Tuesday, March 10, 2015

1. **Operator Concept**

1. Increment and Decrement operators
2. Arithmetic Operators
3. String Concatinations Operators
4. Relational Operators
5. Equality Operators
6. istanceOf Operators
7. Bitwise Operators
8. Short circuit operators
9. Type cast Oprators
10. Assignment Operators
11. [] Oprators
12. Oprators Precedence
13. Evaluation of Oprators
14. New vs newInstance() Oprators
15. istanceOf vs isInstance()
16. ClassNotFoundException vs NoClassDeffoundError
17. Increment and Decrement Oprators

* Increment Operators:-

1. Pre Increment

Ex- y=++x

1. Post Increment

Ex- y=x++

* Decrement Operators:-

1. Pre Decrement

Ex- y=--x

1. Post Decrement

Ex- y=x—

|  |  |  |  |
| --- | --- | --- | --- |
| Expression | Initial Value  Of x | Value of y | Value of x |
| Y = ++x | 10 | 11 | 11 |
| Y = x++ | 10 | 10 | 11 |
| Y = --x | 10 | 9 | 9 |
| Y = x-- | 10 | 10 | 9 |

* We can apply increment and decrement operators only for variables but not for constant values, by mistake if we are trying to applying for constant values then we will get compile time error.

Ex1.0:

Int x=10;

Int y=++x; Sopln(y);//11

Ex1.1

|  |
| --- |
| Int x=10;  Int y=++10; //CE: Unexpected Type  Sopln(y);  Required: variable  Found: value |

->Nesting of increment and decrement operators are not allowed otherwise we will get compile time error.

Ex-

|  |
| --- |
| int x=10;  Int y=++(++x);// CE: unexpected type  Required: variable Found : value  Sopln(y); |

->We cant apply increment and decrement operators for final variables otherwise we will get compile time errors.

Ex-

|  |
| --- |
| final int x=10;  X++;//can not assign a value to final variable x  Sopln(x); |

->Increment and decrement operators are applicable for every primitive types except boolean.

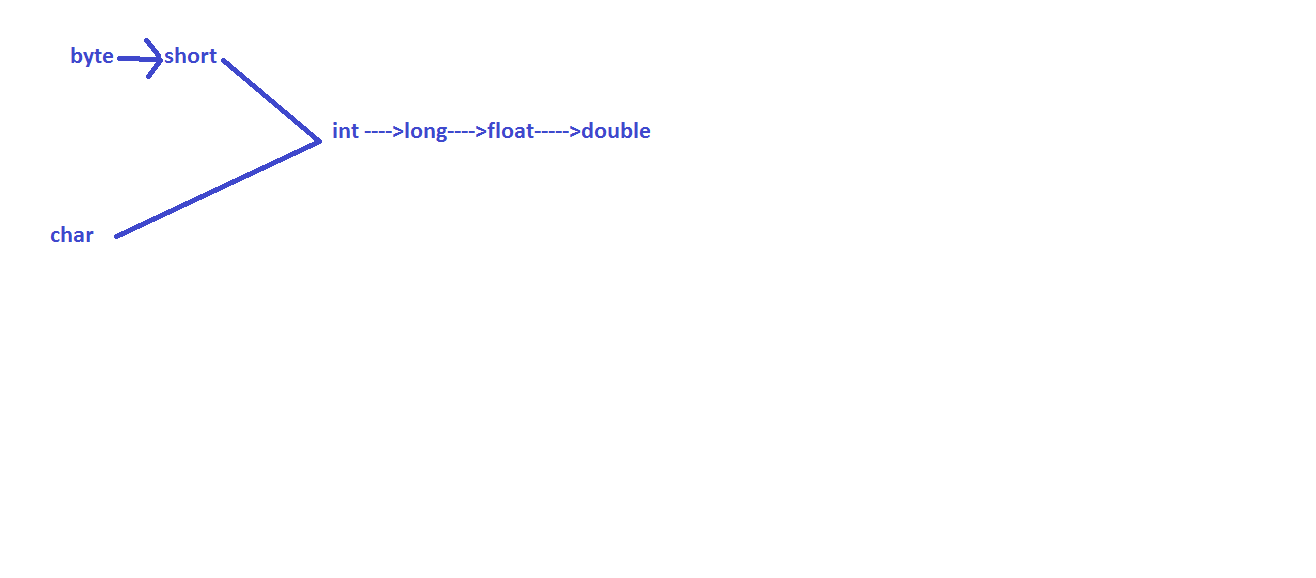
Ex-

|  |
| --- |
| Ex- int x=1; x++; Sopln(x);//11  Ex-char ch=’a’; ch++; Sopln(a);//b  Ex-double d=10.5; d++; Sopln(d);//11.5  Ex-boolean b=true;b++; Sopln(b);// operator ++ can not be applied to boolean. |

Q1.Difference between b++ and b=b+1 ?

Ans: If we apply any arithmetic operation between a and b then result type is :-

MAX(int,TYPE OF a,TYPE OF B)



|  |
| --- |
| Ex-byte b=10;b++;Sopln(b);//11  Ex-byte a=10;byte b=20;byte c=a+b;//CE: Possible loss of pression Found: int required : byte  Ex- byte b=10; b=b+1;//CE: Possible loss of precession , Found: int required: byte |

* To solve this proble , we need to typcast

|  |
| --- |
| Ex-b=(byte)(a+1); |

->But in the increment and decrement operators, internal type casting will be perform automatically.

Ex- b++;-----🡪b=(type of b)(b+1);

Ex- byte b=10; b++; where b++;🡪b=(byte)b+1;

1. Arithmetic Operators(+,-,\*,/,%)

-> If we apply any arithmetic operators are between two variables a and b then result type is always MAX(INT, TYPE OF A, TYPE OF B).

|  |
| --- |
| Ex- byte+byte=int  Ex-byte+short=int  Ex-short+short=int  Ex-byte+long=long  Ex-long+float=float  Ex-int+double=double  Ex-char+char=int  Ex-char+double=double |

|  |
| --- |
| System.out.println(‘a’+’b’);//195  System.out.println(‘a’+0.1);//97.1  System.out.println(10/0);//RE AE: / Divide by zero  System.out.println(0/0);// RE AE: / Divide by zero  System.out.println(10/0.0); infinity  System.out.println(0/0.0); NaN |

11/03/2015

🡪 In Integral arithmetic there is no way to represent infinity; hence if infinity is the result then we will get run time exception saying ArithmeticException, divide by zero.

EX-

|  |
| --- |
| Sopln(10/0);RE:AE:/by zero |

* But in floating point arithmetic, there is a way to represent infinity. For this Float and Double classes contains the following constants:-

POSITIVE\_INFINITY

NEGATIVE\_INFINITY

Hence, even though result is infinity, we won’t get any AE in floating point arithmetic.

Ex-

|  |
| --- |
| Sopln(10/0.0);//infinity  Sopln(-10.0/0);//-infinity |

* NaN(Not a Number)
* In integral arithmetic, there is no way to represent undefined result, Hence if the result is undefined we will get run time Exception saying AE in integral arithmetic.

Ex-

|  |
| --- |
| * Sopln(0/0);//RE:AE: divide by zero |

* But in floating point arithmetic there is a way to represent undefined results. For this Float and Double classes contains NaN constant. Hence if the result is undefined then we won’t get any arithmetic exception in floating point arithmetic.

Ex-

|  |
| --- |
| Sopln(0.0/0);//NaN  Sopln(-0/0.0);//NaN |

->For any x values including NaN, the following expressions returns false:-

|  |  |
| --- | --- |
| X < NaN  X <= NaN  X > NaN  X >= NaN  X == Nan | false  false  false  false  false |

->For any x values including NaN, the following expression returns true:-

|  |
| --- |
| X != NaN |

Ex-

|  |  |
| --- | --- |
| Sopln(10<Float.NaN); | False |
| Sopln(10<=Float.NaN); | False |
| Sopln(10>=NaN); | False |
| Sopln(10>NaN); | False |
| Sopln(10==NaN); | False |
| Sopln(Float.NaN=Float>NaN); | False |
| Sopln(10!=Float.NaN) | True |
| Sopln(Float.NaN!=Float.NaN) | True |

Conclusion:

->ArithmeticException— It is runtime Exception but not compile time error.

->It is possible only in Integral Arithmetic but not in floating point arithmetic.

->The only operators which cause the AE are / and %.

03. String Concatination Oprator(+)

->The only overloaded operator in java is ‘ +’ operator.

-> Sometimes it acts as Arithmetic Addition operator and sometimes it acts String concatenation operator.

Ex- If at least one argument/operand is String type then ‘+' operator acts as String concatenation operator.

Ex- If Both arguments are number type then only + operator acts as arithmetic addition.

Ex-

|  |
| --- |
| String a=”durga”;  Int b=10,c=20,d=30; |

|  |  |
| --- | --- |
| Sopln(a+b+c+d); | Durga102030 |
| Sopln(b+c+d+a); | 60durga |
| Sopln(b+c+a+d); | 30durga30 |
| Sopln(b+a+c+d)’ | 10durga2030 |

Ex-

Consider the following declarations:-

String a=”durga”; Int b=10,c=20,d=30;

* Which are the following expressions are valid :-

|  |  |
| --- | --- |
| a = a+b+c | CE: incompatible |
| a = a+b+c | Ok |
| b = a+c+d | CE: Incompatible |
| b = b+c+d | Ok |

04. Relational Operators(<,<=,>,>=)

* We can apply relational operators for every primitive type except Boolean.
* Ex-

|  |  |
| --- | --- |
| Sopln(10<20); | True |
| Sopln(‘a’<100); | True |
| Sopln(‘a’>10.5); | True |
| Sopln(true>false); | CE: Operator > cannot applied for Boolean, Boolean |

* We can’t apply relational operators for object types otherwise we will get compile time error.

Ex-

Sopln(“durga”<”durga123”);CE: operator < cannot be applied to java.lang.String,java.lang.String

* Nesting of relational operators is not allowed.

Ex-

Sopln(10<20<30);🡺true<30🡺CE: Operator < can’t be applied Boolean , integer.

12-Mar-15

05.Equality Operator(== and !=)

-> We can apply equality operator for every primitive type including Boolean type also.

Ex-

|  |
| --- |
| Sopln(10==20);//false  Sopln(‘a’==97.00);//true  Sopln(false==false);//true  Sopln(true==true);//true |

* We can apply equality operator for Object types also.

For object references R1 and R2  
 R1==R2 returns true if and only both R1 and R2 pointing to the same object that is ‘==’ Operator meant for reference comparison (address comparison).

Ex-

|  |
| --- |
| scjp2.png |

Ex-

Thread t1=new Thread();

Thread t2=new Thread();

Thread t3=t1;

Sopln(t1==t2);//false

Sopln(t1==t3);//true

Ex-2

|  |
| --- |
| * To use equality operator between object types compulsory there should be relationship between argument types or object types (either child to parent or parent to child or same type) otherwise we will get compile time error saying incomparable types. |

Diagram:

Note:

For any object reference or

== null is always false but null==null is always true.

|  |
| --- |
| String s=new String(“durga”);  Sopln(s==null);//false  String s=null;  Sopln(s==null);//true  Sopln(null==null);//true |

|  |
| --- |
| Q. \*\*\*What is difference between == operator and equals () method? |
| Ans. In general, we can use ‘==’ operator for reference comparison (address comparison) whereas .equals () method for content comparison. |
| String s1=new String(“durga”);  String s2=new String(“durga”);  Sopln(s1==s2);//false,bcz different reference  Sopln(s1.equals(s2));//true,because content is same |

06.**istanceof operator**

|  |
| --- |
| -> We can use instanceof operator to check whether the given object is of particular type or not.  Ex-  ArrayList arlist=new ArrayList();  arlist.add(new Student());  arlist.add(new Customer());  arlist.add(new Employee());  arlist.add(new String(“durga”));  Object o=arlist.get(0);  If( o instanceof Student){  Student s=(Student)o;  //perform Student specific functionality  }  Else if(o instanceof Customer){  Customer c=(Customer)o;  //perform Student specific functionality  }  …  … |
| Syntax:-  R(Reference of any object) instanceof X(any class) |

|  |
| --- |
| scjp5.png |

Note:

->To use ‘instanceof’ operator there should be relation between argument/operand types (either to child or parent to child or same type) otherwise we will get compile time error saying inconvertible types.

Ex-

Thread t=new Thread();

Sopln(t instanceof String);//CE: inconvertible types Found: Thread , Required : String

->Whenever we are checking parent object is of child type by using instanceof operator then we will get false as output.

Ex-

Object o=new Object();

Sopln( o instanceof String);//false

Ex-

Object o=new String(“durga”);

Sopln(o instanceof String);//true

Object o1=new StringBuffer(“durga”);

Sopln(o1 instanceof String);//false

->for any class or interface X,

null instanceof X======🡺false

Ex-

Sopln(null instanceof String);//false

Sopln(null instanceof Runnable);//false

Sopln(t instanceof String); CE: inconvertible type

Sopln(null instanceof Object);//false

07. **Bitwise Operators(&,|,^,~,!)**

&->AND=>Returns true iff both arguments are true

^->X-OR=>Returns true iff both arguments are different

|->OR=>Returns true iff at least one argument is true

Ex-

|  |
| --- |
| Sopln(true & false);//false  Sopln(true | false);//true  Sopln(true ^ false);//true |

Friday, March 13, 2015

->We can apply this operator even for integral types also.

Ex-

|  |
| --- |
| Sopln(4 & 5);//4  Sopln(4 | 5);//5  Sopln(4 ^ 5);//1 |

**08.Bitwise complement operator(~):**

->We can apply this operator only for integral types but not for Boolean types.

Ex-

|  |
| --- |
| Sopln(~true);//CE: ~ operator cannot be applied for boolean.  Sopln(~4);//-5  Explaination:  4 => 0 0000 0000 0000 0000 0000 0000 0000 100  ~4=>1 1111 1111 1111 1111 1111 1111 1111 011  0000 0000 0000 0000 0000 0000 0000 100=🡺1’s complement of 4  2’s complement==> +1  101🡺5  Since, Most significant digit is 1.  Hence, resultant sign will be –ve.  ----------------------------------------------------------  ->The most significant bit acts as, 0 means +ve and 1 means –ve number.  -> +ve number will be represented directly in memory whereas –ve will be represented in the form of 2’s complement. |

09. Boolean Complement operator

|  |
| --- |
| -> We can apply this operator only for Boolean types but not for integral types.  Ex-  Sopln(!4);//CE: operator ! cannot be applied for int  Note:  & , | and ^ : these are applicable for integral types  But not for Boolean.  ! : Applicable only for Boolean types but  Not for integral types  ~ : Applicable only for integral types but  Not for Boolean types |

10. Short Circuit Operator (&&, ||)

|  |  |
| --- | --- |
| These are exactly same as bitwise except the following differences: | |
| & , I | && , II |
| Both arguments should be evaluated always. | Second argument evaluation is optional. |
| Relatively performance is low. | Relatively performance is high. |
| Applicable for both Boolean and integral types. | Applicable only for Boolean but not integral types. |
|  |  |

Note:

|  |
| --- |
| X&&y🡺 y will be evaluated iff x is false ie. X is true then y won’t be evaluated.  X||y🡺 y will be evaluated iff x is false ie. X is true then y won’t be evaluated. |

Ex-

|  |
| --- |
| Int x=10;  Int y=15;  If(++x<10 ||++y>15){ ++x;}  Else{++y;}  Sopln(x+”……….”+y);  ->In below table output is given for different expression:- |

|  |  |  |
| --- | --- | --- |
|  | X | Y |
| & | 11 | 17 |
| && | 11 | 16 |
| | | 12 | 16 |
| || | 12 | 16 |

Ex-

|  |
| --- |
| Int x=10;  If(++x<10 &&(x/0>0)){Sopln(“Hello”);}  Else{Sopln(“Hi”);  o/p:- |

|  |  |
| --- | --- |
| 1.RE:AE / by zero |  |
| 2.CE |  |
| 3. Hello |  |
| 4. Hi | Right answer |

Note: In the above code, if we replace && with & then we will get AE: / by zero

11. Typecast Operator

|  |
| --- |
| -> There are two types are typecasting :-  1. Implicit typecasting  2. Explicit typecasting |
| 1. Implicit Typecasting  ->Compiler is the responsible to perform implicit type casting.  -> Whenever we are assigning smaller data type to bigger data type variable then implicit type casting will be performed.  -> It is also known as widening or up casting.  -> There is no loss of information in this type casting.  ->The following of various possible conversions where implicit type casting will be performed.  Byte=>short=>int=>long=>float=>double  Char=>int=>long=>float=>double  Ex-  Double d=10;  Sopln(d);//10.0  Here, Compiler is converting integer 10 to double 10.0 by implicit type casting.  Ex-  Char x='a’;  Sopln(x);//97  Here, Compiler is converting char ‘a’ to int 97 by implicit type casting. |

|  |
| --- |
| 03.Explicit type casting  ->Perogrammaer is responsible explicit type casting.  ->There aere  ->The following are various conversion whre ex |

Ex-

|  |
| --- |
| Int x=50; |

10. Assignment Operator(=)

->There are three types of assignment operators :-

1) Simple Assignment

Ex- int x=10;

2) Chained Assignment

Ex- int a,b,c;

A=b=c=d=20;

Sopln(“a=”+a+”b=”+b+”c=”+c+”=”d);//20

->we cant assigned chained assigment operator directy at the time of declaration.

Ex- int a=b=c=d=20;

CE: can’t file symbol

At the location: Test

3) Compound assignmet operator

->Sometimes assignment operator to form compound assignment;

Int x=10;

X+=20;

Friday, March 20, 2015

**Equivalance between VAR-ARGS parameter and One dimensional array:-**

->case 1: Wherever one dimensional presents , we can replace with var-args parameter.

Ex- m1(String args[])🡺 m1(String…args);

m1(int[] x)🡺 m1(int…x);

=>Wherever var-args parameter, we cant replace into one parameter arry.

=> m1(int …x)

-> we can this method by passing the group of valuesa d . We amt

--