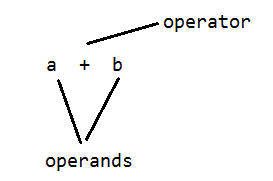
## Operators in Java

**Operator** is a special symbol that tells the compiler to perform specific mathematical or logical Operation.

An operator is a symbol that is used to perform some operation on any values.



The data on which we perform operations is called as operands

**In java we perform operations in following 3 levels**

**1. Unary operators**

If any operator given with single variable or value or expression then it is called as unary operators.

**Eg:**

-a, +a, ....

**2. Binary operators**

If any operator given with 2 variables or values or expressions then it is called as binary operators.

**Eg:**

a+b, a-b, a+b+c, .....

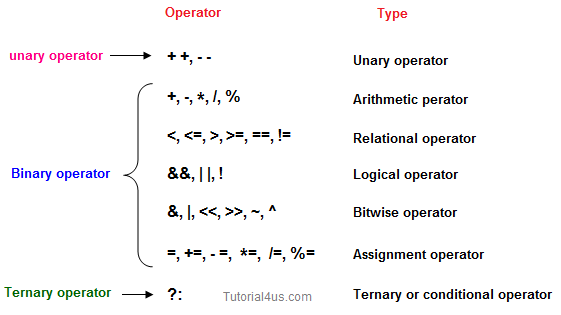
**3. Ternary operators**

If any operator given with 3 variables or values or expressions then it is called as ternary operators.

Java language is rich with built-in operators.

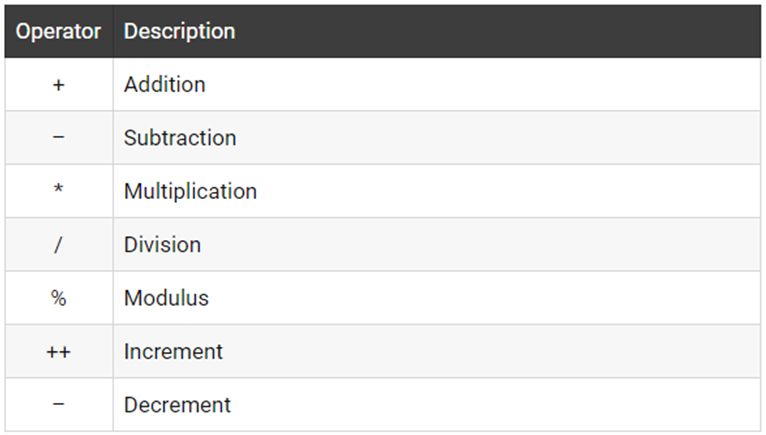
**Java supports following lists of operators.**

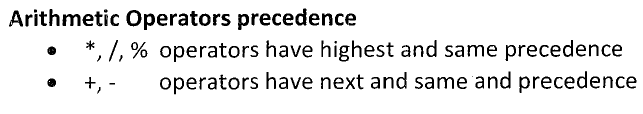
* **Arithmetic Operators**
* **Relational Operators**
* **Logical Operators**
* **Bitwise Operators**
* **Assignment Operators**
* **Ternary or Conditional Operators**

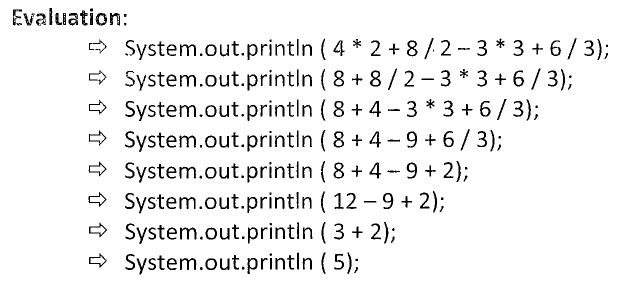


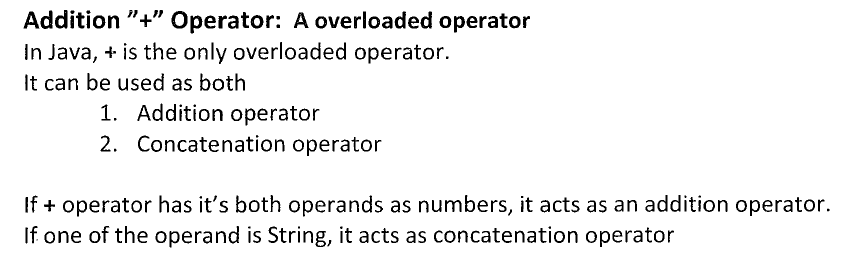
## Arithmetic Operators

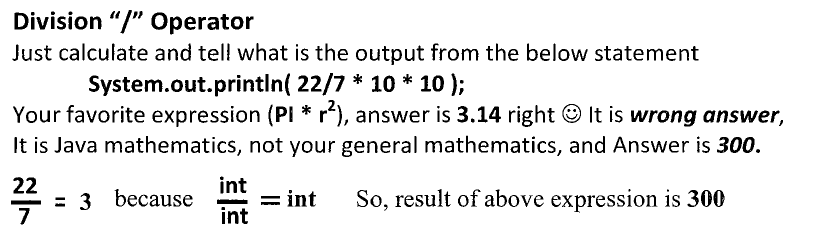
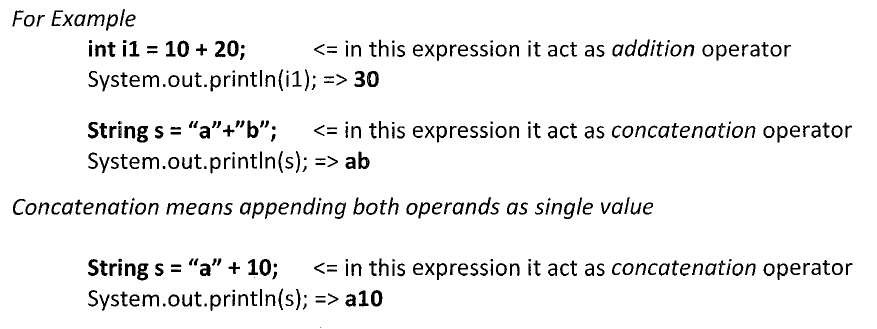
Given table shows the entire Arithmetic operator supported by Java Language.

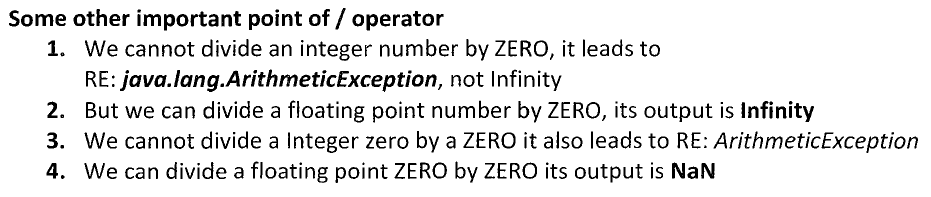


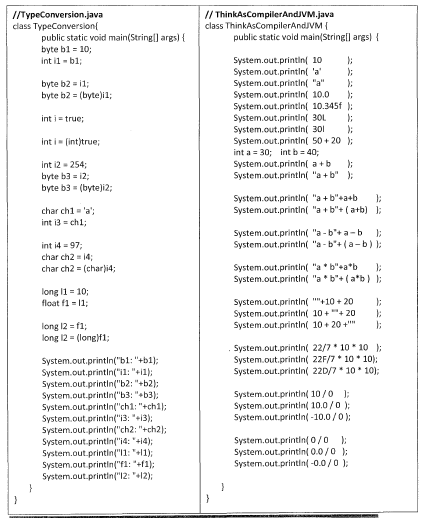
****

****

****

****

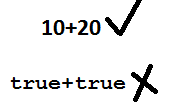
****

****

**Let’s suppose variable A hold 8 and B hold 3.**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Example (int A=8, B=3)** | **Result** |
| + | A+B | 11 |
| - | A-B | 5 |
| \* | A\*B | 24 |
| / | A/B | 2 |
| % | A%4 | 0 |

**Rules**

1. Arithmetical operators can be applied only on numerical data types but we never apply on boolean data types.
2. If any mathematical expression is evaluated then the final result datatype can be decided like follows

|  |  |
| --- | --- |
| **combination of different expressions** | **Final result type** |
| byte,byte | int |
| byte,short,char | int |
| char,char | int |
| int,int | int |
| byte,short,int | int |
| byte,short,int,long | long |
| byte,short,int,long,float | float |
| byte,short,int,long,float,double | double |

1. Mathematical expression is evaluated based on operator precedence.

**Operator precedence (BODMAS)**

**( )**

**/ % \***

**+ -**

**Eg1:**

*int x=2;*

*System.out.println(x\*10+2); //22*

**Eg2:**

*int x=2;*

*System.out.println(x\*(10+2)); //24*

**Eg3:**

*int x=2;*

*System.out.println(2-x\*10/3); //-4*

**Eg4:**

*int x=2;*

*System.out.println(x\*10/3-2); //4*

## Relational//Comparision Operators

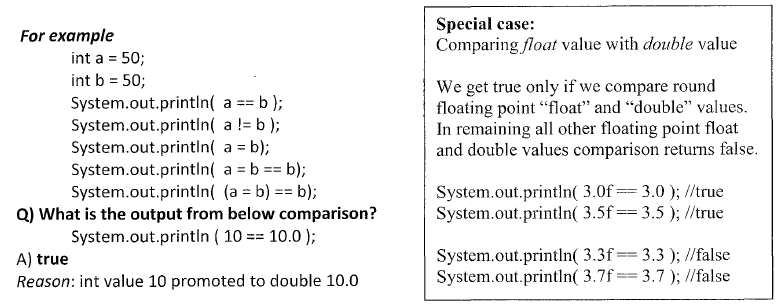
Which can be used to check the Condition, it always returns true or false.



**Let’s suppose variable A hold 8 and B hold 3.**

|  |  |  |
| --- | --- | --- |
| **Operators** | **Example (int A=8, B=3)** | **Result** |
| < | A<B | False |
| <= | A<=10 | True |
| > | A>B | True |
| >= | A<=B | False |
| == | A== B | False |
| != | A!=(-4) | True |

## 



**Rules**

1. Comparison operators <,>,<=,>= can be applied only on numerical types but we never apply on boolean types.

2. Comparison operators ==, != (equality operators) can be applied only on boolean types or only on numerical types separately but not on combination of boolean and numerical types

## Logical Operator

**Logical AND operator (single &)**

This operator will combine multiple conditions and returns result as true only when all the given conditions are true otherwise it returns false.

**Syntax:**

CON1 & CON2 & CON3 & CON4

**Truth table**

|  |  |  |
| --- | --- | --- |
| **C1** | **C2** | **C1&C2** |
| T | F | F |
| F | T | F |
| T | T | T |
| F | F | F |

Here single & operator will evaluate all the conditions and finally returns the result either true or false.

**Eg:**

int a=10;

int b=2;

System.out.println(a<b&++a>++b); //false

System.out.println("a="+a); //11

System.out.println("a="+b); //3

**Logical and operator(double &&)**

**Syntax:**

CON1 && CON2 && CON3 && CON4

**Truth table**

|  |  |  |
| --- | --- | --- |
| C1 | C2 | C1&&C2 |
| T | F | F |
| F | T | F |
| T | T | T |
| F | F | F |

Here double && operator is called as short circuit operator which will not evaluate all the conditions at a time, it will evaluate first condition if it is false then it returns false directly otherwise if first condition is true then it will evaluate the second condition if it is true then third condition,... and finally returns the result. so that the performance of the application will be improved.

**Eg:**

int a=10;

int b=2;

System.out.println(a<b&&++a>++b); //false

System.out.println("a="+a); //10

System.out.println("a="+b); //2

**Logical or operator(single pipe symbol | )**

This operator will combine multiple conditions and returns result as true when any of the given conditions is true otherwise it returns false.

**Syntax:**

CON1 | CON2 | CON3 | CON4

**Truth table**

|  |  |  |
| --- | --- | --- |
| C1 | C2 | C1|C2 |
| T | F | T |
| F | T | T |
| T | T | T |
| F | F | F |

Here this | operator will evaluate all the conditions and finally returns the result either true or false.

**Logical or operator (double pipe symbol(||) )**

**Syntax:**

CON1 || CON2 || CON3 || CON4

**Truth table**

|  |  |  |
| --- | --- | --- |
| C1 | C2 | C1||C2 |
| T | F | T |
| F | T | T |
| T | T | T |
| F | F | F |

Here double || operator is called as short circuit operator which will not evaluate all the conditions at a time, it will evaluate first condition if it is true then it returns true directly otherwise if first condition is false then it will evaluate the second condition if it is false third condition,... and finally returns the result as either true or false. so that the performance of the application will be improved.

**Logical exclusive-or (xor) operator(^)**

This operator will combine multiple conditions and returns result as false when all the given conditions are same otherwise it returns true.

**syntax:**

CON1 ^ CON2 ^ CON3 ^ CON4

**Truth table**

|  |  |  |
| --- | --- | --- |
| C1 | C2 | C1^C2 |
| T | F | T |
| F | T | T |
| T | T | F |
| F | F | F |

Here xor( ^ ) operator will evaluate all the conditions and finally returns the result either true or false.

**Logical not operator(!)**

This operator will complement the given condition or expression.

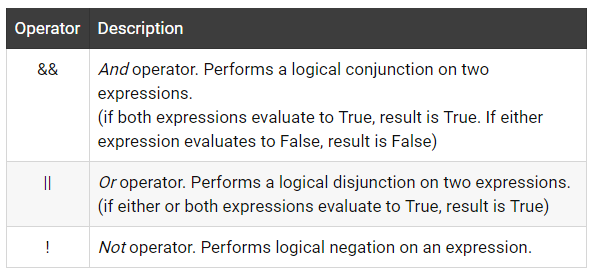
**Syntax:**

!(condition or combinations of conditions)

**Truth table**

|  |  |
| --- | --- |
| C | !C |
| T | F |
| F | T |

Which can be used to combine more than one Condition?. Suppose you want to combined two conditions A<B and B>C, then you need to use Logical Operator like (A<B) && (B>C). Here **&&**is Logical Operator.



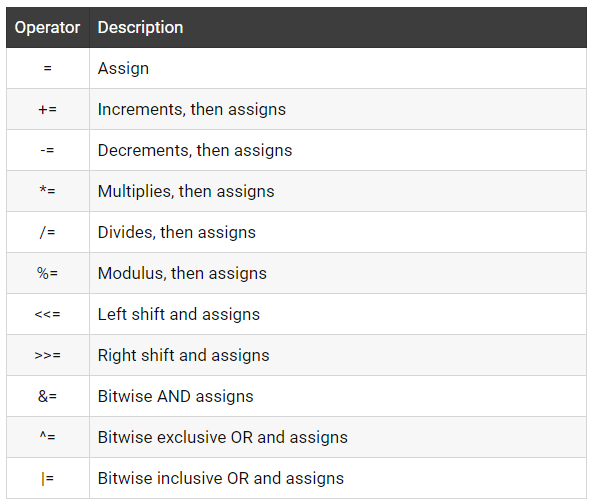
|  |  |  |
| --- | --- | --- |
| **Operator** | **Example (int A=8, B=3, C=-10)** | **Result** |
| && | (A<B) && (B>C) | False |
| || | (B!=-C) || (A==B) | True |
| ! | !(B<=-A) | True |

## Truth table of Logical Operator

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **C1** | **C2** | **C1 && C2** | **C1 || C2** | **!C1** | **!C2** |
| T | T | T | T | F | F |
| T | F | F | T | F | T |
| F | T | F | T | T | F |
| F | F | F | F | T | T |

## Assignment operators

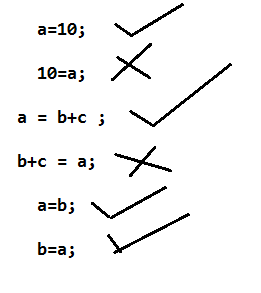
Which can be used to assign a value to a variable.



**Let’s suppose variable A hold 8 and B hold 3.**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Example (int A=8, B=3)** | **Result** |
| += | A+=B or A=A+B | 11 |
| -= | A-=3 or A=A+3 | 5 |
| \*= | A\*=7 or A=A\*7 | 56 |
| /= | A/=B or A=A/B | 2 |
| %= | A%=5 or A=A%5 | 3 |

**Rules**

1. When we write assignment operators then right side we can write anything but left we must write only variable.

1. We can also write assignment operator along with arithmetical operators which are called as **compound assignment.**

Operators, compound assignment operators will do automatic type casting.

**a=a+b --> a+=b**

**a=a-b --> a-=b**

**a=a\*b --> a\*=b**

**a=a/b --> a/=b**

**a=a%b --> a%=b**

***// wap to demo on compound assignment operators***

*class Sample{*

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

*byte a,b;*

*a=10;*

*b=20;*

*// a=(byte)(a+b);*  ***//type casting***

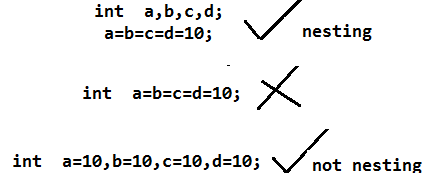
*a+=b;* ***// type casting is done automatically***

*System.out.println("Addition="+a);*

*}*

*}*

1. We can also nest the assignment operators



**Ternary oeprator or ?: operator or conditional operator**

This operator returns the value based on given condition.

**syntax:**

(condition)?value1:value2;

Here if condition is true then it returns value1 otherwise if condition is false then it returns value2.

**Eg1:**

*int a=10;*

*int b=2;*

*int c=(a>b)?a:b;*

*System.out.println(c+"Is Big Number");*

**Eg2:**

*int a=10;*

*int b=20;*

*int c;*

*c = (a>b)?a-b:b-a;*

*System.out.println("Subtraction="+c);*

We can also nest the ternary operator

**Eg3:**

*int a=10;*

*int b=2;*

*int c=6;*

*int d;*

*d=(a>=b&&a>=c)?a:((b>=c&&b>=a)?b:c);*

*System.out.println(d+"Is Big Number");*

Bitwise Operator:

Java defines several bitwise operators that can be applied to the integer types long, int, short, char and byte

|  |  |
| --- | --- |
| **Operator** | **Description** |
| & | Bitwise AND |
| | | Bitwise OR |
| ^ | Bitwise exclusive OR |
| << | left shift |
| >> | right shift |

**Now let’s see truth table for bitwise &, | and ^**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **a** | **B** | **a & b** | **a | b** | **a ^ b** |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 |

The bitwise shift operators shift the bit value. The left operand specifies the value to be shifted and the right operand specifies the number of positions that the bits in the value are to be shifted. Both operands have the same precedence.

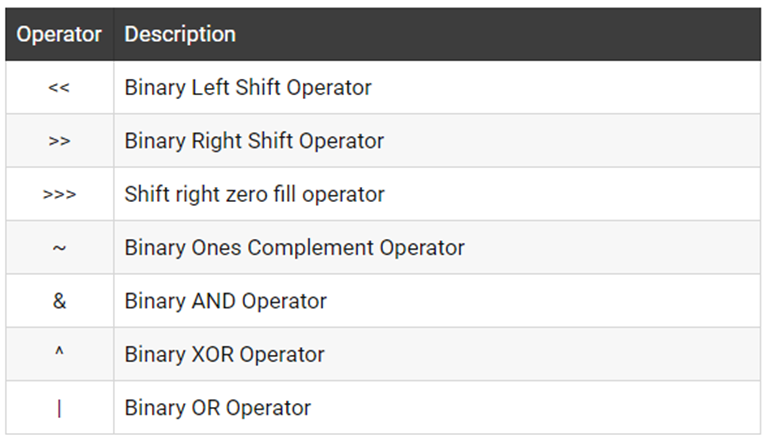
**Example**

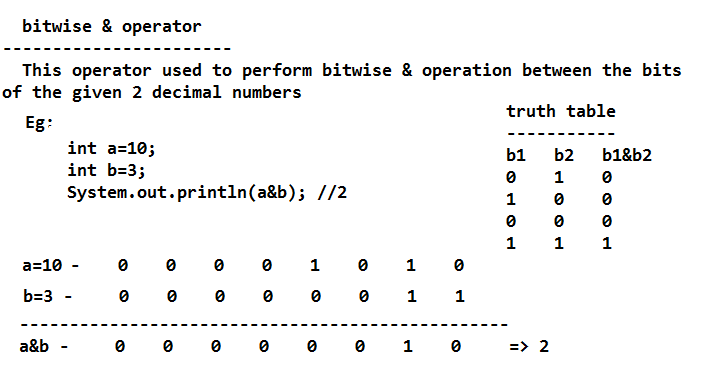
a = 0001000

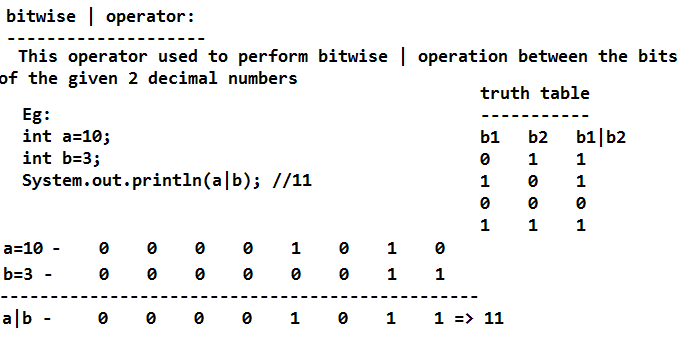
b= 2

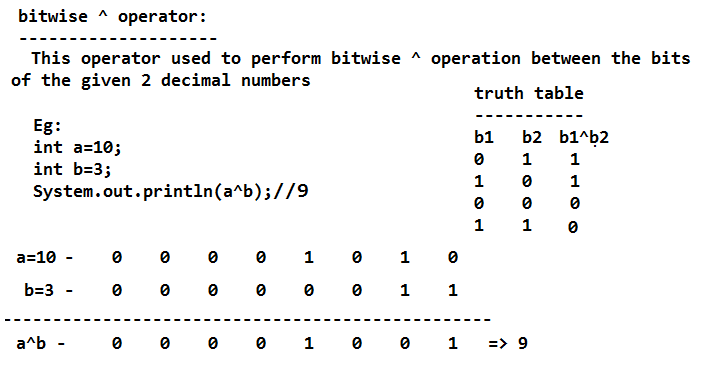
a << b= 0100000

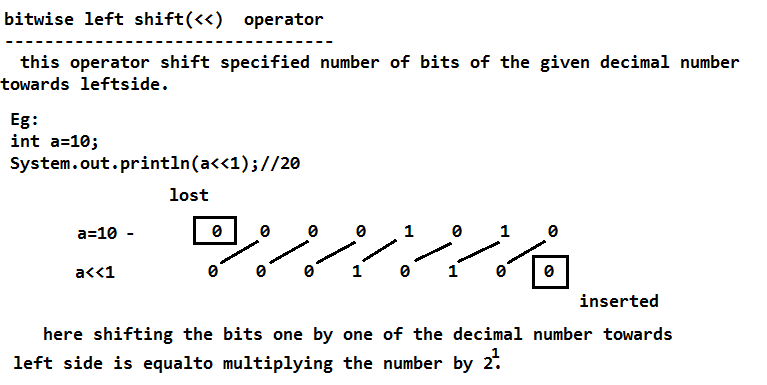
a >> b= 0000010

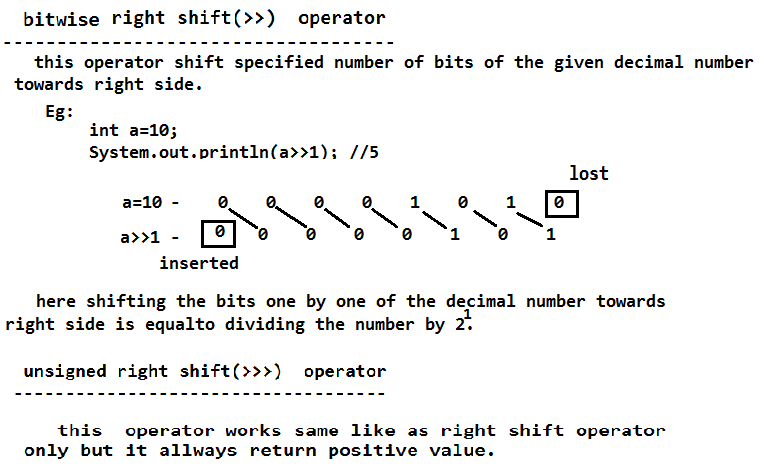


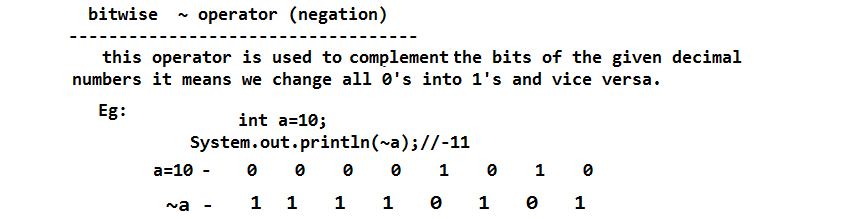


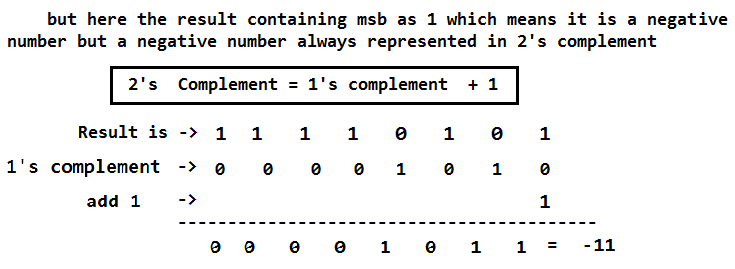










****



#### instanceOf operator

This operator is used for object reference variables. The operator checks whether the object is of particular type (class type or interface type)

**Increment or decrement operators**

Increment or decrement operators are used to increase or decrease the value of the variable by 1.

**Eg:**

++,--

**1. Pre increment(++a)**

pre increment means first value will be increased and next used in the application.

**a=10;**

**p = ++a;**

**p ->11**

**a ->11**

**2. Post increment(a++)**

Post increment means first value will be used before increment and next it will be increased.

**a=10;**

**p = a++;**

**p ->10**

**a ->11**

3. **Pre decrement(--a)**

Pre decrement means first value will be decreased and next used in the application.

**a=10;**

**p = --a;**

**p ->9**

**a ->9**

**4. Post decrement(a--)**

Post decrement means first value will be used before decrement and next it will be

decreased.

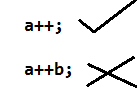
**a=10;**

**p = a--;**

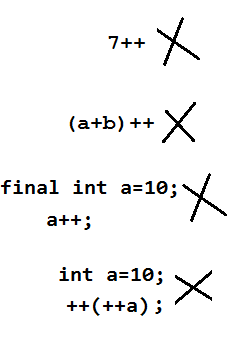
**p ->10**

**a ->9**

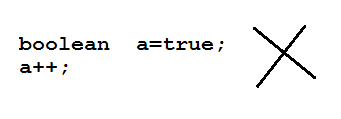
**Rules**

1. Incr/Decr operators are unary operators always used with single variable.

2. We never write Incr/Decr operators along with any value or expression or constant.

 - Nesting Incr/Decr operators is not possible

3. We never apply Incr/Decr operators on boolean data types



**Unary minus**

Unary minus operator is used to change the sign of the given number.

**Eg:**

*int a=10;*

*a = -a;*

*System.out.println(a); //-10*

**new operator**

- new is java keyword or operator which is used to create the object

-We can create an object for both user defined classes and predefine classes

-Creating an object nothing but allocating the memory so that we can use in the application.

-Once an object created that will be located inside the Heap memory of JVM.

**Syntax:**

ClassName referencevariable = new ClassName();

**Eg:**

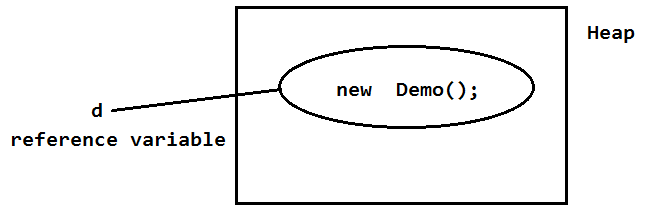
*Declaration of class*

*class Demo{*

*}*

*Declaration of object*

*Demo d = new Demo();*



**. (Dot) operator**

- This operator used to access the members of the class using reference or column like follows

**Eg:**

*reference.variable*

*reference.method()*

*ClassName.variable*

*ClassName.method()*

this operator can also be used to refer or identify the class from a package

**Eg:**

java.lang.NoSuchMethodError

java.lang.String

......

## *Operator Precedence*

Precedencedecides which operator will be evaluated first in a case where more than one operator is present in the same calculation.

## Operator Precedence Table

|  |  |
| --- | --- |
| **Operators** | **Precedence** |
| Postfix | *expr*++ *expr*-- |
| Unary | ++*expr* --*expr* +*expr* -*expr* ~ ! |
| Multiplicative | \* / % |
| Additive | + - |
| Shift | << >> >>> |
| Relational | < > <= >= instanceof |
| Equality | == != |
| bitwise AND | & |
| bitwise exclusive OR | ^ |
| bitwise inclusive OR | | |
| logical AND | && |
| logical OR | || |
| Ternary | ? : |
| Assignment | = += -= \*= /= %= &= ^= |= <<= >>= >>>= |

## Example of Precedence

|  |  |
| --- | --- |
|  | *class OperatorPrecedenceExample {*  *public static void main(String args[]) {*  *int i = 40;*  *int j = 80;*  *int k = 40;*  *int l = i + j / k;*  */\**  *\* In above calculation we are not using any bracket. So which operator*  *\* will be evaluated first is decided by Precedence. As precedence of*  *\* divison(/) is higher then plus(+) as per above table so divison will*  *\* be evaluated first and then plus.*  *\**  *\* So the output will be 42.*  *\*/*  *System.out.println("value of L :" + l);*  *int m = (i + j) / k;*  /\*  \* In above calculation brackets are used so precedence will not come in  \* picture and plus(+) will be evaluated first and then divison()/. So  \* output will be 3  \*/  *System.out.println("Value of M:" + m);*  }  } |

## Operator Associativity

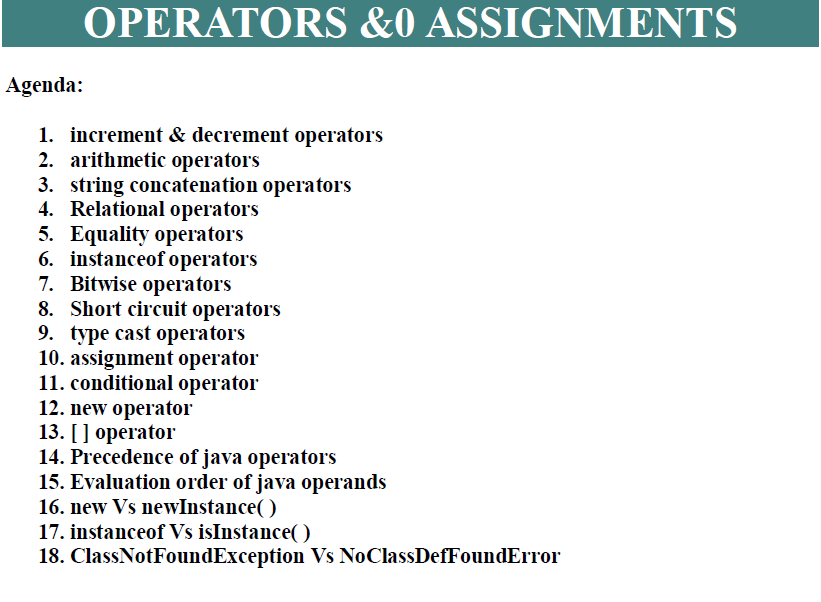
If two operators have the same precedence in the calculation then ***Associativity*** of the operators will be used to decide which operator will be executed first.

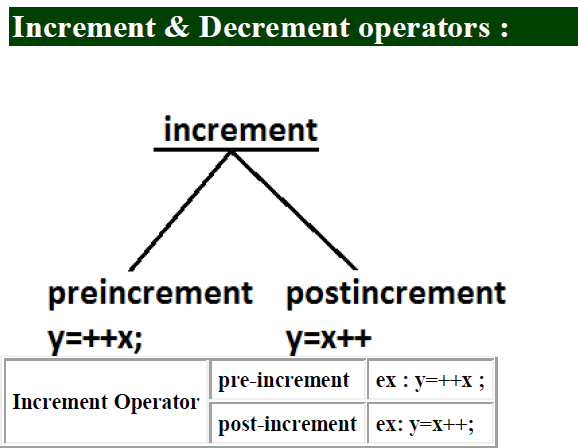
## Example of Associativity

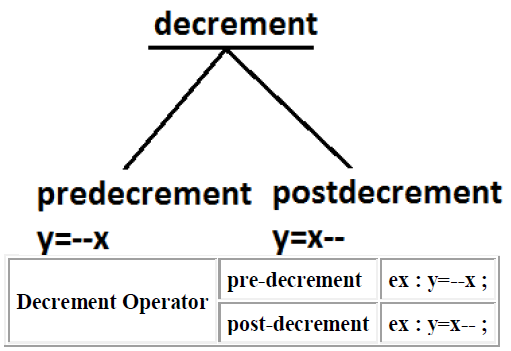
|  |  |
| --- | --- |
|  | package jbt.bean;  /\*  \* Here we will see the effect of precedence in operators life  \*/  public class OperatorAssociativityExample {    public static void main(String args[]) {  int i = 40;  int j = 80;  int k = 40;    int l = i / k \* 2 + j;  /\*  \* In above calculation we are not using any bracket. And there are two  \* operator of same precedence(divion and multiplication) so which  \* operator(/ or \*) will be evaluated first is decided by association.  \* Associativity of \* & / is left to right. So divison will be evaluated  \* first then multiplication.  \*  \* So the output will be 82.  \*/    System.out.println("value of L :" + l);    int m = i / (k \* 2) + j;  /\*  \* In above calculation brackets are used so associativity will not come  \* in picture and multiply(\*) will be evaluated first and then  \* divison()/. So output will be 80  \*/    System.out.println("Value of M:" + m);  }    } |

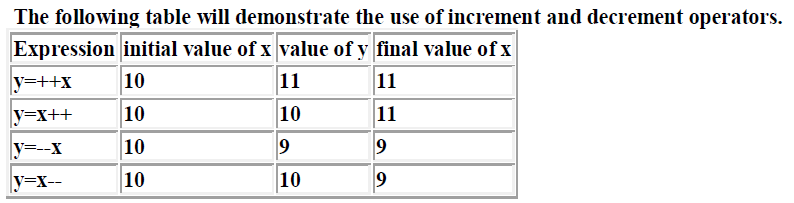
 =================================================

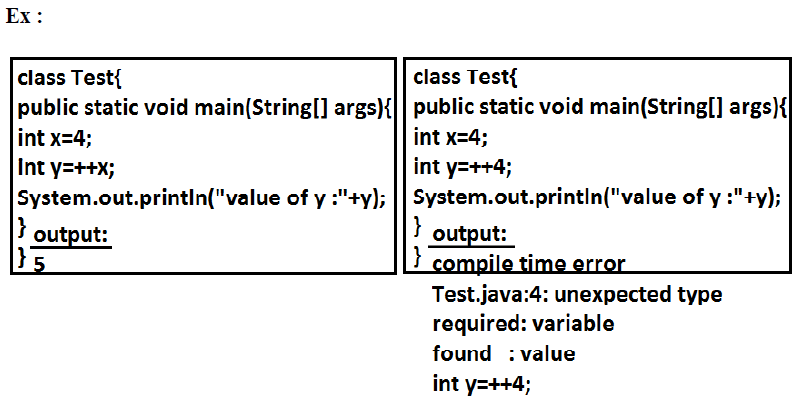
**SCJP:**

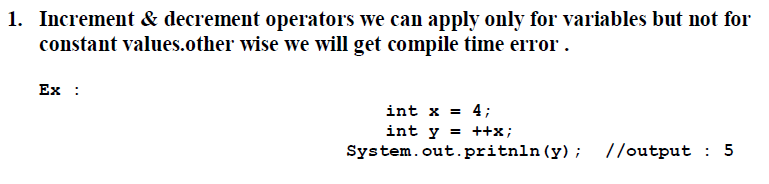


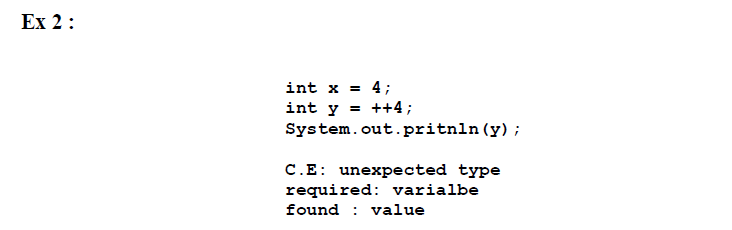


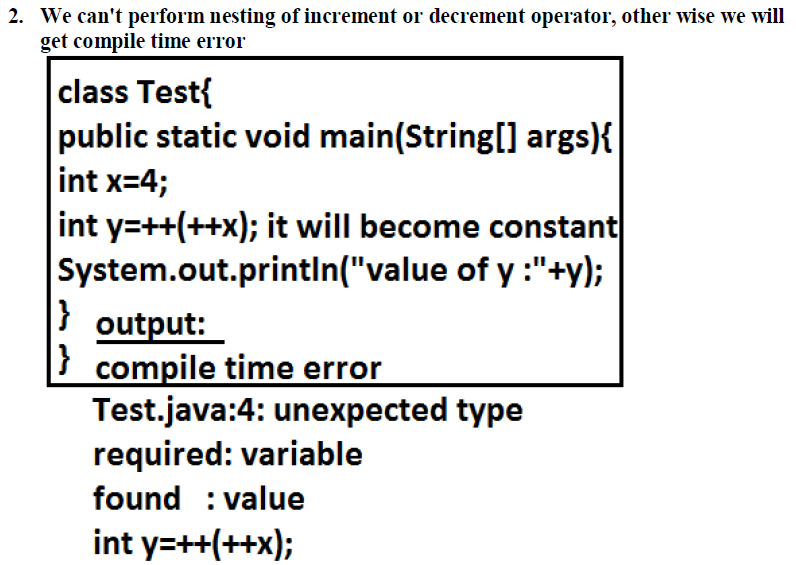


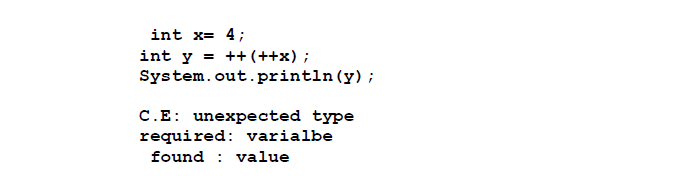


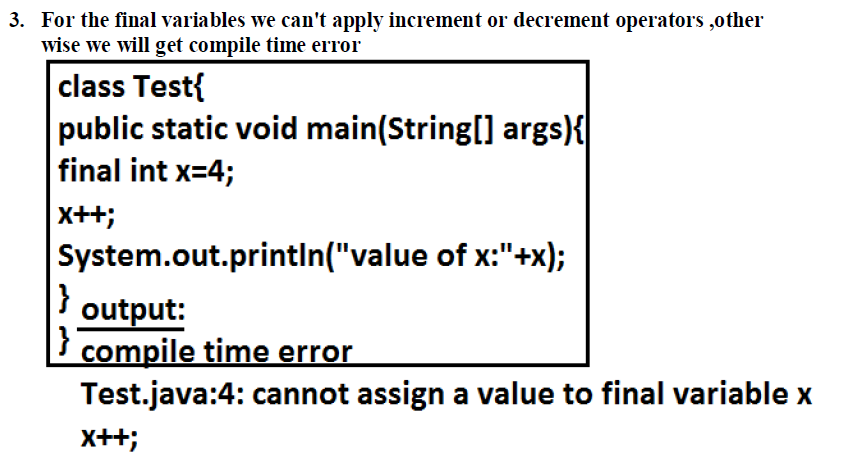


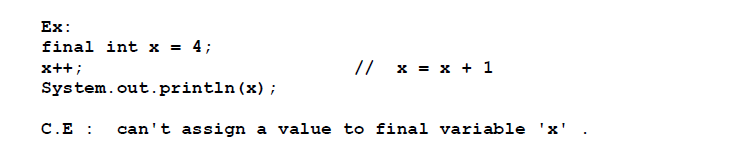


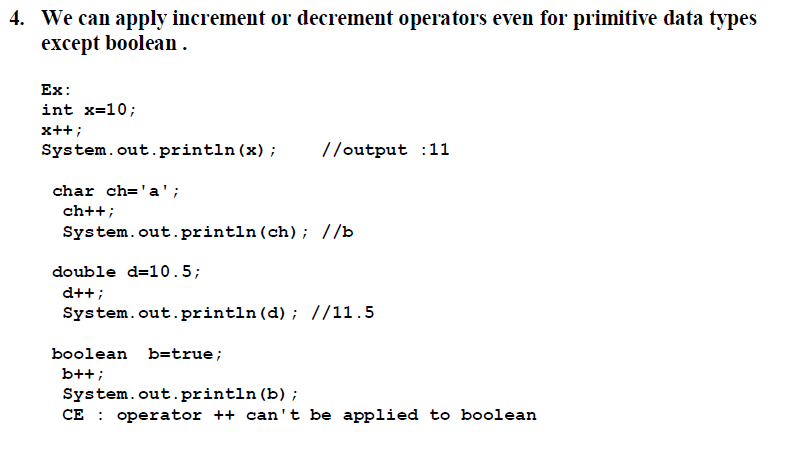


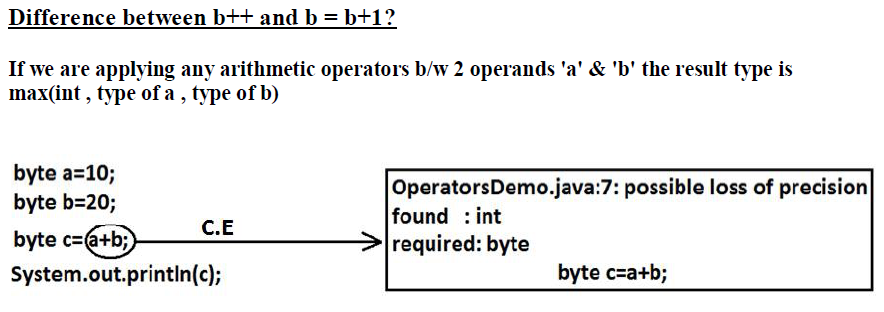


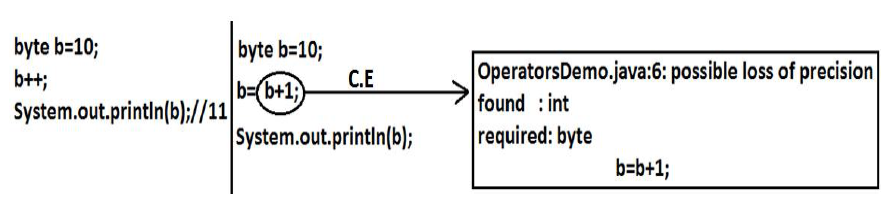


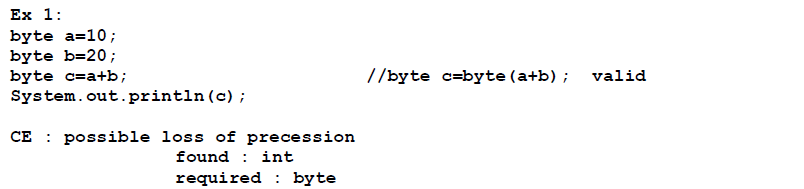


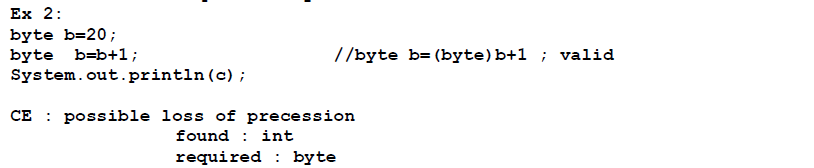


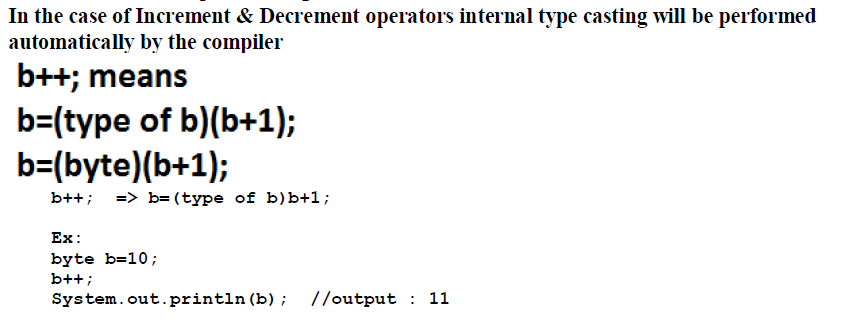


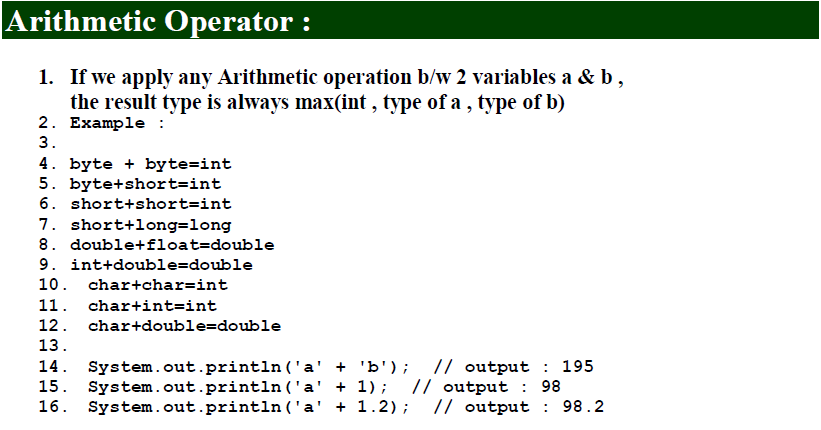


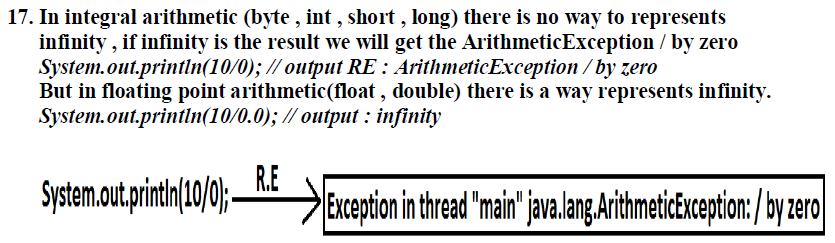


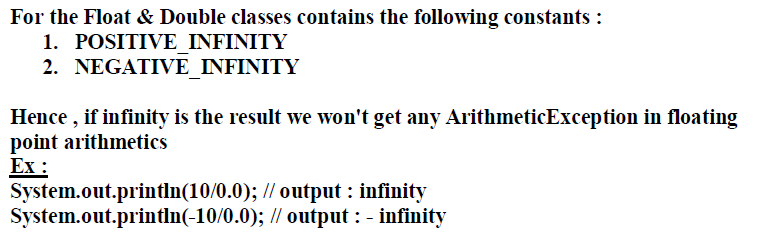


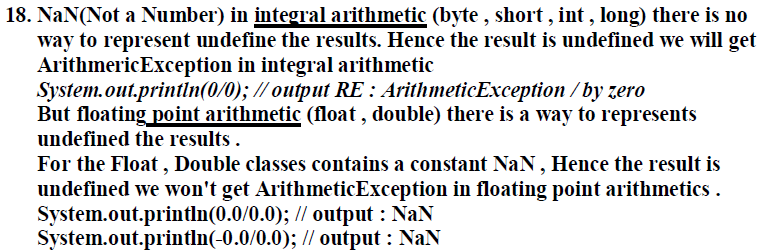


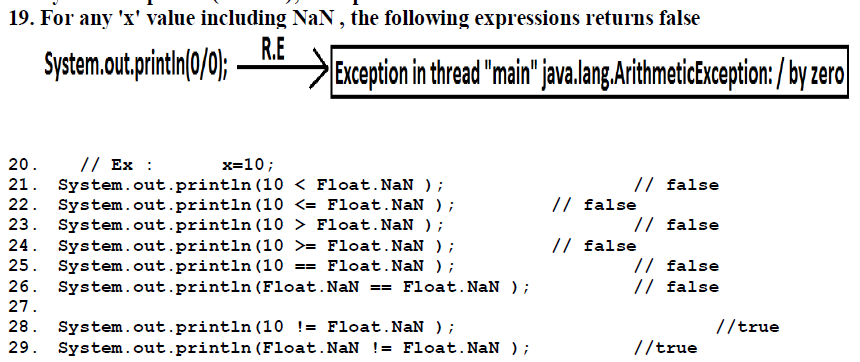


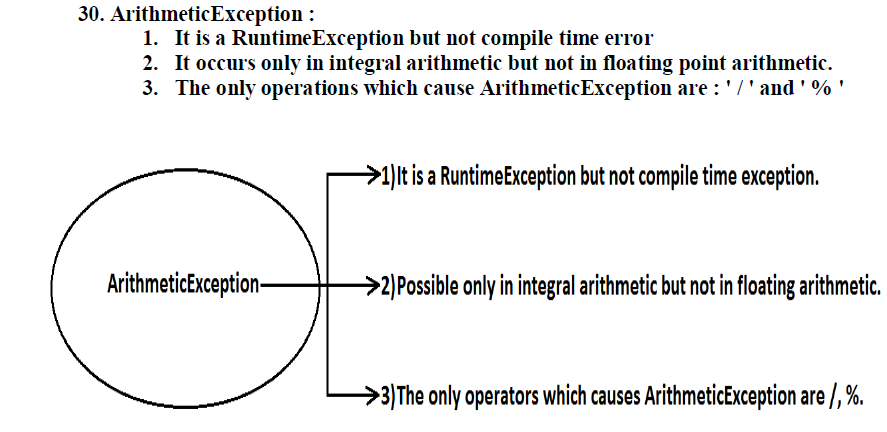


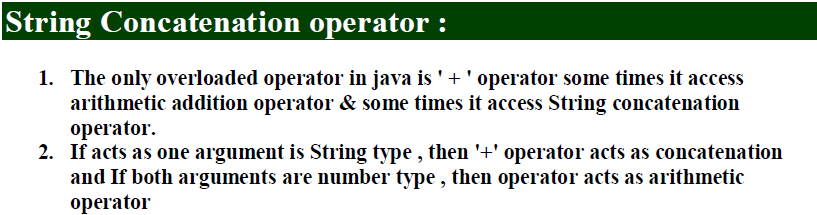


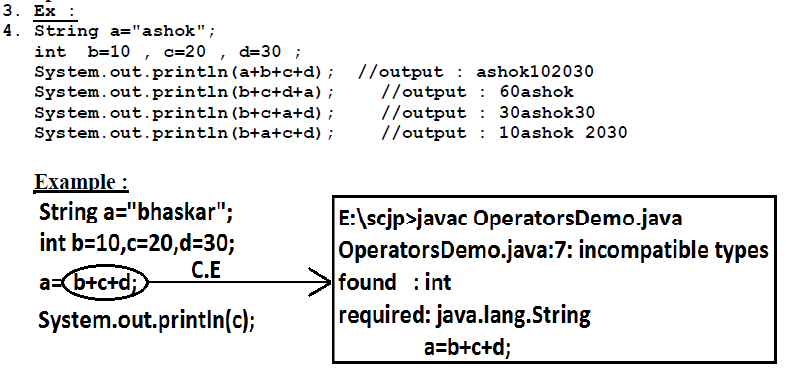


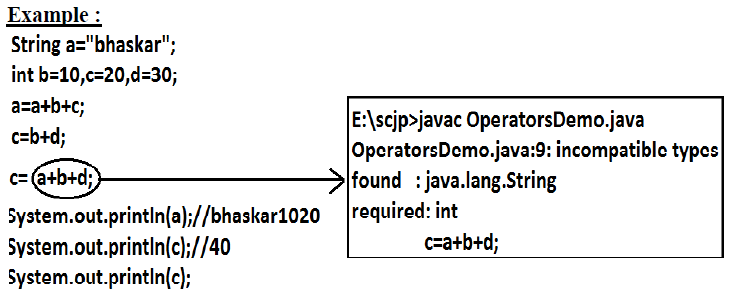


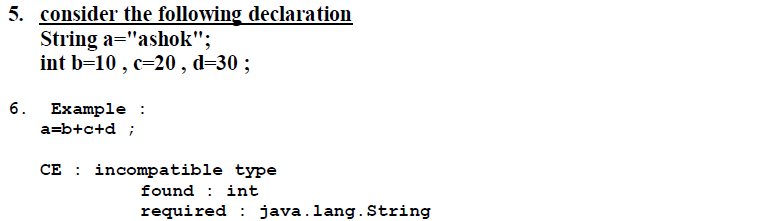


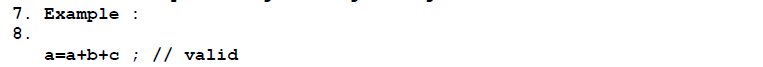




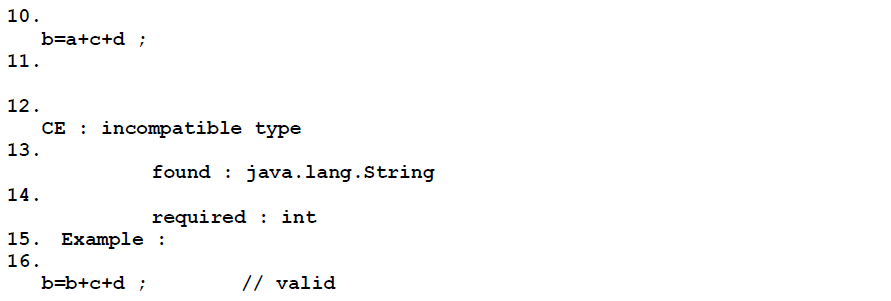


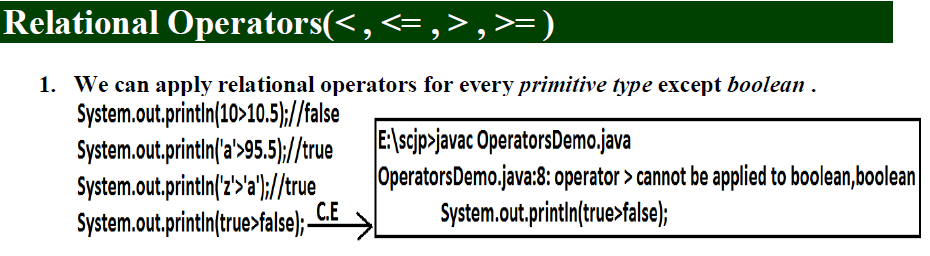


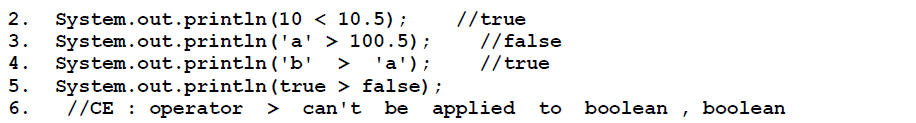


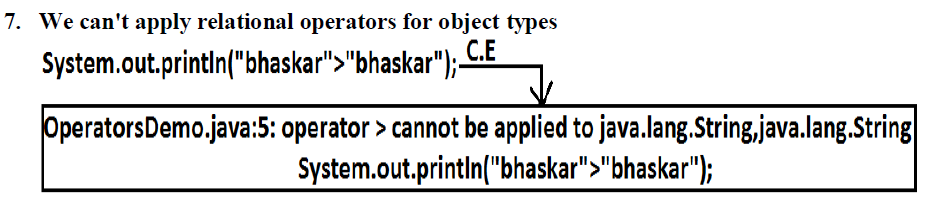


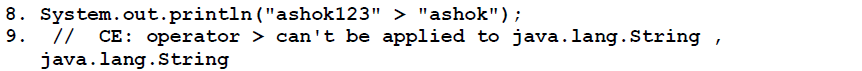


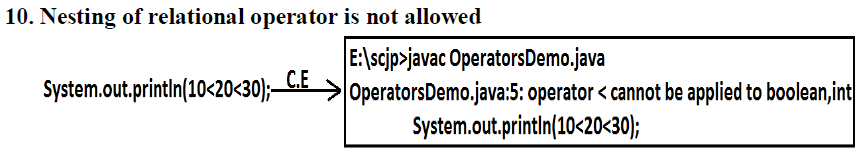


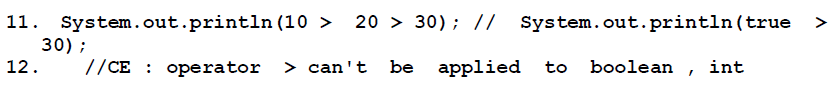


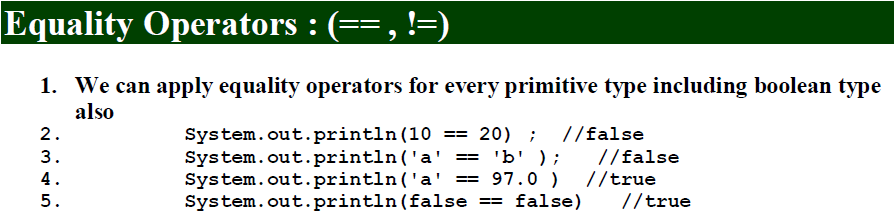


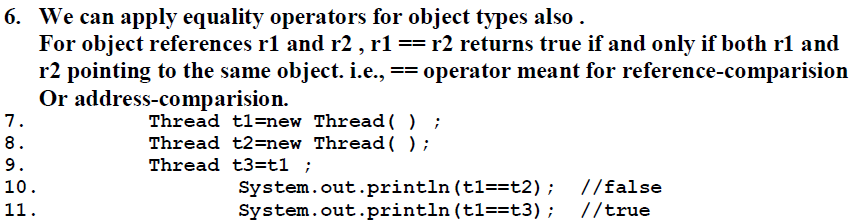


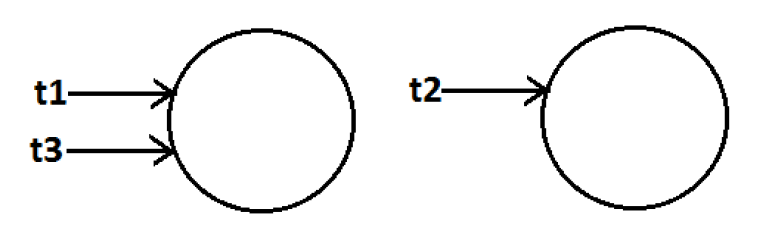


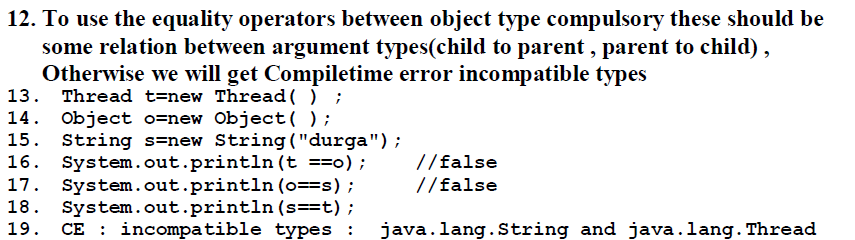


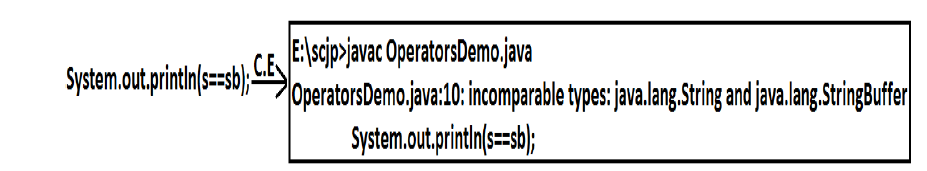


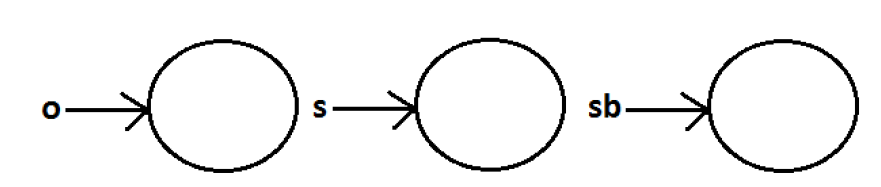


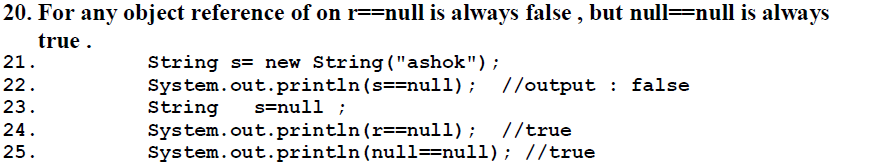




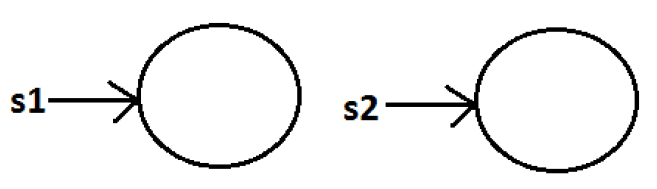


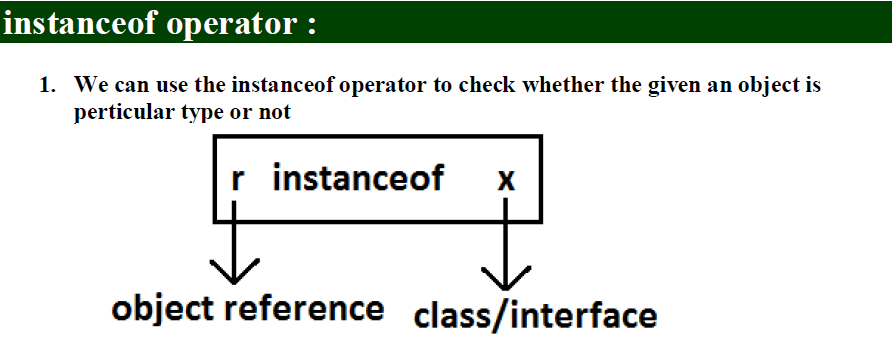


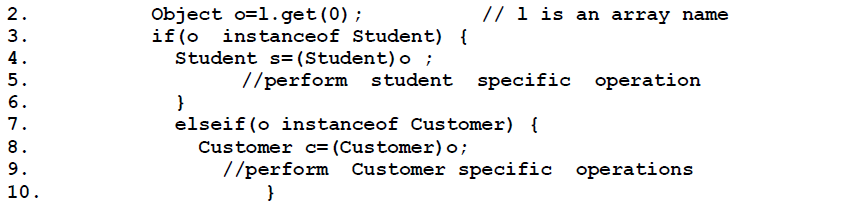


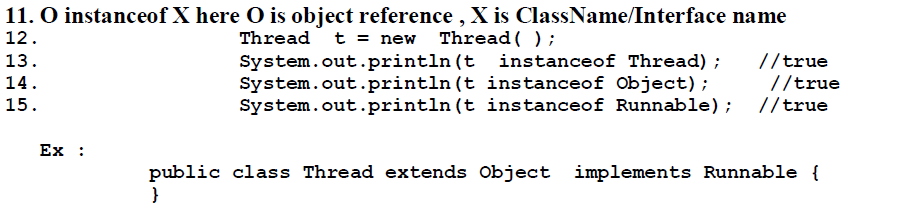


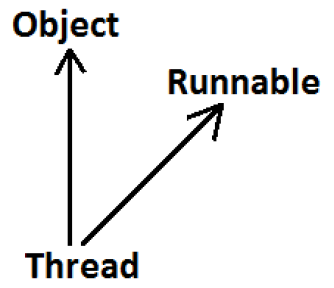


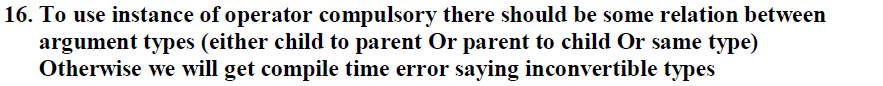


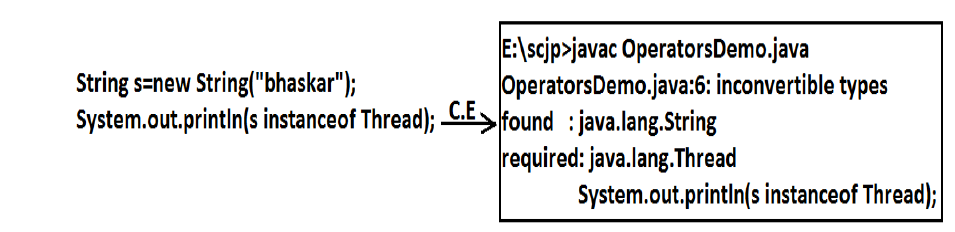


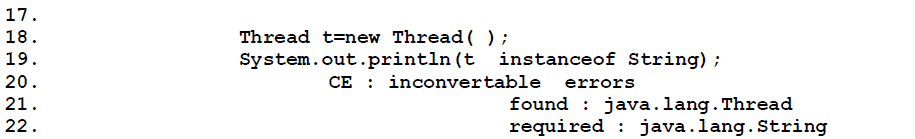


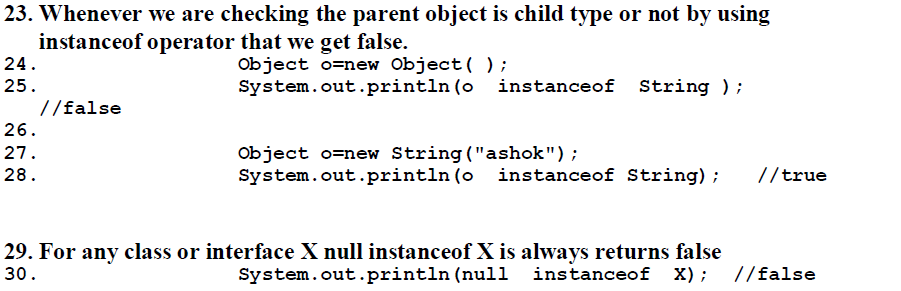


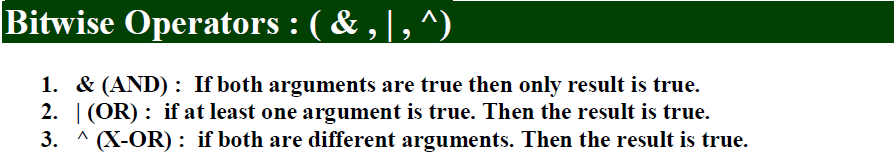


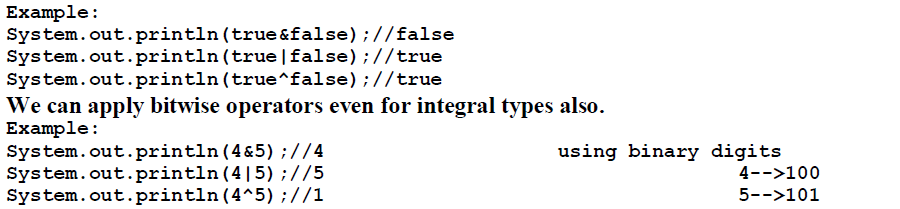


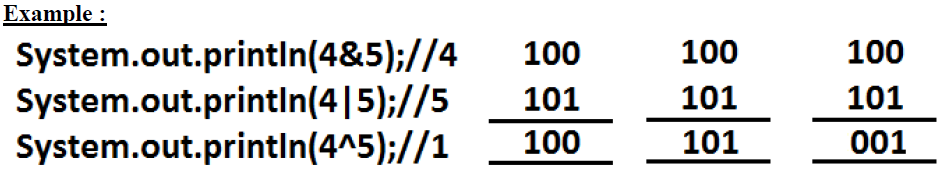


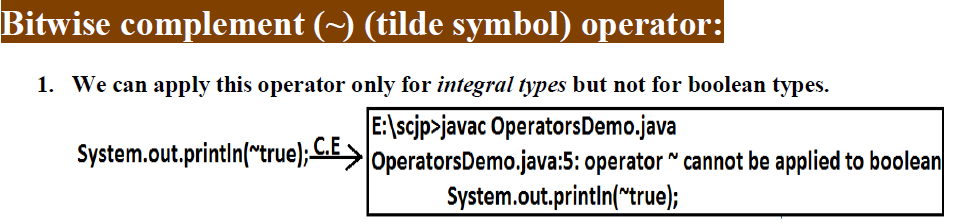


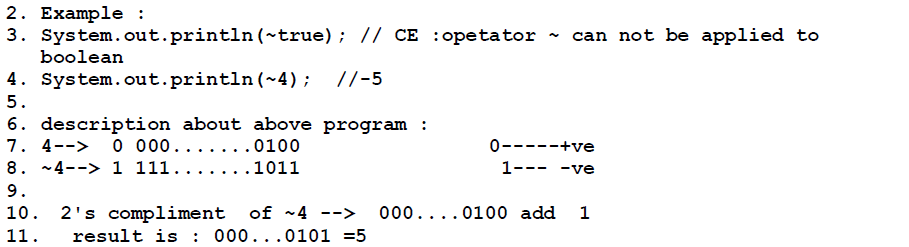


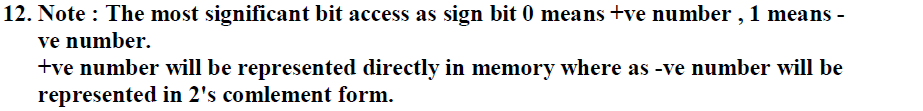


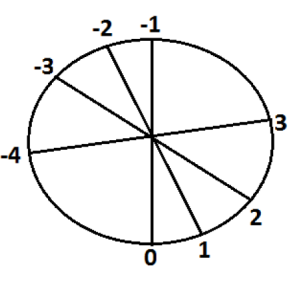


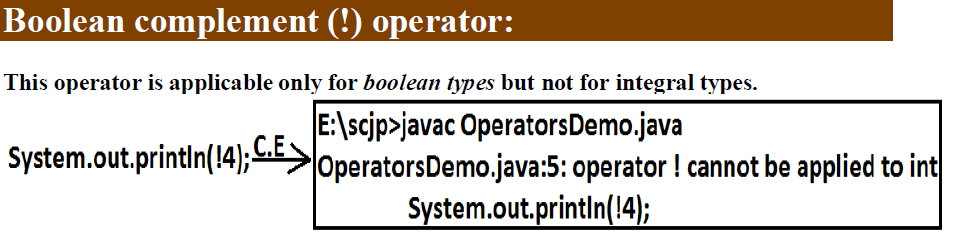


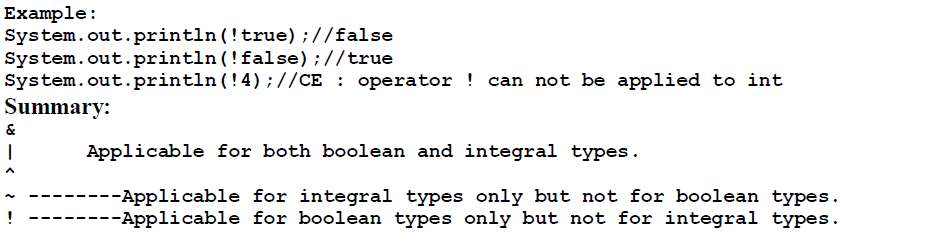


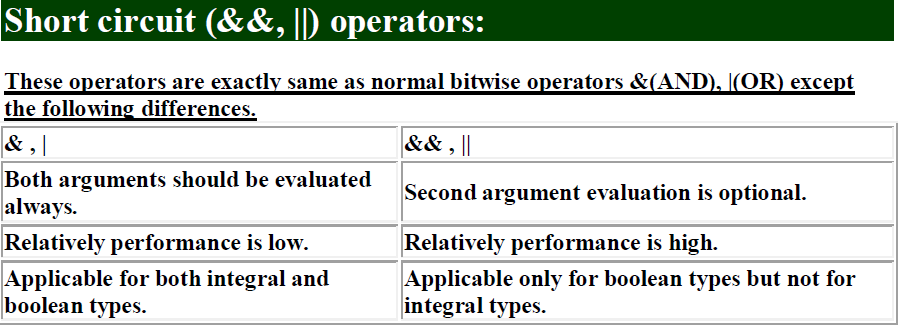


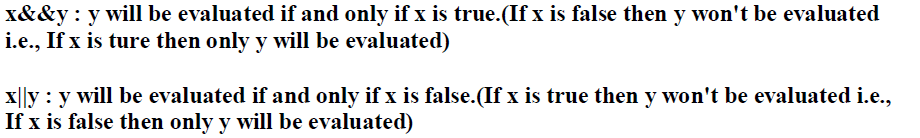


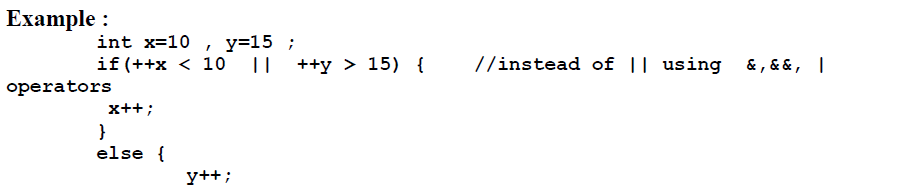


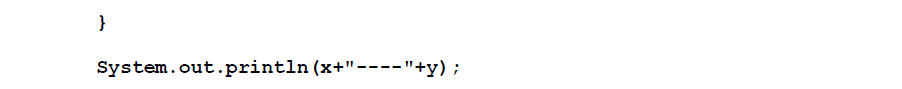


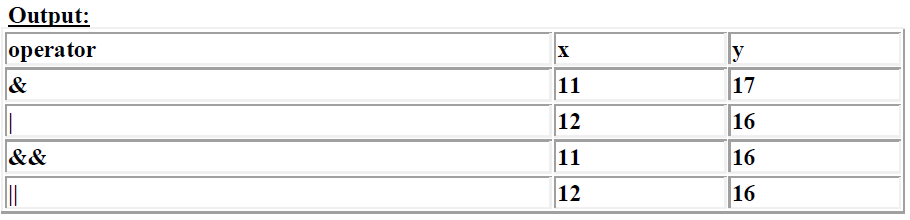


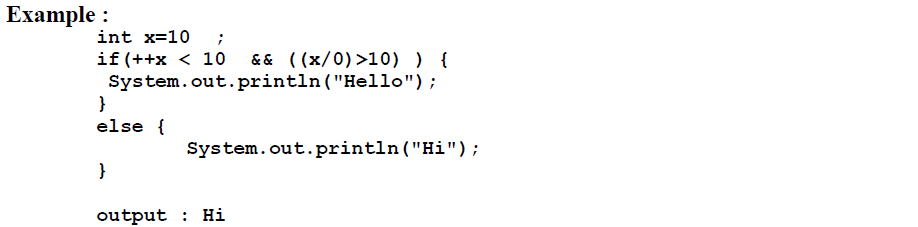








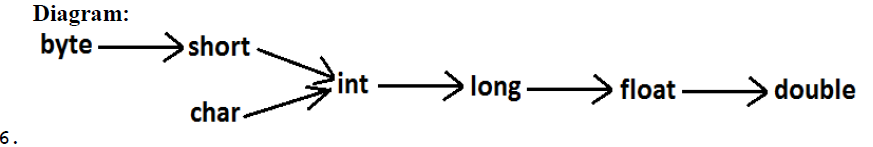




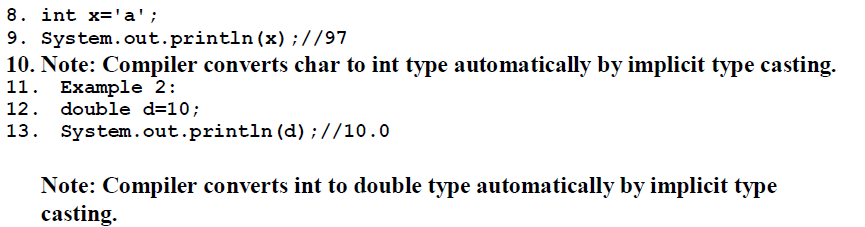


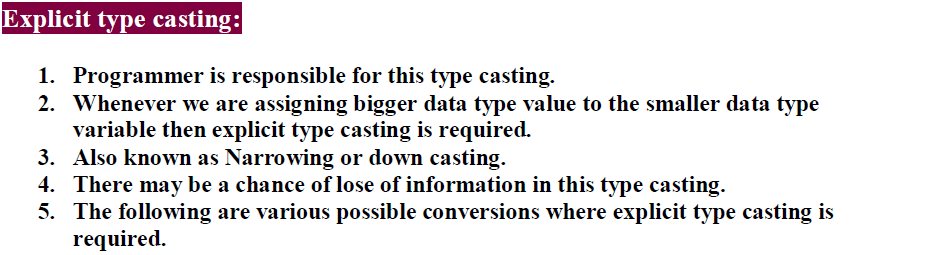


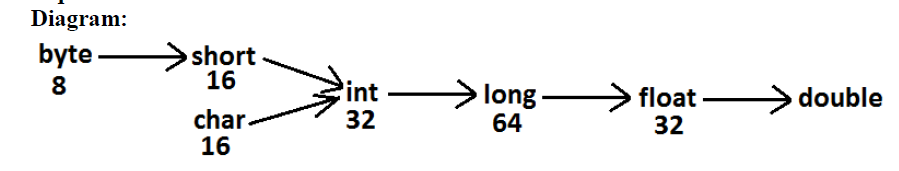


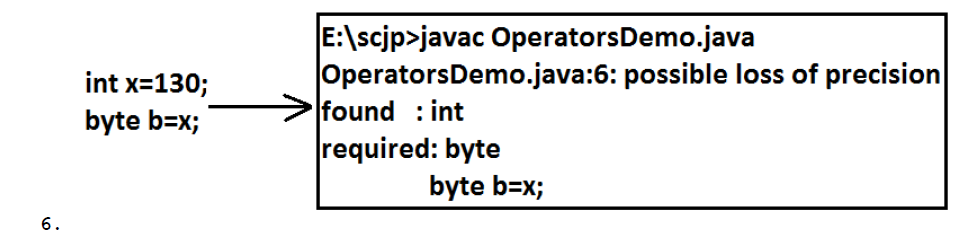


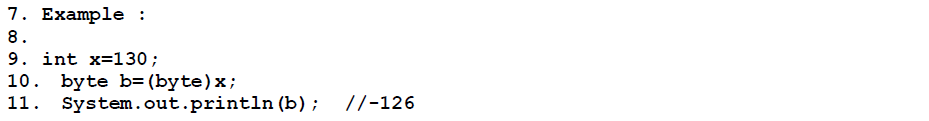




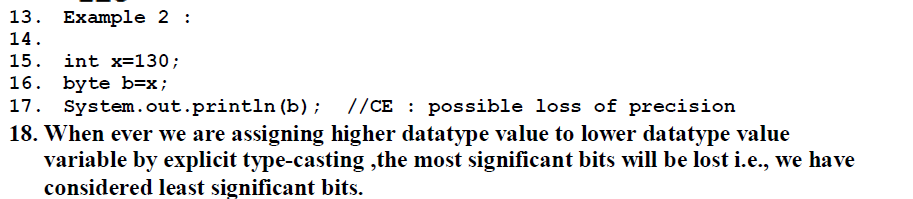


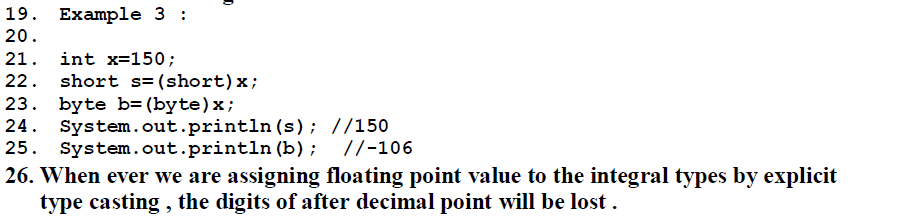


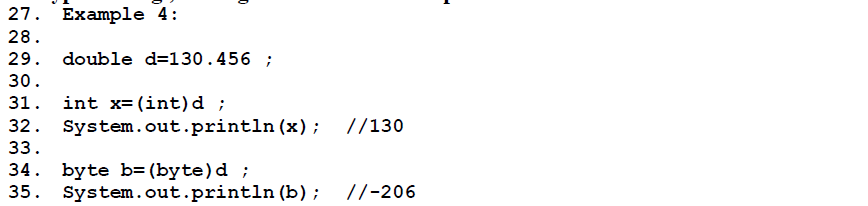


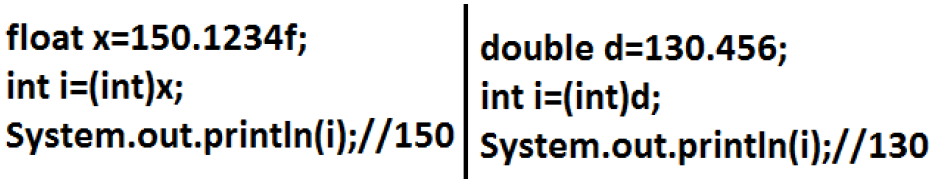


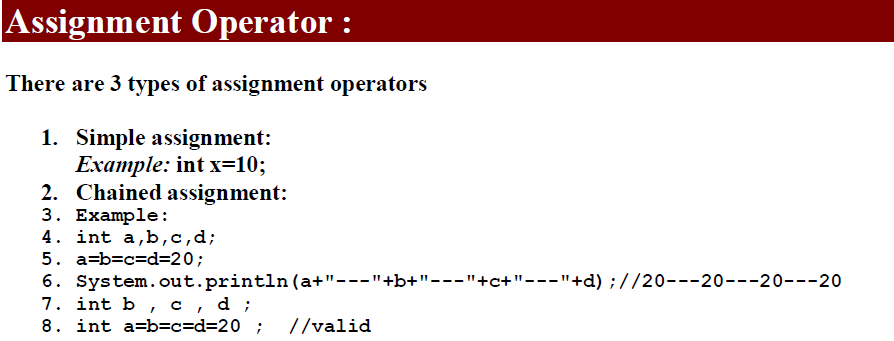


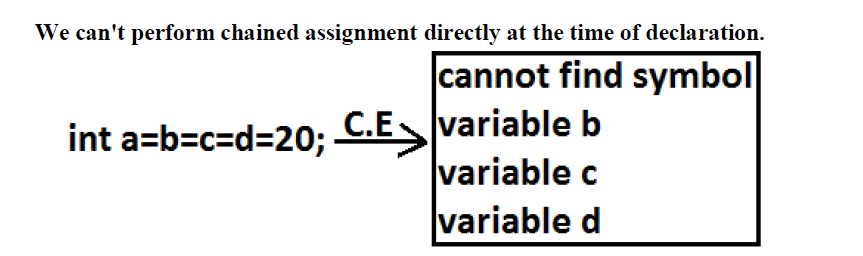


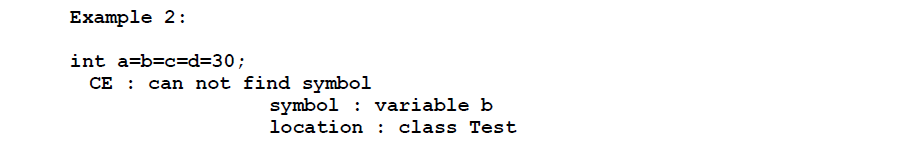


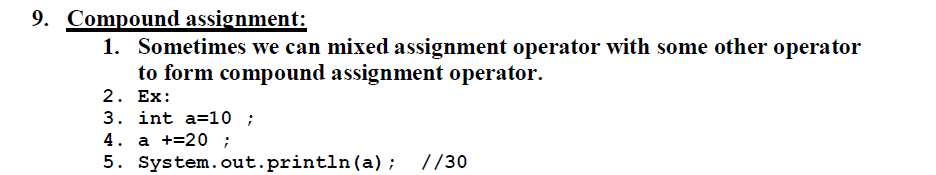


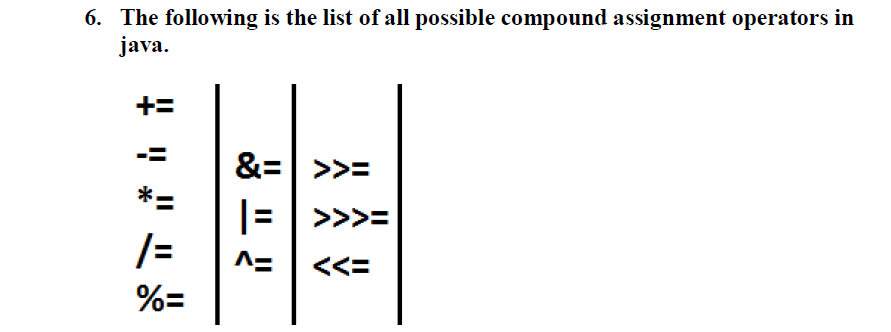


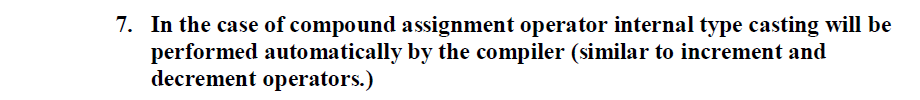


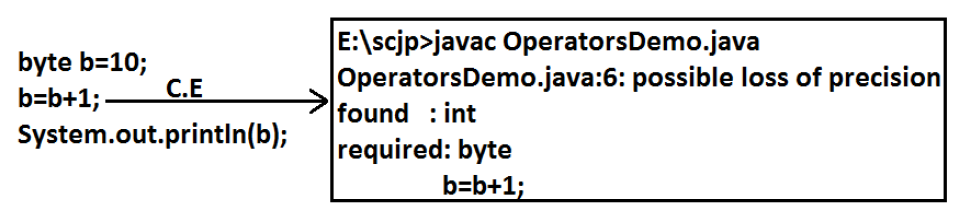


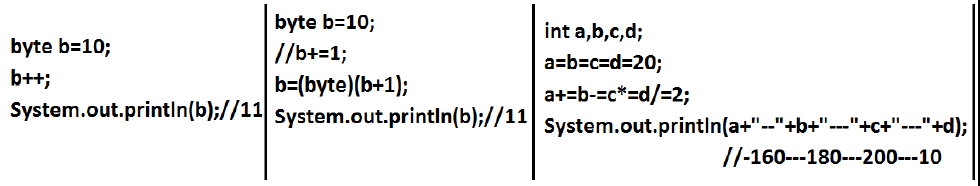


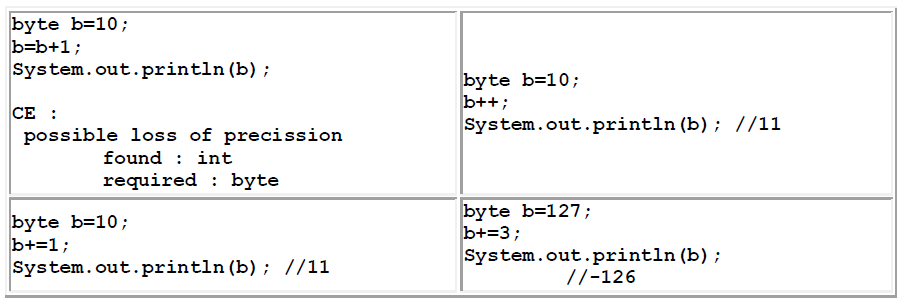


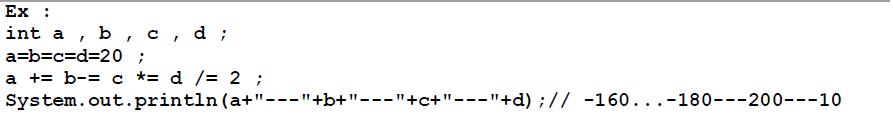


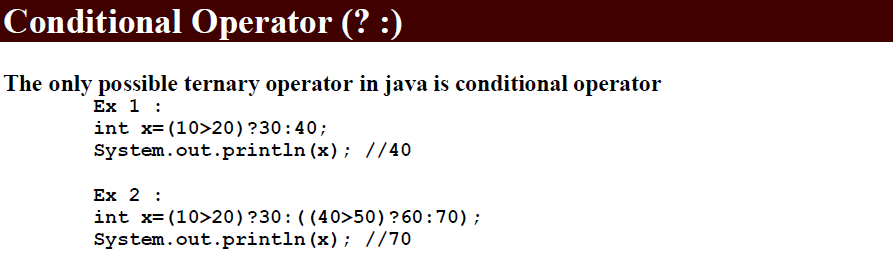


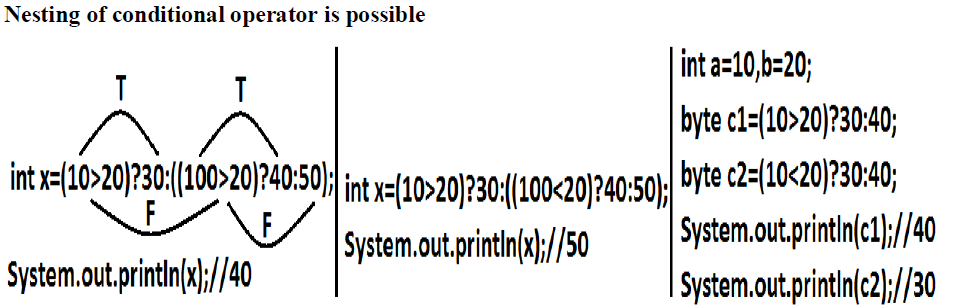


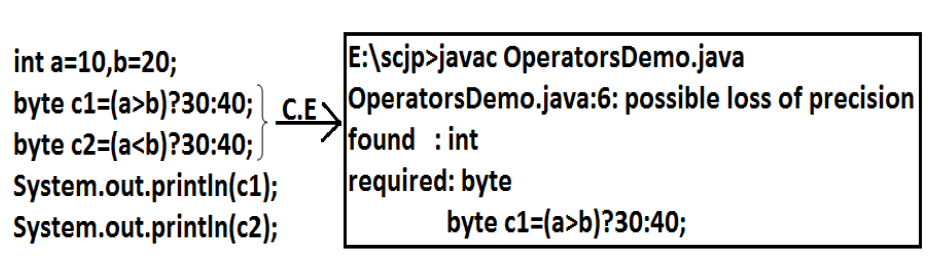


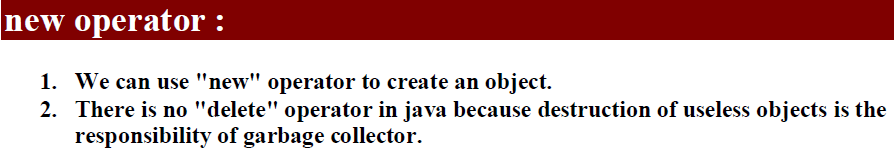




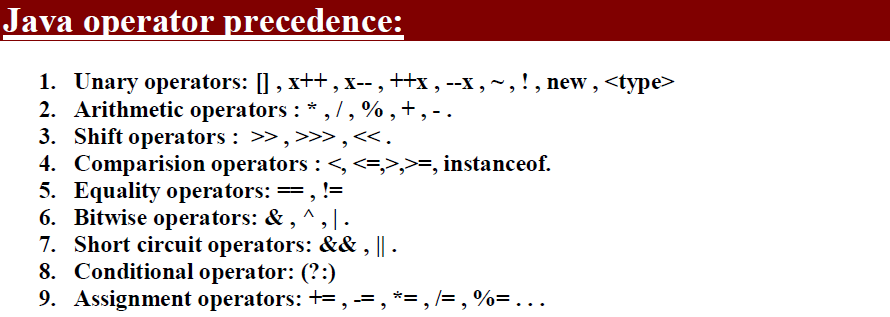


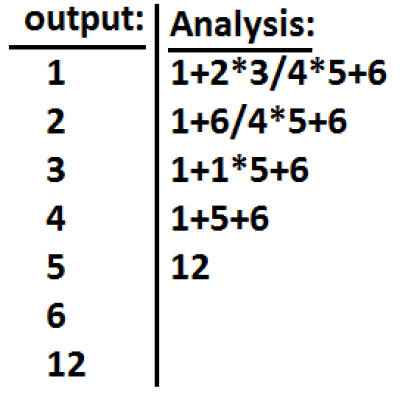
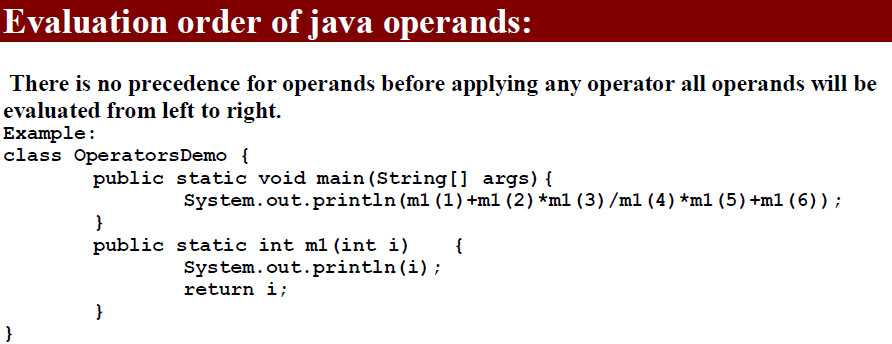


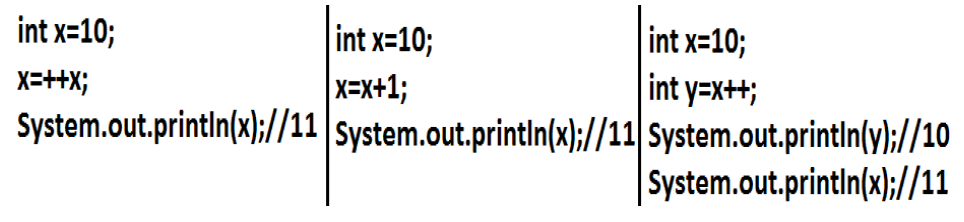


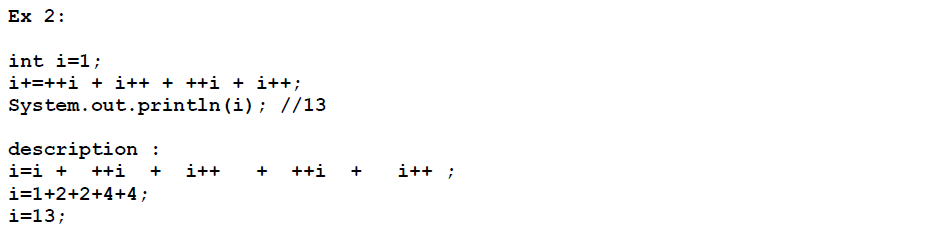


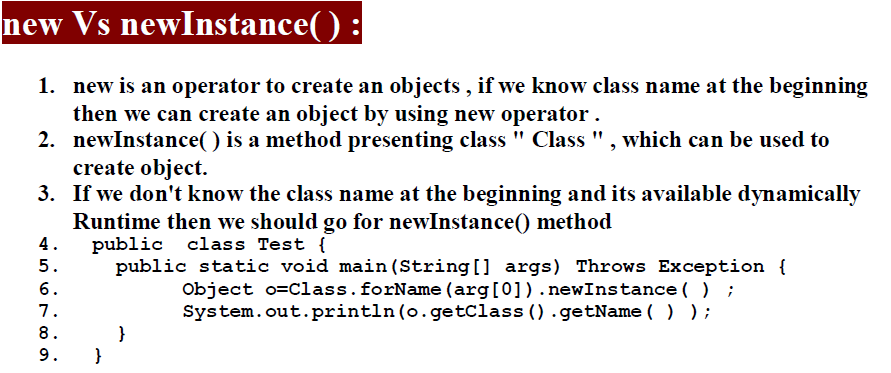


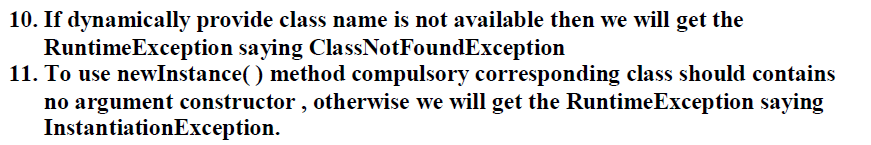


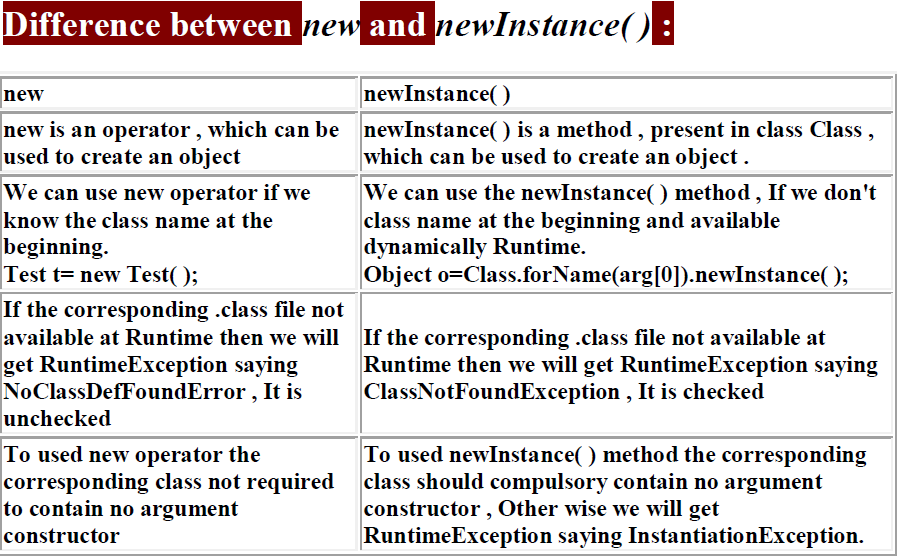


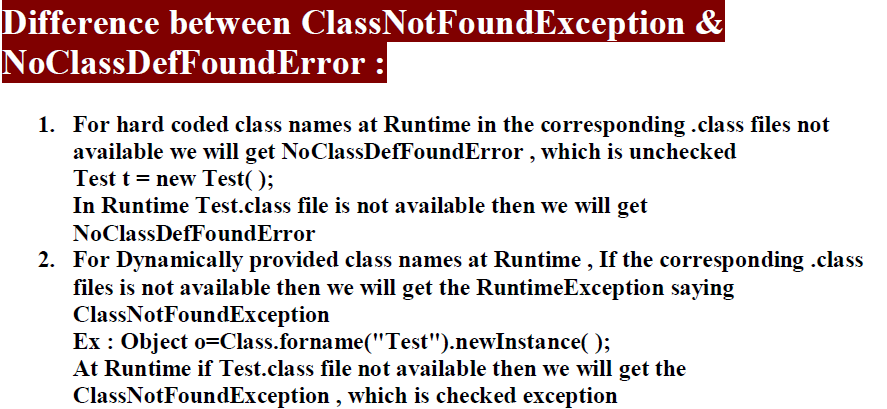


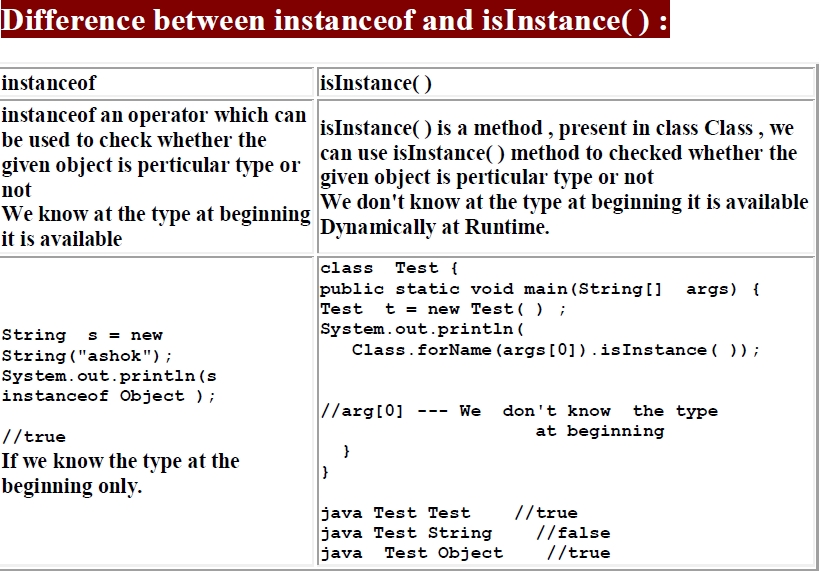


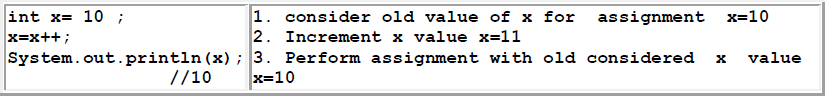












**This program converts the temperature in Fahrenheit to Celsius.**

**The formula used is *C = 5.0 \* (F - 32.0) / 9.0*.**

*class TemperatureConversion{*  
*public static void main(String arg[])    {*  
*double fahrenheit = 98.4;*  
*double celsius  = ( 5.0 \* (fahrenheit - 32.0) ) / 9.0;*  
*System.out.println(fahrenheit + " F is same as " + celsius + " C.");*  
*}*  
*}*

**OUTPUT :** 98.4 F is same as 36.888888888888886 C.

**Change the program to convert from celsius to fahrenheit. The formula will be *F = (( 9.0 \* C ) / 5.0) + 32.0.***

**Top 20 Basic Java Interview Programs on Increment Decrement operators**

**1.Java Interview program on pre increment operator.**

*package com.instanceofjava;*

*class IncrementDemo{*

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

*int a, b;*

*a=10;*

*b=++a;*

*System.out.println(b);*

*}*

*}*

**Output:**

11

**2.Java Interview program on post increment operator.**

*package com.instanceofjava;*

*class IncrementDemo{*

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

*int a, b;*

*a=10;*

*b=a++;*

*System.out.println(a);*

*}*

*}*

**Output:**

11

**3.Java Interview program on pre increment operator in expression.**

*package com.instanceofjava;*

*class IncrementDemo{*

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

*int a=20;*

*a= ++a + 1;*

*System.out.println(a);*

*}*

*}*

**Output:**

22

**4.Java Interview program on post increment operator in expression.**

*package com.instanceofjava;*

*class IncrementDemo{*

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

*int a=20;*

*a= a++ + 1;*

*System.out.println(a);*

*}*

*}*

**Output**:

21

**5.Java Interview program on pre increment operator in expression.**

*package com.instanceofjava;*

*class IncrementDemo{*

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

*int a=20;*

*a= ++a + ++a;*

*System.out.println(a);*

*}*

*}*

**Output:**

43

**6.Java Interview program on post increment operator in expression.**

*package com.instanceofjava;*

*class IncrementDemo{*

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

*int a=20;*

*a= a++ + a++;*

*System.out.println(a);*

*}*

*}*

**Output:**

41

**7.Java Interview program on pre and post increment operator in expression.**

*package com.instanceofjava;*

*class IncrementDemo{*

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

*int a=20;*

*a= a++ + ++a;*

*System.out.println(a);*

*}*

*}*

**Output:**

42

**8.Java Interview program on pre decrement**

*package com.instanceofjava;*

*class IncrementDemo{*

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

*int a, b;*

*a=10;*

*b=--a;*

*System.out.println(b);*

*}*

*}*

**Output:**

9

**9.Java Interview program on post decrement .**

*package com.instanceofjava;*

*class IncrementDemo{*

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

*int a, b;*

*a=30;*

*b=a--;*

*System.out.println(a);*

*}*

*}*

**Output:**

29

**10.Java Interview program on pre decrement in expression.**

*package com.instanceofjava;*

*class IncrementDemo{*

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

*int a=20;*

*a= --a - 1;*

*System.out.println(a);*

*}*

*}*

**Output:**

18

**11.Java Interview program on post decrement in expression.**

*package com.instanceofjava;*

*class IncrementDemo{*

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

*int a=20;*

*a= a-- - 1;*

*System.out.println(a);*

*}*

*}*

**Output:**

19

**12.Java Interview program on pre decrement in expression.**

*package com.instanceofjava;*

*class IncrementDemo{*

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

*int a=20;*

*a= --a - --a;*

*System.out.println(a);*

*}*

*}*

**Output:**

1

**13.Java Interview program on post decrement operator in expression.**

*package com.instanceofjava;*

*class IncrementDemo{*

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

*int a=20;*

*a= a-- - a--;*

*System.out.println(a);*

*}*

*}*

**Output:**

1

**14.Java Interview program on pre and post decrement operator in expression.**

*package com.instanceofjava;*

*class IncrementDemo{*

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

*int a=20;*

*a= a-- - --a;*

*System.out.println(a);*

*}*

*}*

**Output**:

2

**15.Java Interview program on increment and decrement operator in expression.**

*package com.instanceofjava;*

*class IncrementDemo{*

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

*int a=20;*

*a= a-- + ++a;*

*System.out.println(a);*

*}*

*}*

**Output:**

40

**16.increment and decrement operators Inside println() method:**

*package com.instanceofjava;*

*class IncrementDemo{*

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

*int a=1;*

*System.out.println(a++);*

*System.out.println(a++);*

*System.out.println(++a);*

*System.out.println(a++);*

*System.out.println(a++);*

*System.out.println(a--);*

*System.out.println(a--);*

*System.out.println(--a);*

*System.out.println(--a);*

*System.out.println(a--);*

*}*

*}*

**Output:**

1

2

4

4

5

6

5

3

2

2

**17.Java Interview program on increment and decrement operator in println**

*package com.instanceofjava;*

*class IncrementDemo{*

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

*int i=15;*

*System.out.println(i++);*

*System.out.println(i--);*

*}*

*}*

**Output:**

15

16

**18.Java Interview program on increment and decrement operator in println**

*package com.instanceofjava;*

*class IncrementDemo{*

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

*int a=10,b=10;*

*for(int i=0;i<5;i++){*

*if(++a>2||++b>2){*

*a++;*

*}*

*}*

*System.out.println("a= "+a+" b="+b);*

*}*

*}*

**Output:**

20

10

**19.Java Interview program on increment and decrement operator in loops**

*package com.instanceofjava;*

*class IncrementDemo{*

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

*int i;*

*for( i=1;i<4;i++){*

*System.out.println(i);*

*}*

*System.out.println("value of i after completion of loop: "+i);*

*}*

*}*

**Output:**

1

2

3

value of i after completion of loop: 4

**20.Java Interview program on increment and decrement**

*package com.instanceofjava;*

*public class A{*

*static int a = 1111;*

*static {*

*a = a-- - --a;*

*}*

*{  
 a = a++ + ++a;*

*}*

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

*System.out.println(a);*

*}*

*}*

**Output:**

2

**What is the difference between the >> and >>> operators?**

A.

">>" is a signed right shift

">>>" is an unsigned right shift.

If >> is applied on a negative number, the result will still be negative.

>>> ignores the sign of the number.

If >>> is applied on a negative number,the result will be a positive number

The >> fills from the left with the sign bit (0 or 1).

The >>> zero-fills from the left.

**What is the difference between the Boolean & operator and the && operator?**

A.

**For Integers:**

"&" is the "bit-wise AND" operator.

**For boolean arguments:**

"&" constitutes the (unconditional) "logical AND" operator

"&" always evaluates both arguments.

"&&" is defined for two boolean arguments.

It is the "conditional logical AND" operator.

"&&" ealuates the first argument. if it is true, it then evaluates the second.

**Limitations of primitive data types**:

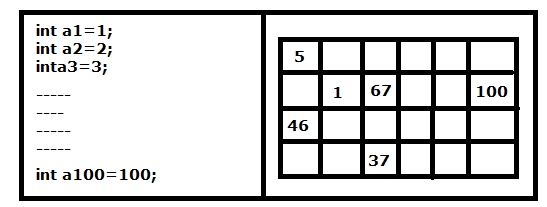
Using primitive data types we can not store multiple values in continuous memory locations

Due to this limitation we have to face 2 problems.

**#1:**

For instance if we want to store multiple values, for example 1 to 100 , we must create 100 variables. All those variables created across JVM at different locations as shown below.

Hence it takes more time to retrieve values.



**#2:**

Also using variables we can not pass all values to the remote computer with single network call, which increases burden on network and also increase lines of code in program.

----------------------------------------------------------------------

**Java - Interview Questions and Answers on New Operator**

**Q1. Difference between Class#getInstance() and new operator ?**

**Ans**. Class.getInstance doesn't call the constructor whereas if we create an object using new operator , we need to have a matching constructor or copiler should provide a default constructor.

**Q2. What are different ways of object creation in Java ?**

**Ans.**

Using new operator - new xyzClass()

Using factory methods - xyzFactory.getInstance( )

Using newInstance( ) method - (Class.forName(“xyzClass”))emp.newInstance( )

By cloning an already available object - (xyzClass)obj1.clone( )

**Q3. Difference between new operator and Class.forName().newInstance() ?**

**Ans**. new operator is used to statically create an instance of object. newInstance() is used to create an object dynamically ( like if the class name needs to be picked from configuration file ). If you know what class needs to be initialized , new is the optimized way of instantiating Class.

**Bit Manipulation Interview Questions and Answers**

**Write a Java method that will return the number of bits that will need to be changed in order to convert an integer, X, into another integer, Y and vice versa. The method should accept two different integers as input. For example, if your method is passed the integers 12 and 16 then your method should return a 3 .**

This problem may confuse you at first, but as always it’s best to break any problem down so that you can understand it better.

Let’s take a closer look at the example given to us. It says that for the input of a 12 and 16, the output of the method should be a 3. Since we are looking for the number of bits that will need to be changed, let’s convert 12 and 16 to binary to further understand this bit manipulation problem. The binary representation of 12 is 01100 and the binary representation of 16 is 10000. Comparing the binary representation of those two numbers, we can see that the first 3 digits are different, which means that to convert 12 to 16 or 16 to 12 we would have to change 3 numbers. And that is why the method that we write should output a 3. Hopefully that’s clear to you now.

So, now we know what we want to do, but the question is how do we actually do it? Here’s a hint: think of the common binary operators and see if you can come up with something that would help solve this problem.

**Using the XOR operator to solve this bit manipulation interview question**

The binary operator that is perfect for this exercise is the XOR operator. If you need a refresher on how that operator works you can read this: XOR in Java. This is because the XOR operator will return true when the binary digits being compared are different, but will return false when they are the same. So, if we just XOR the two numbers being passed in, the result of the XOR operation will be binary number that will have a binary 1 each and every time there is a different bit between the inputs x and y, and a binary 0 when the bits in the input x and y are the same. So, for example, if we XOR 1001 and 0101 the result of the XOR operation would be a 1100 because the 2 leftmost bits are different, so the result of XOR’ing those 2 bits would be a 1 but the 2 rightmost bits in our inputs are the same, so the result of XOR would be a 0.

**The algorithm to solve the bit manipulation interview problem**

So, the algorithm to solve this bit manipulation interview question would be:

First, XOR the two numbers being passed in – call them x and y – and then call the result Z.

Then, just count the number of binary 1’s in Z since that count represents the number of different bits in the two numbers x and y. In order to count the number of binary 1’s in Z, we can just take Z, and perform the binary AND operation with the number 1. When the result of that operation is a 1 then we know that the very last binary digit in Z is a 1. Then, we can shift Z by 1 bit and repeat until there is nothing left to shift – which is when Z is equal to 0.

**Answer in Java to bit manipulation interview question**

Here is the Java implementation of that algorithm and our final answer to this problem:

*public static int findNumberOfBits(int x, int y){*

*int bitCount = 0;*

*int z = x ^ y; //XOR x and y*

*while (z != 0){*

*//increment count if last binary digit is a 1:*

*bitCount += z & 1;*

*z = z >> 1; //shift Z by 1 bit to the right*

*}*

*return bitCount;*

*}*

The code above should be pretty self explanatory as long as you understand the simple algorithm that we used and read the comments in the code. Hopefully you learned something new from this interview question!

-----------------------------------------------------------------------------