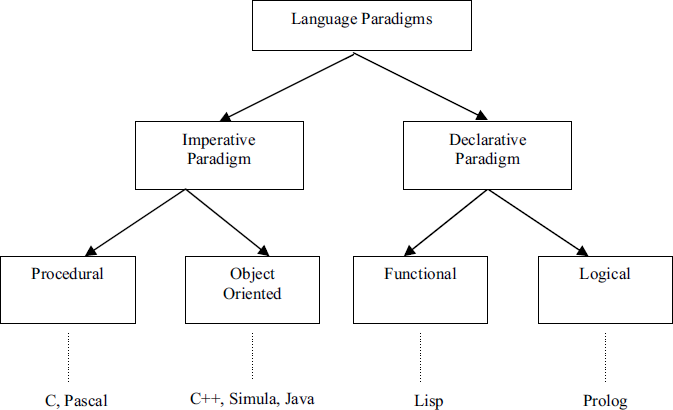
# Java–Unit 1 and Unit 2

# PARADIGMS OF PROGRAMMING LANGUAGES

1. The term paradigm describes a set of techniques, methods, theories and standards
2. Language paradigms were associated with classes of languages. First the paradigms are defined. Thereafter, programming languages according to the different paradigms are classified.
3. The language paradigms are divided into two parts,**imperative** and **declarative paradigms**
4. Imperative languages can be further classified into **procedural**and**object oriented**approach. Declarative languages can be classified into **functional languages** and **logical languages.**

****

1. **Imperative paradigm:**

**The meaning** of imperative is**“expressing a command or order”**,so the programming languages in this category specify the step-by-step explanation of command. Imperative programming languages describe the details of **how**the results are to be obtained. The programs specify step by step the entire set of transitions that the program goes through. The program starts from an initial state, goes through the transitions and reaches a final state. Within this paradigm we have the procedural approach and Object Oriented approach.

* + **Procedural paradigm:**Procedural languages are **statement oriented** with the variables holding values. We have two kinds of statements.**Non executable** statements allocate memory, bind symbolic names to absolute memory locations, and initialize memory.
  + **Executable** statements like computation, control flow, and input/output statements. The popular programming languages in this category are Ada, Fortran, Basic, Algol, Pascal, Cobol, Modula, C,etc.

the simulation of real world entities. The popular programming languages in this paradigm are C++, Simula, Smalltalk and Java.

1. **Declarative paradigm:**In this paradigm programs declare or specify what is to be computed without specifying how it is to be achieved. Declarative programming is also known as Value- oriented programming. Declarative languages describe the relationships between variables in terms of functions and inference rules. Declarative paradigm is further divided into two categories,**functional** and **logical paradigms.**
   * **Functional paradigm:**In this paradigm, a program consists of a **collection of functions**. A function just computes and returns a value. A program consists of calling a function with appropriate arguments, but any function can make use of other functions also. The main programming languages in this category are **Lisp, ML, Scheme**, and **Haskell.**

POP and OOP:-

|  |  |  |
| --- | --- | --- |
| **Subjectof**  **Difference** | **Procedure Oriented Programming**  **(POP)** | **Object Oriented Programming (OOP)** |
| **Problem**  **Decomposition** | Decompose the main problem in small  parts called functions. | Decompose the main problem in small  parts called objects. |
| **Connections of parts** | Connects small parts of the program by  passing parameters & using operating system. | Connects small parts of the program by passing messages. |
| **Emphasizing** | Emphasizes on functions. | Emphasizes on data. |
| **Use of data** | In large programs, most functions use  global data. | Each object controls data under it. |
| **Passing of data** | Data may get passed from one  function to another. | Data never get passed from one object  to another. |
| **Security of data** | Appropriate & effective techniques are  unavailable to secure the data. | Data stay secured as no external  function can use data of an object. |
| **Modification of program** | Modification of a completed program is very difficult and it may affect the  whole program. | Modifications are easy as objects stay independent to declare and define. |
| **Designing**  **Approach** | Employs top-down approach for  designing programs. | Employs bottom-up approach for  designing. |
| **Data identification** | In large programs, it is very difficult to  find what data has been used by which function. | As data and functions stay close, it is easy to identify data. |
| **Used languages** | Languages like C, FORTRAN, COBOL  etc. use POP. | Languages like C++, JAVA etc. use OOP. |

**Object**

**Object** means a real word entity such as pen, chair, table etc.**Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies the software development and maintenance by providing some concepts:

* Object
* Class
* Inheritance
* Polymorphism
* Abstraction
* Encapsulation

Any entity that has state and behavior is known as an object. For example: chair, pen, table, keyboard, bike etc. It can be physical and logical. Object is the physical as well as logical entity whereas class is the logical entity only.

An entity that has state and behavior is known as an object e.g. chair, bike, marker, pen, table, car etc. It can be physical or logical (tangible and intangible). The example of intangible object is banking system. An object has three characteristics:

* **state:**represents data (value) of anobject.
* **behavior:**represents the behavior (functionality) of an object such as deposit, withdrawetc.
* **identity:Object** identity is typically implemented via a unique ID.The value of the ID is not visible to the external user. But, it is used internally by the JVM to identify each object uniquely.

For Example: Pen is an object. Its name is Reynolds, color is white etc. known as its state. It is used to write, so writing is its behavior.

**Object is an instance of a class.**

Class is a template or blueprint from which objects are created. So object is the instance(result) of a class.

# Object Definitions:

* Object is a real world entity.
* Object is a run time entity.
* Object is an entity which has state and behavior.
* Object is an instance of a class.

# Class

**Collection of objects** is called class. It is a logical entity.

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

A class in Java can contain:

# fields

* **methods**
* **constructors**
* **blocks**
* **nested class and interface Syntax to declare a class:**

1. class<class\_name>{
2. field;
3. method;4.}

JAVA was developed by Sun Microsystems Inc in 1991, later acquired by Oracle Corporation. It was developed by James Gosling and Patrick Naughton. It is a simple programming language. Writing, compiling and debugging a program is easy in java. It helps to create modular programs and reusable code.

# Java terminology:-

1. **Java Virtual Machine(JVM)**

This is generally referred as JVM. Before, we discuss about JVM lets see the phases of program execution. Phases are as follows: we write the program, then we compile the program and at last we run the program.

* 1. Writing of the program is of course done by java programmer like you andme.
  2. In third phase, JVM executes the byte code generated by compiler. So, now that we understood that the primary function of JVM is to execute the byte code produced by compiler.**Each operating system has different JVM, however the output they produce after execution of byte code is same across all operating systems**. That is why we call java as platform independent language.

# bytecode

As discussed above, javac compiler of JDK compiles the java source code into bytecode so that it can be executed by JVM. The bytecode is saved in a .class file by compiler.

# Java DevelopmentKit(JDK).

As the name suggests this is complete java development kit that includes JRE (Java Runtime Environment), compilers and various tools like Java Doc, Java debugger etc.

In order to create, compile and run Java program you would need JDK installed on your computer.

# Java RuntimeEnvironment(JRE)

JRE is a part of JDK which means that JDK includes JRE. When you have JRE installed on yoursystem,youcan run a java program however you won’t be able to compile it.JRE includes JVM, browser plugins and applets support. When you only need to run a java program on your computer, you would only need JRE.

# Main Features of JAVA

**Java is a platform independent language**

Compiler (javac) converts source code (.java file) to the byte code(.class file). As mentioned above, JVM executes the bytecode produced by compiler. This byte code can run on any platform such as Windows, Linux, Mac OS etc. Which means a program that is compiled on windows can run on Linux and vice-versa. Each operating system has different JVM, however the output they produce after execution of bytecode is same across all operating systems. That is why we call java as platform independent language.

# Simple

Java is considered as one of simple language because it does not have complex features like Operator overloading, Multiple inheritance, pointers and Explicit memory allocation.

# Robust Language

Robust means reliable. Java programming language is developed in a way that puts a lot of importance. The main features of java that makes it robust are garbage collection, Exception Handling and memory allocation.

# Secure

We don’t have pointers and we cannot access out of bound arrays

# Java is distributed

Using java programming language we can create distributed applications. RMI(Remote Method Invocation) and EJB(Enterprise Java Beans) are used for creating distributed applications in java. In simple words: The java programs can be distributed on more than one systems that are connected to each other using internet connection. Objects on one JVM (java virtual machine) can execute procedures on a remote JVM.

# Multithreading

Java supports multithreading. Multithreading is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of CPU.

# Portable

As discussed above,java code that is written on one machine can run on another machine. The platform independent byte code can be carried to any platform for execution that makes java code portable.

# OOPs Features:-

Java is an object oriented language because it provides the features to implement an object oriented model. These features includes**Abstraction**,**encapsulation**,**inheritance** and **polymorphism**.

**OOPS** is about developing an application around its data, i.e. objects which provides the access to their properties and the possible operations in their own way.

4 main concepts of Object Oriented programming are:

1. Abstraction
2. Encapsulation
3. Inheritance
4. Polymorphism

# Abstraction

One of the most fundamental concept of OOPs is **Abstraction**.

Another example of abstraction: A car in itself is a well-defined object, which is composed of several other smaller objects like a gearing system, steering mechanism, engine, which are again have their own subsystems. But for humans car is a one single object, which can be managed by the help of its subsystems, even if their inner details are unknown.

# Encapsulation

Encapsulation is:

* Binding the data with the code that manipulates it.
* It keeps the data and the code safe from external interference

Looking at the example of a power steering mechanism of a car. Power steering of a car is a complex system, which internally have lots of components tightly coupled together, they work synchronously to turn the car in the desired direction. It even controls the power delivered by the engine to the steering wheel. But to the external world there is only one interface is available and rest of the complexity is hidden. Moreover, the steering unit in itself is complete and independent. It does not affect the functioning of any other mechanism.

Similarly, same concept of encapsulation can be applied to code. Encapsulated code should have following characteristics:

* Everyone knows how to access it.
* Can be easily used regardless of implementation details.
* There shouldn’t any side effects of the code, to the rest of the application.

# Inheritance

* Inheritance is the mechanism by which an object acquires the some/all properties of an other object
* It supports the concept of hierarchical classification.

For example: Car is a four wheeler vehicle so assume that we have a class FourWheeler and a sub class of it named Car. Here Car acquires the properties of a class FourWheeler. Other classifications could be a jeep, tempo, van etc. FourWheeler defines a class of vehicles that have four wheels, and specific range of engine power, load carrying capacity etc. Car (termed as a sub-class) acquires these propertiesfrom FourWheeler, and has some specific properties, which are different from other classifications of FourWheeler, such as luxury, comfort, shape, size, usage etc.

A car can have further classification such as an open car, small car, big car etc, which will acquire the properties from both Four Wheeler and Car, but will still have some specific properties. This way the level of hierarchy can be extended to any level.

Java Swing and [Awt](https://beginnersbook.com/2015/06/java-awt-tutorial/) classes represent best examples for inheritance.

# Polymorphism

* Polymorphism means to process objects differently based on their data type.

Lets us look at same example of a car. A car have a gear transmission system. It has four front gears and one backward gear. When the engine is accelerated then depending upon which gear is engaged different amount power and movement is delivered to the car. The action is same applying gear but based on the type of gear the action behaves differently or you can say that it shows many forms (polymorphism means many forms)

Polymorphism could be static and dynamic both. Method Overloading is static polymorphism while, Method overriding is dynamic polymorphism.

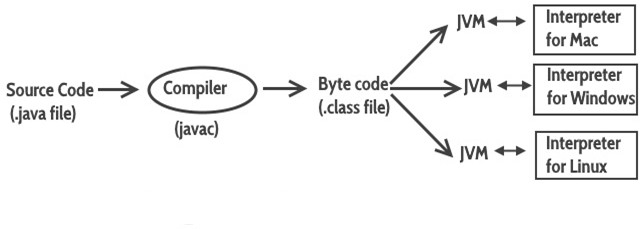
* **Overloading** in simple words means more than one method having the same method
* **Overriding** means a derived class is implementing a method of its super class. The call to overriden method is resolved at runtime, thus called runtime polymorphism

# Java Virtual Machine (JVM), Difference JDK, JRE & JVM–Core Java

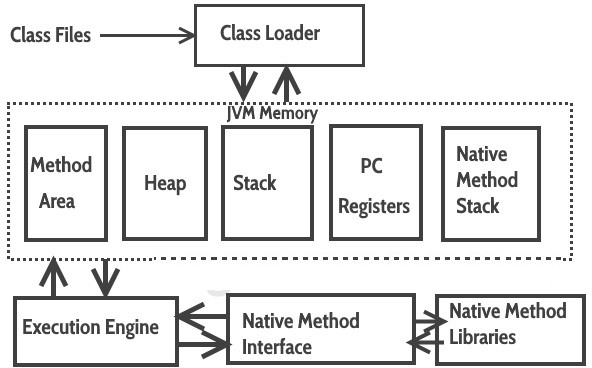
Java is a high level programming language. A program written in high level language cannot be run on any machine directly. First, it needs to be translated into that particular machine language. The **javac compiler**does this thing, it takes java program (.java file containing source code) and translates it into machine code (referred as byte code or .class file).

Java Virtual Machine (JVM) is a virtual machine that resides in the real machine (your computer) and the **machine language for JVM is byte code**. This makes it easier for compiler as it has to generate byte code for JVM rather than different machine code for each type of machine. JVM executes the byte code generated by compiler and produce output.**JVM is the one that makes java platform independent**.

So, now we understood that the primary function of JVM is to execute the byte code produced by compiler.**Each operating system has different JVM, however the output they produce after execution of byte code is same across all operating systems.**Which means that the byte code generated on Windows can be run on Mac OS and vice versa. That is why we call java as platform independent language. The same thing can be seen in the diagram below:



# JVM Architecture

****

**How JVW works**:

**Class Loader:**The class loader reads the .class file and save the byte code in the **method area**.

**Method Area**: There is only one method area in a JVM which is shared among all the classes. This holds the class level information of each .class file.

**Heap**: Heap is a part of JVM memory where objects are allocated. JVM creates a Class object for each

.class file.

**Stack**: Stack is a also a part of JVM memory but unlike Heap, it is used for storing temporary variables.**PC Registers**: This keeps the track of which instruction has been executed and which one is going to be executed since instructions are executed by threads each thread has PC register .

**Native Method stack:**A native method can access the runtime data areas of the virtual machine.**Native Method interface**: It enables java code to call or be called by native applications. Native applications are programs that are specific to the hardware and OS of a system.

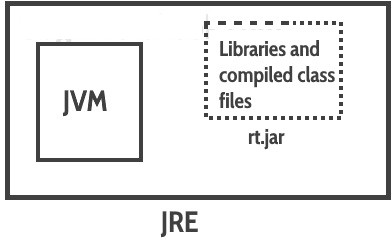
**Garbage collection**: A class instance is explicitly created by the java code and after use it is automatically destroyed by garbage collection for memory management.

**JVM Vs JRE Vs JDK**

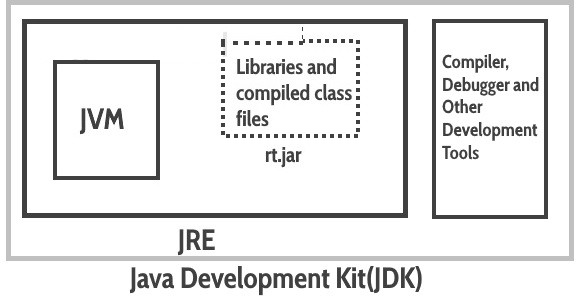
**JRE:**JRE is the environment within which the java virtual machine runs. JRE contains Java virtual Machine(JVM), class libraries, and other files excluding development tools such as compiler and debugger.

Which means you can run the code in JRE but you can’t develop and compile the code in JRE.

**JVM:**As we discussed above, JVM runs the program by using class, libraries and files provided by JRE.



**JDK:**JDK is a superset of JRE, it contains everything that JRE has along with development tools such as compiler, debugger etc.



# Benefits of OOPs:

OOP offers several benefits to both the program developer and the user. The new technology provides greater programmer productivity, better quality of software and lesser maintenance cost. The major benefits are:

1. **Ease in division of job:**Since it is possible to map objects of the problem domain to those objects in the program, the work can be easily partitioned based on objects.
2. **Reduce complexity:**Software complexity can be easily managed.
3. **Provide extensibility:**Object Oriented systems can be easily upgraded from small to large system.
4. **Eliminate redundancy:**Through inheritance we can eliminate redundant code and extend the use of existing classes.
5. **Saves development time and increases productivity:**Instead of writing code from scratch, solutions can be built by using standard working modules.
6. **Allows building secure programs:**Data hiding principle helps programmer to build secure programs that cannot be accessed by code in other parts of the program.
7. **Allows designing simpler interfaces:**Message passing techniques between objects allows making simpler interface descriptions with external systems.

# Applications of OOPs:

1. Object Oriented databases
2. Embedded systems
3. Simulation and modeling
4. Neural networks
5. Decision support systems
6. Office automation systems
7. AI and expert systems
8. CAD/CAM systems
9. Internet solutions.

# API:

An application programming interface (API), in the context of Java, is a collection of prewritten packages, classes, and interfaces with their respective methods, fields and constructors. Similar to a user interface, which facilitates interaction between humans and computers, an API serves as a software program interface facilitating interaction.

Java Development Kit (JDK) is comprised of three basic components, as follows:

* Java compiler
* Java Virtual Machine(JVM)
* Java Application Programming Interface(API)

The Java API, included with the JDK, describes the function of each of its components. In Java programming, many of these components are pre-created and commonly used. Thus, the programmer is able to apply prewritten code via the Java API. After referring to the available API classes and packages, the programmer easily invokes the necessary code classes and packages for implementation.

# Introduction to JAVA

There are many types of Java programs which run differently:

* **Java Applet-**small program written in Java and that is downloaded from a website and executed within a web browser on a client computer.
* **Application- executes on a client computer. If online, it has to be downloaded before being** run.
* **JAR file(Java ARchive)**- used to package Java files together into a single file (almost exactly like a

[.zip](https://simple.wikipedia.org/w/index.php?title=Zip_(file_format)&action=edit&redlink=1)file).

* **Servlet- runs on a**web server and helps to generate web pages.
* **Swing application- used to build an application that has a** GUI(windows, buttons, menus,etc.).
* **EJB- runs on a web server and is used to develop large, complex** web sites. How to Compile and Run your First Java ProgramSimple Java Program:

public class FirstJavaProgram {

public static void main(String[] args){ System.out.println("This is my first program in java");

}//End of main

}//End of FirstJavaProgram Class

**Output:**This is my first program in java

# How to compile and run the above program

**Prerequisite:You need to have java installed on your system.**

**Step 1:Open a text editor, like Notepad on windows and TextEdit on Mac. Copy the above program and paste it in the text editor.**

You can also use IDE like Eclipse to run the java program but we will cover that part later in the coming tutorials.

**Step 2:Save the file as FirstJavaProgram.java**You may be wondering why we have named the file as FirstJavaProgram, the thing is that we should always name the file same as the public class name. In our program, the public class name is FirstJavaProgram, that’s why our file name should be**FirstJavaProgram.java**.

**Step 3:In this step, we will compile the program. For this, open Terminal** in Linux To compile the program, type the following command and hit enter.

javac FirstJavaProgram.java

# Set path in Linux:

example: export PATH=$PATH:/home/jdk1.6.0/bin/

**Step 4:After compilation the .java file gets translated into the .class file(byte code). Now we can run the program. To run the program, type the following command and hit enter:**

java FirstJavaProgram

# Explanation to the First Java Programpublic class FirstJavaProgram {

This is the first line of our java program. Every java application must have at least one class definition that consists of class keyword followed by class name. When I say keyword, it means that it should not be changed, we should use it as it is. However the class name can be anything.

You need to know now that a java file can have any number of classes but it can have only one public class and the file name should be same as public class name.

# public static void main(String[] args) {

This is our next line in the program, lets break it down to understand it:

**public:**This makes the main method public that means that we can call the method from outside the class.**static:**We do not need to create object for static methods to run. They can run itself.

**void:**It does not return anything.

**main:**It is the method name. This is the entry point method from which the JVM can run your program.

**(String[] args):**Used for command line arguments that are passed as strings.

System.out.println("This is my first program in java");

This method prints the contents inside the double quotes into the console and inserts a newline after.

# Variable

A Java variable is a piece of memory that can contain a data value. A variable thus has a data type. Variables are typically used to store information which your Java program needs to do its job. This can be any kind of information ranging from texts, codes to numbers, temporary results of multi step calculations etc.

In the code example below, the main() method contains the declaration of a single integer variable named number. The value of the integer variable is first set to 10, and then 20 is added to the variable afterwards.

public class MyClass{

public static void main(String[] args) { int number =10;

number = number + 20;

}

}

# Java Variable Types

In Java there are four types of variables:

1. Non-staticfields
2. Staticfields
3. Localvariables
4. Parameters
   1. A non-static field is a variable that belongs to an object. Objects keep their internal state in non- staticfields.Non-staticfieldsarealsocalledinstancevariables,becausetheybelongtoinstances (objects) of a class. Non-static fields are covered in more detail in the text on Java fields.
   2. Astatic field is avariable that belongs to a class.A static field has the same value for all objects that access it. Static fields are also called class variables. Static fields are also covered in more detail in the text on Java fields.
   3. A local variable is a variable declared inside a method. A local variable is only accessible inside the method that declared it. Local variables are covered in more detail in the text on Java methods.
   4. A parameter is a variable that is passed to a method when the method is called. Parameters are also only accessible inside the method that declares them, although a value is assigned to them when the method is called. Parameters are also covered in more detail in the text on Java methods.Java Variable Declaration

Exactly how a variable is declared depends on what type of variable it is (non-static, static, local, parameter). However, there are certain similarities that

In Java you declare a variable like this:

type name ;

Instead of the word type, you write the data type of the variable. Similarly, instead of the word name you write the name you want the variable to have.

Here is an example declaring a variable named myVariable of type int. int myVariable;

# Java Variable Assignment

Assigning a value to a variable in Java follows this pattern:

variableName = value ;

Here are three concrete examples which assign values to three different variables with different data types

myInt = 127;

myFloat = 199.99;

myString = "This is a text";

# Java Variable Naming Conventions

There are a few rules and conventions related to the naming of variables. The rules are:

1. Java variable names are case sensitive. The variable name money is not the same as Moneyor MONEY.
2. Java variable names must start with a letter, or the $ or \_ character.
3. After the first character in a Java variable name, the name can also contain numbers

Variable names cannot be equal to reserved key words in Java Here are a few valid Java **variable name examples:** myvar

myVar

\_myVar

$myVar myVar1 myVar\_1

**Data Types:**

**Data type** defines the values that a variable can take, for example if a variable has int data type, it can only take integer values.

In java we have two categories of data type:

1) Primitive data types 2) Non-primitive data types–Arrays and Strings are non-primitive data types.

# Primitive data types

In Java, we have eight primitive data types: boolean, char, byte, short, int, long, float and double. Java developers included these data types to maintain the portability of java as the size of these primitive data types do not change from one operating system to another.

* + **byte**,**short**,**int** and **long**data types are used for storing whole numbers.
  + **Float** and **double** are used for fractional numbers.
  + **Char** is used for storing characters(letters).
  + **Boolean** data type is used for variables that holds either true or false.

# byte:

This can hold whole number between -128 and 127. Mostly used to save memory and when you are certain that the numbers would be in the limit specified by byte data type.

**Default size of this data type**: 1 byte.

# Default value: 0

**Example:**

class JavaExample{

public static void main(String[] args) { bytenum;

num = 113; System.out.println(num);

}

}

Output: 113

Try the same program by assigning value assigning 150 value to variable num, you would get **type mismatch error because**thevalue150 is out of the range of byte data type .The range of byte as I mentioned above is -128 to127.

# short:

This is greater than byte in terms of size and less than integer. Its range is -32,768 to 32767.

**Default size of this data type**: 2 byte

**short num**= 45678;

**int**:Used when short is not large enough to hold the number, it has a wider range: -2,147,483,648 to 2,147,483,647

**Default size:**4 byte**Default value:**0 Example:

class JavaExample {

public static void main(String[] args) { short num;

num = 150; System.out.println(num);

}

}

Output: 150

# long:

Used when int is not large enough to hold the value, it has wider range than int data type, ranging from - 9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.

**size:**8 bytes

**Default value:**0 Example:

class JavaExample {

public static void main(String[] args) { long num = -12332252626L; System.out.println(num);

}

}

Output:

-12332252626

# double:

Sufficient for holding 15 decimal digits

**size**: 8 bytes Example:

class JavaExample {

public static void main(String[] args) {

double num = -42937737.9d; System.out.println(num);

}}

Output:

-4.29377379E7

# float:

Sufficient for holding 6 to 7 decimal digits

**size**: 4 bytes

class JavaExample {

public static void main(String[] args) { float num = 19.98f; System.out.println(num);

}

}

Output: 19.98

# boolean:

Holds either true of false.

class JavaExample {

public static void main(String[] args) {

}

}

Output: false

boolean b = false; System.out.println(b);

# char:

Holds characters.

**Size**: 2 bytes

class JavaExample {

public static void main(String[] args) {

char ch = 'Z'; System.out.println(ch);

}

}

Output: Z

# Literals in Java

A literal is a fixed value that we assign to a variable in a Program. int num=10;

Here value 10 is a Integer literal. char ch = 'A';

Here A is a char literal

# Integer Literal

Integer literals are assigned to the variables of data type byte, short, int and long. byte b = 100;

short s = 200;

int num = 13313131; long l = 928389283L;

# Float Literals

Used for data type float and double. double num1 = 22.4;

float num2 = 22.4f;

Note: Always suffix float value with the “f” else compiler will consider it as double.

# Char and String Literal

Used for char and String type. char ch = 'Z';

String str = "BeginnersBook";

# Java Keywords

Keywords are predefined, reserved words used in Java programming that have special meanings to the compiler.

For example: int score;

Here, int is a keyword. It indicates that the variable score is of integer type.

You cannot use keywords like int, for, class etc as variable name (or identifiers) as they are part of the Java programming language syntax. Here's the complete list of all keywords in Java programming.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Java Keywords List** | | | | |
| abstract | assert | boolean | break | byte |
| case | catch | Char | class | const |
| continue | default | Do | double | else |
| enum | extends | Final | finally | float |
| for | goto | If | implements | import |
| instanceof | int | interface | long | native |
| new | package | Private | protected | public |
| return | short | Static | strictfp | super |
| switch | synchronized | This | throw | throws |
| transient | try | Void | volatile | while |

# Java identifiers

Identifiers are the name given to variables, classes, methods etc. Example: int score;

Here, score is a variable (an identifier). You cannot use keywords as variable name. It's because keywords have predefined meaning.

# Rules for Naming an Identifier

* + Identifier cannot be akeyword.
  + Identifiers are case-sensitive.
  + It can have a sequence of letters and digits. However, it must begin with a letter, $ or \_. The first letter of an identifier cannot be a digit.
  + It's convention to start an identifier with a letter rather and $ or\_.
  + Whitespaces are not allowed.
  + Similarly, you cannot use symbols such as @, #, and soon.

# Java Command Line Arguments

The java command-line argument is an argument i.e. passed at the time of running the java program. The arguments passed from the console can be received in the java program and it can be used as an input. So, it provides a convenient way to check the behavior of the program for the different values.

# Example:

In this example, we are receiving only one argument and printing it. To run this java program, you must pass at least one argument from the command prompt.

.

class CommandLineExample{

public static void main(String args[]){ System.out.println("Your first argument is: "+args[0]);

}

}

compile by > javac CommandLineExample.java run by > java CommandLineExample sonoo

2)to print all

public class CommandLine {

public static void main(String args[]) {

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

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

}

}

}

**Compile** :javac filename.java

**Run :**java CommandLine this is rgukt

Output: args[0]: this

args[1]: is

args[2]: rgukt

Basic Examples:

**Java Program to read integer value from the Standard Input**

In this program we will see how to read an integer number entered by user. Scanner class is in java.util package. It is used for capturing the input of the primitive types like int, double etc. and strings.

# Program to read the number entered by userWe have imported the package java.util.Scanner to use the Scanner. In order to read the input provided by user, we first create the object of Scanner by passing System.in as parameter. Then we are using nextInt() method of Scanner class to read the integer.

import java.util.Scanner; public class Demo {

public static void main(String[] args) { Scanner scan = new Scanner(System.in); System.out.print("Enter any number: ");

// This method reads the number provided using keyboard int num = scan.nextInt();

// Closing Scanner after the use scan.close();

// Displaying the number

System.out.println("The number entered by user: "+num);

}

}

# Output:

Enter any number: 101

The number entered by user: 101

# Program to check whether the given number is positive or negative

In this program we have specified the value of number during declaration and the program checks whether the specified number is positive or negative.

public class Demo

{

public static void main(String[] args)

{

int number=109; if(number > 0)

{

System.out.println(number+" is a positive number");

}

else if(number < 0)

{

System.out.println(number+" is a negative number");

}

else

{

System.out.println(number+" is neither positive nor negative");}

}

}

# Output:

109 is a positive number

# Example 2: Check whether the input number(entered by user) is positive or negative

Here we are using Scanner to read the number entered by user and then the program checks and displays the result.

import java.util.Scanner; public class Demo

{

public static void main(String[] args)

{

int number;

Scanner scan = new Scanner(System.in); System.out.print("Enter the number you want to check:"); number = scan.nextInt();

scan.close(); if(number > 0)

{

System.out.println(number+" is positive number");

}

else if(number < 0)

{

System.out.println(number+" is negative number");

}

else

{

}

# Output:

System.out.println(number+" is neither positive nor negative");

}

}

Enter the number you want to check:-12

-12 is negative number

# Sum of two numbers

public class AddTwoNumbers {

public static void main(String[] args) { int num1 = 5, num2 = 15, sum;

sum = num1 + num2;

Output:

System.out.println("Sum of these numbers: "+sum);

}

}

Sum of these numbers: 20

# Sum of two numbers using Scanner

The scanner allows us to capture the user input so that we can get the values of both the numbers from user. The program then calculates the sum and displays it.

import java.util.Scanner;

public class AddTwoNumbers2 {

public static void main(String[] args) { int num1, num2, sum;

Scanner sc = new Scanner(System.in); System.out.println("Enter First Number: "); num1 = sc.nextInt(); System.out.println("Enter Second Number:");num2 = sc.nextInt();

sc.close();

sum = num1 + num2;

System.out.println("Sum of these numbers: "+sum);

}

}

# Output:

Enter First Number: 121

Enter Second Number: 19

Sum of these numbers: 140

# Control Flow Statements:

The program is a set of statements, which are stored into a file. The interpreter executes these statements in a sequential manner, i.e., in the order in which they appear in the file. But there are situations where programmers want to alter this normal sequential flow of control. For example, if one wants to repeatedly execute a block of statements or one wants to conditionally execute statements.Therefore’ onemore category of statements called control flow statements is provided.

|  |  |
| --- | --- |
| **Statement** | **Type Keyword** |
| Selection | If-else, nested if , if-else-if, switch-case |
| Iteration | While, do-while, for |
| Jump | Break, continue, label;, return |
| Exception handling | Try-catch-finally, throw |

# Java if (if-then) Statement

The Java if statement tests the condition. It executes the*if block,*if condition is true.

# The syntax:

if (expression) {

// statements

}

# Flow chart:

**Example:**

class IfStatement {

public static void main(String[] args) { int number = 10;

if (number > 0) {

System.out.println("Number is positive.");

}

}

Output:

}

System.out.println("This statement is always executed.");

Number is positive.

# Java if...else Statement

The if statement executes a certain section of code if the test expression is evaluated to true. The if statement can have optional else statement. Codes inside the body of else statement are executed if the test expression is false.

# Syntax:

if (expression) {

// codes

}

else {

// some other code

}

# Flowchart:

**Example:**

class IfElse {

public static void main(String[] args) { int number = 10;

if (number > 0) { System.out.println("Number is positive.");

}

else {

System.out.println("Number is not positive.");

Output:}

System.out.println("This statement is always executed.");

}

}

Number is positive.

# Java IF-else-if ladder Statement

The if-else-if ladder statement executes one condition from multiple statements.

# Syntax:

if (expression1)

{

// codes

}

else if(expression2)

{

// codes

}

else if (expression3)

{

// codes

}

.

.

else

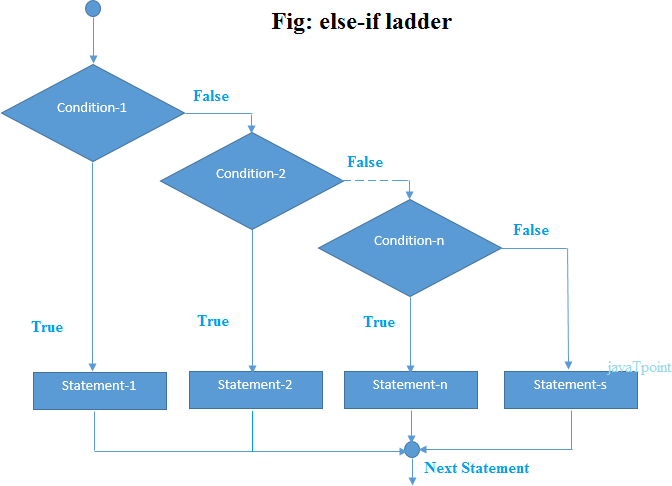
{

// codes

}

The if statements are executed from the top towards the bottom. As soon as the test expression is true, codes inside the body of that if statement is executed. Then, the control of program jumps outside if- else-if ladder. If all test expressions are false, codes inside the body of else is executed.

# Flowchart:



**Example:**

class Ladder {

public static void main(String[] args) { int number = 0;

if (number > 0) { System.out.println("Number is positive.");

}

else if (number < 0) { System.out.println("Number is negative.");

}

else {

System.out.println("Number is 0.");

}

}

}

# Output:

Number is 0.

A nested if is an if statement that is the target of another if statement. Nested if statements means an if statement inside another if statement

# Example:

class Number{

public static void main(String[] args) {

Double n1 = -1.0, n2 = 4.5, n3 = -5.3, largestNumber; if (n1 >= n2) {

if (n1 >= n3) { largestNumber = n1;

}

else {

largestNumber = n3;

}

}

else {

if (n2 >= n3) { largestNumber = n2;

}

else {

largestNumber = n3;

}

}

Output:

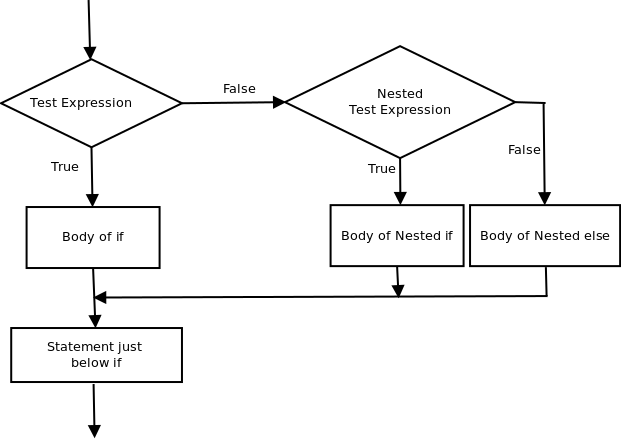
System.out.println("Largest number is " + largestNumber);

}

}

Largest number is 4.5

# Flow chart:



**Switch Statement:**

The Java *switch statement* executes one statement from multiple conditions. It is like if-else-if ladder statement.

# Syntax:

switch (variable/expression) { case value1:

// statements break;

case value2:

// statements break;

.. .....

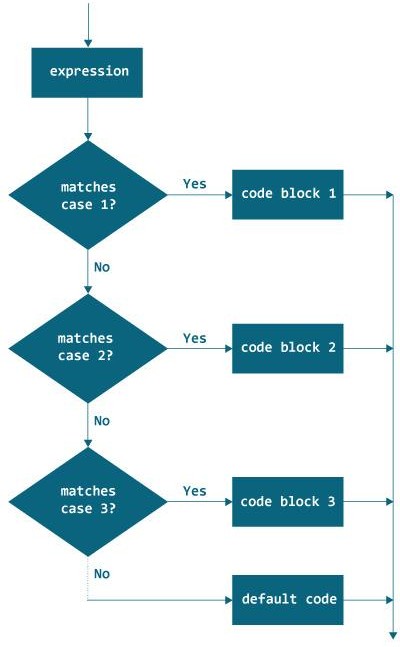
.. .....

default:

// statements

}

It's also important to note that switch statement in Java only works with:

* Primitive data types: byte, short, char andint
* *Enumerated types (Javaenums)*
* Stringclass
* a few classes that wrap primitive types: Character, Byte, Short, and Integer.Flowchart:

**Example1:**

class Day {

public static void main(String[] args) {

int week = 4; String day;

switch (week) { case 1:

day ="Sunday"; break;

case 2:

day ="Monday"; break;

case 3:

day ="Tuesday"; break;

case 4:

day = "Wednesday"; break;case 5:

day ="Thursday"; break;

case 6:

day = "Friday"; break;

case 7:

day="Saturday";

break;

default:

day = "Invalidday";break;

}

Output:

System.out.println(day);

}

}

Wednesday

# Example 2:

The program below takes three inputs from the user: operator and 2 numbers. It performs calculation based on numbers and operator entered. Then the result is displayed on the screen.

We have used Scanner object to take input from the user. import java.util.Scanner;

class Calculator {

public static void main(String[] args) { char operator;

int number1, number2, result; float div;

Scanner scanner = new Scanner(System.in); System.out.print("Enter operator (either +, -, \* or /): "); operator = scanner.next().charAt(0);

System.out.print("Enter number1 and number2 respectively:"); number1 =scanner.nextInt();

number2 =scanner.nextInt();

switch (operator) { case '+':

result = number1 + number2;

System.out.print(number1 + "+" + number2 + " = " + result); break;

case '-':

result = number1 - number2;

System.out.print(number1 + "-" + number2 + " = " + result); break;

case '\*':

result = number1 \* number2;

System.out.print(number1 + "\*" + number2 + " = " +result); break;

case '/':

div =(float) number1 / number2; System.out.print(number1 + "/" + number2 + " = " +result); break;

default:

System.out.println("Invalidoperator!"); break;

}

}

}

# Output:

Enter operator (either +, -, \* or /): \*

Enter number1 and number2 respectively: 1

-5

1\*-5 = -5

# 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. Here is a simple example:

# Example:

public class BreakDemo

{

public static void main(String[] args)

{

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

{

if (i == 5)

{

break; // terminate loop if i is5}

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

}

}

# Output :

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

}

1 2 3 4 Loop is over.

# continue statement

When a continue statement is encountered inside the body of a loop, remaining statements are skipped and loop proceeds with the next iteration. Here is a simple example.

# Example:

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 iseven

}

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

}

}

}

**Output:** 1 3 5 7 9

# Loops in Java:

Looping in programming languages is a feature which facilitates the execution of a set of instructions/functions repeatedly while some condition evaluates to true.

Java provides three ways for executing the loops. While all the ways provide similar basic functionality, they differ in their syntax and condition checking time.

They are:

1. while loop
2. for loop
3. do-whileloop

**while loop:**A while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition. The while loop can be thought of as a repeating if statement.

# Syntax :

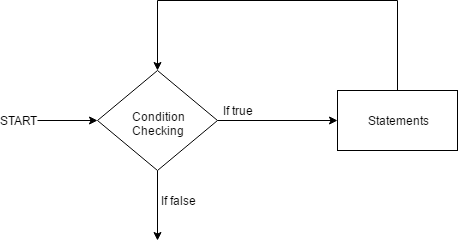
while (boolean condition)

{

loop statements...

}

# Flowchart:

****

* While loop starts with the checking of condition. If it evaluated to true, then the loop body statements are executed otherwise first statement following the loop is executed.
* When the condition becomes false, the loop terminates which marks the end of its life cycle. ExampleProgram:

// Java program to illustrate while loop class whileLoopDemo

{

public static void main(String args[])

{

int x = 1;

// Exit when x becomes greater than 4 while (x <= 4)

{

System.out.println("Value of x:" + x);

//increment the value of x for next iteration x++;

}

}

}

# Output:

Value of x:1 Value of x:2 Value of x:3 Value of x:4

# for loop:

for loop provides a concise way of writing the loop structure. Unlike a while loop, a for statement consumes the initialization, condition and increment/decrement in one line thereby providing a shorter, easy to debug structure of looping.

# Syntax:

for (initialization condition; testing condition; increment/decrement)

{

statement(s)

}

* **Initializationcondition:Here,**weinitializethevariableinuse.Itmarksthestartofaforloop.An already declared variable can be used or a variable can be declared, local to looponly.
* **Testing Condition:**It is used for testing the exit condition for a loop. It must return a Boolean value. It is also an**Entry Control Loop**as the condition is checked prior to the execution of the loop statements.
* **Statement execution:**Once the condition is evaluated to true, the statementsin the loop body areexecuted.
* **Increment/ Decrement:**It is used for updating the variable for nextiteration.
* **Loop termination:**When the condition becomes false, the loop terminates marking the end ofits life cycle.

# for-loop-in-javaFlowchart:

**Example:**

// Java program to illustrate for loop.

class forLoopDemo

{

public static void main(String args[])

{

# Output:

// for loop begins when x=2

// and runs till x <=4

for (int x = 2; x <= 4; x++) System.out.println("Value of x:" + x);

}

}

Value of x:2

Value of x:3

Value of x:4

# Enhanced For loop

Enhanced for loop provides a simpler way to iterate through the elements of a collection or array. It is inflexible and should be used only when there is a need to iterate through the elements in sequential manner without knowing the index of currently processed element.

# Syntax:

for (T element:Collection obj/array)

{

statement(s)

}

Example: Java program to illustrate enhanced for loop public class enhancedforloop

{

public static void main(String args[])

{

String array[] = {"IIIT", "RGUKT", "Basar"};

//enhanced for loop for (String x:array)

{

System.out.println(x);

}

**/\* for loop for same function**

**for (int i = 0; i < array.length; i++)**

**{**

**System.out.println(array[i]);**

**}**

**Output:**

**\*/**

}

}

IIIT

RGUKT

Basar

**do while:**do while loop is similar to while loop with only difference that it checks for condition after executing the statements, and therefore is an example of **Exit Control Loop.**

# Syntax:

do

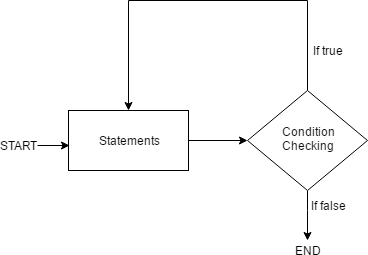
{

statements..

}

while (condition);

# Flowchart:

****

* do while loop starts with the execution of the statement(s). There is no checking of any condition for the first time.
* Aftertheexecutionofthestatements,andupdateofthevariablevalue,theconditionischecked for true or false value. If it is evaluated to true, next iteration of loopstarts.
* When the condition becomes false, the loop terminates which marks the end of its lifecycle.
* It is important to note that the do-while loop will execute its statements atleast once before any condition is checked, and therefore is an example of exit control loop.

# Example: Java program to illustrate do-while loop

class dowhileloopDemo

{

public static void main(String args[])

{

int x = 21; do

{

System.out.println("Value of x:" + x); x++;

}

# Output:

while (x < 20);

}

}

Value of x: 21

## Java Type Casting

Type casting is when you assign a value of one primitive data type to another type.

In Java, there are two types of casting:

* **Widening Casting** (automatically) - converting a smaller type to a larger type size  
  byte -> short -> char -> int -> long -> float -> double
* **Narrowing Casting** (manually) - converting a larger type to a smaller size type  
  double -> float -> long -> int -> char -> short -> byte

## Widening Casting

Widening casting is done automatically when passing a smaller size type to a larger size type:

public class Main {

public static void main(String[] args) {

int myInt = 9;

double myDouble = myInt; // Automatic casting: int to double

System.out.println(myInt); // Outputs 9

System.out.println(myDouble); // Outputs 9.0

}

}

## Narrowing Casting

Narrowing casting must be done manually by placing the type in parentheses in front of the value:

public class Main {

public static void main(String[] args) {

double myDouble = 9.78d;

int myInt = (int) myDouble; // Manual casting: double to int

System.out.println(myDouble); // Outputs 9.78

System.out.println(myInt); // Outputs 9

}

}

**Type conversion from int to String**

class Main {

public static void main(String[] args) {

// create int type variable

int num = 10;

System.out.println("The integer value is: " + num);

// converts int to string type

String data = String.valueOf(num);

System.out.println("The string value is: " + data);

}

}

**Type conversion from String to int**

# class Main {

# public static void main(String[] args) {

# // create string type variable

# String data = "10";

# System.out.println("The string value is: " + data);

# // convert string variable to int

# int num = Integer.parseInt(data);

# System.out.println("The integer value is: " + num);

# }

# }

# Arrays

An *array* is a collection of variables of the same type, referred to by a common name. In Java,

arrays can have one or more dimensions, although the one-dimensional array is the most common. Arrays are used for a variety of purposes because they offer a convenient means of grouping together related variables.

For example, you might use an array to hold a record of the daily high temperature for a month, a list of stock price averages, or a list of your collection of programming books.

The principal advantage of an array is that it organizes data in such away that it can be easily manipulated. For example, if you have an array containing the incomes for a selected group of households, it is easy to compute the average income by cycling through the array. In Java Arrays are implemented as objects.

# One-DimensionalArrays

A one-dimensional array is a list of related variables. For example, you might use a one- dimensional array to store the account numbers of the active users on a network. Another array might be used to store the current batting averages for a cricket team.

Declaration of one Dimensional Arrays:

*type array-name*[ ] = new*type*[*size*];

1. Here,*type* declares the element type of the array.
2. The element type determines the data type of each element contained in the array.
3. The number of elements that the array will hold is determined by *size.*
4. Since arrays are implemented as objects, the creation of an array is a two-step process. First, you declare an array reference variable. Second, you allocate memory for the array, assigning a reference to that memory to the array variable.
5. Thus, arrays in Java are dynamically allocated using the **new operator.**

Here is an example. The following creates an **int** array of 10 elements and links it to an array reference variable named **sample**:

int sample[] = new int[10];

This declaration works just like an object declaration. The **sample** variable holds a reference to the memory allocated by **new**. This memory is large enough to hold 10 elements of type **int**. As with objects, it is possible to break the preceding declaration in two. For example:

int sample[];

sample = new int[10];

In this case, when **sample** is first created, it refers to no physical object. It is only after the second statement executes that **sample** is linked with an array.

An individual element within an array is accessed by use of an index. An *index* describes the position of an element within an array. In Java, all arrays have zero as the index of their first element. Because **sample** has 10 elements, it has index values of 0 through 9. To index an array, specify the number of the element you want, surrounded by square brackets. Thus, the first element in **sample** is **sample[0]**, and the last element is **sample[9]**.

# Example:-

public class Arrays1 {

public static void main(String args[]) { int sample[] = new int[10];

int i;

for(i = 0; i < 10; i = i+1) sample[i] = i;

for(i = 0; i < 10; i = i+1)

System.out.println("This is sample[" + i + "]: " + sample[i]);

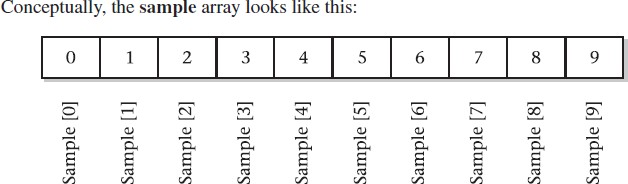
}

}

# Output:

The output from the program is shown here: This is sample[0]:0

This is sample[1]: 1 This is sample[2]: 2 This is sample[3]: 3 This is sample[4]: 4 This is sample[5]: 5 This is sample[6]: 6 This is sample[7]: 7 This is sample[8]: 8 This is sample[9]:9



# Accessing Java Array Elements using for Loop

Each element in the array is accessed via its index. The index begins with 0 and ends at (total array size)-

1. All the elements of array can be accessed using Java forLoop.

// accessing the elements of the specified array for (int i = 0; i < arr.length; i++)

System.out.println("Element at index " + i +" : "+ arr[i]);

# Example:Java Program to find Min and Max element in user given array.

import java.util.Scanner; class MinMax {

public static void main(String args[]) { Scanner scan=new Scanner(System.in);

System.out.println(“EnterSizeoftheArray:“);intlen=scan.nextInt();

int nums[] = new int[len];

int min,max;

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

{

System.out.print("Enter"+i+"th element: "); nums[i]=scan.nextInt();

}

min=max=nums[0];

for(int i=0;i<len;i++){

if(min>nums[i])

min=nums[i]; else

max=nums[i];

}

System.out.println("Min element is: "+min+"\nMax element is: "+max);

}

}

# Multidimensional arrays

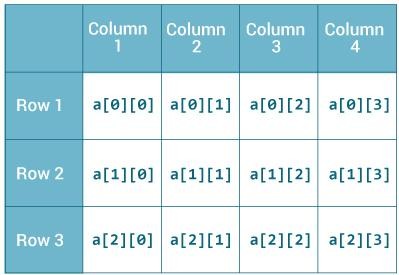
Multidimensional arrays are**arrays of arrays**with each element of the array holding the reference of other array. A multidimensional array is created by appending one set of square brackets ([]) per dimension.

# Examples:

int[][] intArray = new int[10][20]; //a 2D array or matrix int[][][] intArray = new int[10][20][10]; //a 3D array

For example,

int[][] a = new int[3][4];

Here, a is a two-dimensional (2d) array. The array can hold maximum of 12 elements of type int.

Similarly, you can declare a three-dimensional (3d) array. For example, String[][][] personalInfo = new String[3][4][2];

Here,*personalInfo*is a 3d array that can hold maximum of 24 (3\*4\*2) elements of type String.

class MultidimensionalArray {

public static void main(String[] args) {

int[][] a = {

{1, 2, 3},

{4, 5, 6, 9},

{7},

};

System.out.println("Length of row 1: " +a[0].length); System.out.println("Length of row 2: " +a[1].length); System.out.println("Length of row 3: " +a[2].length);

}

}

When you run the program, the output will be:

Length of row 1:3

Length of row 2:4

Length of row 3:1

Since each component of a multidimensional array is also an array (a[0], a[1] and a[2] are also arrays), you can use length attribute to find the length of each rows.

# Example: Print all elements of 2d array Using Loop

class MultidimensionalArray {

public static void main(String[] args) {

int[][] a = {

{1, -2, 3},

{-4, -5, 6, 9},

{7},

};

for (int i = 0; i < a.length; ++i) { for(int j = 0; j < a[i].length; ++j) {

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

}

}

}

}

# Example: Multiply two Matrices

import java.util.Scanner; public class JavaProgram

{

public static void main(String args[])

{

int m, n, p, q, sum = 0, c, d, k; Scanner in = newScanner(System.in);

System.out.print("Enter Number of Rows and Columns of First Matrix : "); m = in.nextInt();

n = in.nextInt();

int first[][] = new int[m][n]; System.out.print("Enter First Matrix Elements : "); for(c=0 ; c<m; c++)

{

for(d=0; d<n; d++)

{

first[c][d] = in.nextInt();

}

}

System.out.print("Enter Number of Rows and Columns of Second Matrix :"); p =in.nextInt();

q =in.nextInt();

if ( n != p )

{

System.out.print("Matrix of the entered order can't be Multiplied..!!");

}

else

{

int second[][] = new int[p][q];

int multiply[][] = new int[m][q]; System.out.print("Enter Second Matrix Elements :\n");

for(c=0; c<p; c++)

{

for(d=0; d<q; d++)

{

second[c][d] = in.nextInt();

}

}

System.out.print("Multiplying both Matrix...\n");

for(c=0; c<m; c++)

{

for(d=0; d<q; d++)

{

for(k=0; k<p; k++)

{

sum = sum + first[c][k]\*second[k][d];

}

multiply[c][d] = sum; sum = 0;

}

}

System.out.print("Multiplication Successfully performed..!!\n"); System.out.print("Now the Matrix Multiplication Result is :\n");

for(c=0; c<m; c++)

{

for(d=0; d<q; d++)

{

System.out.print(multiply[c][d] + "\t");

}

System.out.print("\n");

}

}

}

}

# VECTORS

* 1. Vector class is in java.util package ofjava.
  2. Vector is dynamic array which can grow automatically according to the required need.Vector does not require any fix dimension like String array and int array.
  3. The Vector class implements a growable array of objects. Like an array, it contains components that can be accessed using an integer index. However, the size of a Vector can grow or shrink as needed to accommodate adding and r removing items after the Vector has been created.
  4. Vectors (the java.util.Vector class) are commonly used instead of arrays
  5. Vectors can hold only Objects and not primitive types (eg, int). If you want to put a primitive type in a Vector, put it inside an object (eg, to save an integer value use the Integer class or define your own class). If you use the Integer wrapper, you will not be able to change the integer value, so it is sometimes useful to define your own class.

# To Create a Vector:

Following are the list of constructors provided by the vector class.

# Create a Vector with default initial size

Vector v = new Vector(); //it creates a vector v with initial capacity 10.

1. **Create a Vector with an initial size**//it creates a vector v with initial size.

Vector v=new Vector(initial capacity);

# Create a Vector with an initial size with incremental value

//it creates a vector v with initial size and incremental value

Vector v = new Vector(int initialCapacity,int incrementalCapacity);

# Common Vector Methods

There are many methods in the Vector class and its parent classes. v is a Vector, i is an int index, o is an Object.

|  |  |
| --- | --- |
| *Method* | *Description* |
| v.add(o) | adds Object o to Vector v |
| v.add(i, o) | Inserts Object o at index i, shifting elements up as necessary. |
| v.clear() | removes all elements from Vector v |
| v.contains(o) | Returns true if Vector v contains Object o |
| v.firstElement(i) | Returns the first element. |
| v.get(i) | Returns the object at int index i. |
| v.lastElement(i) | Returns the last element. |
| v.listIterator() | Returns a ListIterator that can be used to go over the Vector. This is a useful  alternative to the for loop. |
| v.remove(i) | Removes the element at position i, and shifts all following elements down. |
| v.set(i,o) | Sets the element at index i to o. |
| v.size() | Returns the number of elements in Vector v. |
| v.toArray(Object[]) | The array parameter can be any Object subclass (eg, String). This returns the vector values in that array (or a larger array if necessary). This is useful when you need the generality of a Vector for input, but need the speed of arrays when processing the  data. |

# Differences between a Vector and an Array

1. A vector is a dynamic array, whose size can be increased, where as an array size can notbe changed.
2. Reserve space can be given for vector, where as for arrays can not.
3. A vector is a class where as an array is not.
4. Vectors can store any type of objects, where as an array can store only homogeneous values.

# 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. Example:-

import java.util.\*; public class Vector1 {

public static void main(String args[]){

Vector v=new Vector();

// Vector v1=new Vector();

//Vector<Integer> v=new Vector<Integer>(); for integer elements

//Vector<String> v=new Vector<String>(); for String elements v.add(7);//adding element

v.add('a');

v.add("RGUKT");

v.add(3, 4); //adding element at index(index,element)

//add at index method moves the already present at that index and added current data

v.set(3, "RGUKT1");//set the element at some index

// set method replaces the actual index information

//v=v1.clone();

boolean b=v.contains("RGUKT");//searching for element="RGUKT" System.out.println(v.indexOf(7));//to know the index of element=7 System.out.println(v.get(2));//to get the element at index=2 System.out.println(v.elementAt(2));// same as get(2); System.out.println(b);// printing vector System.out.println(v.firstElement());//print first element System.out.println(v.lastElement()); // printing last element

System.out.println("size= "+v.size()); // to get the size i.e element present in vector

//System.out.println(v.Capacity()); gives the capacity of vector

// printing vector Using for loop for(int i=0;i<v.size();i++){

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

// Using Iterator class

//Iterator iter = v.iterator(); // creating object

//while (iter.hasNext()){ // calling hasneNext method

//System.out.println(iter.next()); // calling next method to get each element

//}

//Printing Using Enumeration class Enumeration enm=v.elements(); while(enm.hasMoreElements()){

System.out.print(""+enm.nextElement());

}

}

}

}

# Java WrapperClasses

The following two statements illustrate the difference between a primitive datatype and an object of a wrapper class:int x = 25;

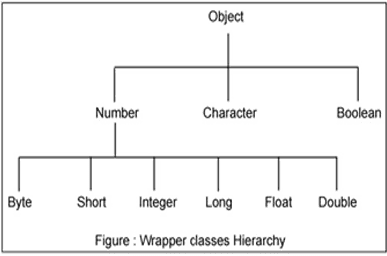
Integer y = new Integer(33);

The first statement declares an int variable named x and initializes it with the value 25. The second statement instantiates an Integer object. The object is initialized with the value 33 and a reference to the object is assigned to the object variable y.

Below table lists wrapper classes in Java API with constructor details.

|  |  |  |
| --- | --- | --- |
| **Primitive** | **Wrapper Class** | **Constructor Argument** |
| boolean | Boolean | boolean or String |
| byte | Byte | byte or String |
| char | Character | char |
| Int | Integer | int or String |
| float | Float | float, double or String |
| double | Double | double or String |
| long | Long | long or String |
| short | Short | short or String |

Below is wrapper class hierarchy as per Java API



As explain in above table all wrapper classes (except Character) take String as argument constructor. Please note we might get NumberFormatException if we try to assign invalid argument in the constructor. For example to create Integer object we can have the following syntax.

Integer intObj = new Integer (25); Integer intObj2 = new Integer ("25");

Here in we can provide any number as string argument but not the words etc. Below statement will throw run time exception (NumberFormatException)

# Example:

/ Java program to demonstrate Wrapping and UnWrapping

// in Java Classes class Wrapper\_Class

{

public static void main(String args[])

{

// byte data type byte a = 1;

// wrapping around Byte object Byte byteobj = new Byte(a);

// int data type int b = 10;

//wrapping around Integer object Integer intobj = new Integer(b);

// float data type float c = 18.6f;

// wrapping around Float object Float floatobj = new Float(c);

// double data type double d = 250.5;

// Wrapping around Double object Double doubleobj = new Double(d);

// char data type char e='a';

// wrapping around Character object Character charobj=e;

// printing the values from objects

System.out.println("Values of Wrapper objects (printing as objects)"); System.out.println("Byte object byteobj: " + byteobj); System.out.println("Integer object intobj: " + intobj); System.out.println("Float object floatobj: " + floatobj); System.out.println("Double object doubleobj: " + doubleobj); System.out.println("Character object charobj: " + charobj);

// objects to data types (retrieving data types from objects)

// unwrapping objects to primitive data types byte bv =byteobj;

int iv = intobj; float fv =floatobj;

double dv = doubleobj; char cv =charobj;

// printing the values from data types 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); System.out.println("char value, cv: " + cv);}

}