**1. Token**

**2. Data Types.**

**3. Variables**

**4. Arrays**

**5.Operators.**

**6.Control Statements.**

**7. Type Casting.**

**TOKEN**

**1.Token:**The smallest Individual unit in java program is known as Token. Java language includes 5 types of tokens.

1. Reserved words/keywords.

2. Identifiers.

3. Literals.

4. Operators.

5. Separators.

**1.1.Reserved Words**:- Keywords are predefined identifiers. The keywords ,combined with operators and separators according to syntax. They have special meaning.

Note:-1. we specify operation to compiler and JVM by Keyword.

2. In java, we have 50 keywords. Among 50 keywords 2 keywords are unused keywords because keyword is defined but they are not implemented, those are const,goto.

3.null,true,false is not keyword. Although it is not used as identifier. Null ,true and false is literal.

Rules:-

1. Keywords can not be used as user defined identifier.

Example: variable name, class name and method name…etc.

1. All keywords must be in lower case. The keywords are case-sensitive.

Exercise:

1. Identfify the invalid keywords.

a)Class(Invalid) b) init(Invalid) c) short 4) try 5)void 6)Public(Invalid)7)Try(Invalid) 8)If (Invalid) 9)if 10)switch 11) swtch(Invalid) 12)For(Invalid) 13)while 14) Double(Invalid)15) While. (Invalid)

2. Write about the null.

**1.2.Identifiers**:- Identifier is name of Basic Programming Element.

Rules:-

* Identifier should only contain
* Alphabets[a to z] [A to Z]
* Digits[0-9]
* Special characters[\_ or $].
* Identifier should not start with digit.
* Identifier is case sensitive.
* Keyword can not be used as user defined identifier.
* There is no limit in identifier length.

Exercise:

Find the valid identifiers.

1. Hai123
2. Sample
3. 1hhh
4. Student\_name
5. \_student
6. Student$name
7. Student\*nam
8. Student-name
9. E9mploy.

**1.3.separator:-** The separator is symbol. It separates one element from another element in java program. There are 6 separators in java.

1) parentheses(): Used to enclose parameters in method defition and method invocation statements.

2) {}: It is used to define block of code for class ,method and local scope.

3) brackets[]: used to declare array type and for referencing array value.

4) semicolon: used to separate statements.

5) comma: used to separate consecutive identifiers in variable declaration.

6) period: Used to separate package names from sub-packages and also used to separate variable or method from reference variable.

**1.4. Literal**:-

The literal is single character or sequence of characters(letters,digits & other characters) that represent value to be stored in variable name. There are 5 major types of literals.

1.Integer literal

2.Floating\_point literal.

3.Character literals.

4.String literals.

5.boolean literals.

6.Backslash character literals.

**DATA TYPES**

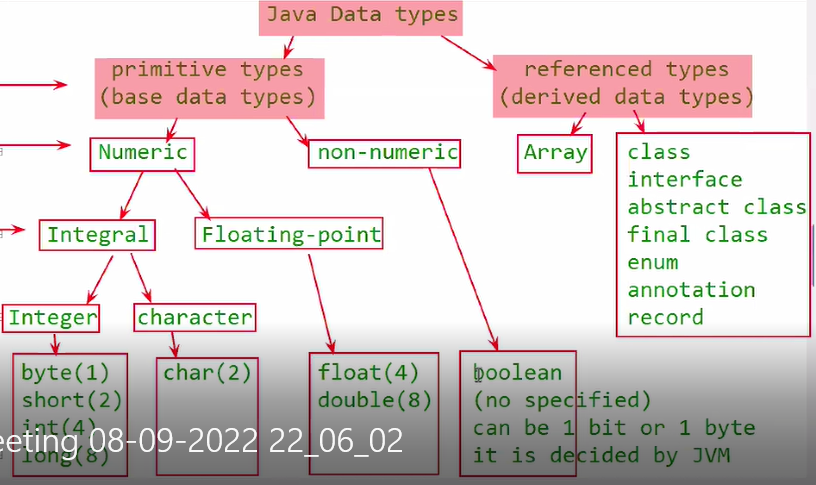
**2.Data Types:-**The keyword or class name which is used for creating variable or object is called data type. The data type keyword will inform compiler and JVM the following information :

* Size of memory to be created for variable .(1/2/4/8 bytes).
* Type of value that can be stored in memory .
* Range of value that can be stored in memory .

**2.1 Java Data Types:** Java supports two types of Data types.

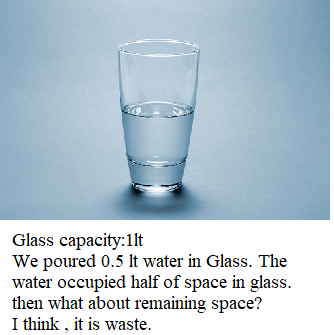
2.1. Primitive Types(Base Data types) .

2.2. Referenced Types(Derived Data types)



**2.1.1. Primitive Data Types:-** We use primitive data types(PDT) for storing a single value. As per mathematical values, the primitive data types were divided into Numeric and non-numeric.

Based on type of value, the int, float, character and Boolean data types are given.



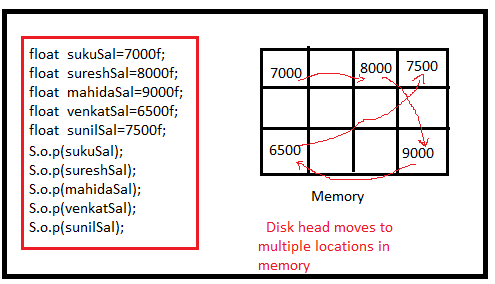
Let us assume, There is only one data type for all integer values. Its size is 4 bytes. Its range is -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807. If we store the 50(or) small value in such type variable, then It requires more time for manipulation , space also wasted and automatically it will affect the speed of execution of program.

Therefore Based on range of value, the byte, short, long and float data types were given.

Limitation of Primitive Types:

1. Primitive Type variable holds only one value.

2. I want to store 5 employees salaries in program. I will declare 5 variables and I will store 5 salaries in 5 variables.



Problem-1: It is difficult to remember all variable names.

Problem-2: program execution time increases.

3.we cannot pass all values to remote computer with single network call, which increases burden on network.

The above limitations and problems have been over came by Referenced Types.

**2.1.2. Referenced Data Types(RDT):-**

* The array must be used for storing multiple values of same type.
* The class must be used for storing multiple values of different type.
* The enum must be used for storing multiple constant values with different names.

**2.2.Integer Data Types:-** java does not support unsigned data type. Therefore all values are signed.

a)Byte:- :- It occupies one byte space in memory. Range is -27 to 27-1(-128 to 127).

b)short:- It occupies 2 bytes space in memory. Its range is -215 to 215-1.

**By Default numeric values are treated as integer types values.**

c)Int:- It occupies 4 bytes space in memory. It contains value which should be inbetween -231 to 231-1.In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java.

d)long:- :- It occupies 8 bytes space in memory. Its range is -263 to 263-1. we can make integers long by appending letter L or l at end of number.

Ex:- 123L or 123l.

Note:-1. Integer Data type variable hold decimal ,octal and hexadecimal number.

2. Lower Range DT variable value can be stored in Higher Range variable.

3. Higher Range DT variable value can’t be stored in Lower Range Variable. When it is tried, we get CE(i.e Possible lossy conversion HRDT to LRDT).

4. If you assign out of range value to variable then we get CE( DT number too large)

Example:1

class Sample

{

public static void main(String[] args)

{

byte a=10;

short b=256;

int c= 1012;

long d= 123456L;

System.out.println(a);

System.out.println(b);

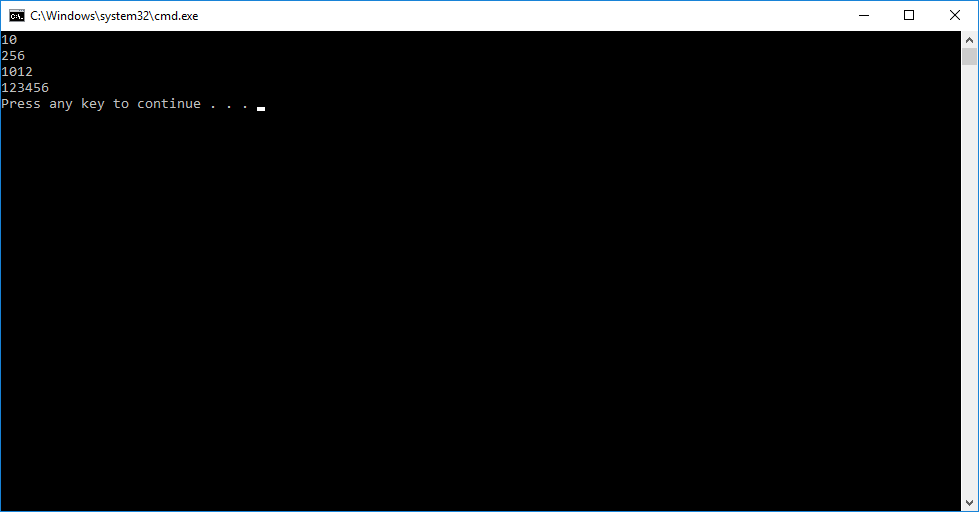
System.out.println(c);

System.out.println(d);

}

}

Output:



Example: 2

class Sample

{

public static void main(String[] args)

{

byte a=011;

short b=023;

int c= 04567;

long d= 0234523L;

System.out.println(a);

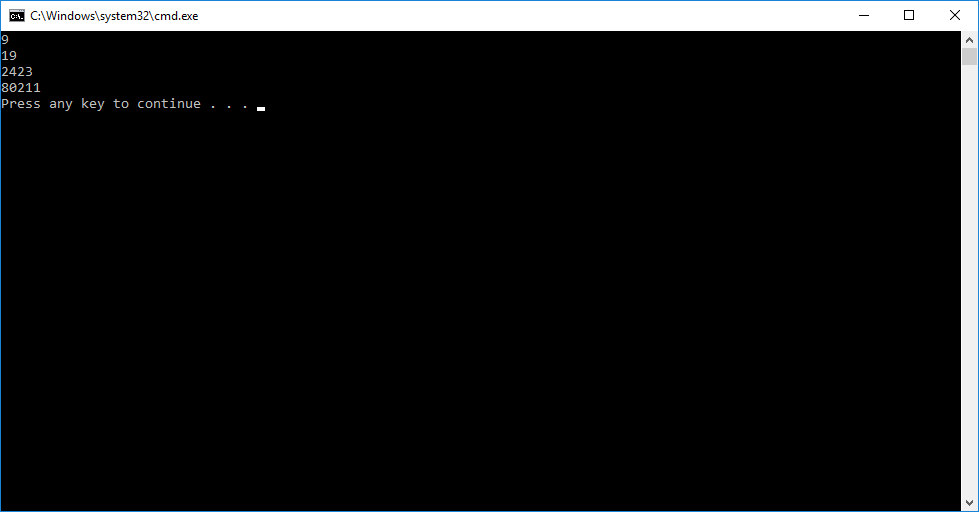
System.out.println(b);

System.out.println(c);

System.out.println(d);

}

}



Example:3

class Sample

{

public static void main(String[] args)

{

byte a=0X11;

short b=0X23;

int c= 0X4567;

long d= 0X234523L;

System.out.println(a);

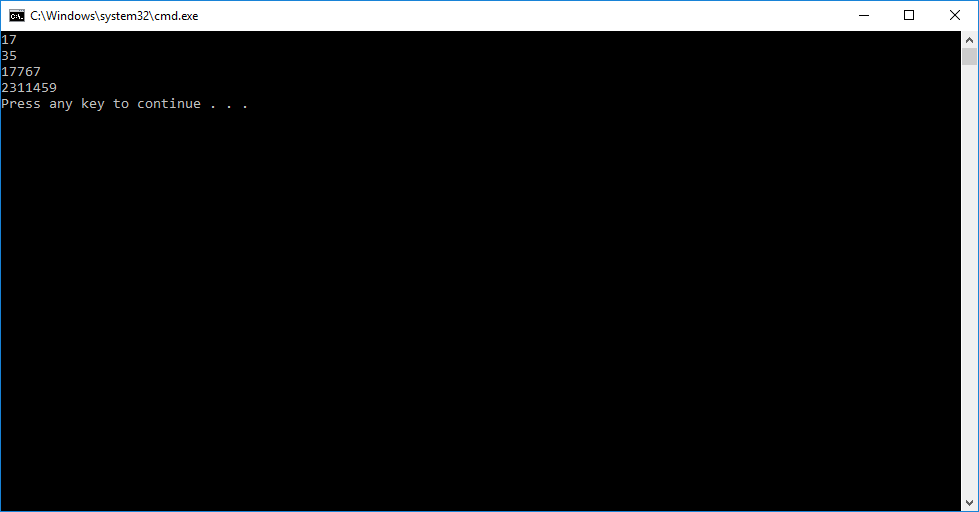
System.out.println(b);

System.out.println(c);

System.out.println(d);

}

}



Note: - even we stored octal or hexadecimal literal to variable, we got decimal numbers. Because they are converted into decimal numbers and then into binary before storing them in memory.

Example:4

class Sample

{

public static void main(String[] args)

{

byte a=2000;

short b=33333;

int c= 123456789123;

long d= 12223334455666777888991122233444L;

System.out.println(a);

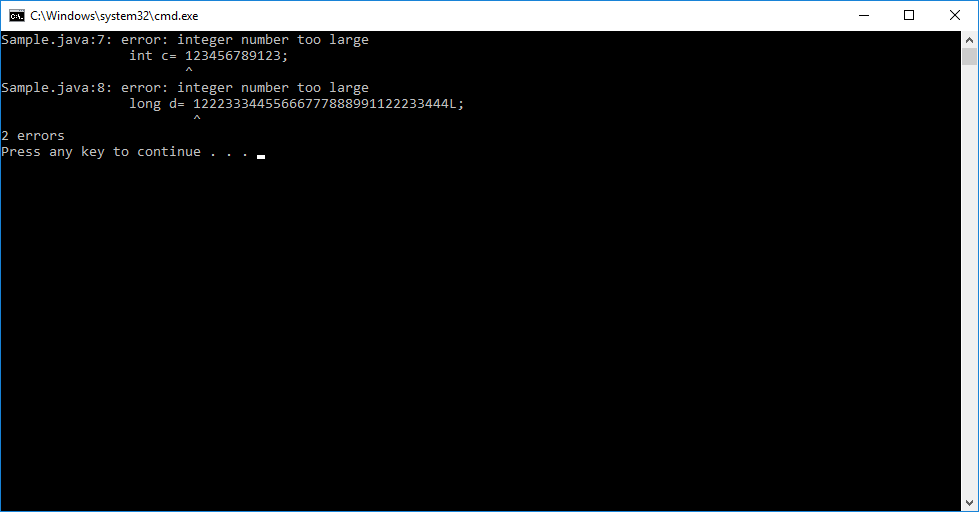
System.out.println(b);

System.out.println(c);

System.out.println(d);

}

}



Example:5

class Sample

{

public static void main(String[] args)

{

byte a=-20;

short b=-3333;

int c= -123456;

long d= -12223334455666777L;

System.out.println(a);

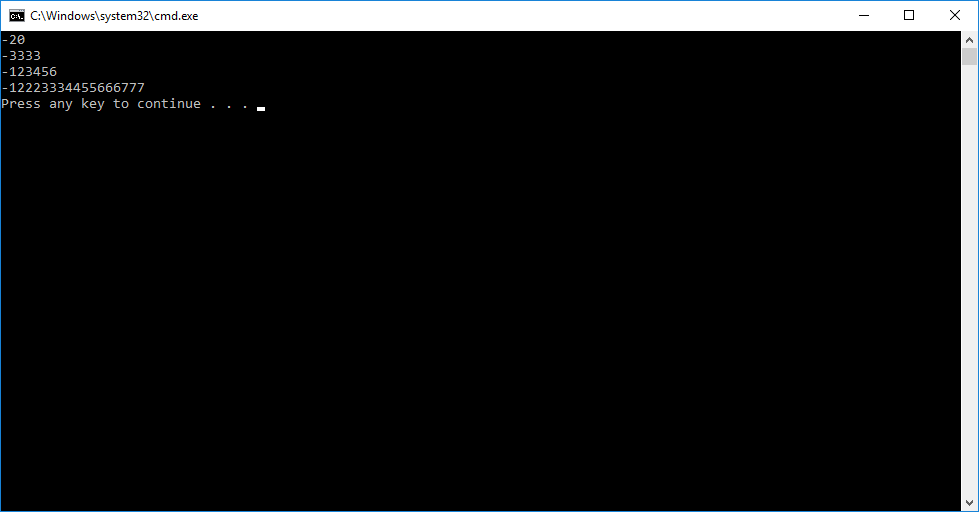
System.out.println(b);

System.out.println(c);

System.out.println(d);

}

}



Example:6

class Sample

{

public static void main(String[] args)

{

byte a=10;

short b=255;

int c=33456;

long d=5444343L;

d=c;

d=b;

d=a;

c=b;

c=a;

b=a;

// HRDT variable value will be place LRDT variable.

a=d;

b=d;

c=d;

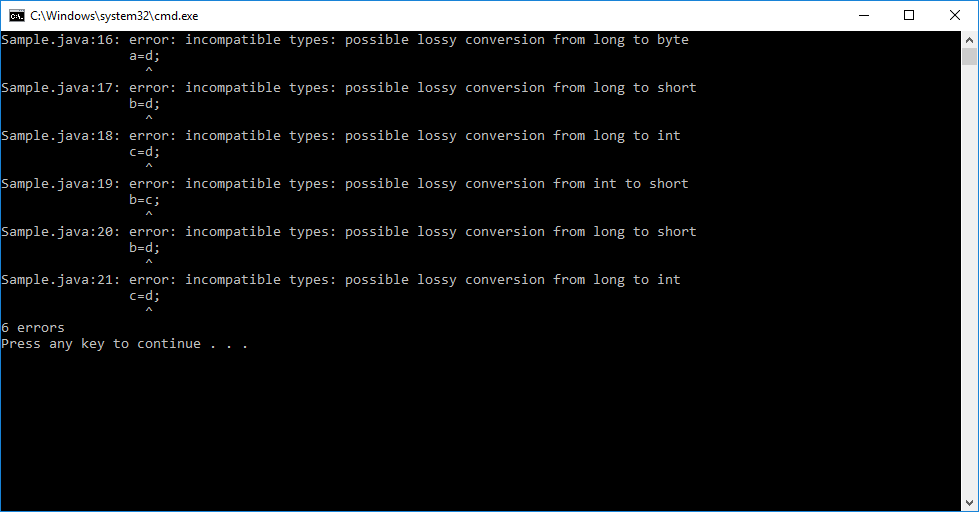
b=c;

b=d;

c=d;

}

}



**2.3.Character Type**:- It stores only single character. If assigned number’s corresponding character is not supporting by your computer JVM internally stores special character “?”.

The size of char is 2 bytes.

The range is -215 to 215-1.

We store the single character within single quotes in character type variable or

We store the integer number in character type variable.

Q. why does char occupies 2 bytes instead of 1 byte?

A. The c /c++ is ascii based language. Java is Unicode based language. c/C++ allows 0-9,A-Z,a-z, and special characters. All these characters are ASCII characters.To represent ASCII Characters, 1 byte space is enough. The no.of Unicode characters is >256 and <65535. Therefore to represent those Unicode character,1 byte is not enough. We need 2 bytes to represent Unicode characters.

Example:

class Sample

{

public static void main(String[] args)

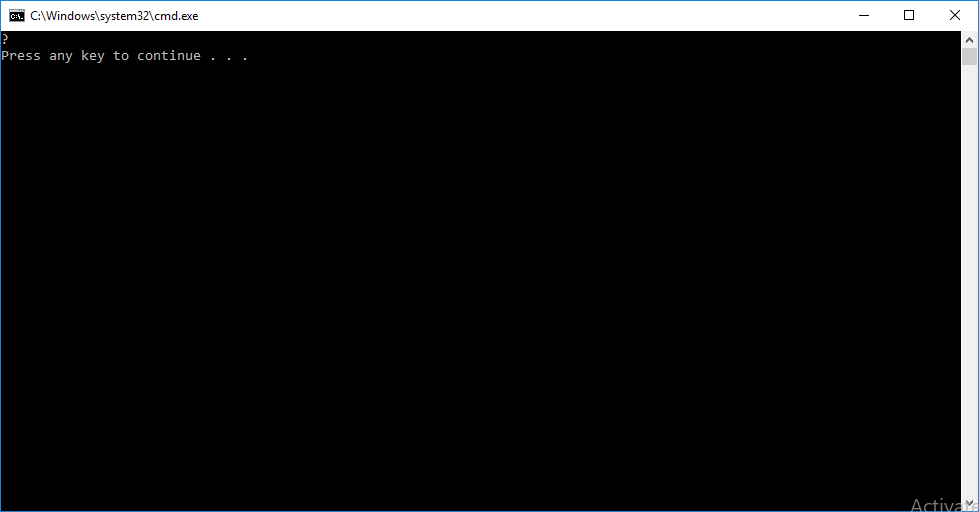
{

char a=459;

System.out.println(a);

}

}



Example:2

class Sample

{

public static void main(String[] args)

{

char a='N';

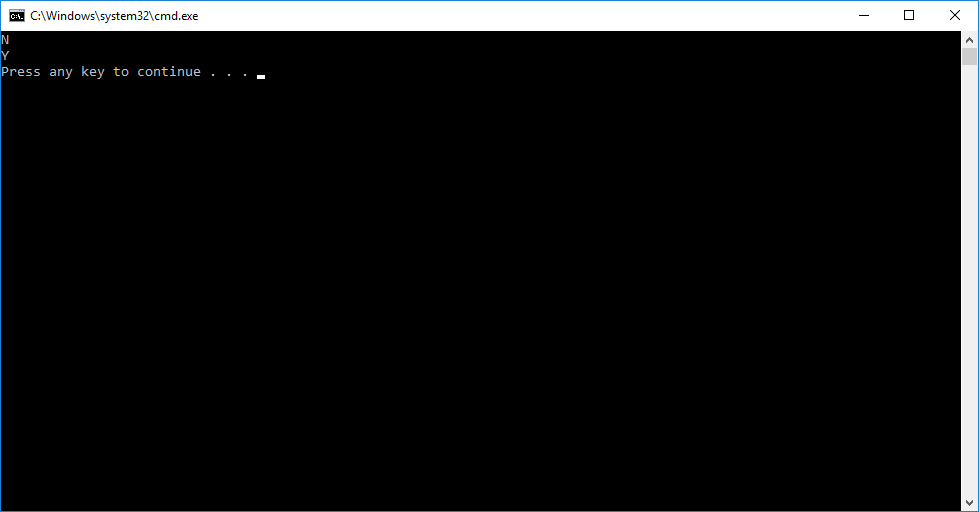
char b=89;

System.out.println(a);

System.out.println(b);

}

}



Note:- The Character Data Type is compatibility to integer Data types. The character DT variable value can be assigned to HRDT integer variable.

Example:-

class Sample

{

public static void main(String[] args)

{

char a='N';

int b=a;

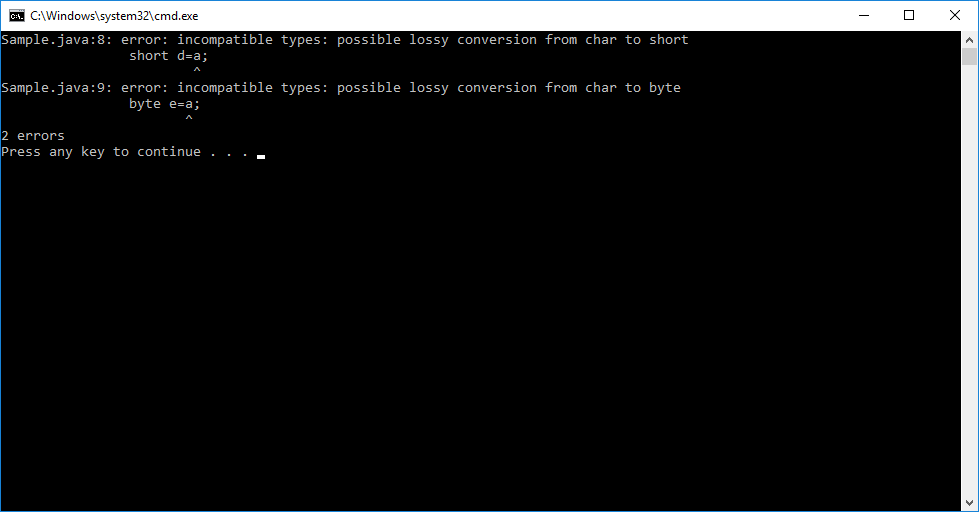
long c=a;

short d=a;

byte e=a;

}

}



**2.4. Boolean Data Type**:- The Boolean type can take either true or false. Boolean variable does not take 1 or 0.The size of Boolean depends on the JVM. In some JVM, Boolean occupies 1bit space. In some JVM, Boolean occupies 1 byte space. Therefore size and range is not applicable to Boolean type.

Example:1

class Sample

{

public static void main(String[] args)

{

boolean a=true;

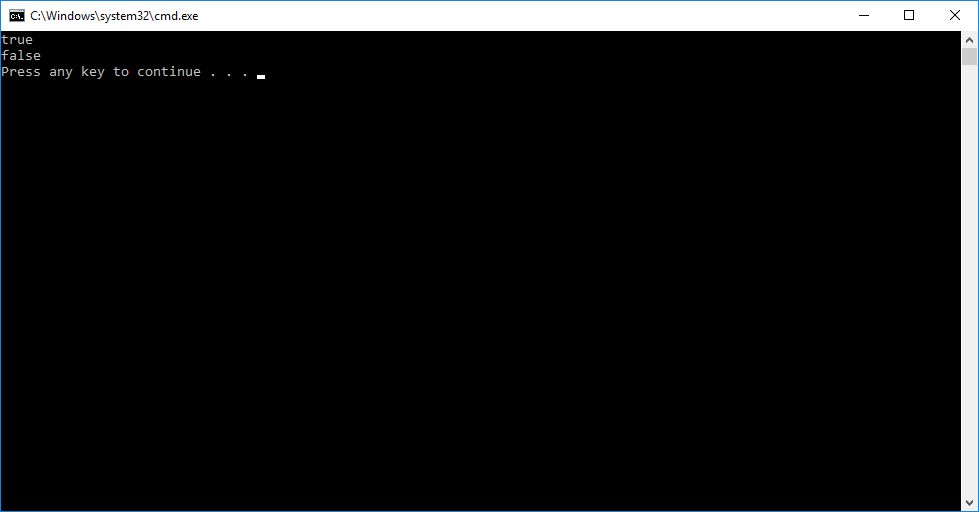
boolean b=false;

System.out.println(a);

System.out.println(b);

}

}



Example:2

class Sample

{

public static void main(String[] args)

{

boolean a=1;

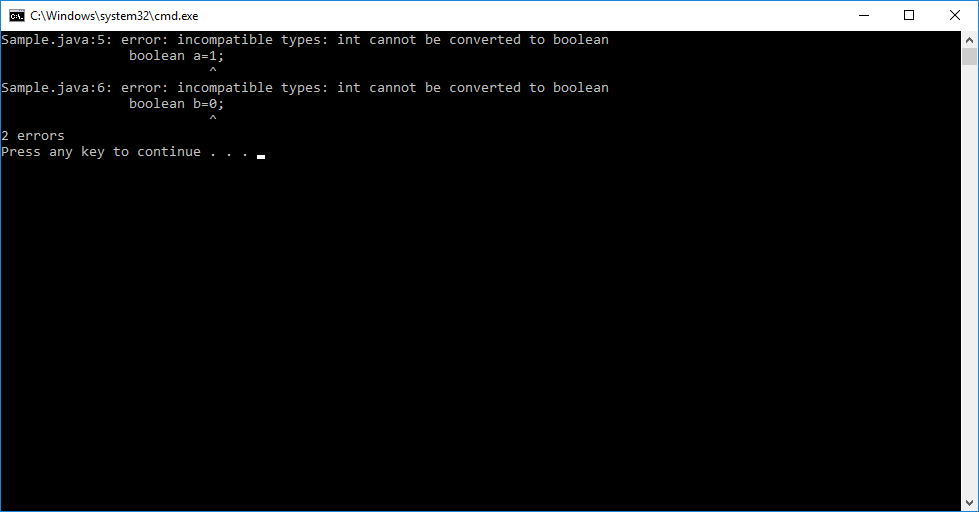
boolean b=0;

System.out.println(a);

System.out.println(b);

}

}



**2.5. Float**:- It occupies 4 bytes space. It range is -231 to 231-1

Example: 12.3f, 15.3F,…etc.

Example:1

class Sample

{

public static void main(String[] args)

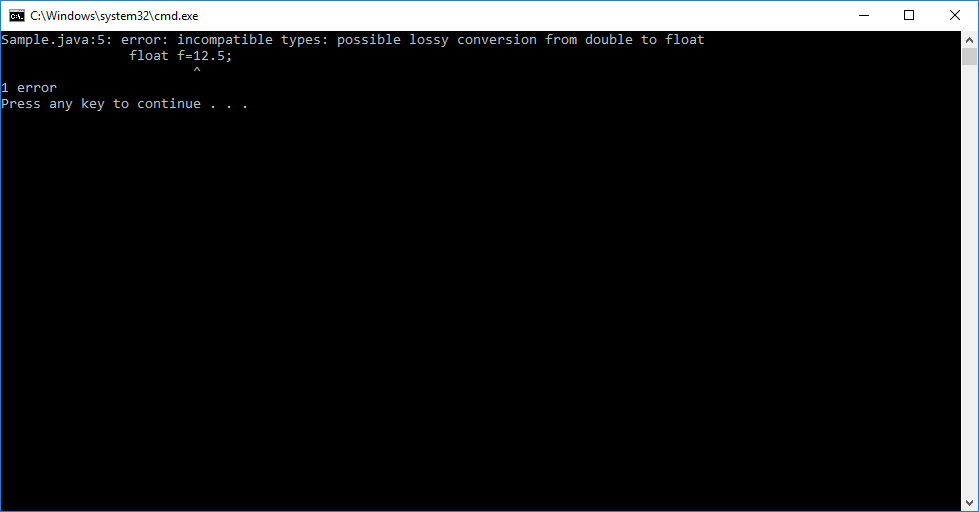
{

float f=12.5;

System.out.println(f);

}

}



Why did we get error?

With out either ‘f’ or ‘F’, if we assign value to float type variable then value is considered as double. Double is higher range DT than Float. That’s why we got error.

Example:2

class Sample

{

public static void main(String[] args)

{

float g=12.5F;

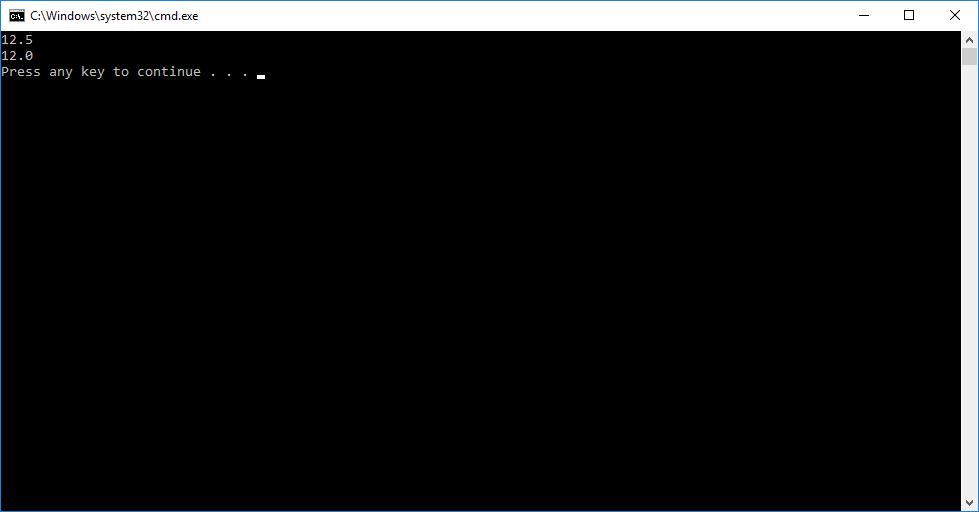
float h=12f;

System.out.println(g);

System.out.println(h);

}

}



**2.6.Double:**  It occupies 8 bytes memory space. Its value range is -263  to 263-1.

Example:1

class Sample

{

public static void main(String[] args)

{

double g=12.5;

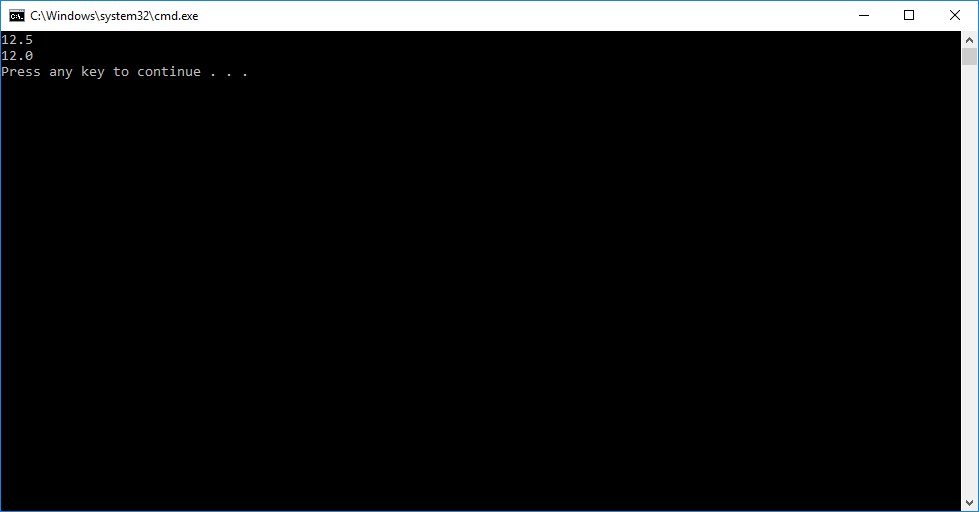
double h=12;

System.out.println(g);

System.out.println(h);

}

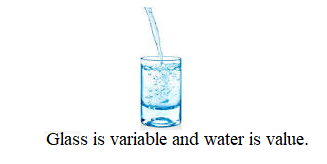
}



**VARIABLES**

**3.Variable:**

**3.1.Definition:**The variable is named memory location. That is used for storing **one value or one object reference temporarly.** The variable can have different value at different time during the program execution.



**3.2. Variable Declaration:**  Creating a variable without value is variable declaration.

Basic Syntax:

Datatype variablename;

**3.** 3.**Variable Initialization:** Storing a value in newly created variable is called variable initializating.

Storing initial value in variable is called variable initialization.

Syntax:

Variablename=value;

**3.4. Defining a variable:**  Declaring and initializing variable in single line is variable defining.

Syntax:

Datatype variablename=value;

**3.5. Variable Types:** Based on storing value in variable, The variables have been divided into two types.

3.5.1. primitive Varible.

3.5.2. Reference Variable.

**3.5.1.Primitive Variable:-** If variable holds PDT(primitive data type) value then variable is said to be primitive variable.

**3.5.2 Reference Variable:-** If variable holds reference/address of object then variable is siaid to be referenced Varaible.

**3.5.2.1 Types of Reference Varibale:-** we have two types of referenced variables based on value we have stored.

1.null referenced variables.

2. object referenced variables.

A) null referenced variables:- if we store ‘null’ value in referenced variable, it is called null referenced variable.

Syntax:

RDT var-name=null;

If we print null referenced variable ,**print() or println() method prints ‘null’**.

B) object referenced variables:- if we store “ object reference” in referenced variable, It is called object referenced variable.

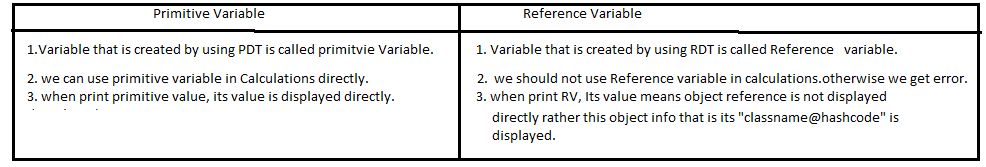
Syntax:

RDT var-name=new RDT();

If we print object referenced variable, print() or println() method print following:

**Classname@hashcode of object in hexadecimal string format**.

Diff B/W primitive and Reference Variable:



**3.6.Rules For Accessing Variable**:

1. The variable must be declared in same line or previous line to assing value,else we get CE.

Example:

B=20; //CE

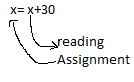
2. The variable must be initialized in previous line for accessing its value, else we get CE.(variable might’have been intialized).

Example:

C=12;//CE

Exercise on PDT Variables:

1. Int x;//variable declaration
2. Int y;//variable declaration.
3. Float z=23.4f; variable defintion.
4. X=24;//Initialization.



1. Y=x; //copying.
2. Double z;

Sop(z);//CE.

Note1: we can assign value to variable with directly literal, another variable or method call.

Syntax:

Variable-name=literal;

Variable-name=another variable-name;

Variable-name=method-name();

On variable ,we basically performs two operation.

1. getting existing value from variable.

2. storing value into variable.

3. creating variable

4. Modifying existing value in variable.

5. copying value from another variable.

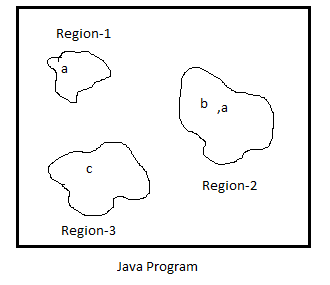
**3.7.LifeTime and Scope of Variable:**

**3.7.1.LifeTime:** The lifetime is time period between varible memory creation and destruction.

Example:



**3.7.2.** **Scope**: The scope is region in which only the variable can accessible.



**ARRAYS**

**4. Array:-** Array is a Container Object. Array contains homogeneous elements. This means that all the elements in array are of same type. These elements are stored in consecutive memory locations.

Q)Why do we use Array?

Imagine we are developing application of T-shirt shop. This shop sales only 3color t-shirts- black,white and orange. We store 3 color name in 3 variables of application.

String col1=”black”;

String col2=”white”;

String col3=”orange”;

When we want to store 3 color names in consecutive memory locations and we want to give single common name to all memory locations, we should use array.

Q. What is index?

A. Index is number/digit. It indicates position/location of element in array.

Note:-1. Array is not keyword. It is concept.

**Limitations:**

1. Array size is fixed i.e We can’t increase or decrease size of array after its creation.

2. Insertion and deletion processes are tricky in arrays.

3. We can’t store different data types values in arrays.

**Application of Array:**

1. Arrays are used in implementing in CPU scheduling algorithms. Array can store the processes .
2. Arrays are used in implementing the other DS like stack,queue,…etc.

**4.1. Declaring Array:-**

Syntax:-1

datatype[] variable-name;

* we can place the [] after data type and before variable name.
* we can place the [] after the variable name.

Example:

Int [] a;

Where a is integer type array variable. It holds reference of integer Array object.

Examples to valid array variable declarations:

1. Int a[]; ------valid.
2. Float[]a; ------valid.
3. Int [][]a;---------valid.
4. Float b[][];-------valid.
5. Int [] a,b;------valid. Both are type of [].
6. Int [][]a,b;----valid. Both are two dimentional array.
7. Int []a[],b;----------valid. A is two dimentional array and b is single dimentional array.

Note:- After declaring Array, we can’t assign values to array.But we can assign the array reference.

Example1:

class A

{

public static void main(String arg[])

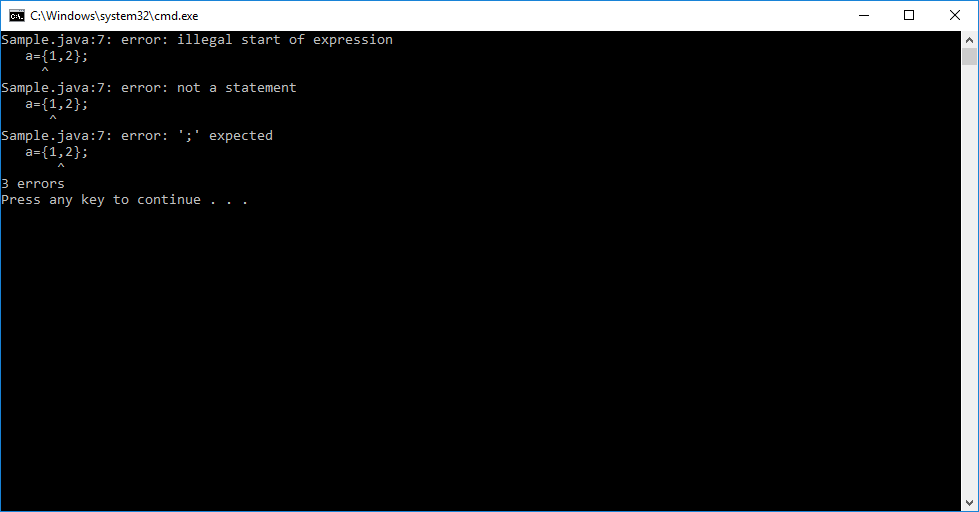
{

int a[];

a={1,2};

}

}



Explanation: The declaration of array does not create array in memory. That’ why we could’t place values in array, after declaration.

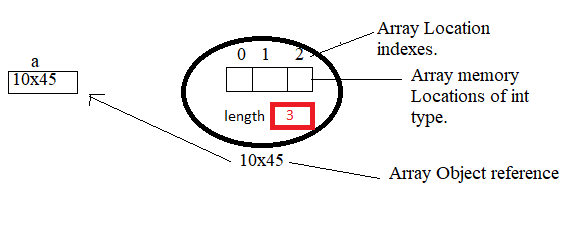
**4.2.Creating An Array:-** we create an array with specified size.

Syntax:-

datatype [] arrayname=new datatype[size];

Example:

Int [] a=new int[3];



Rules for creating Array:

1.Array size is mandatory. Else it leads to CE.

2.we must only specify array size/length only at array object creation side not at declaration side. Otherwise It leads to error.

3.size should be zero, or any positive number. If we pass negative value,then we will see the **java. lang. NegativeArraysizeException**.

4.While storing, reading array values, we must give valid index. Otherwise It displays java. lang**. ArrayIndexOutBoundsException**.

5. When we try to store wrong type object into an array of Object, **java.lang.ArrayStoreException is araised**.

6. array length represents no.of elements the array can have.

Example:

**class** Student{}

**public** **class** Sample {

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

Object s[]=**new** Student[3];

**try** {

s[0]=**new** String("sukumar");

System.***out***.println("Length"+s.length);

}

**catch**(ArrayStoreException e){

System.***out***.println("Program tried to string wrong DT value into Array");

}

}//main

}//class

Output:-

Program tried to string wrong DT value into Array

Example:1

class Sample

{

public static void main(String arg[])

{

int a[];

a=new int[2];

float b[]=new float[2];

a[0]=10;

a[1]=20;

b[0]=2.5f;

b[1]=3.4f;

System.out.println(a[0]);

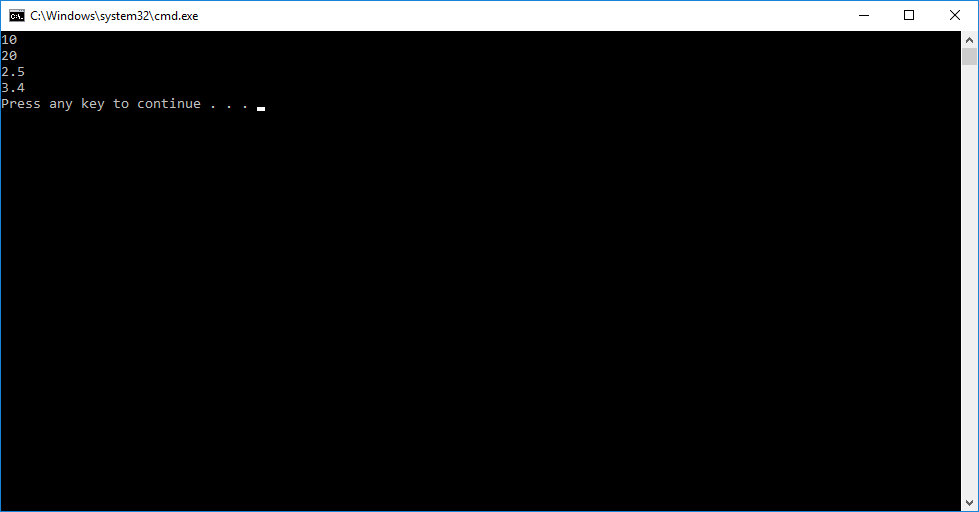
System.out.println(a[1]);

System.out.println(b[0]);

System.out.println(b[1]);

}

}



**4.3.Defining Array:-**The array is created and given values are placed in array locations. we have two ways to define array.

Syntax:1

datatype[]<arrayname>={list of values};

Syntax:2

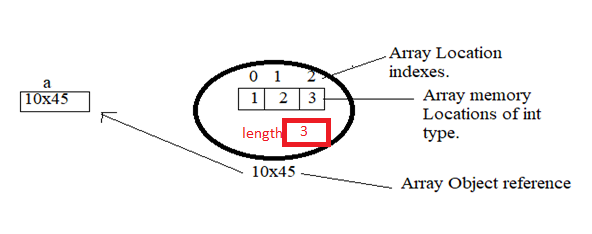
datatype [] <arrayname>=new datatype[]{v1,v2,..vn};

Rule1:-Size should not be specified.

Example:

Int a[]=new int[]{1,2,3};

Int []a={1,2,3}



Example:1

class Sample

{

public static void main(String arg[])

{

int a[]={1,2};

float [] b= new float[]{4.5f,6.7f};

Sample [] c={new Sample(),new Sample()};

System.out.println(a[0]);

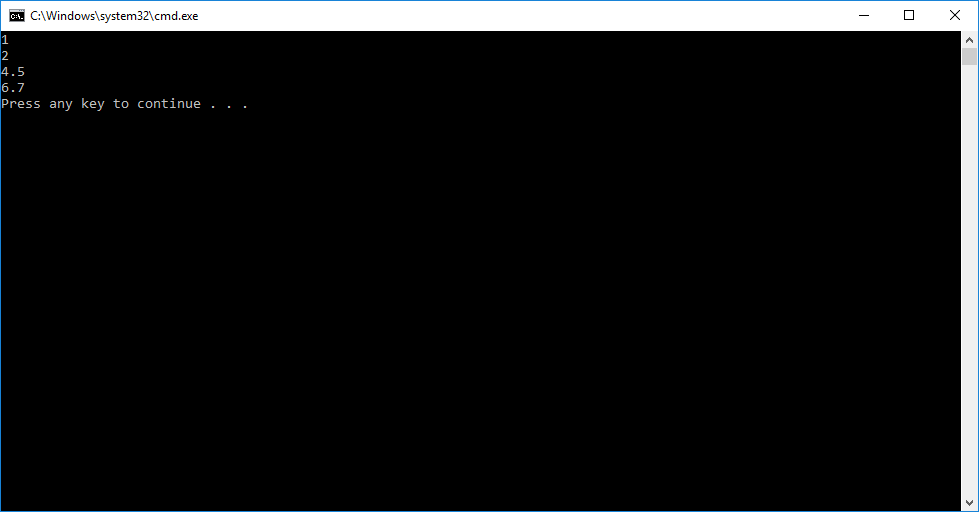
System.out.println(a[1]);

System.out.println(b[0]);

System.out.println(b[1]);

}

}



Example 2:

class Student

{

int eid;

String eName;

int marks[]=new int[2];

}

class Sample

{

public static void main(String arg[])

{

Student a[]=new Student[2];

a[0]=new Student();

a[1]=new Student();

}

}

Example3:

import java.util.Scanner;

class Student{

byte rno;

String name;

String branch;

float marks[]=new float[3];

public void setData(){

Scanner s1=new Scanner(System.in);

System.out.println("Enter rno:");

rno=s1.nextByte();

s1.nextLine();

System.out.print("Enter Student Name:");

name=s1.nextLine();

System.out.print("Enter the Branch Name:");

branch=s1.nextLine();

System.out.print("Enter a Marks in Telugu:");

marks[0]=s1.nextFloat();

s1.nextLine();

System.out.print("Enter a Marks in English:");

marks[1]=s1.nextFloat();

s1.nextLine();

System.out.print("Enter a Marks in Hindi:");

marks[2]=s1.nextFloat();

}

public void getData()

{

System.out.println("Rno:"+rno);

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

System.out.println("Branch:"+branch);

System.out.println("Telugu:"+marks[0]);

System.out.println("English:"+marks[1]);

System.out.println("Hindi:"+marks[2]);

}

}

class ArrayDemo

{

public static void main(String args[])

{

Student s1=new Student();

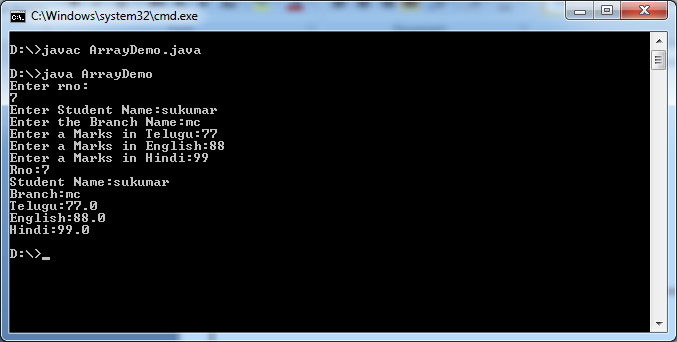
s1.setData();

s1.getData();

}

}

Output:-



**4.4.Types of Arrays based on Dimension:-**

Java supports two types of arrays

1. Single dimensional Arrays.
2. Multidimensional Arrays.

**4.4.1. Multidimensional Array:-** It can be defined as arrays with in array.

Declaration Syntax:

datatype[][]…. <Arrayname>;

Creating n-D Array:

datatype[][]…etc <arrayname>=new datatype[size-1][size-2]…etc;

Rule1:- 1. Size-1 is mandatory.

2.Remaining sizes are optional.

**4.4.1.1)2D Array:-**2D array can be represented as collection of rows and columns. The elements are arranged as table with rows and columns.

a. 2D array Declaration:

Syntax:

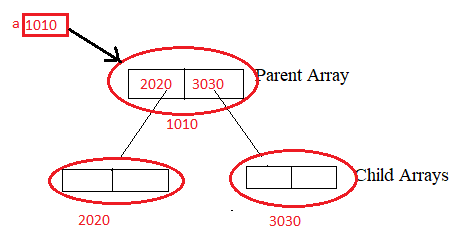
datatype [][] <array-name>;

b.2D Array creation:

Syntax:

datatype [][] <array-name>=new datatype[m][n];

Example: Int [][]a=new int[2][2];



c. 2D array defining:

Syntax:1

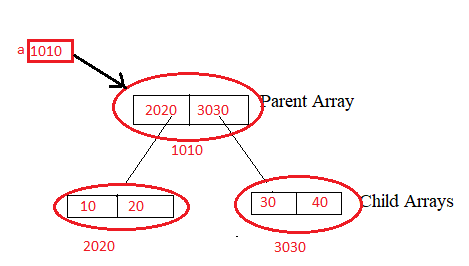
Datatype [][]<array-name>={ {v1,v2,..vn}1,{v3,v4,..vn}2…{}metc}

Syntax:2

Datatype[][]<array-name>=newdatatype[m][n] {v1,v2,..vn}1,{v3,v4,..vn}2…{}metc};

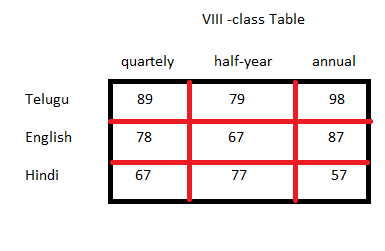
Example:

Int [][]a={{10,20},{30,40}};



**Note:- The length property of 2D array returns no.of rows in array**.

Example:2 2D Array Example.



import java.util.Scanner;

class Student{

int rNo;

String sName;

int marks[][]=new int[3][3];

}

public class Sample {

public static void main(String[] args) {

int row,column;

try(Scanner s=new Scanner(System.in);

){

Student s1=new Student();

System.out.print("Enter the rno:");

s1.rNo=s.nextInt();

System.out.print("Enter the Name:");

s1.sName=s.next();

System.out.print("Enter the No.of Rows:");

row=s.nextInt();

System.out.print("Enter the no.of columns:");

column=s.nextInt();

//Read the elements from user and store in array

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

for(int j=0;j<column;j++) {

System.out.print("Enter the "+i+" "+j+"value:");

s1.marks[i][j]=s.nextInt();

}

}

// Display the Array on console.

System.out.println("================Array=================");

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

for(int j=0;j<column;j++) {

System.out.print(" "+s1.marks[i][j]);

}

System.out.println(" ");

}

}

catch(NegativeArraySizeException e) {

System.out.println("Array Length should be positive");

}

catch(ArrayIndexOutOfBoundsException e) {

System.out.println("Program tried to do operation on Non-existing locaiton in Array");

}

catch(ArrayStoreException e) {

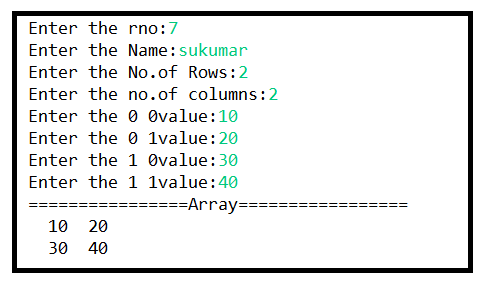
System.out.println("Program tried to store Wrong Type object in Array");

}

}//main

}//class

Output:-



**4.4.1.2)3D-Array:-** The 3D array is collection of one or more two-dimensional arrays.

a. 3D Array Declaration:

Syntax:

Datatype [][][] <arrayname>;

b. 3D Array Creation:

Syntax:

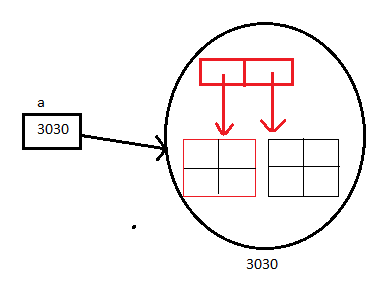
Datatype[][][]<arrayname>=new Datatype[m][n][o];

Where

M,n,o are sizes of dimensions.

Example:

Int [][][] a=new int[2][2][2];



c. 3D Array Defining:

Syntax:1

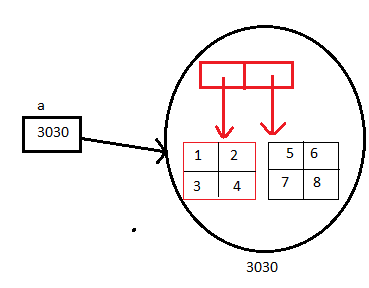
Datatype[][][]<arrayname>=new Datatype[m][n][o]{{2Darray},{2Darry},..{2Darray}};

Syntax:2

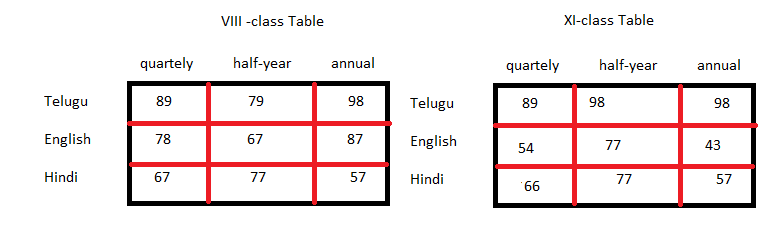
Datatype[][][] a= {{2Darray},{2Darry},..{2Darray}};

Example:

Int [][][] a={{{1,2},{3,4}},{{5,6},{7,8}}};



Example: 3D Array.



import java.util.Scanner;

class Student{

byte rno;

String name;

String branch;

float marks[][][]=new float[2][3][3];

public void setData(){

Scanner s1=new Scanner(System.in);

System.out.print("Enter rno:");

rno=s1.nextByte();

s1.nextLine();

System.out.print("Enter Student Name:");

name=s1.nextLine();

System.out.print("Enter the Branch Name:");

branch=s1.nextLine();

for(int i=0;i<2;i++)

{

System.out.println("Enter values of "+i +" Table:");

for(int j=0;j<3;j++)

{

System.out.println("Enter values of "+j +" Row:");

for(int k=0;k<3;k++)

{

marks[i][j][k]=s1.nextFloat();

}

}

}

}

public void getData()

{

System.out.println("Rno:"+rno);

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

System.out.println("Branch:"+branch);

System.out.println("Marks Table:");

for(int i=0;i<2;i++)

{

for(int j=0;j<3;j++)

{

for(int k=0;k<3;k++)

{

System.out.print(marks[i][j][k]+"\t");

}

System.out.println();

}

System.out.println();

}

}

}

class ArrayDemo

{

public static void main(String args[])

{

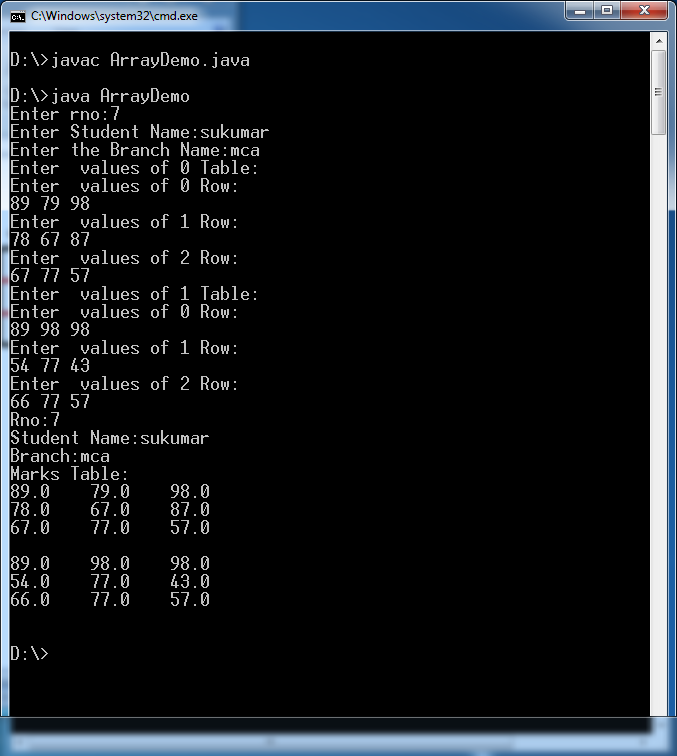
Student s1=new Student();

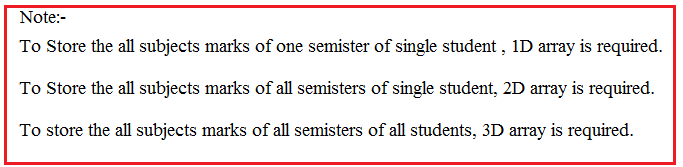
s1.setData();

s1.getData();

}

}





**4.4.1.3) Jogged Arrays:-** Multidimensional array with different sizes of child arrays is called jogged array.

Example:

class A{

public static void main(String arg[])

{

int a[][]=new int[2][];

a[0]=new int[3];

a[1]=new int[2];

}

}

Parent Array

Child Array

Example:1

class Sample

{

public static void main(String arg[])

{

int a[][]=new int[2][2];

a[0][0]=10;

a[0][1]=20;

a[1][0]=30;

a[1][1]=40;

for(int i=0;i<2;i++)

{

for(int j=0;j<2;j++)

{

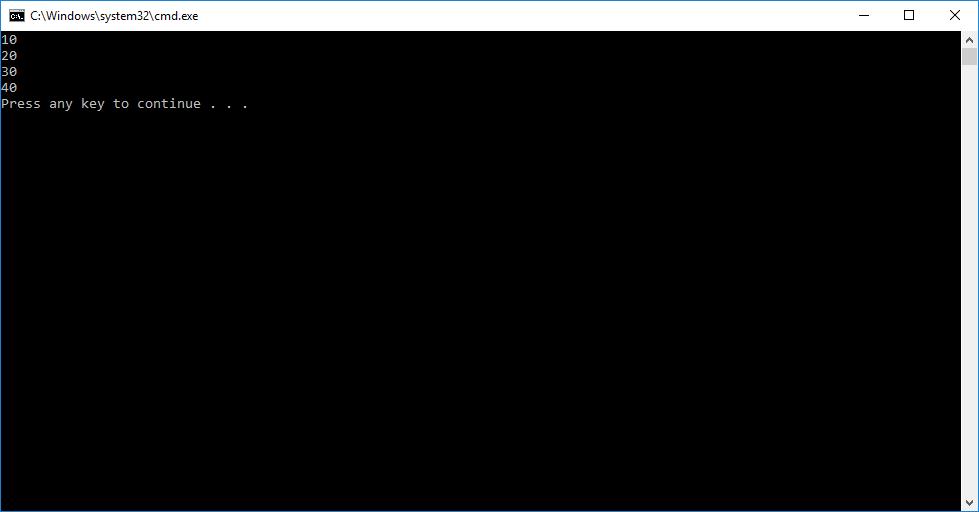
System.out.println(a[i][j]);

}

}

}

}



**OPERATORS**

**5.Operators**:-

**5.1**.Def:- The symbol or word that performs

1. Assignment operation
2. Validation and
3. Calculation operations and returns a result

Is called operator.

* Validation means checking/knowing the given value is correct or wrong. The operators those perform validations will return boolean type result true/false. So these opertors are also called as boolen operators.
* The operators those perform calculations will return number type result.
* The operator that store result in destination variable will return the same copying values as result. This operator is aslo called as assignment operator.

RealWorld Example:-

Mother wrote grocessaires list on paper. She give it to her son. He bring it to super marker to purchase them.

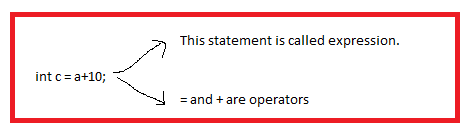
He takes product and check whether product name and quantity is in list or not. It is in a list then he put product in basket.

Where checking is validation operation and putting is assignment operation. The paper on which grocessairy names are written is variable and grocesaries name are the values.

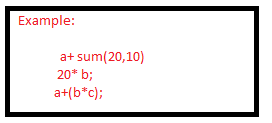
**Operand:-** The value that is passed as input to operator for performing an assignment ,validation and calculation operations is called an operand.

Expression:- The combination of operators and operands is called an expression.

Example:



The thing which is value, which contains value, which returns value can also be written in the place of operand in expression.



Types of Expressions:-There are two types of expressions.

1. Constant Expression:- If we use only literals , final variables or both in expression then such expression is said to be constant expression.

Example:

1. 20+30
2. 50-32.

2. Variable Expression:- If we use at least one variable , non-void method in expression then such expression is said to be variable expression.

Example:

1. 40+b
2. 60\*3/c

**5.2:- Types of Operators:**- Based on number of operands accepts by an operators are divided into three types.

a. Unary operators.(11)

b. Binary Opertors.(27)

c. Ternary Operators.(2)

1. Unary Operator:- An opertor that can take only one operand is called unary operator. We can place unary opertor either before or after given operand.

Ex:-

+a, -b , !true, !false, ++a, a++,b--,--b, new Student();

1. Binary Operators;- An opertor that takes two operands is called binary operator. We can place binary operator only in between two operands.

Example:

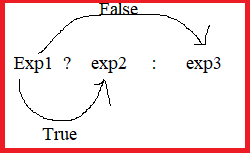
A+b, a-b,a\*b, a&b, a<b,a !=b, a==b,…etc.

1. An operator that takes 3 operands is called ternary operator.

Example:

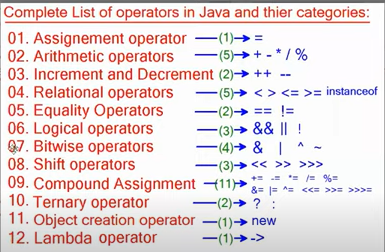
(?:) is ternary operator,It has two opertors ? and :. We should place ternary operator also in between 3 operands.

Example:



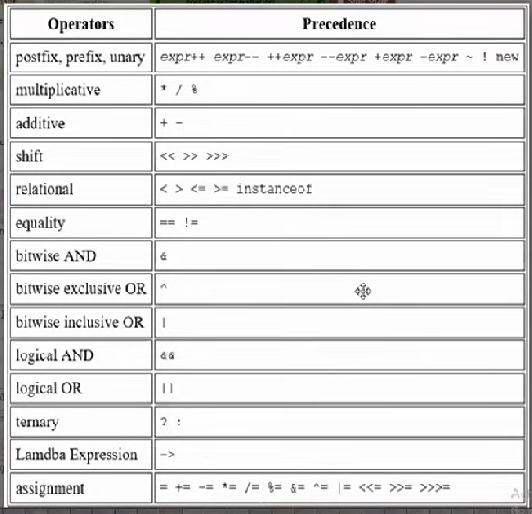
The first expression a should generate boolean result.

* If result is true, expression exp2 is only executed.
* If result is flase, expression exp3 is only executed.

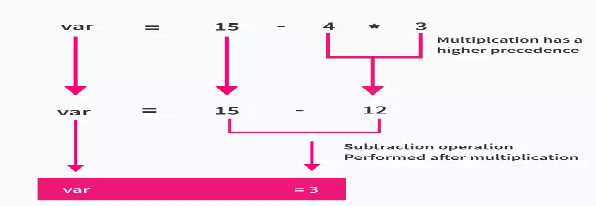


**5.3. Operator Precedence and Associativity:-**  The order of operators execution in expression is caller operator precedence.

If expression consists of several operators then which operator to be executed first and which operator to be executed next this order is decided by operator precedence.

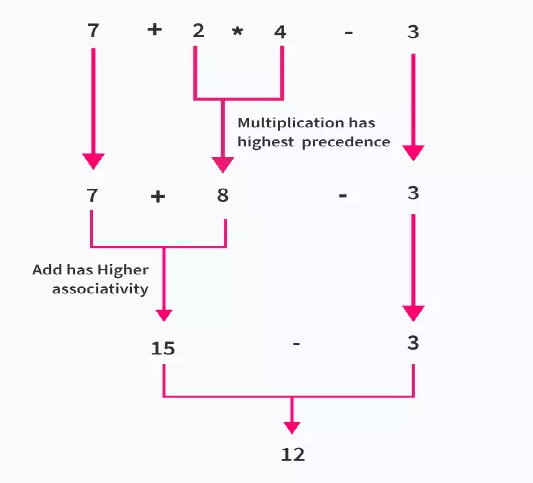
****

Example:-



When two operators have same precedence in expression, operator associativity comes into play. The associativity of operators is used to determine the direction in which same precedence operators are executed.

Example:-



**5.4. Arithmetic Operaors:-**  These are used to performing calculations addition, subtraction,multiplication,division.

The Arithmetic operators are binary operators. These operators takes only byte,short,int,long,float,double,char type of values as input.

These operators always returns numeric value as output. Return value data type is int or heighest range data type used in expression.

Java supports 5 arithmetic operators.

a. addition --- +

b. subtraction -

c. multiplication \*

d. division /

e. Reminder %

Examples:-

1. Byte+byte => int
2. Short+short => short
3. Char + char => char
4. Int + int => int
5. Int +float=float
6. Int+float+long=>float
7. Int+float+long+double=>double
8. Boolean+int// It leads to CE.
9. String-int // It leads to CE.
10. String \* float // It leads to CE.

The + is overloaded operator in java. It can be used as both

1. Addition operator.
2. Concatenations operator.

If + operator has its both operands as numbers or characters, it acts as an addition operator . If one of the operand is string, it acts as concatenation operator.

Example:-

20+30 => in this expression it act as addition operator.

35+”rock”=> in this expression it act as concatenation operator.

1.System.out.println(10+20) //30🡪 numeric type.

2.System.out.println(“”+10+20) //1020🡪string type

3. System.out.println(10+” “+20) //1020🡪 string type

4. System.out.println(10+20+” “) //30🡪 string type .

5. System.out.println(“p+q”+20+30) // p+q2030 🡪 string type.

/ division operator:-

Example: int/int=int

22/7 =3.

Usually, a/0 is infinity in mathemathics.

we can not divide integer number by zero, it leads to

RE: java.lang.ArithmeticExcetion. It is not infinity.

Because, integers does not have ‘infinitity”.

When we divide float ,double,long number by zero, it leads to

RE: java.lang.ArithmeticExcetion. It is not infinity.

When we divide integer zero by zero it also leads to RE:

RE: java.lang.ArithmeticExcetion. It is not infinity.

When we divide floating point zero by zero its output is **NaN**.(Not a Number)

Example:1

import java.io.\*;

class sample

{

public static void main(String[] args) {

System.out.println(10+20);

System.out.println(""+10+20);

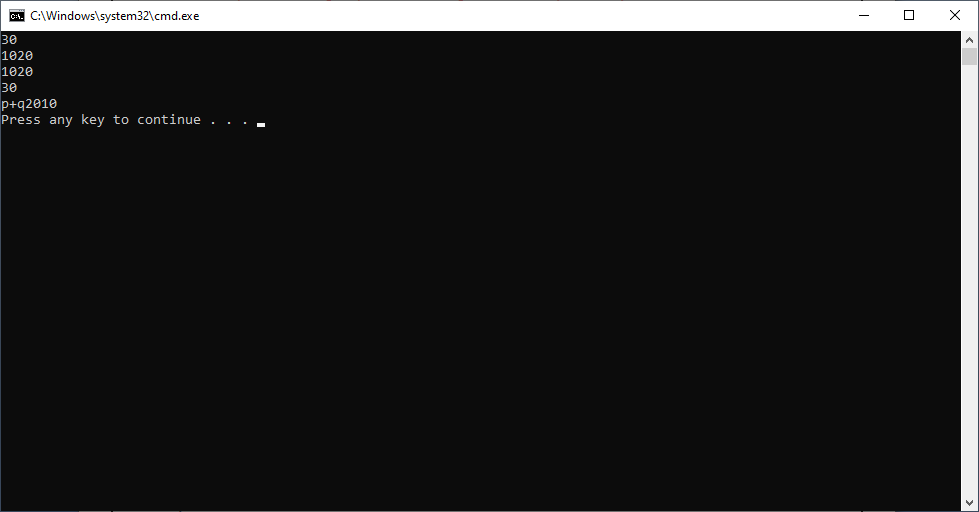
System.out.println(10+""+20);

System.out.println(10+20+"");

System.out.println("p+q"+20+10);

}

}



Example:2

import java.io.\*;

class sample

{

public static void main(String[] args) {

System.out.println(10.4f/0);

System.out.println(20.3/0);

System.out.println(0.0f/0);

System.out.println(0.0/0);

}

}

Output: No compile Time error.

We get following runtime error.



Example:3

import java.io.\*;

class sample

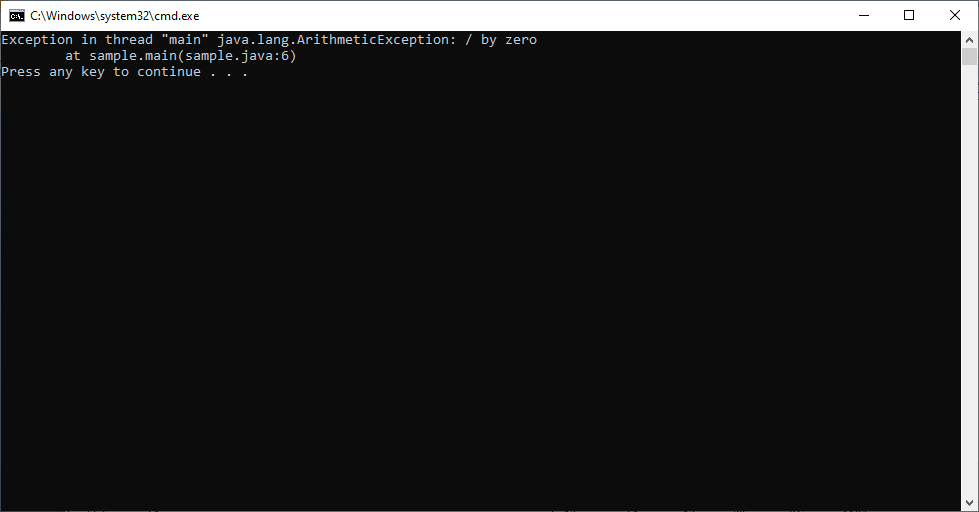
{

public static void main(String[] args) {

System.out.println(10/0);

}

}



import java.io.\*;

class sample

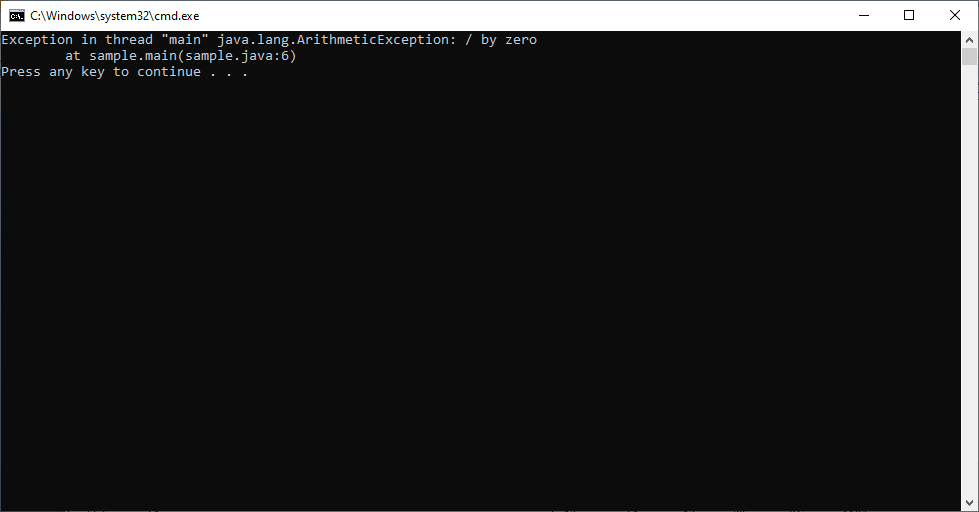
{

public static void main(String[] args) {

System.out.println(0/0);

}

}



**5.5. Increment and Decrement Operator:-**

The ++ is increment operator.

The -- is decrement operator.

1. Pre-Increment and Pre-decrement operator:-If increment operator is before the variable then it is said to be pre-increment operator.

Example:- ++a , ++c , ++b.

If decrement operator is after the variable then it is said to be pre-decrement operator.

Example:-

--b , --d, --x.

**Note:- First increment /decrement value and then read value ,use this in expression.(expression is executed with new value).**

1. Post-increment and post-decrement operator:- if increment operator is after the variable then it is said to be post-increment operator.

Example:- a++ , c++ , b++

If decrement operator is after the variable then it is said to be post-decrement operator.

Example:-

b-- , d --- x—

**Note:- First read value , substitute in expression and then either increase or decrease.(expression is executed with old value)**

Example:-1

Int x=10;

Int y=20;

System.out.println(x);

++x;

System.out.println(x);

System.out.println(y);

Y++;

System.out.println(y);

Output:

10

11

20

21

Example:2

Int x=1;

Int y=x++;

System.out.println(x);

System.out.println(y);

Output:

2

1

Example:3

Int x=1;

Int y =++x;

System.out.println(x);

System.out.println(y);

Output:

2

2

Example:4

X=2;

System.out.println(x++);

System.out.println(x);

Output:

2

3

Example: 5

X=2

System.out.println(++x);

System.out.println(x);

Output:

3

3

**5.6.Unary Operators**:-( +, - , ! ~, ++, --)

a.unary +:- It is used to represent the positive operand. In this use the operator is optional.

Syntax:- + operand

b.unary - :- It is used to convert positive value into negative and negative value into +value.

Syntax:- -operand

c.logical complement operator:- It is used to reverse the value of Boolean value.

Syntax:- ! operand

Example:

import java.io.\*;

class sample

{

public static void main(String[] args) {

System.out.println(+10);

System.out.println(-10);

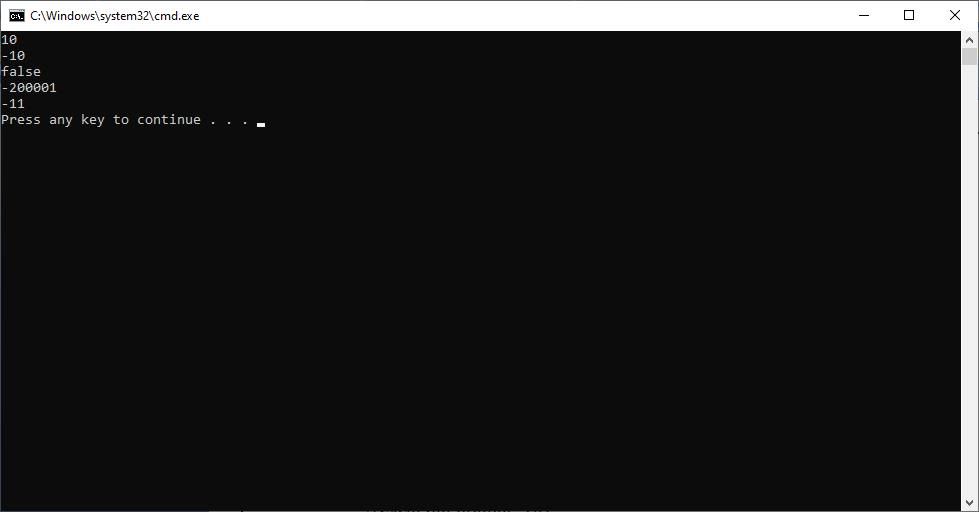
System.out.println(!true);

System.out.println(~200000);

System.out.println(~10);

}

}



**5.7:- Boolean Operators:-** The Boolean operators are >, < , <= ,>=, == ,!=. The Boolean operators returns either true or false.

Syntax:



* If both variables have same values ==operator returns true,otherwise returns false.
* If both variables have different values != operator returns true, otherwise returns false.
* If operand-1 is greater than operand-2 , then > returns true, otherwise returns false.
* If operand-2 is smaller than operand-2 , then < returns true, otherwise returns false.
* If operand-1 is greater than or equal to operand-2 , then >= returns true, otherwise it returns false.
* If operand-1 is smaller than or equal to operand-2 , then <= returns true, otherwise it returns false.

Example:

import java.io.\*;

class sample

{

public static void main(String[] args) {

System.out.println(10==10);

System.out.println(20==10);

System.out.println(10!=10);

System.out.println(40!=10);

System.out.println(30>40);

System.out.println(50>40);

System.out.println(30<40);

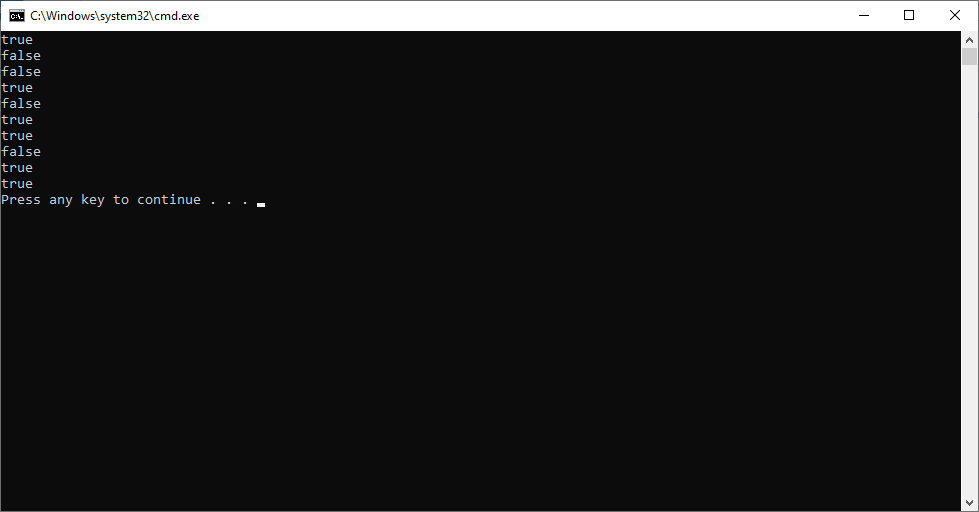
System.out.println(50<40);

System.out.println(30>=30);

System.out.println(50<=50);

}

}



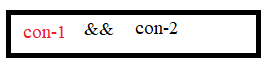
**5.8. Logical Operators**:- The logical operators are &&,||.

The logical operators combines two expressions/operands.

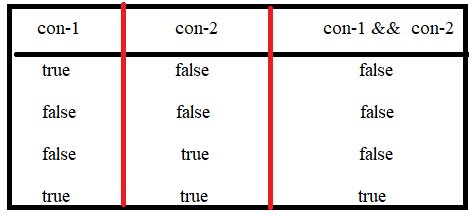
The logical operators returns either true or false.

1. &&- Logical and operator.

Syntax:-



The truth table of logical and operation.

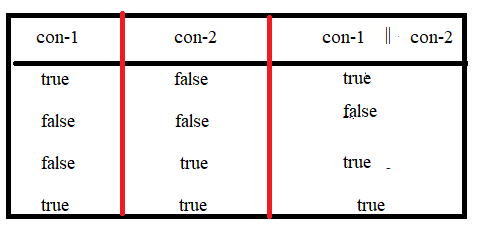


B. || -logical or operator :-

Syntax:



Truth table of logical or operator.



Examples:

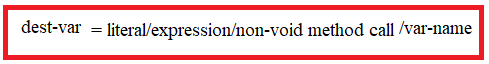
System.out.println(1 && 1) // It leads to CE .

System.out.println( 0 || 0) // It leads to CE.

System.out.println(true && true) // It leads to CE.

**5.9. Assignment Operator:-** The operator that store result in destination variable and will return the same copying values as result. This operator is aslo called as assignment operator.

**Syntax:**

****

a)If right side part of assignement operator is literal/const expression/non-void method call then compiler directly uses the values generates the final result and verifies that whether result is in dest-var range or not and result type is compitable or not.

If yes, store the result in dest-var.

If no, display CE : possible loss of precession.

Example:

Byte a;

1. a = 240 // CE: possible loss of precession.
2. a = 30 + 40 // No Error. Result is 70. Byte range is 127. 70 is less than 120 therefore , we wont get error.
3. a = 30L // CE possible loss of precession.
4. a = 34.5f //CE possible loss of precession.
5. a = 23.2 // CE possible loss of precession.
6. a = (byte)30L //No Error.
7. a = (byte)34.5f // No Error.
8. a = (byte)23.2 // No Error.

b)If left side part of assignment operator is var-name/variable-expression then compiler uses data types of variables and generates data type as result.

Next compiler verifies whether result type is compatable with dest-var data type or not and result type size is equal to or less than dest-var size or not.

If yes compiled successfully.

If no , it display CE: possible loss of precesion.

Example:

byte a=20;

Byte b=10;

Byte c = a + 30;

\|/

Int

Integer is compatable with byte but Its range is greater than byte range. There fore we get “Possible loss of precesion”.

**5.10. new Operator:-**  It is unary operator. One instance for class is created at run time with new operator. After creating , this operator returns reference of instance.

Syntax:



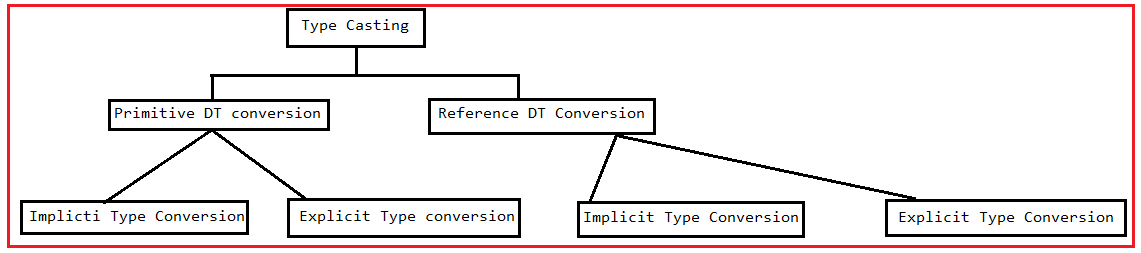
**Limitations:-**

1. Compile time onwards, new operator expects presence of classname. The new operator can be used to create the instance for class which comes into java application dynamically.

**TYPE CASTING**

In Java, type casting is process that converts a one data type into another data type.There are two types of conversions.

1. **primitive DT conversion.**
2. **Referenced DT conversion.**

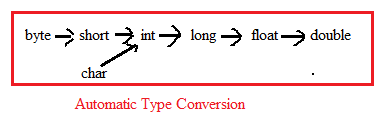


**1.Casting the Primitive Data types:-** The process of changing one type of primitive value into another type primitive value is called PDT(Primitive Data Type) conversion. There are two types of primitive type conversions.

**1.1. Implicit type conversion/widening.**

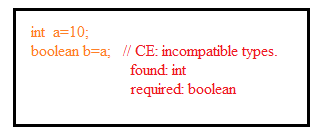
**1.2. Explicit type conversion/narrowing.**

**1.1. Implicit Type Conversion:-**Converting a lower data type into higher one is called widening type conversion. It is also known as implicit conversion or **casting down.** It is done automatically. It is safe because there is no chance to lose data.



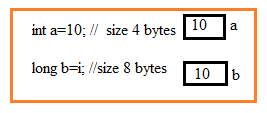
It takes place when:

* Both data types must be compatible with each other.

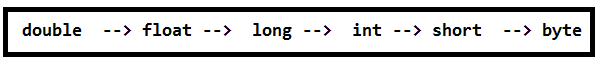


* Both target type must be larger than the source type.

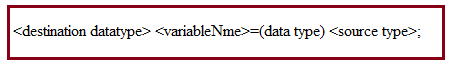
Example:-



**1.2. Explicit type conversion/narrowing: C**onverting a higher Range data type into a lower one is called narrowing type casting. It is also known as explicit conversion or casting up. It is done manually by programmer. If we do not perform casting then compiler reports compile-time error.



Syntax:



Rules:

1. Cast operator data type(data type in syntax) Range must be <= destination data type range else it leads to CE:”possible loss of precession”.
2. The cast operator data type should be compatible with destination type else it leads to CE: “incompatible data type”.

Example:

**float** f=2.5f;

**int** x=(**int**)f;

System.***out***.println("Before Explict conversion:"+f);

System.***out***.println("After Conversion into integer:"+x);

**Output:**-

Before Explict conversion:2.5

After Conversion into integer:2

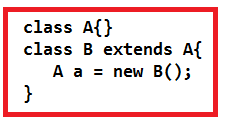
**2.Referenced Type Conversion:-** Converting One RDT into another RDT iscalled **Referenced Type conversion.** There are two types of RDT conversions.

a. Implicit RDT conversion/

b. Explicit RDT conversion.

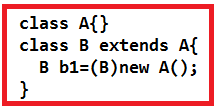
**2.1. Implicit RDT conversion:-** Casting from a subclass to a superclass is called implicit Conversion/UpCasting.

Example:-



**2.2.Explicit RDT conversion:-** Casting from a superclass to subclass is called “Explicit RDT Conversion”/ “Down Casting”.

Example:-

****

**Note:- we can not cast two unrelated classes implicitly or explicitly.**