Core Java Notes

1.History

The history of java starts from Green Team. Java team members (also known as Green Team), initiated a revolutionary task to develop a language for digital devices such as set-top boxes, televisions etc.For the green team members, it was an advance concept at that time. But, it was suited for internet programming. Later, Java technology as incorporated by Netscape.

James Gosling - founder of java

Currently, Java is used in internet programming, mobile devices, games, e-business solutions etc. There are given the major points that describes the history of java.

1) James Gosling, Mike Sheridan, and Patrick Naughton initiated the Java language project in June 1991. The small team of sun engineers called Green Team.

2) Originally designed for small, embedded systems in electronic appliances like set-top boxes.

3) Firstly, it was called "Greentalk" by James Gosling and file extension was .gt.

4) After that, it was called Oak and was developed as a part of the Green project.

5)In 1995, Oak was renamed as "Java" because it was already a trademark by Oak Technologies.

6)Originally developed by James Gosling at Sun Microsystems (which is now a subsidiary of Oracle Corporation) and released in 1995.

7)JDK 1.0 released in(January 23, 1996).

Java Version History

There are many java versions that has been released. Current stable release of Java is Java SE 8.

JDK Alpha and Beta (1995)

JDK 1.0 (23rd Jan, 1996)

JDK 1.1 (19th Feb, 1997)

J2SE 1.2 (8th Dec, 1998)

J2SE 1.3 (8th May, 2000)

J2SE 1.4 (6th Feb, 2002)

J2SE 5.0 (30th Sep, 2004)

Java SE 6 (11th Dec, 2006)

Java SE 7 (28th July, 2011)

Java SE 8 (18th March, 2014)

==========

2.Introduction

Java is a programming language and a platform.

Java is a high level, robust, secured and object-oriented programming language.

Platform: Any hardware or software environment in which a program runs, is known as a platform. Since Java has its own runtime environment (JRE) and API, it is called platform.

Java Example

```
class Simple{

    public static void main(String args[]){

      System.out.println("Hello Java");

        }

        }
```

==========

3.Types of Java Applications

There are mainly 4 types of applications that can be created using java programming:

1) Standalone Application

It is also known as desktop application or window-based application. An application that we need to install on every machine such as media player, antivirus etc. AWT and Swing are used in java for creating standalone applications.

2) Web Application

An application that runs on the server side and creates dynamic page, is called web application. Currently, servlet, jsp, struts, jsf etc. technologies are used for creating web applications in java.

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

An application that is created for mobile devices. Currently Android and Java ME are used for creating mobile applications.

Java Platforms / Editions

There are 4 platforms or editions of Java:

1) Java SE (Java Standard Edition)

It is a java programming platform. It includes Java programming APIs such as java.lang, java.io, java.net, java.util, java.sql, java.math etc. It includes core topics like OOPs, String, Regex, Exception, Inner classes, Multithreading, I/O Stream, Networking, AWT, Swing, Reflection, Collection etc.

2) Java EE (Java Enterprise Edition)

It is an enterprise platform which is mainly used to develop web and enterprise applications. It is built on the top of Java SE platform. It includes topics like Servlet, JSP, Web Services, EJB, JPA etc.

3) Java ME (Java Micro Edition)

It is a micro platform which is mainly used to develop mobile applications.

4) JavaFx

It is used to develop rich internet applications. It uses light-weight user interface API.

==========

4.Where it is used?

According to Sun, 3 billion devices run java. There are many devices where Java is currently used. Some of them are as follows:

1.Desktop Applications such as acrobat reader, media player, antivirus etc.

2.Web Applications such as irctc.co.in, javatpoint.com etc.

3.Enterprise Applications such as banking applications.

4.Mobile

5.Embedded System

6.Smart Card

7.Robotics

8.Games etc.

==========5.

5.Features of Java

There is given many features of java. They are also known as java buzzwords. The Java Features given below are simple and easy to understand.

Java Features

Simple

Object-Oriented

Portable

Platform independent

Secured

Robust

Architecture neutral

Dynamic

Interpreted

High Performance

Multithreaded

Distributed

1.Simple

According to Sun, Java language is simple because:

syntax is based on C++ (so easier for programmers to learn it after C++).

removed many confusing and/or rarely-used features e.g., explicit pointers, operator overloading etc.

No need to remove unreferenced objects because there is Automatic Garbage Collection in java.

2.Object-oriented

Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behaviour.

Object-oriented programming(OOPs) is a methodology that simplify software development and maintenance by providing some rules.

Basic concepts of OOPs are:

Object

Class

Inheritance

Polymorphism

Abstraction

Encapsulation

3.**Platform Independent**

java is platform independent

A platform is the hardware or software environment in which a program runs.

There are two types of platforms software-based and hardware-based. Java provides software-based platform.

The Java platform differs from most other platforms in the sense that it is a software-based platform that runs on the top of other hardware-based platforms. It has two components:

Runtime Environment

API(Application Programming Interface)

Java code can be run on multiple platforms e.g. Windows, Linux, Sun Solaris, Mac/OS etc. Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform-independent code because it can be run on multiple platforms i.e. Write Once and Run Anywhere(WORA).

4.Secured

Java is secured because:

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

5. Robust

Robust simply means strong. Java uses strong memory management. There are lack of pointers that avoids security problem. There is automatic garbage collection in java. There is exception handling and type checking mechanism in java. All these points makes java robust.

6. Architecture-neutral

There is no implementation dependent features e.g. size of primitive types is fixed.

In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. But in java, it occupies 4 bytes of memory for both 32 and 64 bit architectures.

7. Portable

We may carry the java bytecode to any platform.

8. High-performance

Java is faster than traditional interpretation since byte code is "close" to native code still somewhat slower than a compiled language (e.g., C++)

9. Distributed

We can create distributed applications in java. RMI and EJB are used for creating distributed applications. We may access files by calling the methods from any machine on the internet.

10. Multi-threaded

A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for

multi-media, Web applications etc.

==========

6. Simple Program of Java

In this page, we will learn how to write the simple program of java. We can write a simple hello java program easily after installing the JDK.

To create a simple java program, you need to create a class that contains main method. Let's understand the requirement first.

Requirement for Hello Java Example

For executing any java program, you need to
install the JDK if you don't have installed it, download the JDK and install it.
set path of the jdk/bin directory.
create the java program
compile and run the java program
Creating hello java example

Let's create the hello java program:

```
class Simple{
      public static void main(String args[]){
        System.out.println("Hello Java");
      }
}
```

==========

7.Difference between JDK, JRE and JVM

Understanding the difference between JDK, JRE and JVM is important in Java. We are having brief overview of JVM here.

If you want to get the detailed knowledge of Java Virtual Machine, move to the next page. Firstly, let's see the basic differences between the JDK, JRE and JVM.

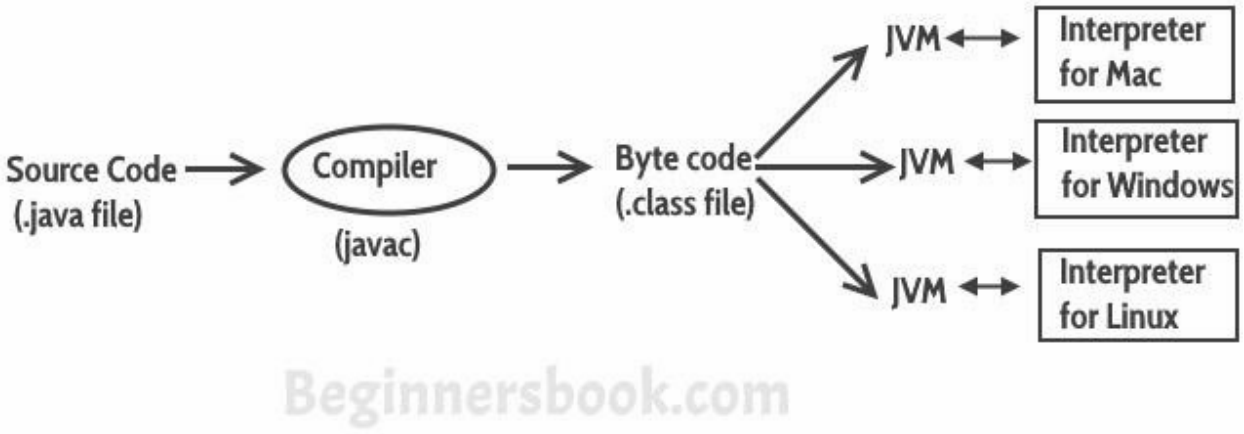
JAVA DEVELOPMENT KIT(JDK) :. As the name suggests this is complete java development kit that includes JRE (Java Runtime Environment), compilers and various tools like JavaDoc, Java debugger etc. In order to create, compile and run Java program you would need JDK installed on your computer

JAVA RUNTIME ENVIRONMENT (JRE) : JRE is a part of JDK which means that JDK includes JRE. When you have JRE installed on your system, you can 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.

JVM (JAVA VIRTUAL MACHINE) : 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).

The Java Virtual machine (JVM) is the virtual machine that runs on actual machine (your computer) and executes Java byte code. The JVM doesn't understand Java source code, that's why we need to have javac compiler that compiles *.java files to obtain *.class files that contain the byte codes understood by the JVM. JVM makes java portable (write once, run anywhere). Each operating system has different JVM, however the output they produce after execution of byte code is same across all operating systems.



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.

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.

CLASSPATH

Classpath is a parameter in the Java Virtual Machine or the Java compiler that specifies the location of user-defined classes and packages. The parameter may be set either on the command- line, or through an environment variable.

Install JDK →C:\Program Files\Java\jdk1.8.0_71\bin (Copy this path) → Rightclick My computer → Advanced system settings → Environment variables → Paste the copied path there.

VARIABLES

A variable is the name given to a memory location. It is the basic unit of storage in a program.

- The value stored in a variable can be changed during program execution.
- A variable is only a name given to a memory location, all the operations done on the variable effects that memory location.
- In Java, all the variables must be declared before they can be used.

How to declare variables?

We can declare variables in java as follows:

datatype : Type of data that can be stored in this variable.

variable_name : Name given to the variable.

value : It is the initial value stored in the variable.

***Examples**:*

float simpleInterest; //Declaring float variable

int time = 10, speed = 20; //Declaring and Initializing integer variable char var =

'h'; // Declaring and Initializing character variable

- Variables naming cannot contain white spaces, for example: int num ber = 100; is invalid because the variable name has space in it.

- Variable name can begin with special characters such as $ and _

- As per the java coding standards the variable name should begin with a lower case letter,for example int number; For lengthy variables names that has more than one words do it like this: int smallNumber; int bigNumber; (start the second word with capital letter).

- Variable names are case sensitive in java

There are 3 types of variables in java

1. Static (or class) variable
2. Instance variable
3. Local variable

STATIC (OR CLASS) VARIABLE : Static variables are also known as Class variables.

- These variables are declared similarly as instance variables, the difference is that static variables are declared using the static keyword within a class outside any method constructor or block.
- Unlike instance variables, we can only have one copy of a static variable per class irrespective of how many objects we create.
- Static variables are created at start of program execution and destroyed automatically when execution ends.

To access static variables, we need not to create any object of that class, we can simply access the variable as:

```
class_name.variable_name;
```

Instance variable : Instance variables are non-static variables and are declared in a class outside any method, constructor or block.

- As instance variables are declared in a class, these variables are created when an object of the class is created and destroyed when the object is destroyed.
- Unlike local variables, we may use access specifiers for instance variables. If we do not specify any access specifier then the default access specifier will be used.

Local Variable : A variable defined within a block or method or constructor is called local variable.

- These variable are created when the block in entered or the function is called and destroyed after exiting from the block or when the call returns from the function.
- The scope of these variables exists only within the block in which the variable is declared. i.e. we can access these variable only within that block.

Example to understand the types of variables in java

```
class A{
int data=50;//instance variable
static int m=100;//static variable
void method(){
int n=90;//local variable
}
}//end of class
```

Datatypes

- **Integers:** This group includes byte, short, int, and long, which are for whole-valued signed numbers.
- **Floating-point numbers:** This group includes float and double, which represent numbers with fractional precision.
- **Characters:** This group includes char, which represents symbols in a character set, like letters and numbers.
- **Boolean:** This group includes boolean, which is a special type for representing true/false values.

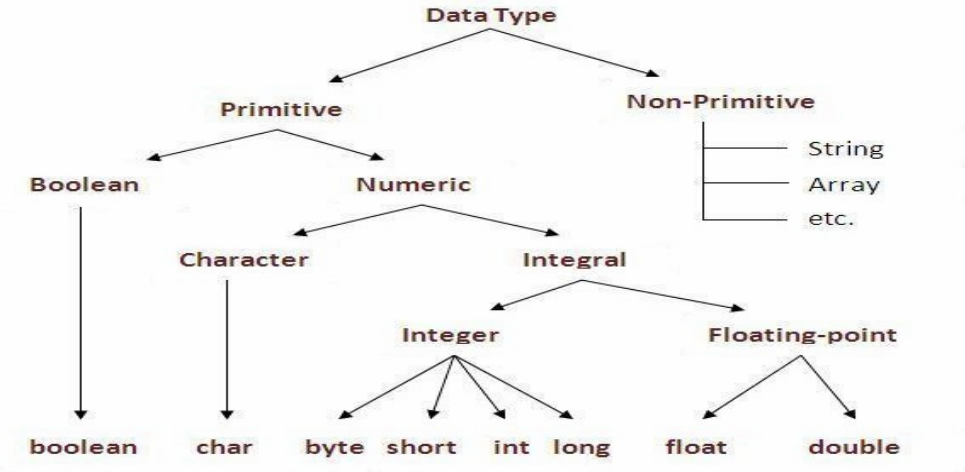


Table 1: List of Java's primitive data types

Type	Size in Bytes	Range
byte	1 byte	-128 to 127
short	2 bytes	-32,768 to 32,767
Int	4 bytes	-2,147,483,648 to 2,147,483, 647
long	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

float	4 bytes	approximately ±3.40282347E+38F (6-7 significant decimal digits) *Java implements IEEE 754 standard*
double	8 bytes	approximately ±1.79769313486231570E+308 (15 significant decimal digits)
char	2 byte	0 to 65,536 (unsigned)
boolean	not precisely defined*	true or false

Operators in java

Operator in java is a symbol that is used to perform operations. For example: +, -, *, / etc.

There are many types of operators in java which are given below:

Unary Operator,
Arithmetic Operator,
shift Operator,
Relational Operator,
Bitwise Operator,
Logical Operator,
Ternary Operator and
Assignment Operator.

Operator Type	Category	Precedence
Unary	postfix	*expr*++ *expr*--
	prefix	++*expr* --*expr* +*expr* -*expr* ~ !
Arithmetic	multiplicative	* / %
	additive	+ -
Shift	shift	<< >> >>>
Relational	comparison	< > <= >= instanceof
	equality	== !=
Bitwise	bitwise AND	&

	bitwise exclusive OR	^
	bitwise inclusive OR	|
Logical	logical AND	&&
	logical OR	||
Ternary	ternary	? :
Assignment	assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>

Java Unary Operator Example: ++ and --

```
class OperatorExample{
public static void main(String args[]){
int x=10;
System.out.println(x++);//10 (11)
System.out.println(++x);//12
System.out.println(x--);//12 (11)
System.out.println(--x);//10
}}
```

Output:

```
10
12
12
10
```

Java Unary Operator Example 2: ++ and --

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=10;
System.out.println(a++ + ++a);//10+12=22
System.out.println(b++ + b++);//10+11=21

}}
```

Output:

22
21

Java Arithmetic Operator Example

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
System.out.println(a+b);//15
System.out.println(a-b);//5
System.out.println(a*b);//50
System.out.println(a/b);//2
System.out.println(a%b);//0
}}
```

Output:

15
5
50
2
0

Java Arithmetic Operator Example: Expression

```
class OperatorExample{
public static void main(String args[]){
System.out.println(10*10/5+3-1*4/2);
}}
```

Output:
21

Java Assignment Operator Example

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int  b=20;
a+=4;//a=a+4 (a=10+4)
b-=4;//b=b-4 (b=20-4)
System.out.println(a);
System.out.println(b);
```

}}
Output:

14
16

Java Assignment Operator Example
class OperatorExample{
public static void main(String[] args){
int a=10;
a+=3;//10+3
System.out.println(a);
a-=4;//13-4
System.out.println(a);
a*=2;//9*2
System.out.println(a);
a/=2;//18/2
System.out.println(a);
}}
Output:

13
9
18
9

Java Assignment Operator Example: Adding short

16.Reading from console using scanner

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 bydefault. 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 a
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

Conditional and looping statements

Java If-else Statement

The Java if statement is used to test the condition. It checks boolean condition: true or false. There are various types of if statement in java.

1. if statement
2. if-else statement
3.nested if statement
4.if-else-if ladder

Java IF Statement

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

Syntax:

```
if(condition){
//code to be executed
}
```

Example:

```
public class IfExample {
public static void main(String[] args) {
      int age=20;
      if(age>18){
            System.out.print("Age is greater than 18");
      }
}
}
```

Output:

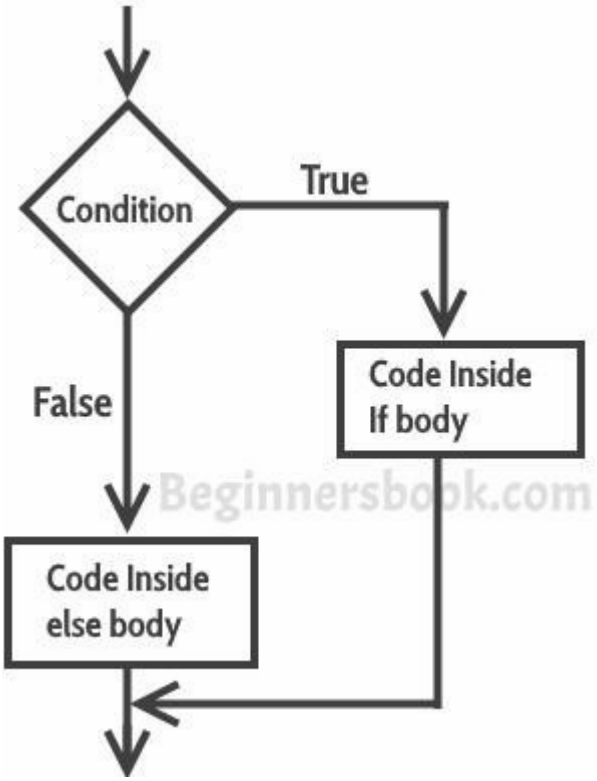
Age is greater than 18

Java IF-else Statement

The Java if-else statement also tests the condition. It executes the if block if condition is true otherwise else block is executed.

Syntax:

```
if(condition){
//code if condition is true
}else{
```




```
//code if condition is false
}
```

Example:

```
public class IfElseExample {
public static void main(String[] args) {
        int number=13;
        if(number%2==0){
                System.out.println("even number");
        }else{
                System.out.println("odd number");
        }
}
}
```

Output:

```
odd number
```

Java IF-else-if ladder Statement

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

Syntax:

```
if(condition1){
//code to be executed if condition1 is true
```

```
}else if(condition2){
//code to be executed if condition2 is true
}
else if(condition3){
//code to be executed if condition3 is true
}
...
else{
//code to be executed if all the conditions are false
}
```

Example:

```
public class IfElseIfExample {
public static void main(String[] args) {
        int marks=65;
```

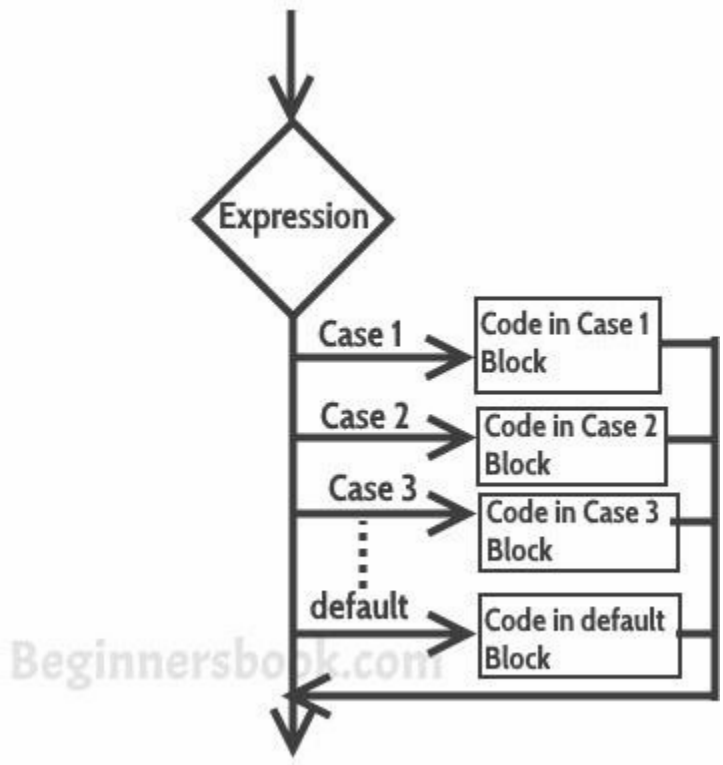
```
        if(marks<50){
            System.out.println("fail");
        }
        else if(marks>=50 && marks<60){
            System.out.println("D grade");
        }
        else if(marks>=60 && marks<70){
            System.out.println("C grade");
        }
        else if(marks>=70 && marks<80){
            System.out.println("B grade");
        }
        else if(marks>=80 && marks<90){
            System.out.println("A grade");
        }else if(marks>=90 && marks<100){
            System.out.println("A+ grade");
        }else{
            System.out.println("Invalid!");
        }
}
}
```

Output:

C grade

Java Switch Statement

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



Syntax:

```
switch(expression){
case value1:
  //code to be executed;
  break;    //optional
case value2:
  //code to be executed;
  break;    //optional
.....

default:
  code to be executed if all cases are not matched;
}
```

Example:

```
public class SwitchExample {
```

```
public static void main(String[] args) {
	int number=20;
	switch(number){
	case 10: System.out.println("10");break;
	case 20: System.out.println("20");break;
	case 30: System.out.println("30");break;
	default:System.out.println("Not in 10, 20 or 30");
	}
}
}
```

Output:

20

Java For Loop

The Java for loop is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop.

There are three types of for loop in java.

Simple For Loop

For-each or Enhanced For Loop

Labeled For Loop

Java Simple For Loop

The simple for loop is same as C/C++. We can initialize variable, check condition and increment/decrement value.

Syntax:

```
for(initialization;condition;incr/decr){

//code to be executed

}
```

Example:

```
public class ForExample {
public static void main(String[] args) {
	for(int i=1;i<=10;i++){
		System.out.println(i);
	}
```

```
}
}
```
Output:

```
1
2
3
4
5
6
7
8
9
10
```

Java For-each Loop

The for-each loop is used to traverse array or collection in java. It is easier to use than simple for loop because we don't need to increment value and use subscript notation.

It works on elements basis not index. It returns element one by one in the defined variable.

Syntax:

```
for(Type var:array){

//code to be executed

}
```

Example:

```
public class ForEachExample {
public static void main(String[] args) {
        int arr[]={12,23,44,56,78};
        for(int i:arr){
                System.out.println(i);
        }
}
}
```
Output:

```
12
23
44
56
78
```

Java While Loop

The Java while loop is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop.

Syntax:

```
while(condition){

//code to be executed

}
```

Example:

```
public class WhileExample {
public static void main(String[] args) {
        int i=1;
        while(i<=10){
                System.out.println(i);
        i++;
        }
}
}
```

Output:

```
1
2
3
4
5
6
7
8
9
10
```

Java Infinitive While Loop

If you pass true in the while loop, it will be infinitive while loop.

Syntax:

```
while(true){
//code to be executed
}
```

Example:

```
public class WhileExample2 {
public static void main(String[] args) {
	while(true){
		System.out.println("infinitive while loop");
	}
}
}
```

Output:

```
infinitive while loop
infinitive while loop
infinitive while loop
infinitive while loop
infinitive while loop
ctrl+c
```

Now, you need to press ctrl+c to exit from the program.

Java do-while Loop

The Java do-while loop is used to iterate a part of the program several times. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use do-while loop.

The Java do-while loop is executed at least once because condition is checked after loop body.

Syntax:

```
do{

//code to be executed

}while(condition);
```

Example:

```
public class DoWhileExample {
public static void main(String[] args) {
	int i=1;
	do{
		System.out.println(i);
	i++;
	}while(i<=10);
}
}
```

Output:

1
2
3
4
5
6
7
8
9
10

Java Infinitive do-while Loop

If you pass true in the do-while loop, it will be infinitive do-while loop.

Syntax:

```
do{
//code to be executed
}while(true);
```

Example:

```
public class DoWhileExample2 {
public static void main(String[] args) {
	do{
		System.out.println("infinitive do while loop");
	}while(true);
}
}
```

Output:

```
infinitive do while loop
infinitive do while loop
infinitive do while loop
```

Java Break Statement

The Java break is used to break loop or switch statement. It breaks the current flow of the program at specified condition. In case of inner loop, it breaks only inner loop.

Syntax:

```
jump-statement;

break;
```

Java Break Statement with Loop

Example:

```
public class BreakExample {
public static void main(String[] args) {
	for(int i=1;i<=10;i++){
		if(i==5){
			break;
		}
		System.out.println(i);
	}
}
}
```

Output:

```
1
2
3
4
```

Java Break Statement with Inner Loop

It breaks inner loop only if you use break statement inside the inner loop.

Example:

```
public class BreakExample2 {
public static void main(String[] args) {
		for(int i=1;i<=3;i++){
			for(int j=1;j<=3;j++){
				if(i==2&&j==2){
					break;
				}
				System.out.println(i+" "+j);
			}
		}
}
}
```

Output:

```
1 1
1 2
1 3
2 1
3 1
3 2
3 3
```

Java Continue Statement

The Java continue statement is used to continue loop. It continues the current flow of the program and skips the remaining code at specified condition. In case of inner loop, it continues only inner loop.

Syntax:

jump-statement;

continue;

Java Continue Statement Example

Example:

```
public class ContinueExample {
public static void main(String[] args) {
	for(int i=1;i<=10;i++){
		if(i==5){
			continue;
		}
		System.out.println(i);
	}
}
}
```

Output:

```
1
2
3
4
6
7
8
9
10
```

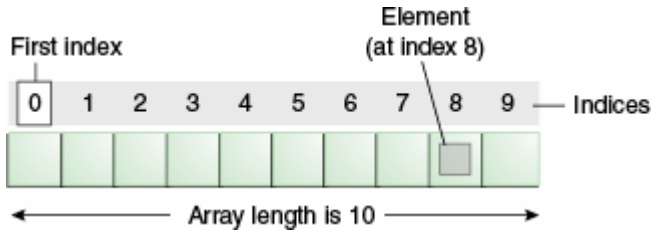
==========

Java Array

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

Java array is an object the 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.

Types of Array in java

There are two types of array.

1.Single Dimensional Array

2.Multidimensional Array

Single Dimensional Array in java

Syntax to Declare an Array in java

dataType[] arr; (or)

dataType arr[];

Instantiation of an Array in java

Datatype arrayname[]=new

datatype[size];

Example of single dimensional java array

Let's see the simple example of java array, where we are going to declare, instantiate, initialize and traverse an 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]);

}}
```

Test it Now

Output: 10
20
70
40
50

Declaration, Instantiation and Initialization of Java Array

We can declare, instantiate and initialize the java array together by:

```
int a[]={33,3,4,5};//declaration, instantiation and initialization
```

Let's see the simple example to print this array.

```
class Testarray1{
public static void main(String args[]){
int a[]={33,3,4,5};//declaration, instantiation and initialization
//printing array
for(int i=0;i<a.length;i++)//length is the property of array
System.out.println(a[i]);

}}
```

Test it Now

Output:33
3
4
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[];

Example to declare Multidimensional Array in java

Datatype arrayname[][] =new datatype [size][size];

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();
}

}}
```
Test it Now

Output:1 2 3

2 4 5

4 4 5

Addition of 2 matrices in java

Let's see a simple example that adds two matrices.

```
class Testarray5{
public static void main(String args[]){
//creating two matrices
int a[][]={{1,3,4},{3,4,5}};
int b[][]={{1,3,4},{3,4,5}};

//creating another matrix to store the sum of two matrices
int c[][]=new int[2][3];

//adding and printing addition of 2 matrices
for(int i=0;i<2;i++){
for(int j=0;j<3;j++){
c[i][j]=a[i][j]+b[i][j];
System.out.print(c[i][j]+" ");
}
System.out.println();//new line
}

}}
```
Test it Now

Output:2 6 8

6 8 10

==========