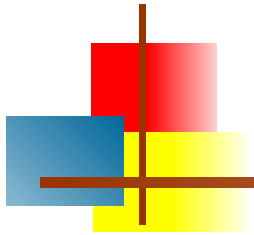


Java Programming

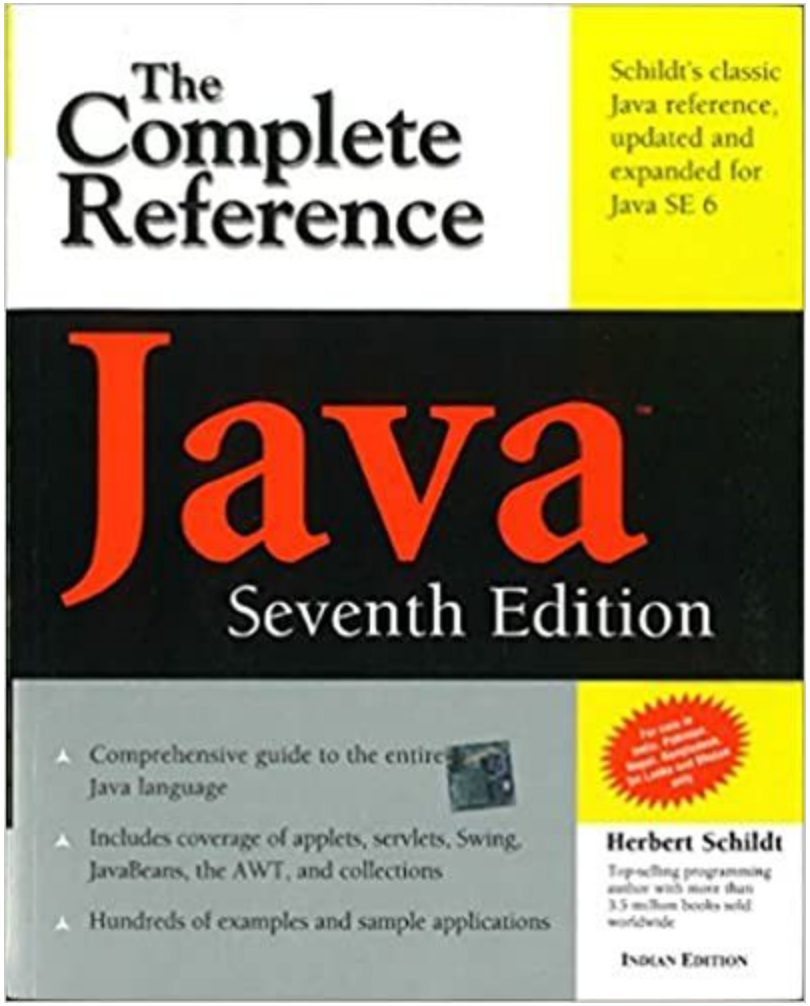
Basic syntactical Contents of Java Programming

Marks =10

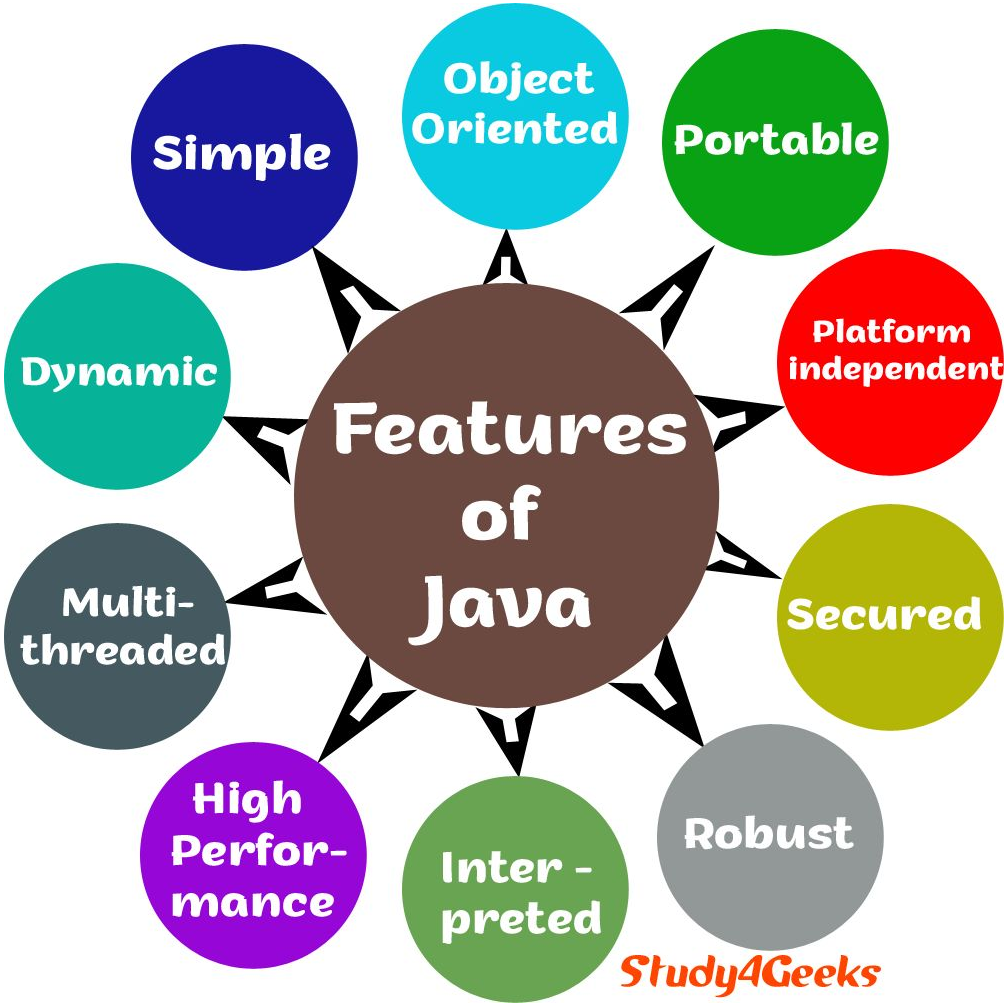
Books


Programming with
Java
3e
A Primer
DOWNLOAD NOW
E. BALAGURUSAMY

Reference Book to be used



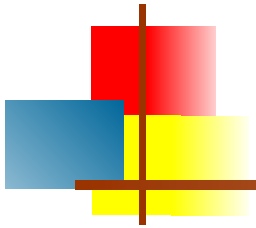
Features of java



Features of Java

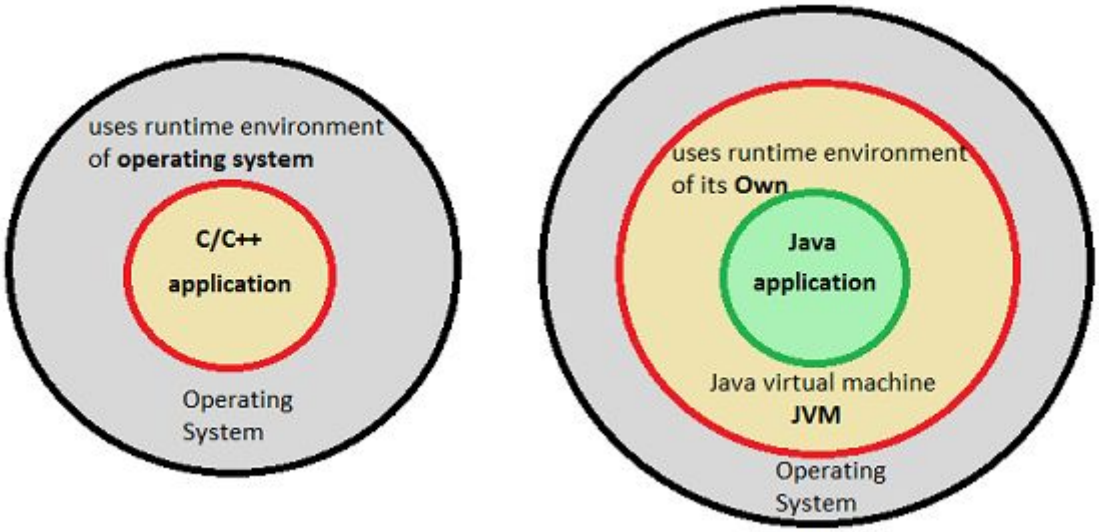
Features of Java

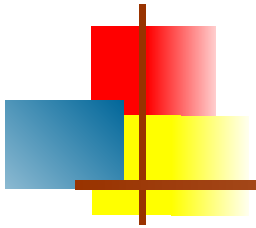
- Simple
- Secured
- Object-oriented
- Platform-independent
- Robust
- Portable
- Interpreted
- Multithreaded
- Architecture-neutral
- Distributed
- High-performance
- Dynamic
- Concurrent
- Class-based
- Network-savvy



- **Simple**
- As defined by Sun Microsystems,
- Java is simple language because it does not some potentially confusing features such as multiple-inheritance, pointers, operator overloading, and explicit memory allocation used in C and C++. There is an "**Automatic Garbage Collection**" in Java that automatically reclaims the objects and arrays i.e. no need to remove the unreferenced objects.
- And obviously, its syntax is based on C and C++ because it was initiated after C and C++. So it's easy to understand Java if one knows about either C or C++.

Secured


uses runtime environment
of **operating system**
C/C++
application
Operating
System
uses runtime environment
of its **Own**
Java
application
Java virtual machine
JVM
Operating
System



- Java programs runs within **Java Virtual Machine** (JVM).
- Omission of explicit pointer makes it robust.
- The class-loader adds the security by separating the package for the classes of the local file system especially from those which are imported from the network sources.
- The code fragments are checked by byte-code verifier for illegal codes which can violate access right to objects.
- The security manager determines what resources a class can access such as reading and writing to the local disk.

- Together these all features prevent viruses and other dangerous threats and make it secured.

JVM

3.9 Java Virtual Machine

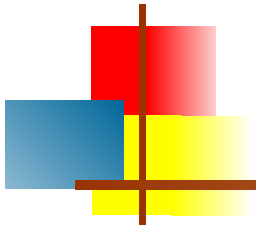
All language compilers translate source code into *machine code* for a specific computer. Java compiler also does the same thing. Then, how does Java achieve architecture neutrality? The answer is that the Java compiler produces an intermedia code known as *bytecode* for a machine that does not exist. This machine is called the *Java Virtual Machine* and it exists only inside the computer memory. It is a simulated computer within the computer and does all major functions of a real computer. Figure 3.6 illustrates the process of compiling a Java program into bytecode which is also referred to as *virtual machine code*.

Fig. 3.6 *Process of compilation*

The virtual machine code is not machine specific. The machine specific code (known as machine code) is generated by the Java interpreter by acting as an intermediary between the virtual machine and the real machine as shown in Fig. 3.7. Remember that the interpreter is different for different machines.

Fig. 3.7 *Process of converting bytecode into machine code*

Figure 3.8 illustrates how Java works on a typical computer. The Java object framework (Java API) acts as the intermediary between the user programs and the virtual machine which in turn acts as the intermediary between the operating system and the Java object framework.



Object Oriented

- There are few rules/concepts of object-oriented programming (oops) for development and maintenance of software.

- **Object:** Objects are a key to understand object-oriented programming. It is the physical (tangible) as well as logical (intangible) entity. An entity has state and behaviour is known as object. For example- bike, car, table, pen, etc.

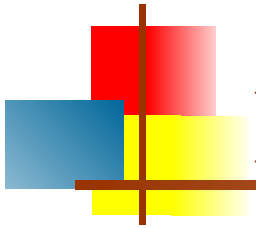
- **Class:** A class is a group of objects that has common properties and considered as a logical (intangible) entity. It is a template or blueprint from which object are created. For example- banking system.

- **Inheritance:**

- **Polymorphism:**

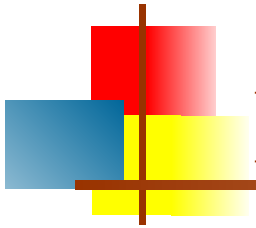
- **Abstraction:**

- **Encapsulation:**



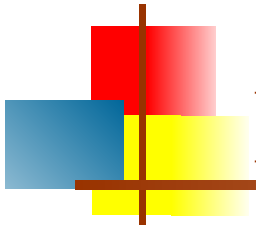
Platform-independent

- Java is a platform independent language.
- Run program on any operating system
- Run program on any hardware set up
- The byte code is distributed over network and interpreted by respective platform JVM on whichever it is being run. That is why it is called as platform independent.



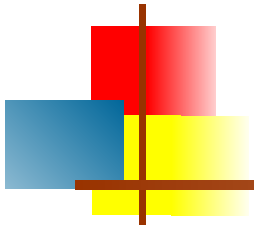
Robust

- Java language becomes robust (strong) by handling the following problems efficiently
- No Mistakes in memory management
- Mishandling of runtime errors can also cause failure of program. And this is handled by exception handling and type checking mechanism.
- Security problems can be avoided by lack of pointers.



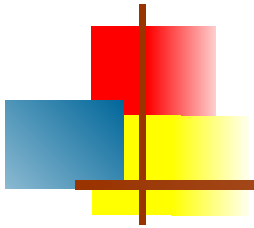
Portable & Interpreated

- As we know that, Java programs written on one platform can run on another platform.
- The Java program compiled into byte codes which are platform independent and can be carried to any platform for execution that makes Java a portable language.
- Interpreted
- Interpretation is the process of converting code to low level language line by line.
- The virtual machine translates the byte code (instructions and associated data) to platform specific instructions through interpretation and then executes those instructions.
- The development process is more rapid and analytical since the linking is an incremental and light weight process.



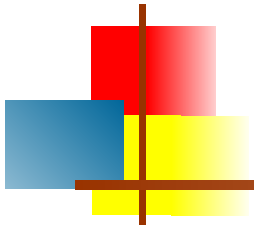
Distributed

- Java is basically designed for the distributed environment of the internet.
- We can access files by calling methods from any machine on the internet.
- Actually, Java programs can be distributed on more than one machine that is connected to each other through internet connection.
- Objects on one machine (JVM) can execute procedure of a remote machine (JVM).
- We can create distributed applications in Java by using Enterprise Java Beans (EJB) and Remote Method Invocation (RMI).



Multithreaded

- A thread is like a separate program, executing concurrently.
- With this feature of Java, it is possible to write programs that can do many tasks simultaneously by defining multiple threads.
- The main advantage of multi-threading is that it shares the same memory.
- This design feature allows developers to construct smoothly running interactive applications such as multi-media, web apps, etc. Multi-threading also contributes to high-performance.



High performance

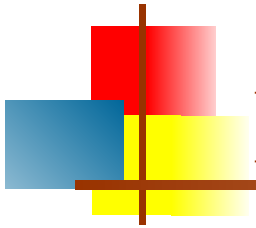
- Good interpretation helps to achieve the level of performance that is usually more than adequate.
- Java is faster than traditional interpretation since it use the “Just in Time” (JIT) compiler to do the interpretation.
- Since, byte code is close to native code, it analyses the interpreted byte code and shows high performance.

History of Java

- There was a small team named as “Green Team” whose members were three college friends named as “James Gosling”, “Mike Sheridan”, “Patrick Naughton”
- who originally initiated a revolutionary task to develop a language for digital devices like television, set-up boxes
- After some time they came to know that it suited for internet programming. So, they move forward to work more on it by giving (by James Gosling) it first name i.e. “Greentalk” and file extension was “.gt”. After that it was called as “Oak” after an oak tree that stood outside Gosling’s office. Later it came under the project called “Green Project” for further development and then finally renamed as “Java” form Java coffee

Histrory of Java

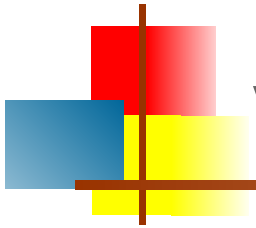
- Java is a name of an island of Indonesia where first coffee was produced and called as Java coffee.
- That is why the language got named as Java.
- The word Java has no meaning as it's just a name not an acronym.



Principles to be achieved(Features)

The Green team gave some opinion about the project that should have the following:

- It should be simple and gather tested fundamentals and features from the earlier language in it.
- It should have standards sets of libraries and APIs with basic and advanced features bundled in it.
- It should get rid of concepts requiring direct manipulation of hardware (here, memory) to make the language safe.
- It should be platform independent and may be written on every platform once i.e. “Write once Run anywhere” (WORA).
- It should be embeddable in web browsers.
- It should be able to manipulate network programming out of the box.
- It should have the ability for a single program to multi-task and do multiple things at a time.
- The first public implementation was released as Java 1.0 by Sun Microsystem in 1995 with a promise of WORA (Write once, Run anywhere) providing no cost run-time on popular platforms.



Version History

- The major release versions of Java are as following.
- **Note:** Current stable version is Java SE 8.

JDK alpha and beta (1995)

JDK 1.0 (Initial Release – 21st January, 1996)

JDK 1.1 (Initial Release – 19th February, 1997)

J2SE 1.2 (Java 2 – 8th December, 1998)

J2SE 3 (Kestrel – 8th May, 2000)

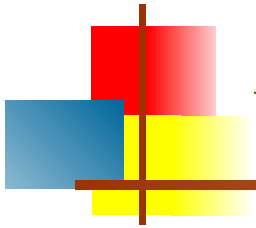
J2SE 1.4 (Merlin – 6th February, 2002)

J2SE 5.0 (Version 1.5.0) (Tiger – 30th September, 2004)

Java SE 6 (Version 1.6.0) (Mustang – 11th December , 2006)

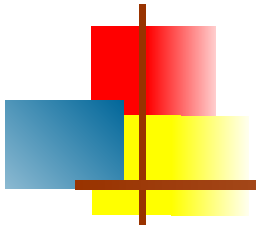
Java SE 7 (Version 1.7.0) (Dolphin – 28th July, 2011)

Java SE 8 (Version 1.8.0) (18th March, 2014)



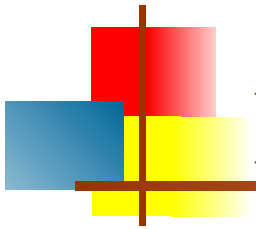
Version Histrory

- Java Standard Edition(Java SE)-jdk
- Java Enterprise Edition(Java EE)-full development
- Java Micro Edition(Java ME)-mobile app



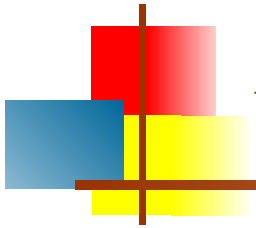
Syntax of java Program

- Import section
- Global variable declaration
- Package declaration
- Interface declaration
- Declare public class{
- Calling of main method()
- Public static void main(String args[])
- {
- //Program logic
- }



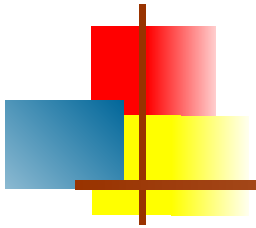
First Java Program

```
public class MyFirstJavaProgram
{
  /* This is my first java program. */
/   * This will print 'Hello World' as the output    */
public static void main(String args[])
{
 System.out.println("Hello World"); // prints Hello World
  }
}
```



Why?

- Why static?
- Why void?
- What is String args[]? S capital?



Creating class and an Object

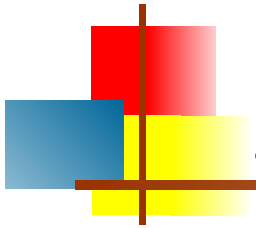
- Create a Class
- To create a class, use the keyword class:
- Create a class named “FirstJava" with a variable x:

- public class FirstJava
- {
- int x = 5;
- }

Create an object

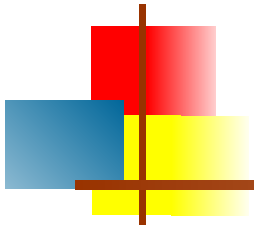
- To create an object of MyClass, specify the class name, followed by the object name, and use the keyword new and constructor
- Example:Create an object called "myObj" and print the value of x using dot operator

```
public class MyClass
 {
  int x = 5;
  public static void main(String[] args)
{
    MyClass myObj = new MyClass();
    System.out.println(myObj.x);
  }
}
```



Java Tokens

- Java Tokens are the smallest individual building block or smallest unit of a Java program;
- the Java compiler uses it for constructing expressions and statements.
- Java program is a collection of different types of tokens, comments, and white spaces.
- When we write a program, we need different important things. We require language tokens, white spaces, and formats.
- There are various tokens used in Java:
  - Reserved Keywords
  - Identifiers
  - Literals
  - Operators
  - Separators
  - White space is also considered as a token.

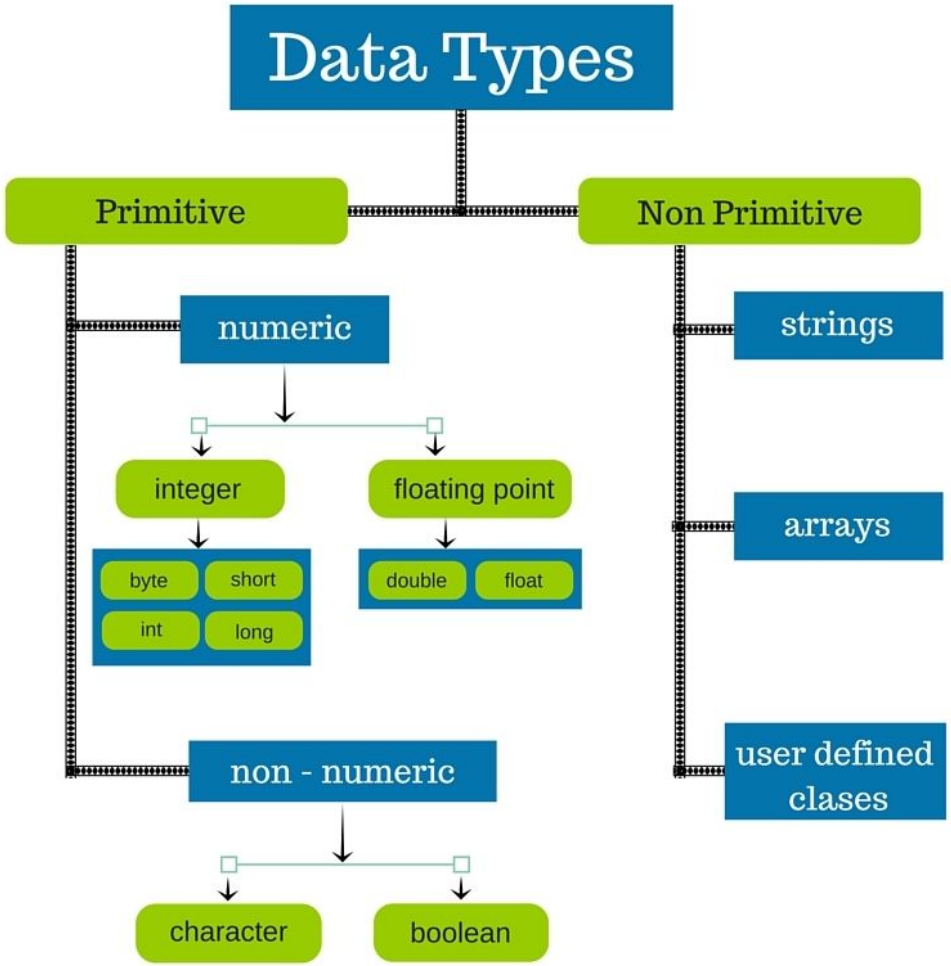


Keywords

- Keywords are words that have already been defined for Java compiler.
- They have special meaning for the compiler.
- Java Keywords must be in your information because you can not use them as a variable, class or a method name.

- abstract assert boolean breakbyte case catch charclass const continue default do double else enum extends final finally float for goto ifimplements 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 true false null
- true, false and null are not reserved words but cannot be used as identifiers, because it is literals of built-in types.

Java Data Types

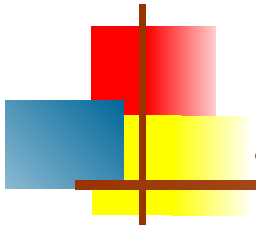


Data Types with size

Data Types in Java

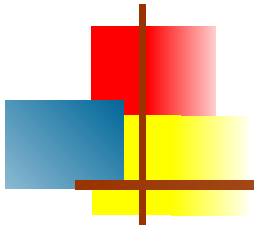
1 byte = 8 bit

Data Type	Default Size	Value Range
boolean	1 bit (which is a special type for representing true/false values)	true/false
char	2 byte (16 bit unsigned unicode character)	0 to 65,535
byte	1 byte (8 bit Integer data type)	-128 to 127
short	2 byte (16 bit Integer data type)	-32768 to 32767
int	4 byte (32 bit Integer data type)	-2147483648 to 2147483647.
long	8 byte (64 bit Integer data type)	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 byte (32 bit float data type)	approximately ±3.40282347E+38F (6-7 significant decimal digits) *Java implements IEEE 754 standard*
double	8 byte (64 bit float data type)	approximately ±1.79769313486231570E+308 (15 significant decimal digits)



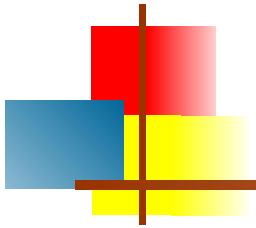
Java Variables

- A variable is a container which holds the value while the Java program is executed.
- A variable is assigned with a data type.
- Variable is a name of memory location. There are three types of variables in java: local, instance and static
- Variable is name of reserved area allocated in memory. In other words, it is a name of memory location. It is a combination of "vary + able" that means its value can be changed.
- variables in java
- int data=50;//Here data is variable



Types of Variables

- There are three types of variables in Java:
- local variable
- instance variable
- static variable
- 1) Local Variable
- A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.
- A local variable cannot be defined with "static" keyword.



Types of Variable

2) Instance Variable

A variable declared inside the class but outside the body of the method, is called instance variable. It is not declared as static.

It is called instance variable because its value is instance specific and is not shared among instances.

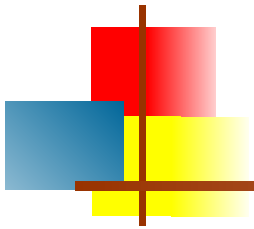
3) Static variable

A variable which is declared as static is called static variable.

It cannot be local.

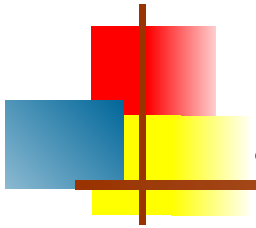
You can create a single copy of static variable and share among all the instances of the class.

Memory allocation for static variable happens only once when the class is loaded in the memory.



Example

- 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

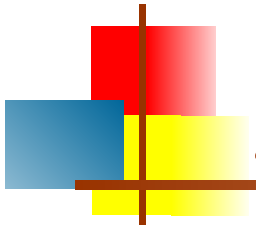


Java Constants

- Constant is a value that cannot be changed after assigning it.
- There is an alternative way to define the constants in Java by using the non-access modifiers static and final.

example : int rate=10;

```
final int interest=10000;
static float principal=10000.99;
final static float PI=3.14;//symbolic constant
```



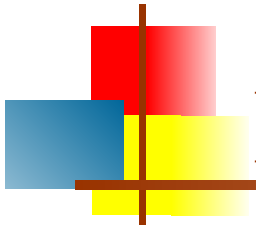
Java Operators

- Java operators are symbols that are used to perform mathematical or logical manipulations.
- Java is rich with built-in operators.
- Operators are tokens that perform some calculations when they are applied to variables.
- There are many types of operators available in Java such as:
- Arithmetic Operators
- Unary Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Compound Assignment Operators
- Conditional Operator
- instanceof Operator
- Member Selection or Dot Operator

Arithmetic Operators

Operator	Meaning	Work
+	Addition	To add two operands.
-	Subtraction	To subtract two operands.
*	Multiplication	To multiply two operands.
/	Division	To divide two operands.
%	Modulus	To get the area of the division of two operands.

Example	Description
val = a++;	Store the value of "a" in "val" then increments.
val = a--;	Store the value of "a" in "val" then decrements.
val = ++a;	Increments "a" then store the new value of "a" in "val".
val = --a;	Decrements "a" then store the new value of "a" in "val".



Relational Operators

- Why is it called as Relational?
- Relational operators compare between operands and determine the relationship between them.
- There are six types of relational operators in Java, these are:

Operator	Meaning
==	Is equal to
!=	Is not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Logical operators(based on logical gates)

Operator	Meaning	Work
&&	Logical AND	If both operands are true then only "logical AND operator" evaluate true. True && true=true
||	Logical OR	The logical OR operator is only evaluated as true when one of its operands evaluates true. True || false = true
!	Logical Not	Logical NOT is a Unary Operator, it operates on single operands. It reverses the value of operands, if the value is true, then it gives false, and if it is false, then it gives true.

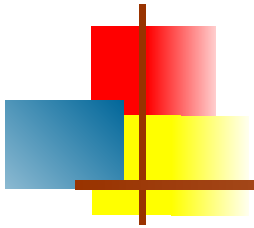
Program to Show Logical Operators Works

- public class logicalop

```
{
 public static void main(String[] args)
 {
      //Variables Definition and Initialization
      boolean bool1 = true, bool2 = false;
      //Logical AND
       System.out.println("bool1 && bool2 = " + (bool1 && bool2));
      //Logical OR
      System.out.println("bool1 || bool2 = " + (bool1 | bool2) );
      //Logical Not
      System.out.println("!(bool1 && bool2) = " + !(bool1 && bool2)); } }
```

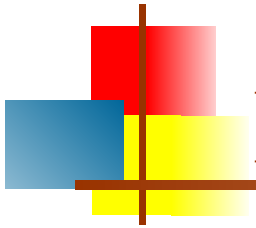
- Output:

```
bool1 && bool2 = false
 bool1 || bool2 = true
 !(bool1 && bool2) = true
```



Finding greatest of 3 numbers

- A=67, b=90, c=4;
- If a>b and a>c then a is greatest
- Else
- If b>a and b>c then b is greatest
- Else c is greatest



Program

```
public class Largest {

    public static void main(String[] args) {

        int a = -4, b = 3, c = 2;

        if( a >= b && a >= c)
            System.out.println(a + " is the largest number.");

        else if (b >= a && b >= c)
            System.out.println(b + " is the largest number.");

        else
            System.out.println(c + " is the largest number.");
    }
}
```

Bit wise Operators

- Operate on each bit of byte
- Can operate on integer data only
- Works on concept of logical gates AND, OR, NOT…

Operators	Symbol	Uses
Bitwise AND	&	op1 & op2
Bitwise exclusive OR	^	op1 ^ op2
Bitwise inclusive OR	|	op1 | op2
Bitwise Compliment	~	~ op
Bitwise left shift	<<	op1 << op2
Bitwise right shift	>>	op1 >> op2
Unsigned Right Shift Operator	>>> op >>>	number of places to shift

Bitwise AND (&)

It returns 1 if and only if both bits are 1, else returns 0

x	y	x & y
0	0	0
0	1	0
1	0	0
1	1	1

```
public class BitwiseAndExample
{
public static void main(String[] args)
{
int x = 9, y = 8;
// bitwise and
// 1001 & 1000 = 1000 = 8
System.out.println("x & y = " + (x & y));
} }
```

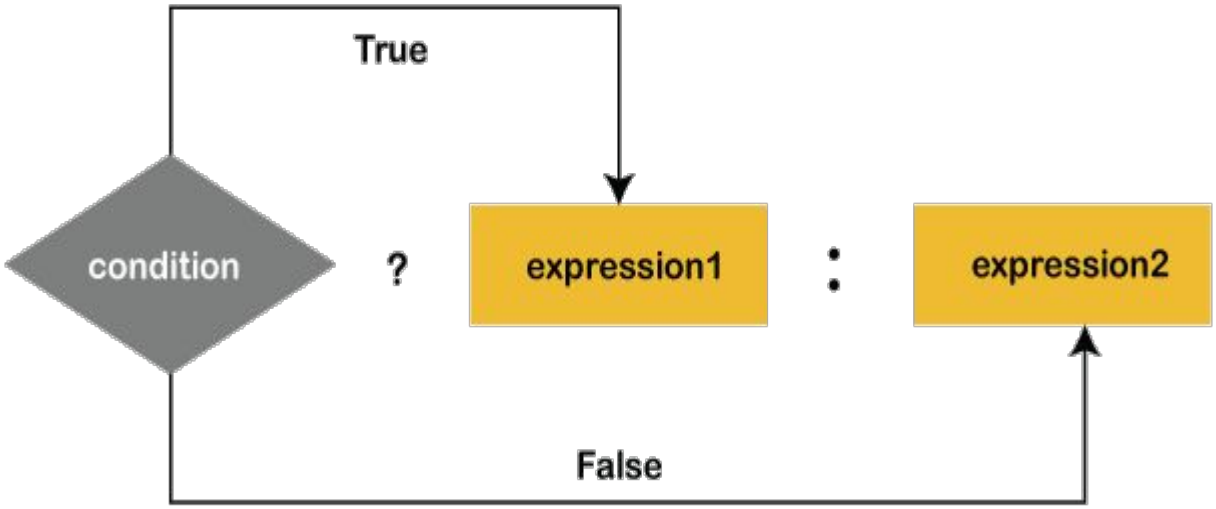
Output

x & y = 8

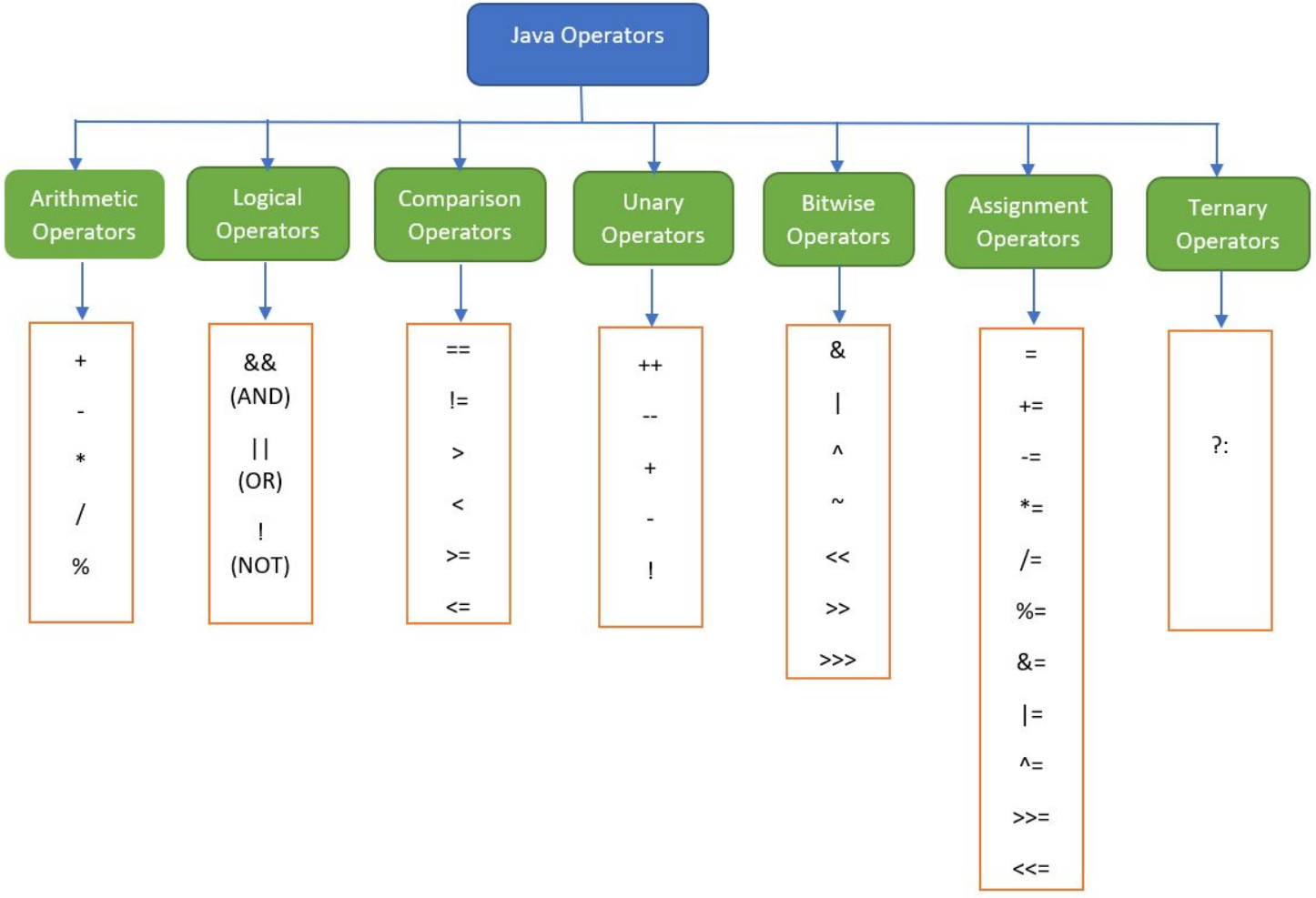
Conditional or Ternary Operator

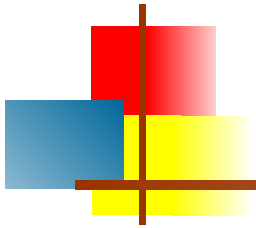
- The meaning of ternary is composed of three parts.
- The ternary operator (? :) consists of three operands.
- It is used to evaluate Boolean expressions.
- The operator decides which value will be assigned to the variable.
- It can be used instead of the if-else statement. It makes the code much more easy, readable, and shorter.
- Syntax:
- variable = (condition) ? expression1 : expression2
- The above statement states that if the condition returns true, expression1 gets executed, else the expression2 gets executed and the final result stored in a variable.

Conditional or Ternary Operator



Precedence of Operators





Type Casting/type conversion

Converting one primitive datatype into another is known as type casting (type conversion) in Java.

You can cast the primitive datatypes in two ways namely, Widening and, Narrowing.

Casting=temporary conversion

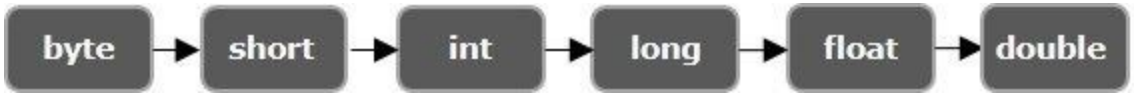
Widening −

Converting a lower datatype to a higher datatype is known as widening.

In this case the casting/conversion is done automatically therefore, it is known as implicit type casting.

In this case both datatypes should be compatible with each other.

Example



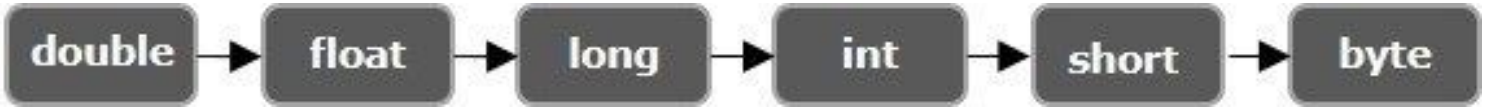

```
public class WideningExample {
   public static void main(String args[]){
      char ch = 'C';
      int i = ch;
      System.out.println(i);
   }
}
```

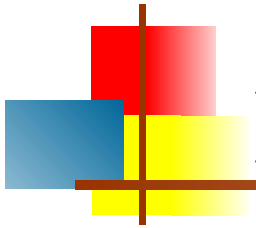
Output

Integer value of the given character: 67

Narrowing −

- Converting a higher datatype to a lower datatype is known as narrowing.
- In this case the casting/conversion is not done automatically, you need to convert explicitly using the cast operator “( )” explicitly.
- Therefore, it is known as explicit type casting. In this case both datatypes need not be compatible with each other.





Example

```
import java.util.Scanner;
public class NarrowingExample {
   public static void main(String args[]){
      Scanner sc = new Scanner(System.in);
      System.out.println("Enter an integer value: ");
      int i = sc.nextInt();
      char ch = (char) i;
      System.out.println("Character value of the given integer: "+ch);
   }
}
```

Output

Enter an integer value:

67

Character value of the given integer: C

Statements

- the statement is the instruction given to the computer that performs specific types of work
- There are 3 types of statement

Statement	Description
Type declaration Statement	To declare the type of variables being used in the program. int data=50;
Arithmetic Statement	To complete the arithmetic operation between constants and variables. int c=a+b;
Control Statement	To control the sequence of execution of different statements of the program. If,if-else

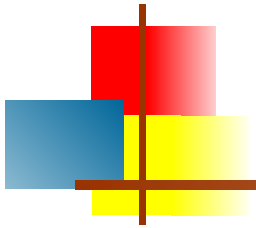
If & if-else statement

- The basic format of if statement is:

```
if(test_expression)
{
   statement 1;
   statement 2;
   ...
}
```

- The basic format of if else statement is:

```
if(test_expression)
{
  //execute true part
}
else
{
  //execute false part
}
```



Odd-even program

```
import java.util.Scanner;

public class EvenOdd {

    public static void main(String[] args) {

        Scanner reader = new Scanner(System.in);

        System.out.print("Enter a number: ");
        int num = reader.nextInt();

        if(num % 2 == 0)
            System.out.println(num + " is even");
        else
            System.out.println(num + " is odd");
    }
}
```

Ladder of if-else(else-if)

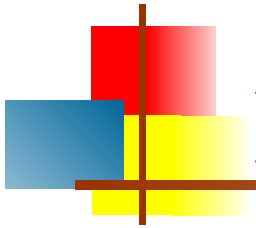
The basic format of else if statement is:

```
if(test_expression)
{
   //execute your code
}
else if(test_expression n)
{
   //execute your code
}
else
{
   //execute your code
}
```

```
public class Sample {

    public static void main(String args[]) {
        int a = 30, b = 30;

        if (b > a) {
            System.out.println("b is greater");
        }
        else if(a > b){
            System.out.println("a is greater");
        }
        else {
            System.out.println("Both are
equal");
        }
    }
}
```



Looping Statements

- Sometimes it is necessary for the program to execute the statement several times
- loops execute a block of commands a specified number of times until a condition is met.
- The process of repeatedly executing a collection of statement is called looping.
- Java supports following types of loops:
  - while loops
  - do while loops
  - for loops

While loop

- while loop is the most basic loop in Java.
- It has one control condition and executes as long the condition is true.
- The condition of the loop is tested before the body of the loop is executed; hence it is called an entry-controlled loop.

Syntax:

```
Counter initiation;
while(counter condition check)
{
//statements
Counter++;
}
```

example:

```
int i=0;
while(i<=10)//entry controlled loop
{
 System.out.println("I=" +i);
i++;
}
```

Do-while loop

- Java do-while loops are very similar to the while loops
- it always executes the code block at least once and goes on excecuting as long as the condition remains true.
- This loop is an exit controlled loop

Syntax:

```
Counter initiation;
do
{
//statements
Counter++
}while(condition check of counter);
//Exit controlled loop
```

Example:

```
int n = 1, times = 3;
      do {
System.out.println("Java do while loops:" + n);
           n++;
       } while (n <= times);
```

Difference Between While & do-while

while	Do-while
Entry controlled loop	Exit controlled
Condition checking is done at the start of loop	Condition checking is done at the end of loop
Execution is available if condition is true	Executed at least once
Keyword used-while	Keywords used-do & while
At the while statement ; is not given	At the while statement ; is given

For loop

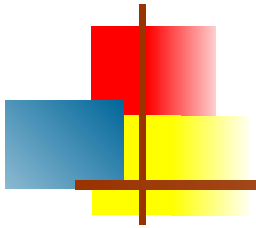
- Java for loops is very similar to Java while loops in that it continues to process a block of code until a statement becomes false
- everything is defined in a single line.

Syntax:

```
for ( initiation of counter; condition check of counter; increment the counter )
{
  // statement(s);
}
```

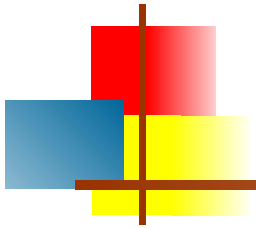
Example:

```
int times=5;
for (int ni= 1; i <= times; i++)
{
System.out.println("Java for loops:" + n);
}
```



Switch-case statement

- Drawbacks of nested if-else
  - Putting {}
  - Length of code gets increased
  - Coding becomes complex
- To avoid this switch-case structure is defined
- After the end of each block it is necessary to insert a break statement because if the programmers do not use the break statement, all consecutive blocks of codes will get executed from each case onwards after matching the case block.



Syntax of switch-case

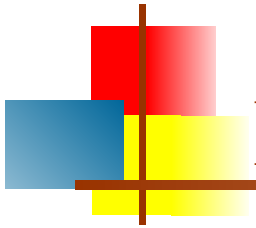
```
switch(variable)
{
case 1:
   //execute your code
break;

case n:
   //execute your code
break;

default:
   //execute your code
break;
}
```

Example of switch-case

- Program click here
- Program on conditional operators
- public class condiop {
- public static void main(String[] args) {
- String out;
- int a = 6, b = 12;
- out = a==b ? "Yes":"No";
- System.out.println("Ans: "+out);
- }
- }



Programs on loops

- While loop:

  sum of numbers between 10 & 25

  reversing the digits of a integer number
- For loop

  factorial calculation

  display of * * * *

  * * *

  * *

  *
- Do-while display odd numbers between 10 to15

Sum of numbers

int sum=0;

sum= sum + i;

Sum	i	Sum=sum+i
0	0	0
0	1	1
1	2	3
3	3	6
6	4	10
10	5	15

Numbers between 10 &25

Logical operator: i>=10&&i<=25

Reverse the digit of a integer number

- Input =764 output=467
- Remainder=number%10
- Number=number/10

Remainder=number%10	Number=number/10	
764%10=4	764/10=76	Number=76
76%10=6	76/10=7	Number=7
7%10=7	7/10=0	Number=0

Factorial Calculation

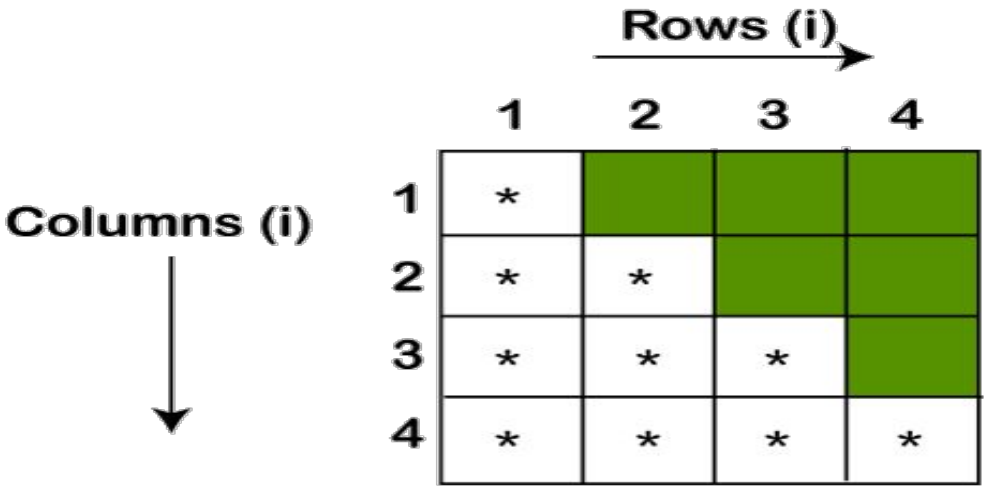
- Fact of 1=1
- N!=N*(N-1)!

N!=N*(N-1)!	
5!=5*(5-1)!	5*4!
4!=4*(4-1)!	5*4*3!
3!=3*(3-1)!	5*4*3*2!
2!=2*(2-1)!	5*4*3*2*1!
1!=1	5*4*3*2*1=120

Thought of a Day

Star pattern program

- Whenever you design logic for a pattern program, first draw that pattern in the blocks, as we have shown in the following image. The figure presents a clear look of the pattern.



- Each pattern program has two or more than two loops. Nested for loops. The first for loop works for the row and the second loop works for the column.
- In the above pattern, the row is denoted by i and the column is denoted by j. We see that the first row prints only a star. The second-row prints two stars, and so on. The colored blocks print the spaces.
- If(row<=columns), stop display of *

Display of command line arguments

- Command line arguments are the arguments for which we are able give values at the command line(Command Prompt)

```
class MyClass1
{
public static void main(String args[])
 {
  for(int i=0;i<args.length;i++)
  System.out.println(args[i]);
  }
}
```

compile by > javac MyClass1.java

run by > java MyClass1 MIT Pune 1 3 abc

Output: MIT

Pune

1

3

abc

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) {
    //Declaring an array
    int arr[]={12,23,44,56,78};
    for(int i:arr){
        System.out.println(i);
    }
}
}
```

Math functions of Java

Method	Description
abs(x)	Returns the absolute value of x
sin(x)	Returns the sine of x, in radians
tan(x)	Returns the tangent of x as a numeric value in radians
cbrt(x)	Returns the cube root of x
ceil(x)	Returns the value of x rounded up to its nearest integer
cos(x)	Returns the cosine of x (x is in radians)
exp(x)	Returns the value of E^x
floor(x)	Returns the value of x rounded down to its nearest integer
log(x)	Returns the natural logarithm (base E) of x
max(x, y)	Returns the number with the highest value
min(x, y)	Returns the number with the lowest value
pow(x, y)	Returns the value of x to the power of y
random()	Returns a random number between 0 and 1
round(x)	Returns the value of x rounded to its nearest integer
sqrt(x)	Returns the square root of x