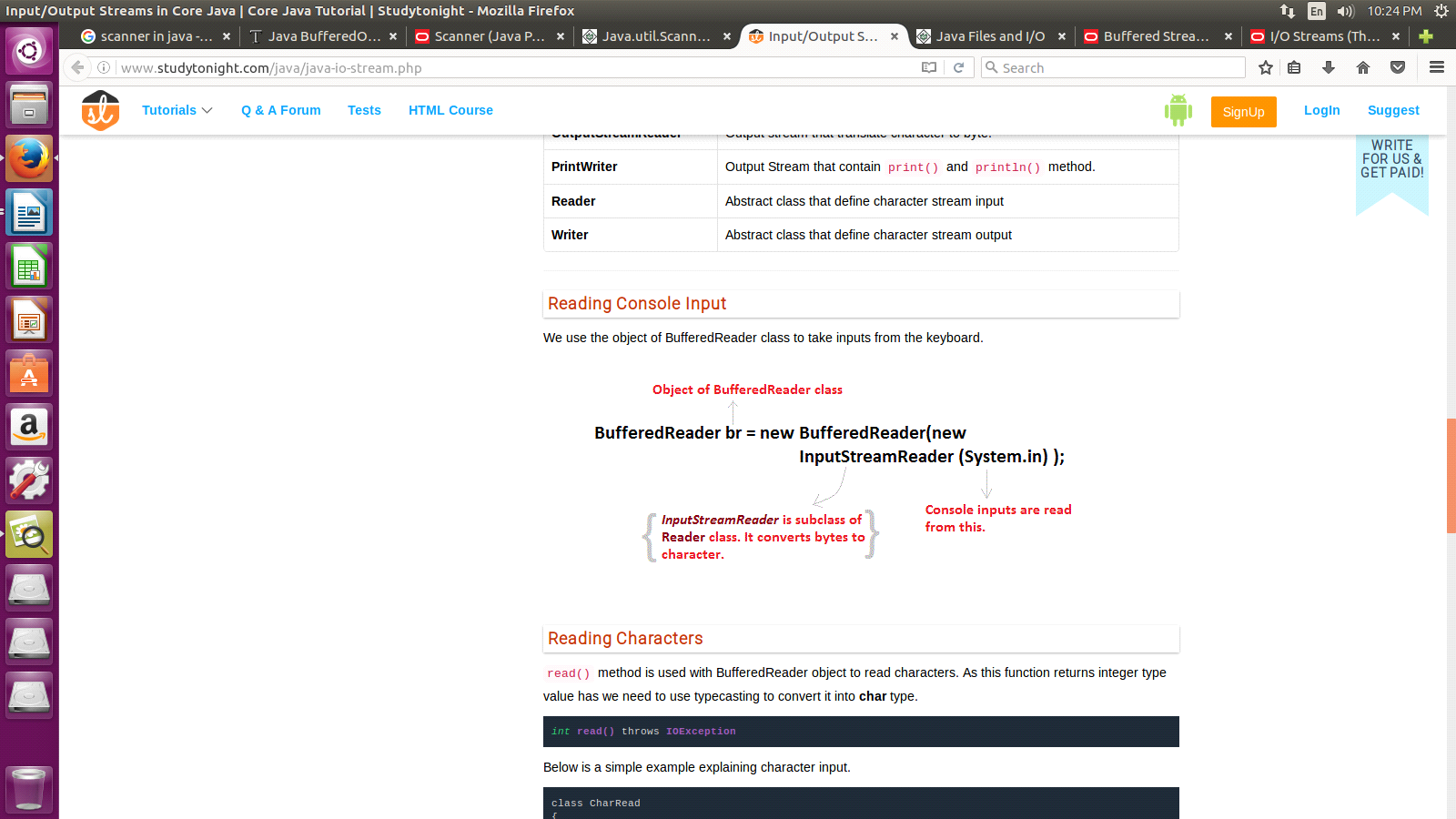
These two abstract classes have several concrete classes that handle unicode character.

**Some important Character stream classes.**

|  |  |
| --- | --- |
| **Stream class** | **Description** |
| **BufferedReader** | Handles buffered input stream. |
| **BufferedWriter** | Handles buffered output stream. |
| **FileReader** | Input stream that reads from file. |
| **FileWriter** | Output stream that writes to file. |
| **InputStreamReader** | Input stream that translate byte to character |
| **OutputStreamReader** | Output stream that translate character to byte. |
| **PrintWriter** | Output Stream that contain print()and println() method. |
| **Reader** | Abstract class that define character stream input |
| **Writer** | Abstract class that define character stream output |

**Reading Console Input**

We use the object of BufferedReader class to take inputs from the keyboard.



**Reading Input Characters**

To read a character from a BufferedReader we will use int read() throws IOException. Each time that read() is called, it reads a character from the input stream and returns it as an integer value. It returns -1 when the end of the stream is encountered. If there goes something wrong it can throw an IOException. The following program demonstrates read() by reading characters from the console until the user types a 'q'. Notice that any I/O exceptions that might be generated are simply thrown out of main(). Such an approach is common when reading from the console, but you can handle these types of errors yourself, if you chose.

// BufferedReader to read characters from the console.

import java.io.\*;

class BufferedReaderCharDemo

{

public static void main([String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) args[]) throws [IOException](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+ioexception)

{

char c;

[BufferedReader](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+bufferedreader) br = new [BufferedReader](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+bufferedreader)(new [InputStreamReader](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+inputstreamreader)([System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).in));

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println("Enter characters, press 'q' to quit.");

// read characters

do

{

c = (char) br.read();

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println(c);

} while(c != 'q');

}

}

**OUTPUT**

D:\JavaPrograms>javac BufferedReaderCharDemo.java

D:\JavaPrograms>java BufferedReaderCharDemo

Enter characters, press 'q' to quit.

krishanq

k

r

i

s

h

a

n

q

### Reading Input Strings

To read a string from the keyboard we use String readLine() throws IOException that is a member of the BufferedReader class. As you can see, it returns a String object. The following program demonstrates BufferedReader and the readLine() method; the program reads and displays lines of text until you enter the word "stop":

// Reading string from console.

import java.io.\*;

class BufferedReaderStringDemo

{

public static void main([String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) args[]) throws [IOException](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+ioexception)

{

// create a BufferedReader using System.in

[BufferedReader](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+bufferedreader) br = new [BufferedReader](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+bufferedreader)(new [InputStreamReader](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+inputstreamreader)([System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).in));

[String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string) str;

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println("Enter lines of text.");

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println("Enter 'stop' to quit.");

do

{

str = br.readLine();

[System](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+system).out.println(str);

} while(!str.equals("stop"));

}

}

**OUTPUT**

D:\JavaPrograms>javac BufferedReaderStringDemo.java

D:\JavaPrograms>java BufferedReaderStringDemo

Enter lines of text.

Enter 'stop' to quit.

This is a [String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string)

This is a [String](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string)

stop

stop

**JAVA SCANNER CLASS**

* There are various ways to read input from the keyboard, the **java.util.Scanner** class is one of them.
* The **Java Scanner** class breaks the input into tokens using a delimiter that is whitespace by default.
* It provides many methods to read and parse various primitive values.
* Java Scanner class is widely used to parse text for string and primitive types using regular expression.
* Java Scanner class extends Object class and implements Iterator and Closeable interfaces.

**Commonly used methods of Scanner class**

There is a list of commonly used Scanner class methods:

|  |  |
| --- | --- |
| **Method** | **Description** |
| public String next() | it returns the next token from the scanner. |
| public String nextLine() | it moves the scanner position to the next line and returns the value as a string. |
| public byte nextByte() | it scans the next token as a byte. |
| public short nextShort() | it scans the next token as a short value. |
| public int nextInt() | it scans the next token as an int value. |
| public long nextLong() | it scans the next token as a long value. |
| public float nextFloat() | it scans the next token as a float value. |
| public double nextDouble() | it scans the next token as a double value. |

**Java Scanner Example to get input from console**

Let's see the simple example of the Java Scanner class which reads the int, string and double value as an input:

import java.util.Scanner;

class ScannerTest

{

 public static void main(String args[])

{

   Scanner sc=new Scanner(System.in);

   System.out.println("Enter your rollno");

   int rollno=sc.nextInt();

   System.out.println("Enter your name");

   String name=sc.next();

   System.out.println("Enter your fee");

   double fee=sc.nextDouble();

   System.out.println("Rollno:"+rollno+" name:"+name+" fee:"+fee);

   sc.close();

 }

}

**Output:**

Enter your rollno

111

Enter your name

Ratan

Enter

450000

Rollno:111 name:Ratan fee:450000