TEKNOTURF

INTRODUCTION TO JAVA

OVERVIEW

TEKNOTURF

JAVA is an open source technology for developing platform independent application.

Introduction to JAVA

TEKNOTURF

- Developed by James Gosling at Sun Micro Systems
- Introduced in 1995
- Is one of the fastest growing programming languages of all times

Java

Features of JAVA

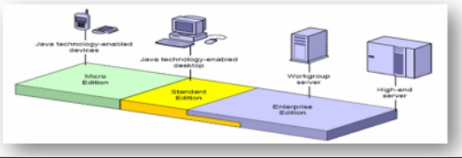
TEKNOTURF

- Object Oriented Programming Language
- Platform independence
- Architecture Neutral
- Secure
- Robust
- Memory management
- Multi thread
- Java can run:
  - in a browser as an Applet.
  - on the desktop as an Application.
  - in a server to provide Services e.g. Database access.
  - embedded in a device.

JAVA Editions

TEKNOTURF

- Java Standard Edition (J2SE or JSE)
  - Basic tools and libraries to build applications and applets.(Standalone applications)
- Java Enterprise Edition (J2EE or JEE)
  - Tools and libraries to create servlets , javaServerPages ,etc. (web applications)
- Java Micro Edition (J2ME or JME)
  - Libraries to create application to run on hand-held devices such as PDAs and cellular phones. (device programs and mobile applications)

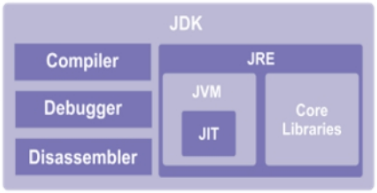


Java platform

TEKNOTURF

JAVA DEVELOPMENT TOOL KIT

- Program development environment to develop applets and applications.
- JDK consists of JRE and tools like javac(compiler) , java Interpreter.



JAVA RUN TIME ENVIRONMENT

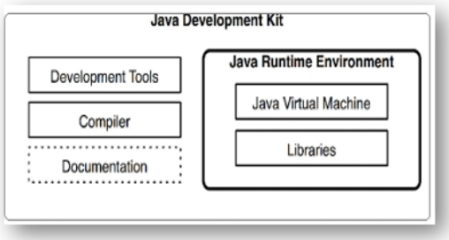
- JRE is used for running java applications and applets.
- JRE consists of JVM, core classes and the supporting files. JVM converts byte code into machine code.

Just-In-Time (JIT): JIT compiler improves the performance of java application by compiling bytecode into native machine code at run time

JAVA Toolkit - JDK

A Java Development Kit (JDK) is a program development environment for writing Java applets and applications.

JDK consists of a runtime environment called JRE, which sits on top of the operating system and the tools like interpreter (java) and compiler (javac).

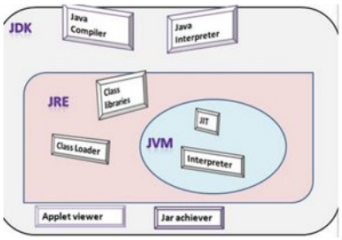


JAVA Compiler

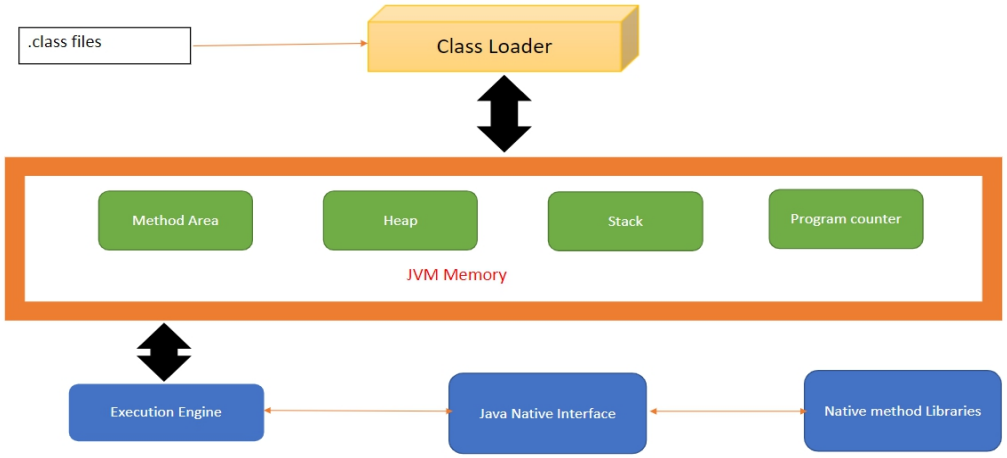
- Java programs are translated into an intermediate code called byte code with the help of a java compiler
- Byte code is platform independent and can run on any operating system

JAVA Run-Time Environment - JRE

- The Java Runtime Environment (JRE), also known as Java Runtime, is a part of the Java Development Kit (JDK)
- The Java Runtime Environment consists of the Java Virtual Machine (JVM), core classes, and supporting files.
- JVM converts Java byte code into machine language and executes it.



JVM Architecture



JVM Architecture

Three main sub systems of JVM are

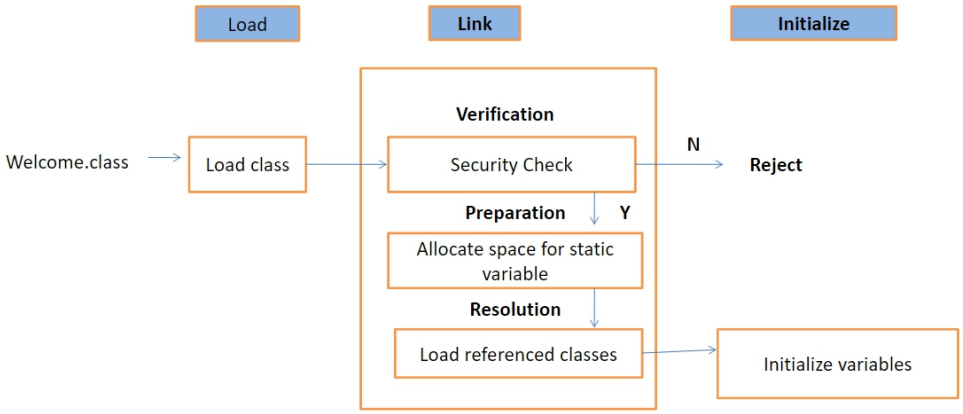
- Class Loader
- Runtime Data Area (JVM Memory)
- Execution Engine

Class Loader

- JVM resides on the RAM.
- When a file is executed, the needed class files with .class extension are brought on the RAM. This is termed as dynamic class loading.
- When a class loader refers to a class for the first time, it loads, links and initializes the class file.

JVM Architecture

Class Loader



JVM Architecture

Runtime Data Area comprises of Method area, Heap, Stack and Program counter

- **Heap**

  Heap memory is used to store objects of classes with instance variables and arrays.

  Garbage collector clears the unused memory automatically.

  JVM will throw OutOfMemoryError when the allocated memory is not sufficient

- **Method Area**

  Method area stores class level data like static fields and structures, code for method and constructor, method data, run-time constant pool

- **Stack**

  Stack is used to used to store local variables and the partial results

  It is used whenever a method is invoked and returns a value.

- **Program Counter**

  Program Counter (PC) keeps track of the current instruction that is being executed at any moment in sequence of instructions in a program.

JVM Architecture

- **Execution engine**

  Execution engine executes the byte code which is assigned to the Runtime Data Area.
  It reads the byte code and executes one by one

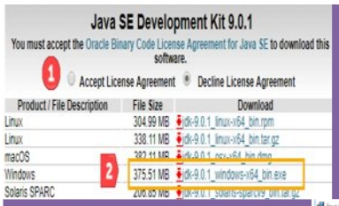
- **Java Native Interface(JNI)**

  Java Native Interface will interact with the Native Method Libraries and provide the Native Libraries required for the Execution Engine

- **Native Method Libraries**

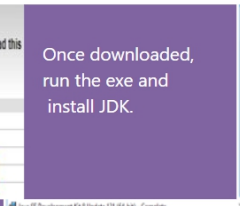
  It is a Collection of the Native Libraries which is required for the Execution Engine.

Java environment set up

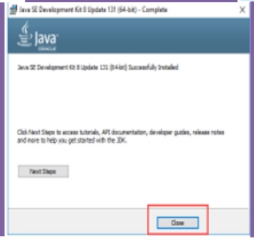


Download JDK from the oracle official website https://www.oracle.com/technetwork/java/javase/downloads/jdk9-downloads-3848520.html

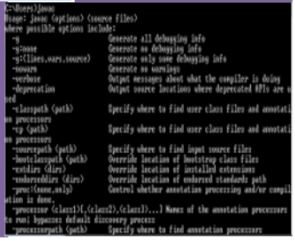
Once downloaded, run the exe and install JDK.



To check java installation go to the command prompt and run the javac command

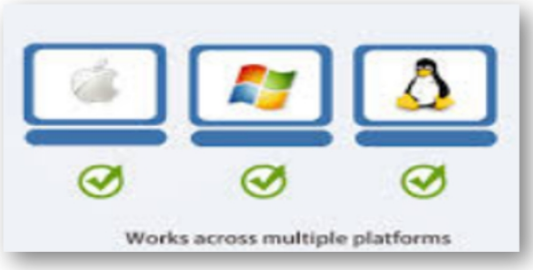


PATH ENVIRONMENT VARIABLE
SET PATH to the bin folder of JDK
PATH is set in the advanced System settings

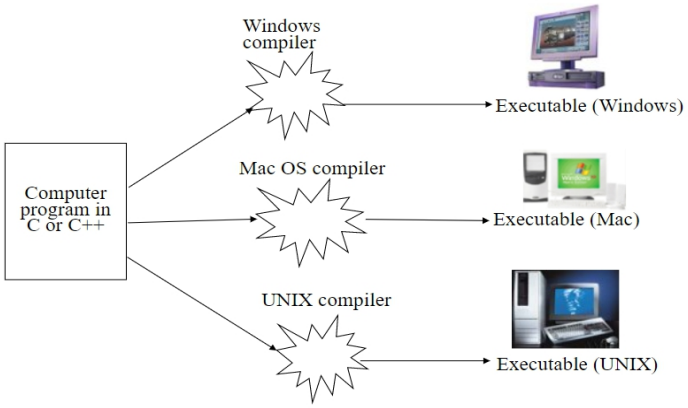


Platform Independent

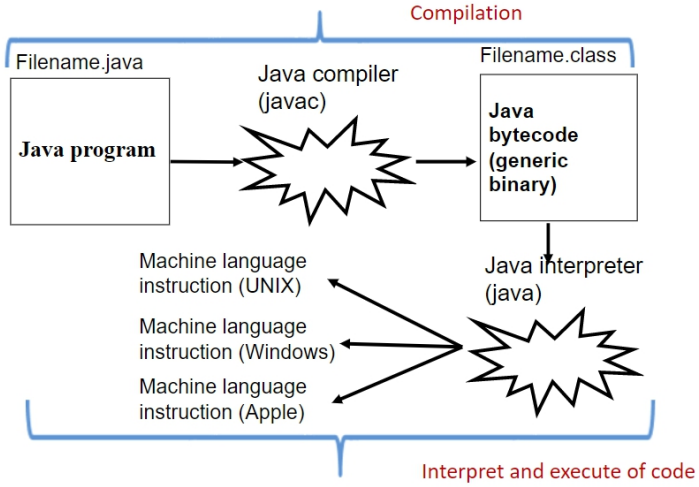
- The main feature of Java is platform independent
- The generated byte code which is the intermediate code is independent of the operating system and hence we say it is platform independent.



Platform Independent

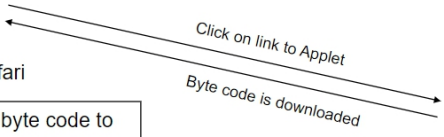


Platform Independent



Write Once; Run Anywhere

Mac user running Safari



Web page stored on Unix server

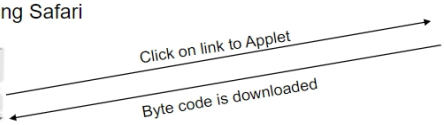
Virtual machine translates byte code to native Mac code and the Applet is run

Windows user running Internet Explorer

Write Once; Run Anywhere

Mac user running Safari

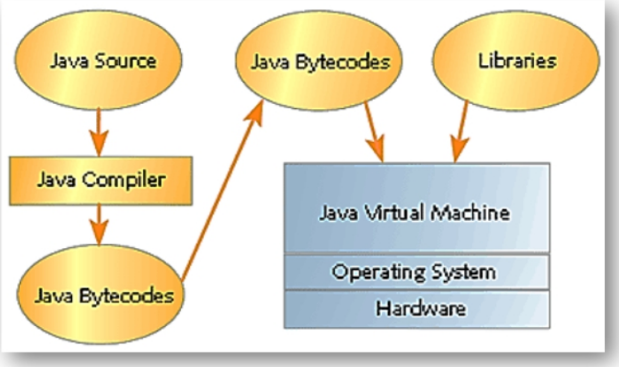
Web page stored on Unix server



Windows user running Internet Explorer

Virtual machine translates byte code to native Windows code and the Applet is run

JAVA Program Execution



Writing a Program in JAVA

TEKNOTURF

Any code written in java is written within a block termed as class

The main method in a java program acts as an entry point signature

All java programs start their execution from the main method

Main method

- has to be declared as public static
- takes String[] as an argument
- return type is void

Writing a Program in JAVA

TEKNOTURF

- Example : *Type the code in a notepad*

```
public class  TestProgram {
    public static void main(String a[]) {
        System.out.println("Welcome to Java");
    }
```

- Save this file with an extension .java
- Name of the file should be same as class name
  TestProgram.java

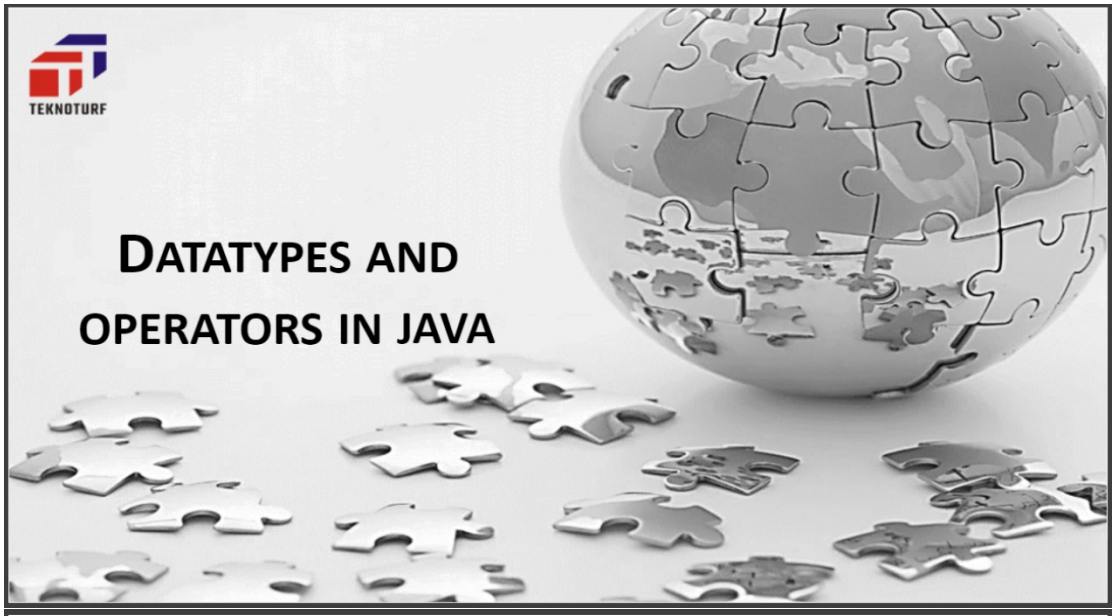
Writing a Program **in JAVA**

TEKNOTURF

To compile and execute the code

- Click "**run**"
- Type "**cmd**"
- Get to the location where the file is located, say Desktop
  .../Desktop>
- Compile the code as
  .../Desktop> **javac** TestProgram.java
- To execute the code
  .../Desktop> **java** TestProgram
- Output will be
  Welcome to Java

Note : when the code is compiled, it generates the intermediate code TestProgram.class, called the bytecode



Datatypes - Overview

TEKNOTURF

Assume David is a customer in ABC Bank.
The bank needs to create the customer with the following fields.

Name	Alphabets
Gender	Single alphabet
Account Number	Number
Balance	Number
Status	Alphabets – Yes/No

Java Comments

TEKNOTURF

Java supports the following comment styles

Single line comment

```
// comment on one line
```

Multi line comment

```
/* comment on one
 * or more lines
 */
```

Java documentation comment

```
/** documentation comment
 *can also span one or more lines
 */
```

Semicolons, Blocks and White Space

A statement is one or more lines of code terminated by a semicolon (;)

```
totals = a + b + c+ d + e + f;
```

A block is a collection of statements bound by opening and closing braces

```
{
   x = y + 1;
}
```

A class definition uses a special block:

```
public class MyDate {
      private int day;
      private int month;
      private int year;
}
```

Any amount of white space is permitted in a Java program.

Java Identifiers

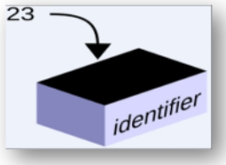
Identifiers are names given for class, variables , methods

Identifiers have the following characteristics:

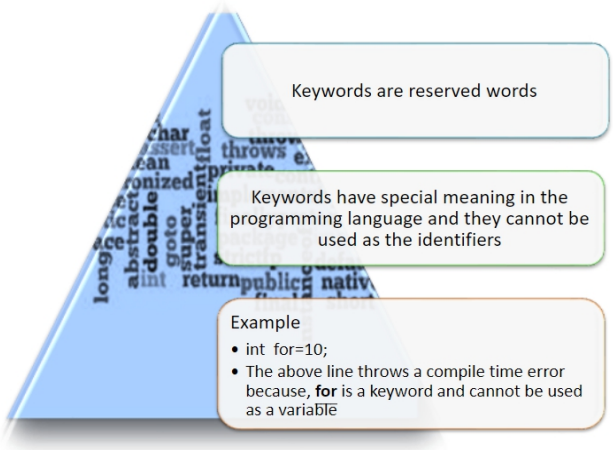
- Identifier names given to a variable, class, or method
- Identifier names can start with an alphabet, underscore (_), or dollar sign ($) and followed by alphabets / numbers
- Are case-sensitive and have no maximum length

Examples:

- identifier
- userName
- user_name
- _sys_var1
- $change



Java Language Keywords



Keywords are reserved words

Keywords have special meaning in the programming language and they cannot be used as the identifiers

Example

- int for=10;
- The above line throws a compile time error because, **for** is a keyword and cannot be used as a variable

Java Language Keywords

Keywords in Java				
abstract	default	if	private	this
assert	do	implements	protected	throw
boolean	double	import	public	throws
break	else	instanceof	return	transient
byte	enum	int	short	try
case	extends	interface	static	void
catch	final	long	strictfp	volatile
char	finally	native	super	while
class	float	new	switch	
continue	for	package	synchronized	

Data Types

Data type defines the set of values a variable can hold in its life time

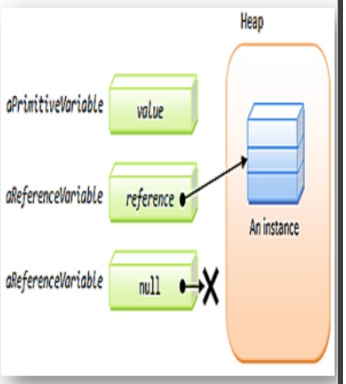
Java supports the following data type

Primitive data type

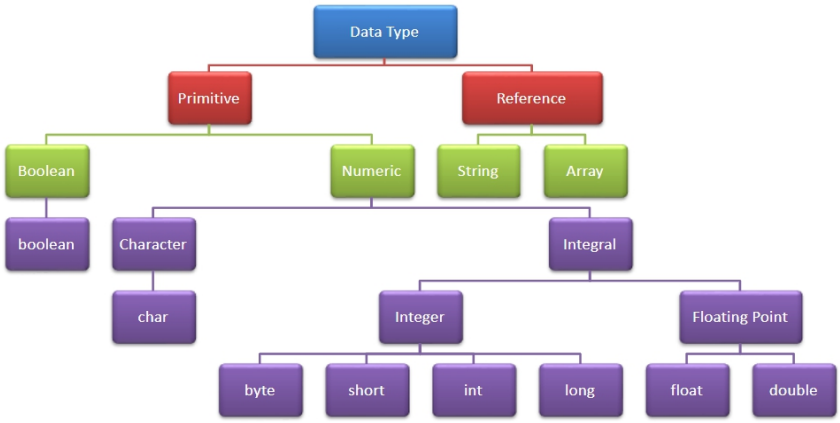
- Holds a scalar value
- int age=23;

Reference data type

- Holds an object
- Employee emp = new Employee();
- emp is a reference holding an employee object
- All Objects are allocated memory in the heap

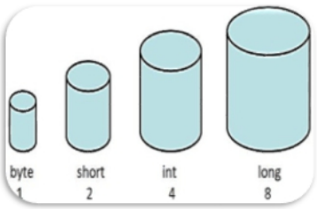


Data Types



Data Types

Data type	Size (in Bytes)	Default Value
byte	1	0
short	2	0
int	4	0
long	8	0
float	4	0.0
double	8	0.0
char	2(UNICODE)	\u0000
boolean	Depends on JVM	false



Formula for Signed datatype: -2^{n-1} to $2^{n-1}-1$

Formula for Unsigned datatype: 0 to $2^{n}-1$

Choose the data types appropriately based on the requirement

All Integer and Floating point data types are signed

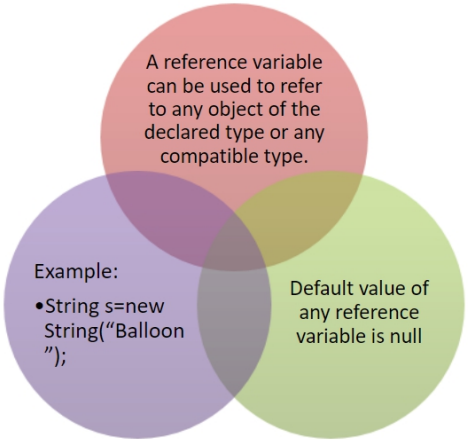
Character data type is unsigned

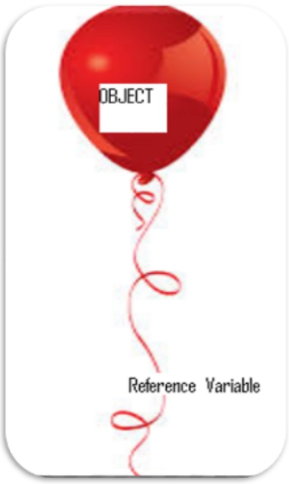
Datatypes - Overview

Having known, Bank needs to create the customer with the following fields.

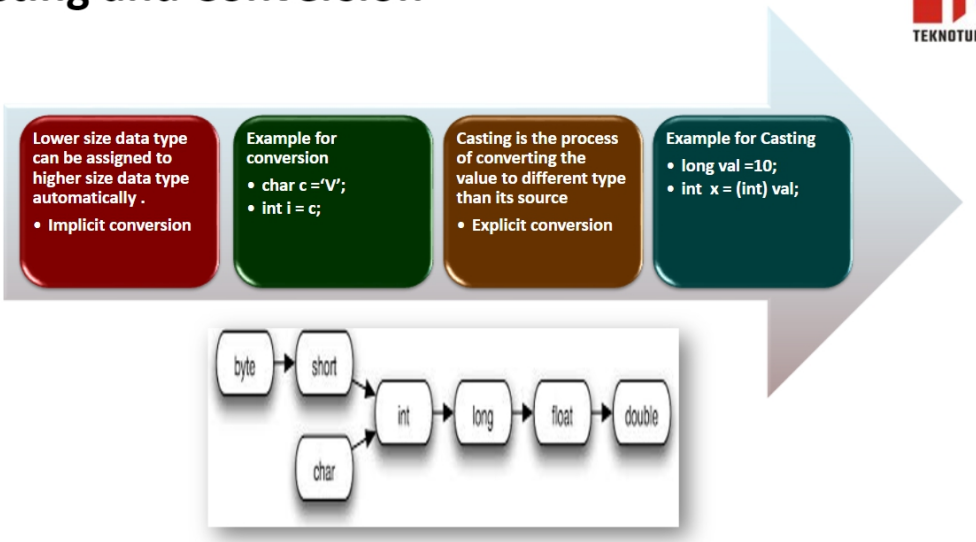
Name	String
Gender	char
Account Number	int
Balance	double
Status	boolean

Reference Data Types





Casting and Conversion



Arithmetic Operation challenges

Result of an expression involving anything int-sized or smaller is always an int.

```
public class UserInterface {

        public static void main(String a[]) {

                byte num1=5;
                byte num2=5;
                byte result=num1+num2;

        }
}
```

The above code produces compile time error.

```
public class UserInterface {

        public static void main(String a[]) {

                byte num1=5;
                byte num2=5;
                byte result=(byte) (num1+num2);
        }
}
```

To overcome the compile time error either typecast explicitly the expression to byte or change the datatype of result to int.

Operators - Overview

Fields for a customer in a Bank

- Name
- Gender
- Account Number
- Balance
- Status

How to calculate the interest amount based on the balance available ?

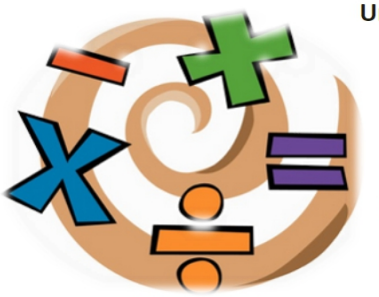
To perform mathematical and logical manipulations on data we have to use **Operators.**

Types of Operators

Arithmetic

Miscellaneous

Unary Arithmetic



Ternary

Relational and logical

Bitwise

Arithmetic and Unary Operators

Operator	Description	Example(X=10,Y=20)
+	Addition – Adds values on either side of the operator	X + Y will give 30
-	Subtraction - Subtracts right hand operand from left hand operand	X - Y will give -10
*	Multiplication - Multiplies values on either side of the operator	X * Y will give 200
/	Division - Divides left hand operand by right hand operand	Y / X will give 2
%	Modulus - Divides left hand operand by right hand operand and returns remainder	Y % X will give 0
++	Increment - Increases the value of operand by 1	Y++ gives 21
--	Decrement - Decreases the value of operand by 1	Y-- gives 19

Relational Operators

Operator	Description	Example(A=5,B=10)
==	Checks if the values of two operands are equal or not; if equal, then condition results in true.	(A==B) is false.
!=	Checks if the values of two operands are equal or not; if values are not equal, then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand; if yes, then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand; if yes, then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand; if yes, then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand; if yes, then condition becomes true.	(A <= B) is true.

Bitwise Operators

TEKNOTURF

Operator	Description	Example (A=15,B=10)
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 10 which is 0000 1010
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 15 which is 0000 1111
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 5 which is 0000 0101
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -16 which is 1111 0000 in 2's complement form due to a signed binary number.
<<	Binary Left Shift Operator. The left operand's value is moved left by the number of bits specified by the right operand	A << 2 will give 60 which is 0011 1100
>>	Binary Right Shift Operator. The left operand's value is moved right by the number of bits specified by the right operand.	A >> 2 will give 3 which is 0000 0011
>>>	Shift right zero fill operator. The left operand's value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.	A >>>2 will give 3 which is 0000 0011

Logical Operators

TEKNOTURF

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverse the logical state of its operand. If a condition is true then Logical NOT operator will make it false.	!(A && B) is true.

Assignment Operators

TEKNOTURF

Operator	Description	Example
=	Simple assignment operator; it assigns values from right side operands to left side operand	C = A + B will assign value of A + B into C
+=	Add AND assignment operator; it adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator; it subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator; it multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A

Assignment Operators

Operator	Description	Example
/=	Divide AND assignment operator. It divides left operand with the right operand and assigns the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator. It takes modulus using two operands and assigns the result to left operand	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator	C ^= 2 is same as C = C ^ 2
|=	bitwise inclusive OR and assignment operator	C |= 2 is same as C = C | 2

Miscellaneous Operators

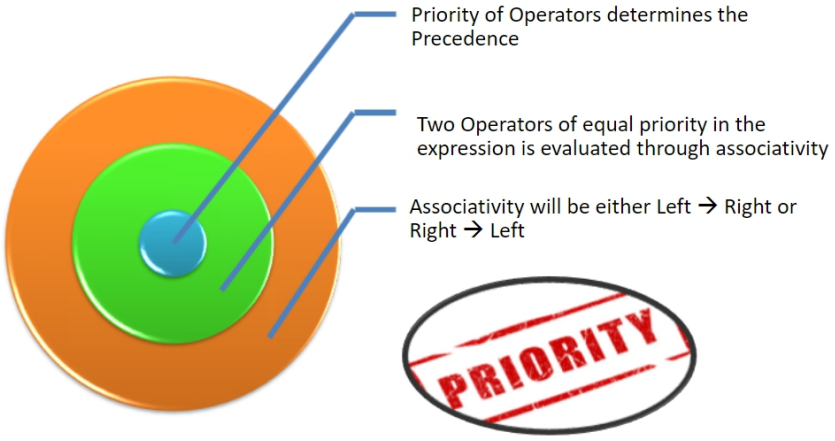
Conditional Operator

- ?:
- variable x = (expression) ? value if true : value if false
- Example
  - int a=10,b=2;
  - b = a > 10?a:b;

instanceof operator

- This operator is used to check whether the object is of a particular type(class or interface)
- (Object reference variable) instanceof (class/interface type)
- String s=new String("Balloon");
- s instanceof String

Precedence and Associativity



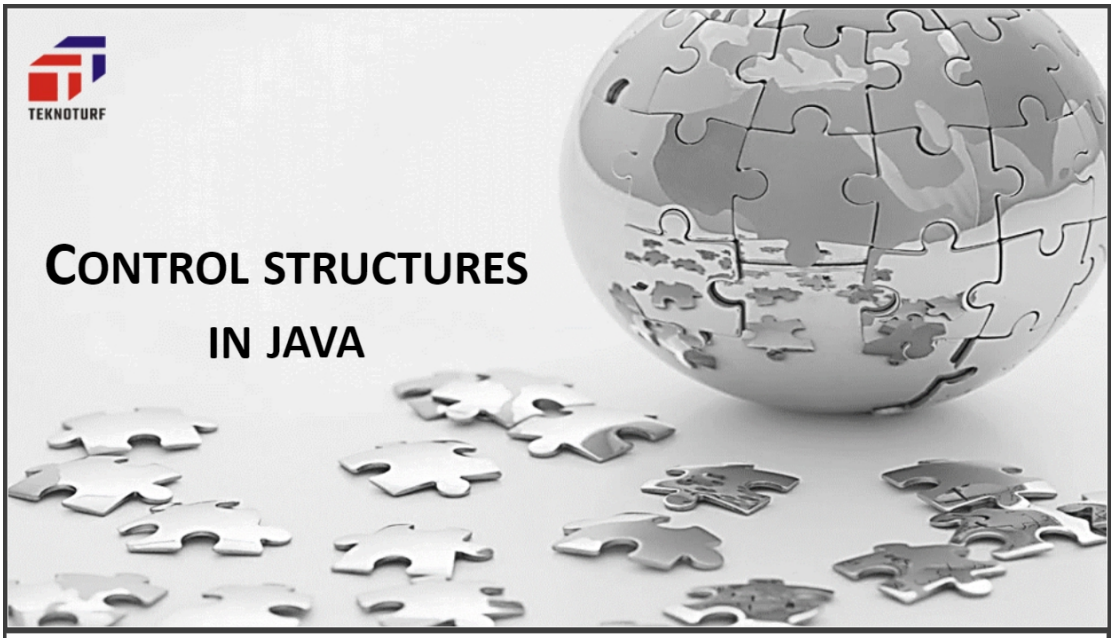
Precedence and Associativity

Category	Operator	Associativity
Postfix	() [] . (dot operator)	Left toright
Unary	++ - - ! ~	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	>> >>> <<	Left to right
Relational	> >= < <=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right

Precedence and Associativity

Category	Operator	Associativity
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= |=	Right to left
Comma	,	Left to right

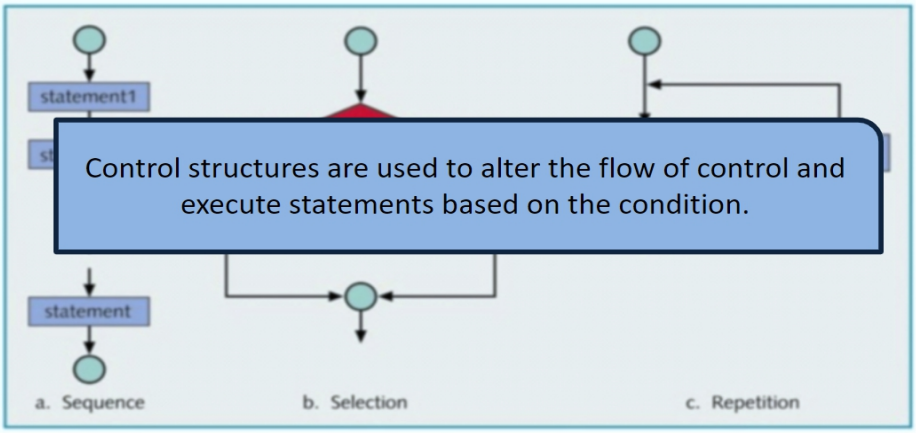
TEKNOTURF



CONTROL STRUCTURES IN JAVA

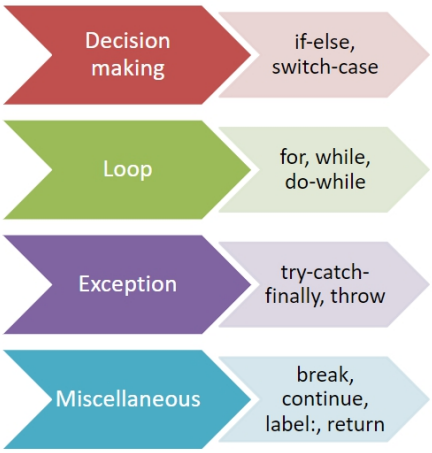
Control Structures - Overview

TEKNOTURF



Control Structures

TEKNOTURF

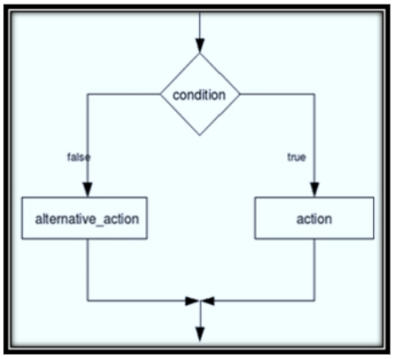


Decision Making – if else

TEKNOTURF

SYNTAX

```
if(condition) {
    //true part
}
else  {
    //false part
 }
```



Example

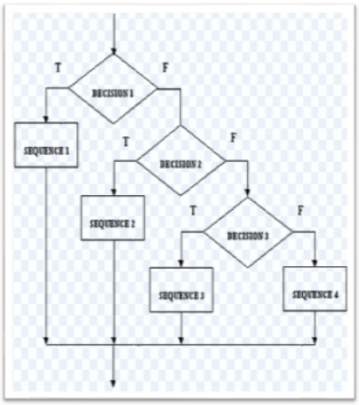
```
if(num1 > num2)
        System.out.println("num1 is greater");
else
        System.out.println("num2 is greater");
```

Decision Making – Nested If

TEKNOTURF

SYNTAX

```
if(condition1){
    if(condition2) {
        // true part of both conditions
    }
    else {
        // false part of condition2
     }
}
else{
    // false part of condition1
}
```



Decision Making – Nested If

TEKNOTURF

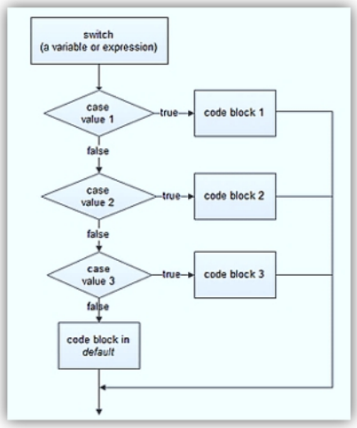
- **Example**

```
if (age <= 19 ) {
      if (age >= 13) {
          System.out.println( "You are in your teen age");
      }
      else {
          System.out.println( "You are a Child");
      }
}
else {
      System.out.println(" You are an adult ");
 }
```

Decision Making – switch - case

SYNTAX

```
switch(parameter) {
    case 1:
      //block executed when
      // the int parameter value is 1
      break;
    case 3:
      //block executed when
      //the int parameter value is 3
      break;
    default:
      // block executed when
      // the int parameter value does not match  any specific value
}
```



Decision Making – switch - case

Example

```
switch(grade) {
    case 'A' :
       System.out.println("Excellent!");   break;
    case 'B' :
      System.out.println("Well done");   break;
    case 'D' :
       System.out.println("You passed");  break;
    case 'F' :
       System.out.println("Better try again");  break;
    default :
       System.out.println("Invalid grade");
}
System.out.println("Your grade is " + grade);
```

From Java 7, a string literal / constant can be used to control a switch statement

DECISION MAKING – SWITCH CASE WITH STRINGS

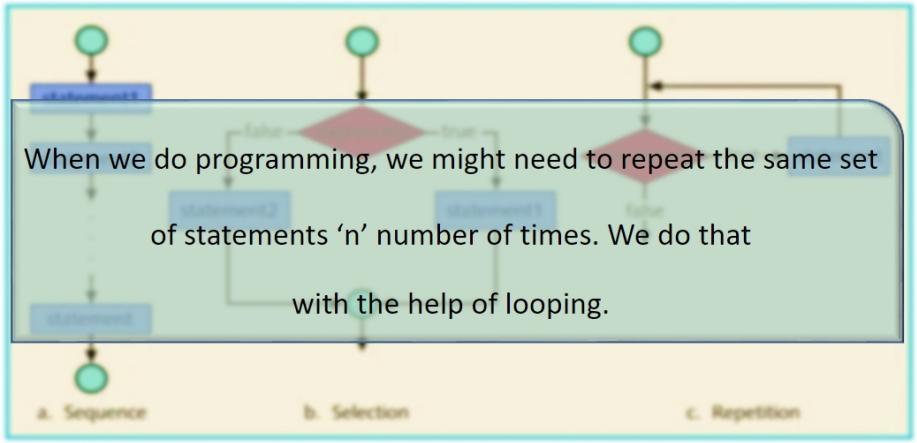
From java 7 a String literal constant can also be used to control a switch statement

```
public class UserInterface {

    public static void main(String a[]) {

        String value = "Three";
        switch (value) {
        case "One": {
               System.out.println("The value is: " + 1);
               break;
        }
        case "Two": {
               System.out.println("The value is: " + 2);
               break;
        }
        case "Three": {
               System.out.println("The value is: " + 3);
               break;
        }
        case "Four": {
               System.out.println("The value is: " + 4);
               break;
        }
        }
    }
}
```

Output
The value is: 3

Note:
The String in the switch expression should be exactly equal (case-sensitive) to the case label expression.

Overview



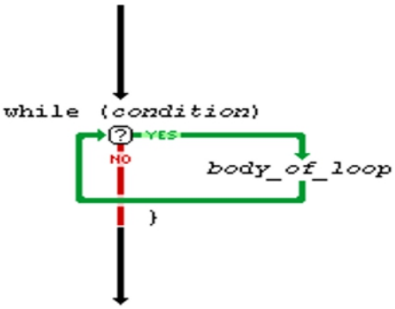
When we do programming, we might need to repeat the same set of statements ‘n’ number of times. We do that with the help of looping.

While Loop

Example

```
int count = 1;
while (count < 11) {
    System.out.println(count);
     count++;
}
```

SYNTAX

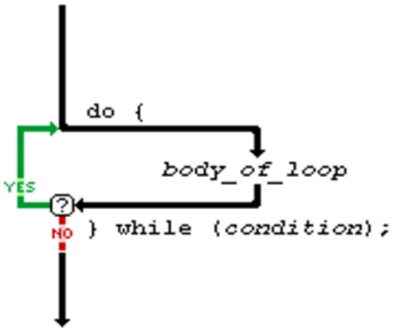


Do-While Loop

Example

```
int count = 1;
    do {
       System.out.println(count);
       count++;
    } while (count < 11);
```

SYNTAX

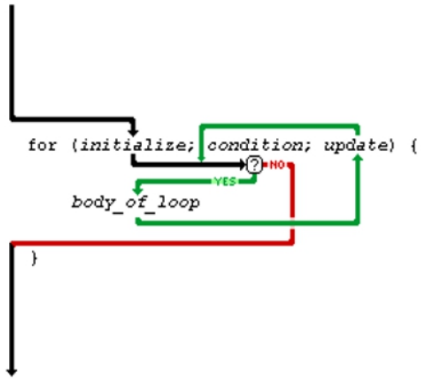


For Loop

Example

```
for(int i=1; i<11; i++) {
      System.out.println("Count is: " + i);
}
```

SYNTAX



break statement

break statement

- Used to terminate a loop as well as switch - case block
- When executed, the control flow will be transferred to the statement that follows after the end of the loop
- Example

```
int a=0,b=1,c=0,cnt=2;
System.out.print(a+", "+b);
while(cnt<20)
{
    c=a+b;
    System.out.print(", "+c);
    if(c%11==0)
        break;
    cnt++;
    a=b;
    b=c;
}
```

Output :

0,1,1,2,3,5,8,13,21,34,55

continue Statement

continue statement

- Used to bypass the execution of rest of the statements in the current iteration of a loop
- It skips to the end of the inner loop and continues the loop
- Example

```
int a=0,b=1,c=0,cnt=2;
System.out.print(a+", "+b);
while(cnt<15)
{
    c=a+b;
    cnt++;
    a=b;
    b=c;
    if(c%11==0)
        continue;
     System.out.println(", "+c);
}
```

Output :

0,1,1,2,3,5,8,13,21,34, 89,144,233,377

Labeled Statements

Labeled Statements - Example

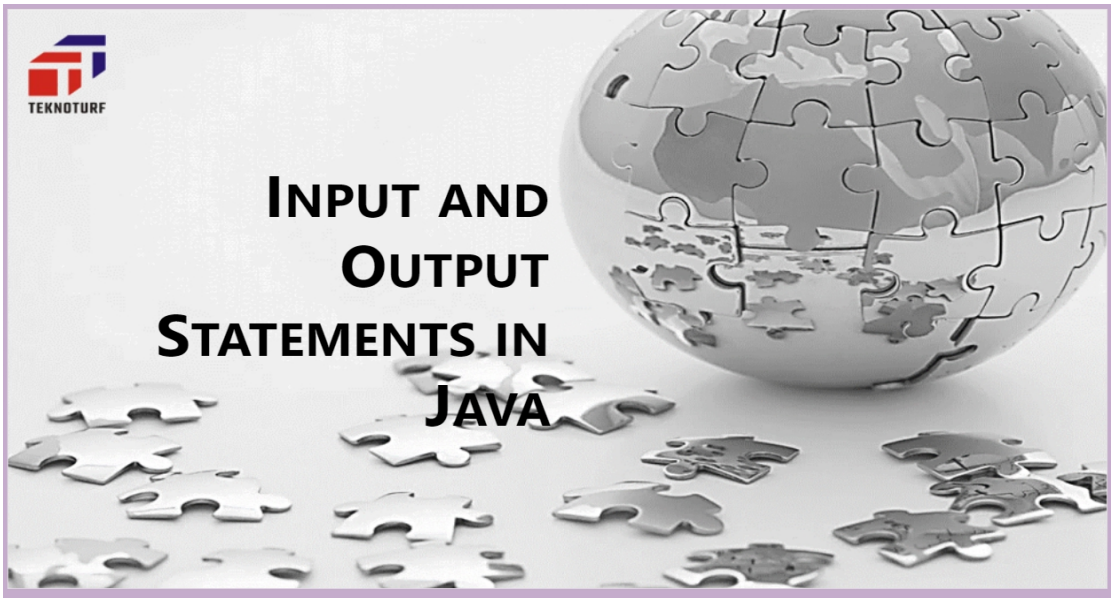
Example

```
boolean isTrue = true;
outer:
  for(int i=0; i<5; i++) {
    while (isTrue) {
        System.out.println("Hello");
        break outer;
    } // end of inner while loop
    System.out.println("Outer loop."); // Won't print
  } // end of outer for loop
System.out.println("Good-Bye");
```

Running this code produces

Hello

Good-Bye



OUTPUT STATEMENT

TEKNOTURF

Various ways to display output on the monitor Using system.out

System.out.println()
System.out.print()
System.out.printf(y)

- System is a predefined class
- out is an object of the predefined class PrintStream
- This method takes a String or a primitive data type or an Object as argument
- println and print method takes a String or a primitive data type or an Object as parameter
- println and print displays the argument passed as parameter
- println adds a new line after printing the data
- print method is similar to println, except that , it does not add a new line.

USAGE OF PRINTLN VS PRINT

TEKNOTURF

When using println, new line is added

```
public class Test
{
    public static void main(String args[])
    {
        String name = "Peter";
        int id=1001;
        double salary = 85250.75;

        System.out.println("Welcome "+name);
        System.out.println("ID is "+id);
        System.out.println("Your salary is "+salary);

    }
}
```

Output

Welcome Peter
ID is 1001
Your salary is 85250.75

When using print, data is printed in the same line ie..new line is not added

```
public class Test
{
    public static void main(String args[])
    {
        String name = "Peter";
        int id=1001;
        double salary = 85250.75;

        System.out.print("Welcome "+name);
        System.out.print("ID is "+id);
        System.out.print("Your salary is "+salary);

    }
}
```

Output

Welcome PeterID is 1001Your salary is 85250.75

USAGE OF ESCAPE SEQUENCES

Escape Sequences

A character preceded by a backslash (\) is an escape sequence and has special meaning to the compiler.

Escape Sequence	Description
\t	Insert a tab in the text at this point.
\b	Insert a backspace in the text at this point.
\n	Insert a newline in the text at this point.
\r	Insert a carriage return in the text at this point.
\f	Insert a form feed in the text at this point.
\'	Insert a single quote character in the text at this point.
\"	Insert a double quote character in the text at this point.
\\	Insert a backslash character in the text at this point.

USAGE OF ESCAPE SEQUENCES

Sample Code for Usage of common escape sequences like \n, \t, \' and \"

```
public class Test
{
    public static void main(String args[])
    {
        String name = "Peter";
        int id=1001;
        double salary = 85250.75;
        double nettSalary=salary-salary*0.15;

        System.out.println("Welcome \""+name+"\"");
        System.out.print("ID is \'"+id+"\'.\nYour salary is "
        +salary+".\tYour nett salary is "+nettSalary);

    }
}
```

Output

```
Welcome "Peter"
ID is '1001'.
Your salary is 85250.75.   Your nett salary is
72463.1375
```

PRINTF STATEMENT

PRINTF METHOD USED TO DISPLAY OUTPUT IN A FORMATTED MANNER

Example : To print the net salary of an employee with multiple digits in decimal to 2 decimal places, requires formatting which needs to be mentioned in the format

Syntax : System.out.printf(format, arguments)

Specify the formatting rules using the format specifier.

Rules start with the '%' character.

The format specifiers for general, character, and numeric types have the following syntax:

%[argument_index$][flags][width][.precision]conversion-character

Here except the conversion-character all the rest are optional

PRINTF STATEMENT

The [argument_index] is a decimal integer that specifies the position of the argument in the argument list. The first argument is referenced by "1$", the second by "2$", etc.

The [flags] define standard ways to modify the output and are most common for formatting integers and floating point numbers

The [width] specifies the field width for outputting the argument. It represents the minimum number of characters written to the output

The [.precision] specifies the number of digits of precision when outputting floating-point values. Additionally, we can use it to define the length of a substring to extract from a String.

The conversion-character determines how the argument is formatted. Conversion characters are only valid for certain data types. Some common characters are :

- s - formats strings
- d - formats decimal integers
- f - formats the floating-point numbers
- c – formats character
- t - formats date/time values
- n – line separator

PRINTF STATEMENT

Example :

```
public class Test
{
    public static void main(String args[])
    {
        String name="Raghu";
        int id = 1001;
        float salary = 72463.1375f;
        boolean status = true;

        //%2$ - Here 2 specifies the 2nd argument name, %3$,.2f – 3$ means 3rd argument salary
        //"," specifies the flag, ".2f" specifies correct to 2 decimal places
        System.out.printf("%2$s %3$,.2f %1$d %b",id,name,salary,status);

        System.out.printf("%n%d %S %,.2f %B",id,name,salary,status);

        System.out.printf("\n\nID is %d %nName is %s. Salary is %.2f",id,name,salary);
    }
}
```

Output

```
Raghu 72,463.14 1001 true
1001 RAGHU 72,463.14 TRUE

ID is 1001
Name is Raghu.
Salary is 72463.14
```

STRING FORMATTING

- %s is the format specifier for a simple string

- To specify a length for the string, a width can be specified
  Example : ***System.out.printf("%15s","Welcome");***
  Output : ' Welcome'

- To left justify the string, we can use '-' flag
  Example : ***System.out.printf("%-15s","Welcome");***
  Output : 'Welcome '

STRING FORMATTING

Example :

```
public class Test
{
    public static void main(String args[])
    {
        String name="Raghu";
        int id = 1001;
        float salary = 72463.1375f;

        System.out.printf("%-5s %-15s %-15s\n","ID","Name","Salary");

        //Convert primitive to String using String.valueOf
        String str1 = String.format("%-5s %-15s %-15s",String.valueOf(id),name,String.format("%.2f",salary));
        System.out.println(str1);


        System.out.printf("%-5s %-15s %-15s\n","ID","Name","Salary");
        //Alternate way to convert primitive to String
        String str2 = String.format("%-5s %-15s %-15s",id+"",name,""+String.format("%.2f",salary));
        System.out.println(str2);
    }
}
```

Output

```
ID    Name           Salary
1001  Raghu          72463.14

ID    Name           Salary
1001  Raghu          72463.14
```

STRING FORMATTING

Example :

```
public class Student {

    private int studentId;
    private String name;
    private float cgpa;
    private float tutionFees;

    public Student(int studentId, String name, float cgpa, float tutionFees) {
        this.studentId = studentId;
        this.name = name;
        this.cgpa = cgpa;
        this.tutionFees = tutionFees;
    }

    public String toString(){
        return String.format("%-15s %-20s %-5s %-15s",String.valueOf(studentId),
                name,String.valueOf(cgpa),String.valueOf(tutionFees));
    }

    public static void main(String arg[]){
        Student s1=new Student(101,"Raghu",9.7f,30000);
        System.out.printf("%-15s %-20s %-5s %-15s\n","Student ID","Name","CGPA","Fees");
        System.out.println(s1);
    }
}
```

Output

```
Student ID      Name                 CGPA  Fees
101             Raghu                9.7
30000.0
```

READ INPUT FROM USER

- In the real world, it is necessary for programs to get input from the user
  Input can be read from keyboard in many ways
- One such way is to use the inbuilt class Scanner
- Predefined / Inbuilt classes in Java are organized in the form of packages.
  Scanner class is present in java.util package
- To use Scanner class, include package java.util using import statement as :
  import java.util.Scanner ;
  // imports just the Scanner class

  or

  import java.util.*;
  // imports the entire package java.util
- Next create a Scanner object
  Scanner sc = new Scanner(System.in); // System.in indicates input will
  //be given to System from keyboard

```
import java.util.Scanner;

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

        System.out.println("Enter id");
        int id = sc.nextInt();
        System.out.println("Enter name");
        String name = sc.next();

        System.out.println(id+"  "+name);
    }
}
```

READ INPUT FROM USER

Scanner class has various methods to get input of various data types from the user

Method	Data type
nextInt()	Integer
nextFloat()	Float
nextDouble()	Double
nextLong()	Long
nextShort()	Short
nextBoolean()	Boolean
next()	Single word (String without space)
nextLine()	String with spaces

```
import java.util.Scanner;

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

        float salary = sc.nextFloat();
        int count = sc.nextInt();
        byte b1 = sc.nextByte();
        long phone = sc.nextLong();
        boolean status = sc.nextBoolean();
        double price = sc.nextDouble();
        String word = sc.next();
        String sentence = sc.nextLine();
    }
}
```

READ INPUT FROM USER

READ STRING

String is a collection of characters.

next() method in Scanner reads a String, with only one word

Even if the input has multiple words, next() takes the first word alone as input

To get a String as input with spaces, use the nextLine() method.

```
import java.util.Scanner;

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

        System.out.println("Enter the name");
        String name = sc.nextLine();

        System.out.println("Enter the city");
        String city = sc.next();


        System.out.println("Name is "+name+". City is "+city);
    }
}
```

READ STRING USING NEXTLINE

- ✓ Observe the code below
- ✓ On executing the above code, city will not be taken as input. Scanner skips the nextLine()

```
Scanner sc = new Scanner(System.in);

System.out.println("Enter the name");
String name = sc.next();

System.out.println("Enter the city");
String city = sc.nextLine();

System.out.println("Name is "+name+". City is "+city);
```

- ✓ Scanner skips nextLine() when it is used after next() or after any nextXXX() method.
- ✓ This is because next() reads only the String, not the newline (when Enter key is pressed) nextLine() consumes the nextline(\n) as input. Hence no input is received.
- ✓ Easy way to resolve :

  When using nextLine after next() or nextXXX(), consume the newline by including the statement sc.nextLine().

READ STRING USING NEXTLINE

TEKNOTURF

USING NEXTLINE() AFTER NEXT() OR NEXTXXX()

```
import java.util.Scanner;

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

        System.out.println("Enter the id");
        int id = sc.nextInt();
        sc.nextLine(); //captures the newline(Enter key)
        System.out.println("Enter the name");
        String name = sc.nextLine();

        System.out.println("Enter the salary");
        double salary = sc.nextDouble();
        sc.nextLine(); //captures the newline(Enter key)

        System.out.println("Enter the city");
        String city = sc.nextLine();

        System.out.println("Name is "+name+". City is "+city);
    }
}
```

- After getting the id using nextInt we have the name using nextLine.
- nextInt captures only the number and hence nextLine captures the new line.
- So it will skip getting input for name and get the input for salary.
- To solve this issue, we have included the statement sc.nextLine to capture the new line.
- Similarly we have obtained the input for salary and city.