24/08/2023

OPERATORS & ASSIGNMENTS

Part-1

1. Increment and decrement operators
2. Airthmetic operators
3. String concatenation operators
4. Relational operators
5. Equality operators
6. instanceof operators
7. bitwise operators
8. Short cut operators
9. type Cast operators
10. assignment operators
11. Conditional operators
12. new operators
13. [] operators
14. operators precedence
15. Evaluation of operands
16. new vs newInstance() (or)
17. instanceof vs isInstance()
18. classNotFoundException vs NoClassDefFoundError

Date

Increment

- Pre-increment
  $y = ++x;$
  first assign then
- Post Increment
  $y = x++$

Decrement

- $y = --x$
  pre decrement
- $y = x--$
  post decrement

expression	initial value of x	value of y	final value of x
$y = ++x$	10	11	11
$y = x++$	10	10	11
$y = --x$	10	9	9
$y = x--$	10	10	9

Date

1)

```
int x=10
int y= ++x;      //x=11
Sopln(y);        y=11
```

```
int x=10
int y= ++10;
Sopln(y);
```

C.E: unexpected type
found: value
required: variable

∴ increment and decrement operator are for variable type not constant

2)

```
int x=10;
int y= ++(++x);
Sopln(y);
```

++x will become value

C.E - Same as above

∴ listing of increment & decrement operator are not allowed

3)

```
final int x=10;
x=11;
Sopln(x);
```

```
final int x=10;
x++;
Sopln(x);
```

Cannot assign a value to final variable

∴ for final variable we can't apply increment & decrement operator

4.

int x=10;	char ch = 'a';	double d=10.5;
x++;	ch++	d++
Sopln(x);	Sopln(ch);	Sopln(d);
	output: b	output 11.5

```
boolean b = true;
b++;
Sopln(b);
```

C.E operator ++ Can not be applied to boolean

5. Clifference b/w b++ and b=b+1;

```
byte b = 10;
b = b+1;
Sopln(b);
```

C.E

max(int, byte, byte) ⇒ int

Solve b = byte(b+1);

```
byte b = 10;
b++;
Sopln(b);
```

Valid

1byte

```
byte a=10;
byte b=20;
byte c = a+b
Sop ln c
```

4byte int

* / + -

max(int, type a, type b)

C.E possible loss of precision

found : int

required : byte

Any Arithmetic operator between two variables 'a' & 'b' the result type is always max(int, type of a, type of b)

Date

We can solve this error (C.E) by performing Type Casting.

```
Class Test
{
    P s v m (String[] args)
    {
        byte a=10;
        byte b=20;
        byte c = (byte) (a+b);
        Sopln (c);
    }
}
```

Output: 30

b++;

⇓

internal type casting is performed automatically

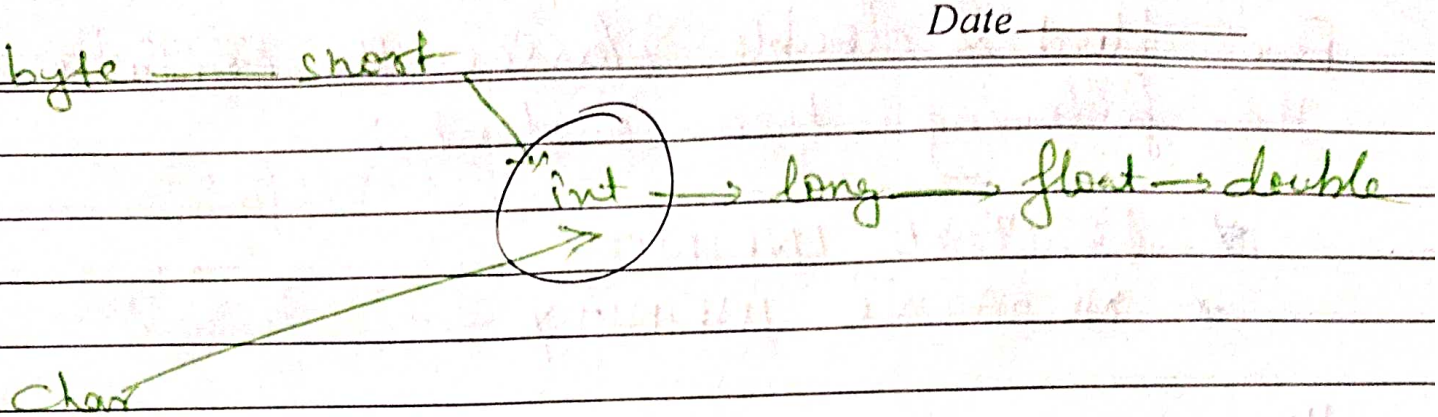
b = (type of b) (b+1);

This is why we don't get compile time error

Airthmetic operator (+, −, *, /, %)

Spiral

Date



byte + byte = int
byte + short = int
~~byte + byte =~~
Short + short = int
byte + long = long
long + double = double
float + long = float
char + char = int

Eg: Sopln ('a' + 'b'); // 195 ✓
(a → 97, b → 98)

Eg: Sopln ('a' + 0.89);
// 97.89 ✓

INFINITY

Sopln ((10/0)); int int int → R.E A.E / by zero
Sopln ((10/0.0)); int → double, double → double
output: Infinity

Sopln (0/0);
Sopln (0.0/0);

in Integral Airthmetic (byte, short, int, long) there is no way to represent infinity, Hence if infinity is the result we will Airthmetic exception in Integral Airthmetic.

For float & double classes the Containing the following two Constant:

Date

* POSITIVE INFINITY
* NEGATIVE INFINITY

Hence even though result is infinity we wont get any airthmetic Exception in floating point airthmetic

Sop (10/0.0); INFINITY

Sop (-10/0.0); - INFINITY

Sop (0/0); int R.E. A.E / by zero
int

Sop (0.0/0) output NAN
NaN

NaN (not a number)

in integral airthmetic (byte, short, int, long) there is no way to represent undefined result. Hence the result is undefined we get A.E.

Eg:

But in floating point Airthmetic (float, double) there is a way to represent Undefined result for this Float Double class contains NaN constant

Hence if the result is Undefined we won't get any A.E Exception. In floating point Airthmetic.

Spiral

Sop(0.0/0) ; NaN

Sopln(-0/0.0) ; NaN

loophole of Nan

False Sopln(10<Float.NaN);

False Sopln(10<=Float.NaN);

False Sopln(10>Float.Nan);

False Sopln(10>=Float.NaN);

False Sopln(10==Float.NaN);

False Sopln(Float.NaN==Float.NaN);

True Sopln(10!=Float.NaN);

True Sopln(Float.NaN!=Float.NaN);

For any x value including NaN the following expressions return false

x<NaN
x<=NaN
x>NaN
x>=NaN
x==NaN

⇒ false

For any x value including NaN the following expressions return True

x!=NaN ⇒ true

Spiral

Date

Airthmetic exception

1. It is runtime exception but not compile-time error
2. In integral Airthmetic possible but not in Floating point Airthmetic
3. The only operators which cause A.E are / and %

- The only operator which is overloaded in java is "+" operator. Sometime it acts as airthmetic addition operator and Some time Concatenation operator.

- if at least one argument is string type then "+" operator acts as Conicatentation and if both argument are number type then it acts as airthmetic addition operator.

Example:

```
String a = "bhaskar"
int b=10, c=20, d=30;
Sopln (a+b+c+d); bhaskar 102030
Sopln (b+c+d+a); 60 bhaskar
Sopln (b+c+a+d); 630 bhaskar30
Sopln (b+a+c+d); 10 bhaskar2030
```

Example 2:

```
String a="bhaskar"
int b=10, c=20, d=30;
a=b+c+d; → C.E
Sopln (c);
```

incompatible type
found: int
required: java.lang.String
a=b+c+d;

Example 3:

```
String a = "bhaskar"
int b=10, c=20, d=30;
a = a+b+c;
c = b+d;
c = a+b+d; ← C.E
```

C.E
incompatible types
found: Java.lang.String
required: int
c=a+b+d;

Spiral

Date

```
Sopln(a);// bhaskar1020
sopln(c);// 40
Sop ln(c);
```

Relational operator:

(<, <=, >, >=)

- We can apply relational operator for every primitive type except boolean

Example

```
Sopln(10>10.5);
Sopln('a'>95.5);
Sopln('z'>'a');
Sopln(True>False);
```

- We can't apply relational operators for the object types.

Examples

```
Sopln("Amrit" > "Amrit")
```

C:\E

operator > cannot be applied to java.lang.String, java.lang.String

```
Sopln("Amrit" > "Amrit");
```

Example:

System.out.println(10<20<30); Date

C.E → operator < cannot be applied to boolean, int

sopln(10<20<30)

(True<30)

Boolean < int

Equality operator:

(==, !=)

sopln(10==20); false

sopln('a'=='b'); false

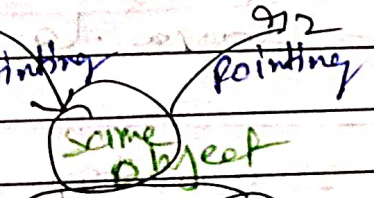
soln('a'==97.0); True

sopln(false==false); true

// we can apply equality operator for every primitive type including Boolean type also.

Universal operator

Equality operator



r1==r2 ⇒ True

we can apply for object type also

for object references r1, r2

return True if and only if both references pointing to same object

Reference comparison are address Compari.

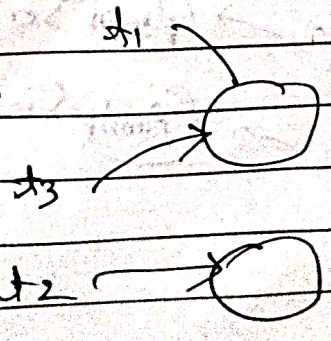
// Thread t1 = new Thread();

Thread t2 = new thread();

thread t3 = t1

sopln(t1==t2); false

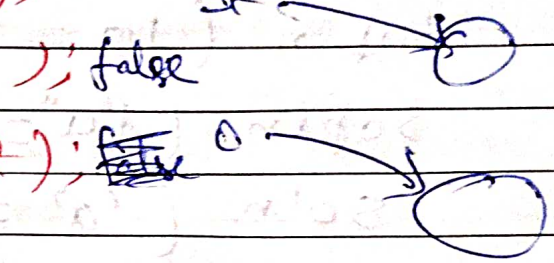
sopln(t1==t3); True



Spiral

Date

```
Thread t = new Thread();
Object o = new Object();
String s = new String("Amrit");
Sopln(t==o); false
Sopln(o==s); false
Sopln(s==t); C.E
```



There should be relation b/w argument type other parent to child or child to parent or same reference type other wise we will C.E incompatible type Java.lang.String & Java.lang.Thread

---

Difference b/w == operator and equals() method

```
String S1 = new String("Amrit");
String S2 = new String("Amrit");
Sopln(S1==S2); // False
Sopln(S1.equals(S2)); // True
```

S1 → Amrit
S2 → Amrit

Date

In general we can use == operator for reference comparison (address comparison) and .equals() for content comparison.

r1 == null ⇒ false

for any reference are, == null is always false

null == null ⇒ True

~~int null~~

String s = new String("durga");
Sop(s == null); false.

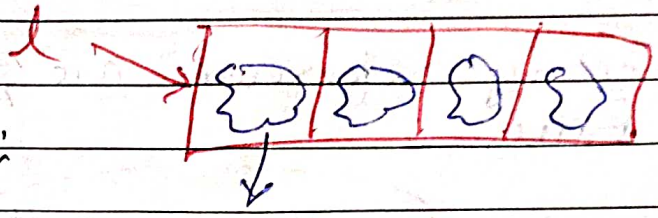
String s = null;
Sop(s == null); True

Sop(null == null); True

instanceof operator

Date

We can use instanceof operator to check whether the given object is of particular type or not



```
Object o = l.get[0];
if (o instanceof Student)
{
    Student s2 = (Student)o;
    // perform Student specific functionality
}
else if (o instanceof Customer)
{
    Customer c = (Customer)o;
    // perform Customer specific
       functionality
}
:
```

Syntax:

r instanceof X

r → object reference

X → class / Interface name

Object ← Thread — Runnable (I)

Example 1:

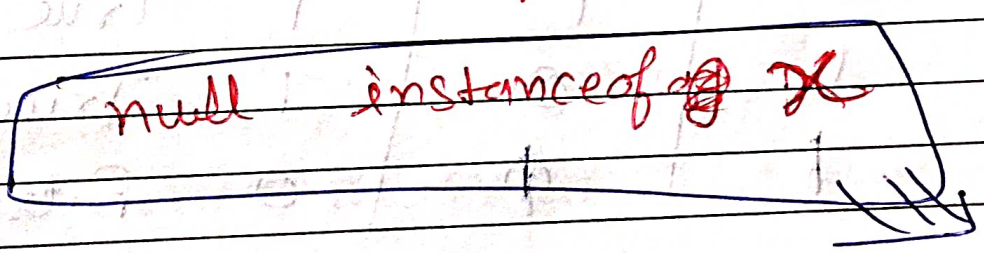
Date _____

```
Thread t = new Thread();
Sopln(t instanceof Thread); true
Sopln(t instanceof Object); true
Sopln(t instanceof Runnable); true
```

Example 2

```
Thread t = new Thread();
Sopln(t instanceof String);
```

C.E inconvertible types
found : j.l.Thread
required : j.l.String




```
Sopln(null instanceof Thread); false
      null instanceof Object); false
      null instanceof Runnable); false
}
```

Bitwise Operator (&, |, ^) Date

& → AND ⇒ Return True if and only if both arguments are True

| → OR ⇒ Return True if and only if atleast one argument is True

^ → X-OR ⇒ Return True if and only if both argument are different

A	B	XOR	
0	0	0	false
0	1	1	True
1	0	1	True
1	1	0	False

eg: Sopln (true & false); false

Sopln (true | false); true

Sopln (true ^ false); true

Sop (4 & 5); 4 → 100, 101 → 100

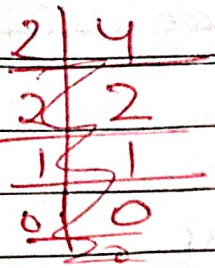
Sopln (4|5); 5 → 100, 101 → 101

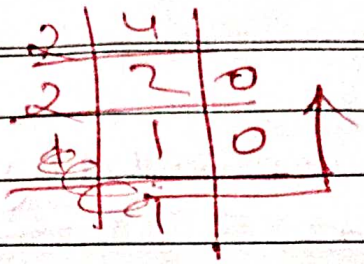
sopln (4^5); 4 → 100, 101 → 001

Spiral

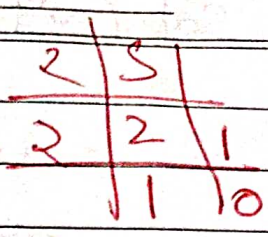
Decimal into Binary

Date

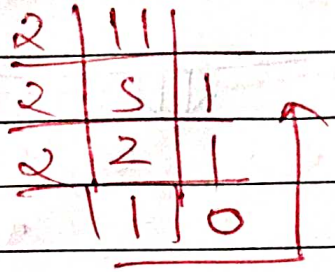




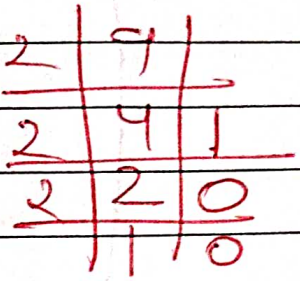
100



S=101



1011



Bitwise Complement Operator (~)

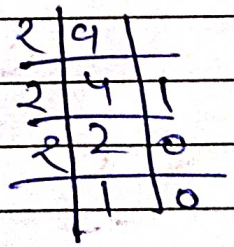
~ operator only for integral not for boolean

Sopln (~true); C.E operator ~ Cannot be applied to boolean

Sopln (~4); output

~ operator is the bitwise NOT operator.

- ~0 result in '-1'
  ~1 result in '-2'
  ~2 result in '-3'
  ~9 result in -10

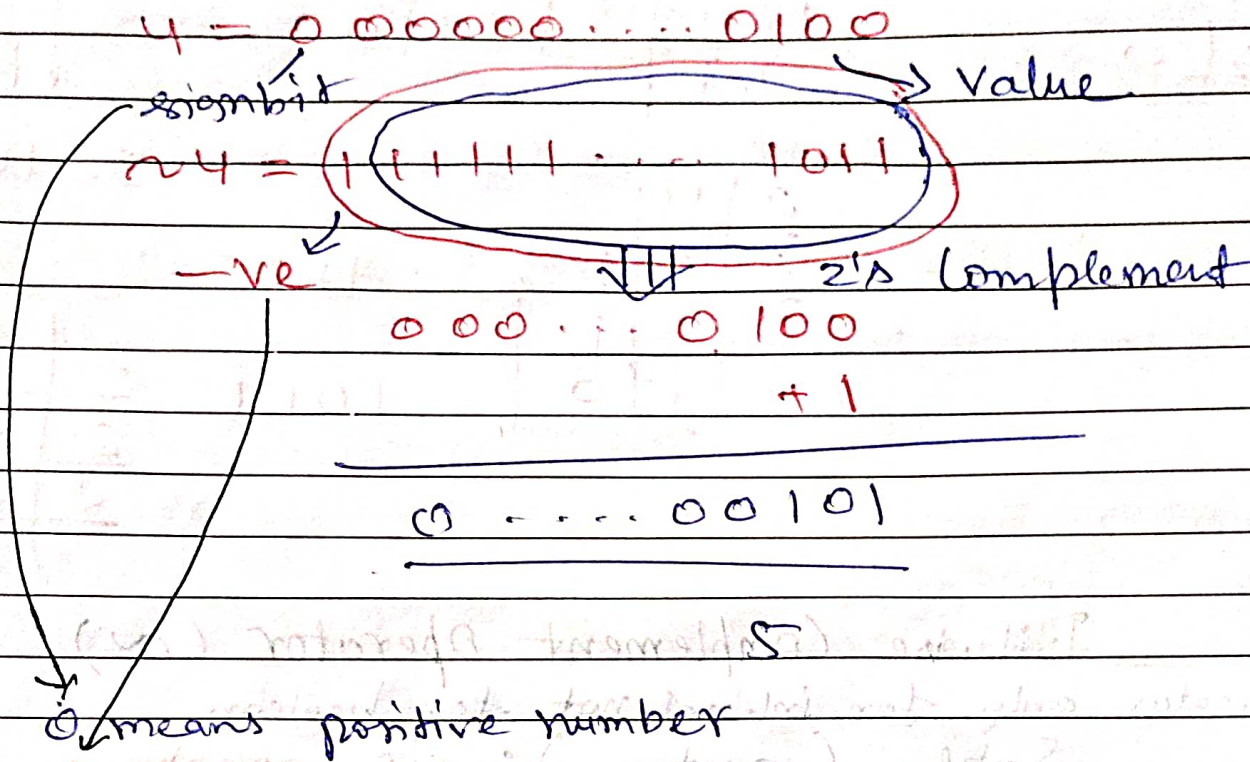


9 = 1001

Not operator = 0110 = 6

Sop(~4); -5 ✓

Date



0 means positive number
1 means negative number

Positive number in the memory directly
Negative numb in the memory indirectly
in 2's complement form

Boolean Complement operator (!)

Sopln (!4);

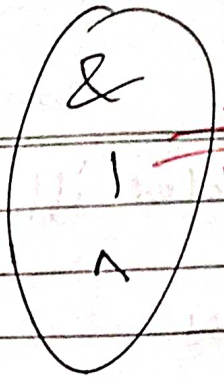
C.E the operator ! is undefined for the argument type(s) int

Sopln(!false); True

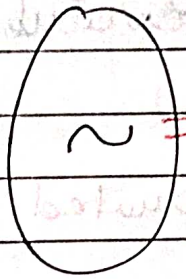
✗ Applicable for boolean types not for int types

Spiral

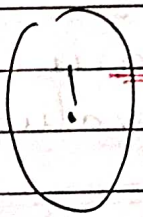
Date



⇒ Applicable for both boolean & integral type



⇒ Applicable for only integeral type but not for boolean type.



⇒ Applicable for only boolean type not for integral

SHORT CIRCUIT OPERATORS (&&, ||)

& |	&& , ||
① Both arguments	① Second option are i's argument
② Relative performance is low	② high
③ Applicable for both boolean & integral type	③ Applicable for boolean type only

Spiral

Note:

Date

x && y ⇒ y will be evaluated iff x is true
i.e if x is false
then y won't be evaluated

x || y ⇒ y will be evaluated iff
x is false
i.e, if x is true then
y won't be evaluated

```
int x=10, y=15;
if (++x < 10 && ++y > 15)
{
    x++;
}
else
{
    y++;
}
sopln(x+"..."+y);
```

	x	y
&&	11	17
&&	11	16
|	12	16
|	12	16

Spiral

Date

```
int x=10;
if ( ++x<10 && (x/0 >10))
  { Sopln("Hello");
  }
else
  { sopln ("Hi");
  }
```

output: Hi

options are :

1. C.E
2. R.E : AE / by zero
3. Hello
4. Hi

replacing :

&& → C.E → second option.

Type Cast operator

- implicit Type-Casting
- Explicit type Casting

```
int x= 'a';
Sopln (x); 97
```

Compiler converts char to int automatically by implicit type. Casting

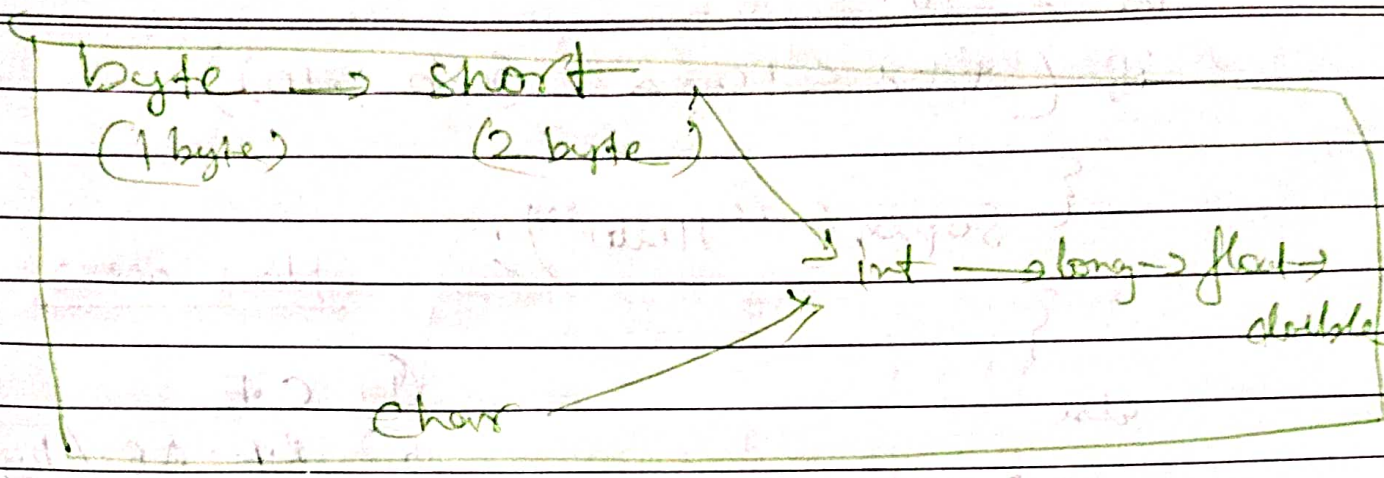
```
double d = 10;
Sopln (d); 10.0 output
```

compiler converts int to double by implicity type casting

1. Compiler (responsible)
2. Smaller ⟶ bigger
3. Widening up casting (Tomorrow cs)
4. No loss of information

Spiral

Date



Explicit Type Casting

```
int x=130;
byte b = x;
Sopln(b);
```

CE possible loss of precision
found: int
required: byte

```
byte b = (byte) x;
```
← Explicit type casting

```
Sopln(b);
```

output (-126) → programmer selected loss of precision

① Who is responsible for using explicit type casting
programmer

② When ETC is performed
Bigger data type → Smaller data type

③ Narrowing or down casting

④ chance of loss of precision

Left

Right

Date

byte ← short ← int ← long ← float ← double

char ←

L → R ⇒ implicit type casting

R → L ⇒ explicit type casting

130 → −126 Explanation

2	130	
2	65	0
2	32	1
2	16	0
2	8	0
2	4	0
2	2	0
2	1	0

int x = 130 ⇒ 00000000...10000010 (32 bits)

byte b = (byte) x; ⇒ 10000010 (8 bits)

M.S.B = −ve

2's complement

1111101

+1

1111110

$2^6 2^5 2^4 2^3 2^2 2^1 2^0$

64 + 32 + 16 + 8 + 4 + 2 + 0

= −126

four rules of binary addition which are

1. 0 + 0 = 0
2. 0 + 1 = 1
3. 1 + 0 = 1
4. 1 + 1 = 10

#

```
int x=150;
short s = (short) x;
Sopln (s);
```

16bit

32bit

int x=150 ⇒

~~0 0 0 ... 0 1 0 0 1 0 1 1 0~~

⇒ [0] 0 0 ... 0 1 0 0 1 0 1 1 0

+ve 150

Date

```
byte b = byte (x);
Sopln (b);
```

byte b=(byte) x;

⇒ [1] 0 0 1 0 1 1 0

-ve

1 1 0 1 0 0

+ 1

(1 1 0 1 0 1 0) x

(2^6 2^5 2^4 2^3 2^2 2^1 2^0)

= 64+32+8+4+2+0

= 106

2	150	
2	75	0
2	37	1
2	18	1
2	9	0
2	4	1
2	2	0
	1	0

```
double d = 130.456;
int x = (int) d;
Sopln (x); 130 output

byte b = (byte) d
Sopln (b); -126
```

if we assign floating point value to the integral type by explicite type casting the digit after the decimal point will be lost

Assignment operators

Date

① Simple:

```
int x=10;
```

② Chained:

```
a=b=c=d=20;
```

③ Compound:

```
a+=20;
```

mixed with some other operator

Chained Assignment operator

```
{ int a,b,c,d;
  a=b=c=d=20;
  sopln(a+" "+b+"....."+c+"....."+d);
```

20 20 20 20

combining two lines into Simple line gives C.E.

```
int a=b=c=d=20;
```

only one variable declared. rest of other are not declared

C.E: Cannot find Symbol
Symbol: Variable b
location: Class Test

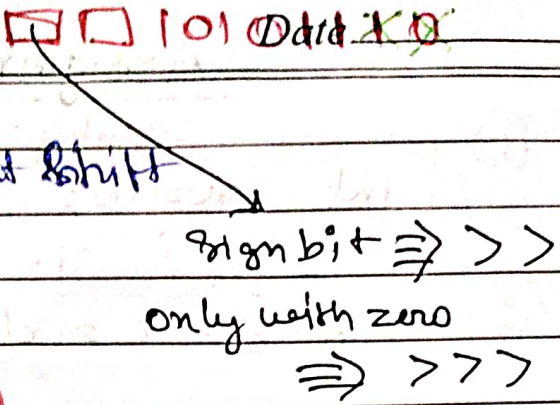
```
int b,c,d;
int a=b=c=d=20; // NO C.E
```

Compound Assignment

```
int a=10;
a+=20; (or) a=a+20;
Sopln(a); 30 output
```

x >> 2

+=	&=	Right shift >>=
-=	|=	>>>= Unsigned right shift
*=	^=	<<=
/=		<<<= (unsigned)
%=		



∴ Assignment operator mixed with another operator

```
byte b=10;
b=b+1;
Sopln(b);
```

C.E
(max (int, byte, int))
P.L.P
found int
require byte

```
byte b=10;
b++;
Sopln(b);
```

output=11

b=byte(b+1);

```
byte b=10;
b+=1;
Sopln(b);
```

output = 11

b=byte(b+1);

```
byte b=127;
b+=3;
Sopln(b);
```

output :- -126

* In the case of compound assignment operators internal type casting will be performed automatically

```
int a,b,c,d;
a=b=c=d=20;
a+=b-=c*=d/=2;
Sopln("a+" "+b+" "+c+" "+d);
```

Right to left

a = 160, b = 180, c = 200, d = 10

Conditional operator (? :)

Syntax:

```
int x = (10<20) ? 30:40;
Sopln(x);
```

True → 30

False → 40

output 30

a++; ++a; Unary

a+b ⇒ Binary

(c)? b:d ⇒ Ternary

```
int x = (10>20) ? 30: ((40>50)? 60:70);
Sopln(x);
```

(10>20) False

(40>50) False

Output 70

Date

new operator :

```
Test t = new Test();
```

1. (Constructor) not to Create object
   It performs initialisation of object
   as it's executed after object creation
   (Destruction of Useless object)
2. Delete object is responsibity of
   garbage collector not programmer
   Hence "delete" key word is not
   in Java

3. new keyword is used to create
   object

[ ] operator

We can use this operator to create anddeclare
Arrays

```
int [ ] x = new int [10];
```

Date

Java operator Precedance

1. Unary operators:
   [ ], x++, x--
   ++x, --x, ~, !
   new <type>
2. Arithmetic operators:
   *, /, %
   +, -
3. Shift operators:
   >>, >>>, <<
4. comparison operator
   <, <=, >, >=, instanceof
5. Equality operator
   ==, !=
6. Bitwise operator
   &
   ^
   |
7. Short Circuit operator
   &&
   ||
8. Conditional operator
   ? :
9. Assignment operator
   =, +=, -=, *=

Date

Java operator Precedance

1. Unary operators:
   [ ], x++, x--
   ++x, --x, ~, !
   new <type>
2. Arithmetic operators:
   *, /, %
   +, -
3. Shift operators:
   >>, >>>, <<
4. comparison operator
   <, <=, >, >=, instanceof
5. Equality operator
   ==, !=
6. Bitwise operator
   &
   ^
   |
7. Short Circuit operator
   &&
   ||
8. Conditional operator
   ? :
9. Assignment operator
   =, +=, -=, *=

Date

Evaluation Order of Java operands

1. In java we have only operator precedence but not operand precedance before applying any operator all operands will be evaluated from left to right

```
class Test
{
    psvm (String[] args)
    {
        sop(m1(1) + m1(2) * m1(3) / m1(4) + m1(5) * m1(6));
    }
    public static int m1(int i)
    {
        sop(i);
        return(i);
    }
}
```

→ left to right

o/p
1
2
3
4
5
6
32

1 + 2 * 3 / 4 + 5 * 6

1 + 6 / 4 + 5 * 6

1 + 1 + 5 * 6

1 + 1 + 30

2 + 30

32

As per operator precedance

Date

① New vs newInstance()

② instanceof vs isInstance()

③ ClassNotFoundException vs NoClassDefFoundError

New vs newInstance()

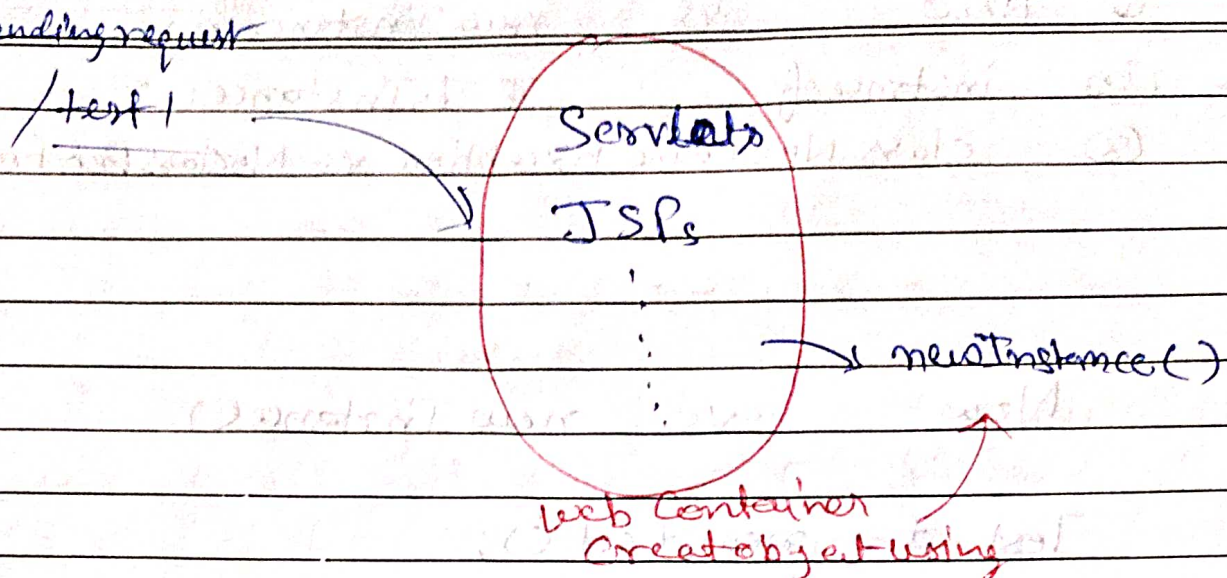
```
Test t = new Test();
Student s = new Student();
Customer c = new Customer();
class Student { }
class Customers { }
class Test
{
    P S V main(String[] args)
    {
        Object o = Class.forName(args[0]).newInstance();
                   → class.Class object
        System.out.println("Object created for " + o.getClass().getName());
    }
}
```

Java Test Student ↵
o/p: Object created for Student
Java Test Customer ↵
o/p: Object created for Customer
Java Test j.l.String ↵
o/p: object created for java.lang.String

- we can new operator to create an object if we know class name at the beginning.
- newInstance() is a method present in class Class we can use newInstance method to create object if we don't know class name at the beginning and it is available dynamically at runtime

Spiral

Date



In the case of new operator based on our requirement we can invoke any constructor

eg. `Test t = new Test();`
`Test t1 = new Test(10);`
`Test t2 = new Test("durga");`

But new instance method internally called no-arg-Constructor, Hence to use newInstance() Compulsory Corresponding class should contain no-arg Constructor,

Otherwise we will get R.E saying InstantiationException

#

While using new operator at syn time if Corresponding .Class file not found during Runtime then we will get RuntimeException

no class def found error : Test

#

```
Package datatypes;
public class NoClassDefFoundExample{
    Public static void main (String[] args)
    {
        NonExistenceClass obj = new NonExistentclass();
    }
class NonExistenceclasses
    {
    }
```

RE: (NoClassDefFoundError Example)

While using newInstance() at Runtime if the Corresponding .class file not available then we will get R.E classNotFound Example

```
Object O=Class.forName(args[0]).newInstance();
```

Java Test Test123 ↵

at Runtime if Test123.class Java file is not available saying

R.E ClassNotFound : Test123

new	newInstance()
① it is operator in Java	It is a method present in java.lang.Class
② We Can use new operator to Create object if we know class name at the begining	we can use this method to create object if we don't know Class name at the begining and it is available dynamically at run time.
③ To use new operator class not required to Contain no-arg Constructor	To use newInstance() Compulsory class should Contain no-arg Constructor Other wise we will get R.E InstantiationException
④ At run time if class file not available then we R.E NoClassDefFoundError	R.E ClassNotFoundException

Date

ClassNotFoundException vs NoClassDefFoundError

For Hard Coded class names, at runtime if the corresponding .class file is not available then we will get RuntimeException: NoClassDefFoundError, which is unchecked

```
Test t = new Test();
```

at Runtime if Test.class file is not available then we will get R.E NoClassDefFoundError : Test

For Dynamically provided class at runtime if the corresponding .class file is not available then we will get R.E saying ClassNotFoundException, which is checked exception

```
Object O = Class.forName(args[0]).newInstance();
```

Java Test Student

At Runtime if Student.class file not available then we will get R.Time Exception saying ClassNotFoundException: Student

instanceof vs isInstance()

①
```
Thread t = new Thread();
Sopln(t instanceof Runnable);
Sopln(t instanceof Object);
```
we know (type of) the beginning. then use instanceof

②
```
class Test
{
    public static void main (String [] args) throws Exception
    {
        Thread t = new Thread();
        Sopln(Class.forName(args[0]).isInstance(t));
    }
}
```

Java Test Runnable ↵
o/p: True

Java Test String ↵
o/p: false

we don't know the (type of) beginning then use isInstance

instanceof is an operator in java, we can use instanceof to check, whether the given object is of particular type or not and we know the type of the beginning eg: ①

isInstance is a method present in java.lang.Class, we can use isInstance method to check whether the given object is of particular type or not and we don't the type of the beginning and it is available dynamically at Runtime. eg ②

instanceof vs isInstance()

①
```
Thread t = new Thread();
Sop(t instanceof Runnable);
Sop(t instanceof Object);
```
we know (type of) the beginning then use **Instance of**

②
```
class Test
{
    public static void main (String [] args) throws Exception
    {
        Thread t = new Thread();
        Sop(Class.forName(args[0]).isInstance(t));
    }
}
```

Java Test Runnable ↵
o/p: True

Java Test String ↵
o/p: false

we don't know the (type of) beginning then use **isInstance**

instanceof is an operator in java, we can use instanceof to check, whether the given object is of particular type or not and we know the type of the beginning eg. ①

isInstance is a method present in java.lang.class, we can use isInstance method to check whether the given object is of particular type or not and we don't the type of the beginning and it is available dynamically at Runtime. eg ②