

# Object Oriented Programming JAVA

Dr. Prafulla Kalapatapu
Computer Science Engineering
Mahindra Ecole Centrale
prafulla.kalapatapu@mechyd.ac.in



### LANGUAGE FUNDAMENTALS

# Language Fundamentals



- 1. Identifiers
- Reserved words
- 3. Data Types
- 4. Literals
- 5. Arrays
- 6. Type of Variables



# **Operators**

# **Arithmetic Operators**



- The Arithmetic operators are (+,-,\*,/,%)
- If we are applying any arithmetic operator between two variables a and b, the result type is always max(int, type of a, type of b).

### Ex:

```
byte + byte = int
byte + short = int
int + long = long
long + float = float
double + char = double
char + char = int
```

```
System.out.println(10+0.0);
o/p: 10.0
System.out.println(100+'a');
o/p: 197
```

# **Infinity**



### Case (i):

 In case of integral division (int, short, long, byte), there is no way to represent infinity. Hence if the infinity is the result, we will get ArithmeticException: divide by zero.

Ex: System.out.println(10/0);

R.E: ArithmeticException: divide by zero.

### Case (ii):

 In case of floating point division (float, double) there is always a way to represent infinity for this float and double classes contains the following two constants.

```
POSITIVE_INFINITY = Infinity
NEGATIVE_INFINITY = - Infinity
```

 Hence, in the case of floating point division, we wont get any ArithmeticException.

```
System.out.println(10/0.0); Infinity
```

System.out.println(-10/0.0); - Infinity

# NaN (Not a Number)



### Case (i):

 In integral division, there is no way to represent undefined results. Hence, if the result is undefined we will get ArithmeticException.

```
Ex: System.out.println(0/0);
RE: AE: Divide by zero.
```

### Case (ii):

- In case of floating point division there is always a way to represent undefined results for this float and double classes contains NaN constants.
- Hence, even though the result is undefined we wont get any RuntimeException.

```
Ex: System.out.println(0/0.0);

NaN

System.out.println(0.0/0);

NaN

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

LEAD NATREPRENEUR INNOVATOR
```



• Note: for any x value, including NaN the relational operations always returns false except for (!=) Expression, this returns true.

```
x >= NaN
         x < NaN
                       false
         x \le NaN
         x == NaN
Fx. \Delta t x = 10
System.out.println(10 > Float.NaN);
                                                false
System.out.println(10 < Float.NaN);
                                                false
System.out.println(10 == Float.NaN);
                                                false
System.out.println(10 != Float.NaN);
                                                true
                                                false
System.out.println(Float.NaN == Float.NaN);
System.out.println(Float.NaN != Float.NaN);
                                                 true
 LEADER 

ENTREPRENEUR 

INNOVATOR
```

x!= NaN returns true

x > NaN

### **String Concatenation Operator**



- 1. Addition operator.
  2. Concatenation operator.
  operand1 + operand2
- If either of the operands (both operands) are string type, '+' acts as concatenation operator otherwise '+' acts as addition operator.

```
System.out.println(10+20+"30");
30+"30" = 3030
System.out.println(10+'A'+"X");
197X
```

### Instanceof operator



 By using this operator. We can check whether the given object is of a particular type (or) not.

### Syntax:

```
r instanceof x

Reference type class name/ interface keyword
```

System.out.println(e instanceof Student); CE: Inconvertible type

Found: Employee Required: Student

System.out.println(null instanceof Employee); false

# **Bitwise Operator**



- 1. & -> AND -> if both operands are true, result is true.
- 2. | -> OR -> if atleast one operand is true, result is true.
- 3. ^ -> XOR -> if both operands are different, result is true.

### Ex:

System.out.println(T & T) true
System.out.println(T | F) true

System.out.println(F ^ F) false

We can apply these operators even for integral data types also.

### Ex:

System.out.println(4 & 5)

System.out.println(4 | 5)

System.out.println(4 ^ 5)

# Bitwise Compliment Operator (~)



 We can apply bitwise complement operator only for integral types but not for boolean type.

```
Ex (i):
System.out.println(~true); CE: operator ~ can't be applied to boolean
Ex (ii):
System.out.println(~4); -5
```

Note: +ve numbers will be represented directly in the memory. Where as -ve numbers will be represented in 2's complement form.

# **Ternary Operators (?:)**

LEADER 

ENTREPRENEUR 

INNOVATOR



Syntax: variable = expression1 ? expression2 : expression3 ; It means if( expression1 is true) variable = expression2; else variable = expression3; Ex: max = (a>b) ? a : b ;It means if(a>b) max = a;else max=b;

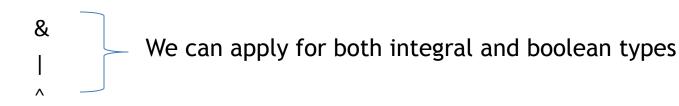
# Boolean Compliment Operator (!) Mahindra École Centrale COLLEGE OF ENGINEERING

We can apply these operator only for boolean type but not for integral type

```
Ex:
    System.out.println(!4);    CE : operator ! can't be applied to int
    System.out.println(!true);    false
    System.out.println(!false);    true
```

# **Summary**





~ -> we can apply only for integral types but not for boolean types

! -> we can apply only for boolean types but not for integral types

# Short circuit operators (&&, ||)

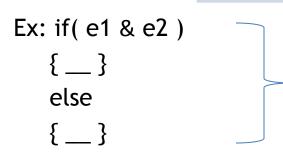


We can use these operators just to improve performance of the system.

These are exactly same as normal bitwise operators, but with few following

differences

| &,   | &&,  |
|--|--|
| Both operands should be evaluated always       | 2 <sup>nd</sup> operand evaluation is optional |
| Relatively low performance                     | Relatively High performance                    |
| Applicable for both boolean and integral types | Applicable only for boolean types              |



Suppose e1 takes 10 sec, e2 takes 10 sec, & takes 1 sec Then total it takes 21 sec to evaluate condition

- (1) x && y -> y will be evaluated iff x is true
- (2)  $x \mid \mid y \rightarrow y$  will be evaluated iff x is false

LEADER = ENTREPRENEUR = INNOVATOR

# **Type Casting**



- There are two types of primitive type castings possible
  - 1. Implicit type casting
  - 2. Explicit type casting

| Implicit type casting  | Explicit type casting   |  |
|--|---|--|
| Compiler is responsible for this type casting  | Programmer is responsible for this type casting   |  |
| Performed automatically when ever we are assigning smaller value to the bigger data type | It requires when ever we are trying to assign bigger data type value to the smaller data type |  |
| It is also known as widening or up casting   | It is also known as narrowing or down casting   |  |
| There is no loss of information in this type casting                                     | There may be a chance of loss of information in this type casting                             |  |

LEADER ENTE



The following is the list of all possible automatic promotions.

Compiler automatically promotes 10 to 10.0 in ex(i).

Compiler automatically promotes char to int type in ex(ii)

LEADER 

ENTREPRENEUR 

INNOVATOR



The following conversions requires explicit type casting.

```
byte <- short <- int <- long <- float <- double char
```

```
Ex (i):
```

byte b=130;

CE: possible loss of precision

found: int

required: byte

### Ex (ii):

byte b=(byte)130;

System.out.println(b);

o/p:-126



 When ever we are assigning bigger data type values to the smaller datatype by explicit type casting the most significant bits will be lost.

```
Ex: int a=150;
byte b=(byte)a;
short s=(short)a;
System.out.println(b);
System.out.println(s);
O/P: -106
150
```

 Whenever we are trying to assign floating point values to the integral type by explicit typecasting the digits after the decimal point will be lost

```
Ex: double d=10.235; byte b=(byte)130.625 int a=(int)d; System.out.println(b); System.out.println(a); O/p: 10 o/p: -126
```

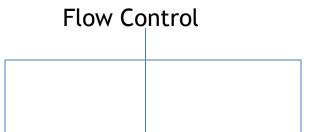


# Flow Control

### Flow Control



It describes the order in which statements will be executed at runtime.



### **Selection Statements**

- if-else
- Switch

- while
- do-while
- For
- For each loop

#### Iterative Statements Transfer Statements

- break
- continue
- return

### **Selection Statement**



 The argument to the if statement should be boolean type, if we are providing anyother type, we will get compile time error.

# **Examples**



a.
int x=0;
if(x)
{ ---- }
else
{ ----}

b.
int x=10;
if(x =20)
{ ---- }
else
{ ----}

CE: Incompatible types Found : int Required : boolean

c. int x=10; if(x ==20) { ---- } else { ----}

d.
boolean b= false;
if(b)
{ ---- }
else
{ ----}

boolean b=false;
if(b= =true)
{ ---- }
else
{ ----}

No Error



- Curly braces ( {,}) are optional.
- Without curly braces, we can take only one statement. Which should not be a declarative statement.

### Ex:

```
a.
if(true)
if(true);

System.out.println("hi");

C.
if(true)
if(true)
if(true)
if(true)
compile time Error
int x=10;
int x=10;
}
```



### 2. Switch Statement:

• If several options are possible then it is never recommended to use if-else, we should go for switch statement.

Curly braces are mandatory here.



Both case and default are optional inside a switch.

```
Ex: int x=10;
switch(x)
{
}
```

With in the switch, every statement should be under some case or default.
 Independent statements are not allowed.

```
Ex: int x=10;
    switch(x)
    {
        System.out.println("hi");
     }
     CE: case,default or '}' expected
```



 Until 1.4V the allowed data types for switch arguments are byte

short

int

char

 But from 1.5V onwards in addition to these, the corresponding wrapper classes are allowed.

| 1.4V  | 1.5V      | 1.7V   |
|-------|-----------|--------|
| byte  | +         | +      |
| short | Byte      | String |
| int   | Short     |        |
| char  | Character |        |
|       | Integer   |        |

• Apart from these, if we are passing any other type we will get compile LEADER-ENGINEER INNOVATOR

### **Examples**



```
a.
byte b=10;
switch(b)
{
}
```

```
b.
long l=10l;
switch(l)
{
}
```

CE: Possible loss of precision Found : long Required : int

```
c.
char ch='a';
switch(ch)
{
}
```

```
d.
boolean b=true;
switch(b)
{
}
```

CE: incompatible types
Found: boolean
Required: int



 Every case label should be with in the range of switch argument type, otherwise we will get compile time error.

Ex:

```
a.
b.
byte b=10;
switch(b)
{
    case 10 : System.out.println("10");
    case 100 : System.out.println("100");
    case 1000 : System.out.println("1000");
    case 1000 : System.out.println("1000");
}
case 1000 : System.out.println("1000");
case 1000 : System.out.println("1000");
}
```

CE: Possible loss of precision

Found: int

Required: byte

LEADER 

ENTREPRENEUR 

INNOVATOR



 Every case label should be a valid compile time constant, if we are taking variable as case label, we will get compile time error.

Ex:

```
a.
int x=10;
int y=20;
switch(b)
{
  case 10 : System.out.println("10");
  case y : System.out.println("20");
}

b.
int x=10;
final int y=20;
switch(b)
{
  case 10 : System.out.println("10");
  case y : System.out.println("20");
}
```

### CE: constant expression required

If we declare y as final then we wont get any compile time error.



 Expressions are allowed for both switch argument and case label but case label should be constant expression.

```
Ex: a.
    int x=10;
    switch(x+1)
    {
       case 10 : System.out.println("10");
       case 10+20 : System.out.println("10+20");
    }
```

Duplicate case labels are not allowed.

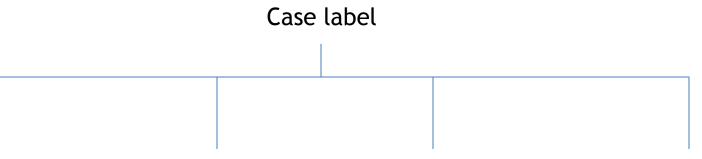
```
Ex: b.
int x=10;
switch(x)

CE: duplicate case label

case 97: System.out.println("97");
case 98: System.out.println("98");
case 'a': System.out.println("a");
```

# Summary – case labels





- 1. It should be compile time constant
- constant expressions switch argument type
- Expressions also 3. Value should be 4. Duplicates allowed but should be with the range of are not allowed



### Fall- through inside switch:

 With in the switch statement, if any case is matched from that case onwards all statements will be executed until break statement or end of the switch

• Fall-through inside switch is useful to define some common action for several LECASES ENTREPRENEUR - INNOVATOR