Keywords, Data types , Variables and Operators

Keywords

- **Reserved Words** that have a **predefined meaning** in the java language.
- Programmers cannot use keywords as names for variables, methods, classes, or as any other identifier

Java Keywords

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

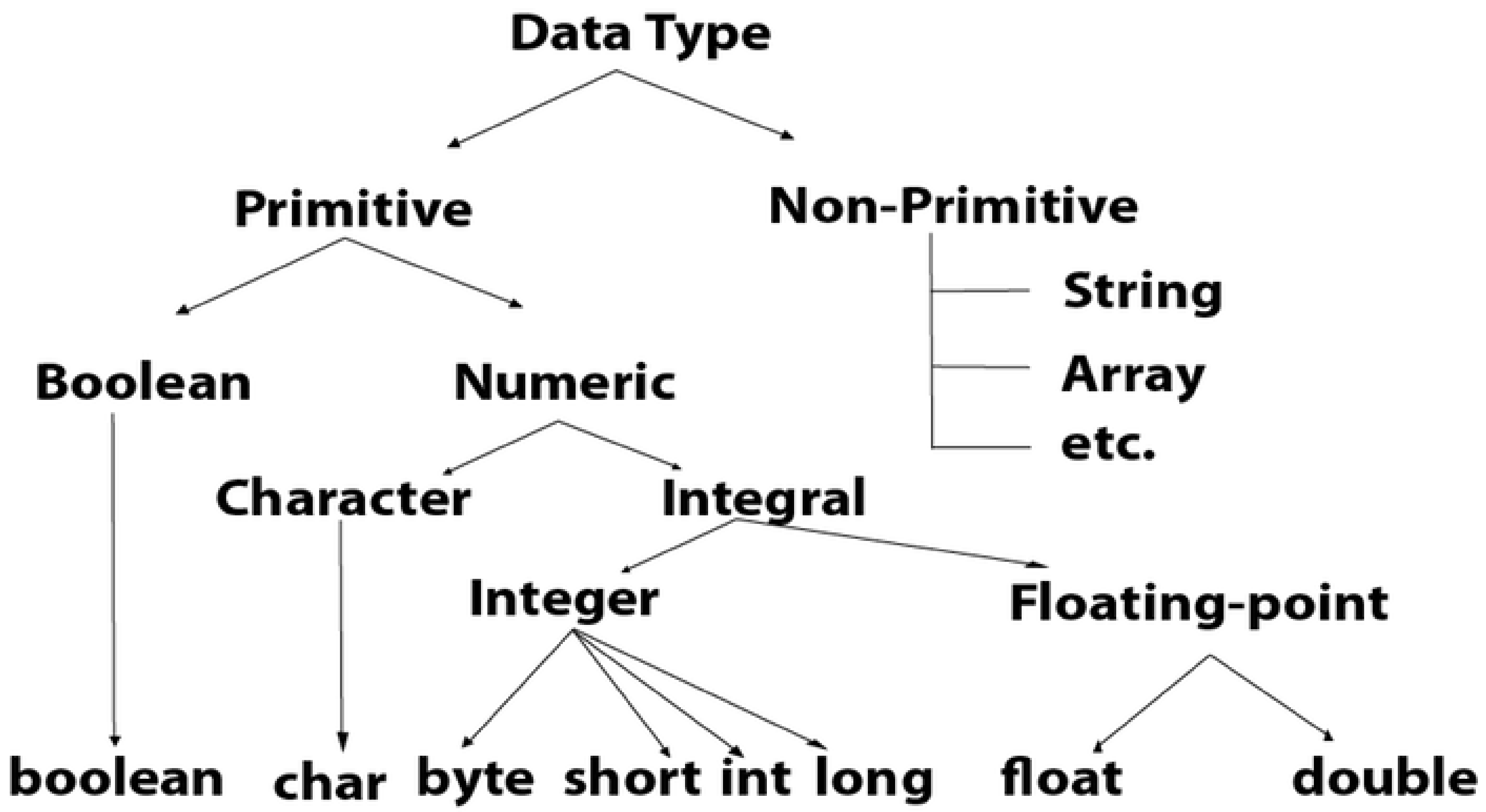
Quiz

What will be the result, if we try to compile and execute the following code?

```
class Test {
    public static void main(String [ ] ar) {
        int for=2;
        System.out.println(for);
    }
}
```

Data types

(Defines the type of data)



Types

- **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.

- **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.

Primitive Data Types

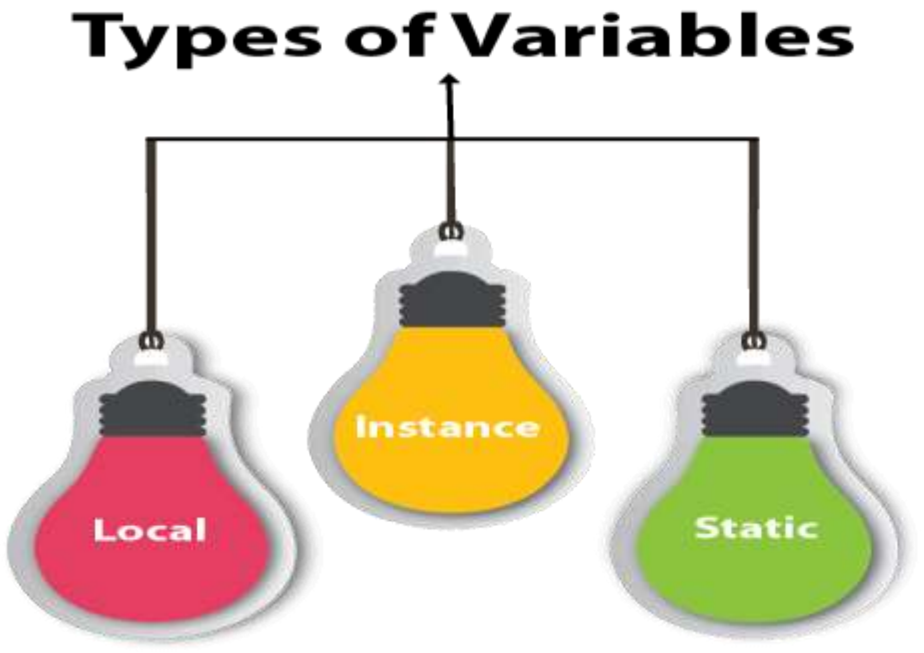
Data Type	Size (in bits)	Minimum Range	Maximum Range	Default Value (for fields)
byte	8	-128	+127	0
short	16	-32768	+32767	0
int	32	-2147483648	+2147483647	0
long	64	-9223372036854775808	+9223372036854775807	0L
float	32	1.40E-45	3.40282346638528860e+38	0.0f
double	64	4.94065645841246544e-324d	1.79769313486231570e+308d	0.0d
char	16		0 to 65,535	'\u0000'
boolean	1	NA	NA	false

Identifiers

- Identifiers are the names which are used to represent any variable, class, reference, package, method.
- No spaces are allowed.
- All alphabets both uppercase and lowercase are allowed.
- No special characters are allowed except $(dollar) and _(underscore).
- Digits are allowed but the name cannot start with a digit.
- The name should not be a reserved keyword.

Variables

- A variable is a container which holds the value while the Java program is executed.
- A variable is assigned with a data type.
- Variable is a name of memory location



Types of Variables

- **Local Variables**
  - A variable declared inside the body of the method is called local variable.
  - Scope of a local variable is within the method
  - A local variable cannot be defined with "static" keyword.
  - You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.
- **Instance Variables (Non-static)**
  - A variable declared inside the class but outside the body of the method, is called instance variable.
  - It is not declared as static.
  - Scope of an instance variable is the whole class.

- **Static Variables**
  - A variable which is declared as static is called static variable.
  - It cannot be local.
  - You can create a single copy of static variable and share among all the instances of the class.

- Example Program:

Operators

- Java provides a set of operators to manipulate operations.
- Types of operators in java are,
  - Arithmetic Operators
  - Unary Operator
  - Relational Operators
  - Logical Operators
  - Simple Assignment Operator
  - Ternary Operator
  - Bitwise Operators

Arithmetic Operators

- Java arithmetic operators are used to perform basic arithmetic operations like addition, subtraction, multiplication, and division.
- They act as basic mathematical operations.

Operator	Description	Example
+	Addition	A + B
-	Subtraction	A - B
*	Multiplication	A * B
/	Division	A/B
%	Modulus	A%B

Arithmetic Operators - Example

```
/* Example to understand  Arithmetic operator */

class Sample{
  public static void main(String[ ] args){
    int a = 10;
    int b = 3;
    System.out.println("a + b = " + (a + b) );
    System.out.println("a - b = " + (a - b) );
    System.out.println("a * b = " + (a * b) );
    System.out.println("a / b = " + (a / b) );
    System.out.println("a % b = " + (a % b) );
  }
}
```

<u>Output:</u>

```
a + b = 13
a - b = 7
a * b = 30
a / b = 3
a % b = 1
```

Unary Operators

- The Java unary operators require only one operand.

Unary operators are used to perform various operations i.e.:

- incrementing/decrementing a value by one

Operator	Description	Example
+	Unary plus operator	+A
-	Unary minus operator	-A
++	Increment operator	++A or A++
--	Decrement operator	--A or A--

Unary Operator - Example

```
/* Example for Unary Operators*/

class Sample{
   public static void main(String args[]) {
   int a = 10;
   int b = 20;
   System.out.println("++a   = " +  (++a) );
   System.out.println("--b   = " +  (--b) );
   }
}
```

Output:

```
a++   = 11
b--   = 19
```

Relational Operators
(Used to relate 2 values or expressions)

Operator	Description	Example
==	Two values are checked, and if equal, then the condition becomes true	(A == B)
!=	Two values are checked to determine whether they are equal or not, and if not equal, then the condition becomes true	(A != B)
>	Two values are checked and if the value on the left is greater than the value on the right, then the condition becomes true.	(A > B)
<	Two values are checked and if the value on the left is less than the value on the right, then the condition becomes true	(A < B)
>=	Two values are checked and if the value on the left is greater than equal to the value on the right, then the condition becomes true	(A >= B)
<=	Two values are checked and if the value on the left is less than equal to the value on the right, then the condition	(A <= B)

Comparing Strings

- For comparing Strings instead of using == operator, use equals method

String s1=“hello”;

String s2=“Wipro”;

System.out.println(s1.equals(s2)); ➔false

Relational Operators - Example

```
/* Example to understand  Relational operator */

  class Sample{
   public static void main(String[] args){
         int a = 10;
         int b = 20;
         System.out.println("a == b = " + (a == b) );
         System.out.println("a != b = " + (a != b) );
         System.out.println("a > b = " + (a > b) );
         System.out.println("a < b = " + (a < b) );
         System.out.println("b >= a = " + (b >= a) );
         System.out.println("b <= a = " + (b <= a) );
   }
  }
```

Output:

```
a == b = false
a != b = true
a > b = false
a < b = true
b >= a = true
b <= a = false
```

Logical Operators

- Used to compare 2 Expressions.

Operator	Description	Example
&&	• This is known as Logical AND & it combines two variables or expressions and • if and only if both the operands are true, then it will return true	(A && B) is false
||	• This is known as Logical OR & it combines two variables or expressions and • if either one is true or both the operands are true, then it will return true	(A || B) is true
!	• Called Logical NOT Operator. • It reverses the value of a Boolean expression	!(A && B) is true

Logical Operators - Example

```
/* Example to understand  logical operator */

class Sample{
   public static void main(String[] args){
   boolean a = true;
   boolean b = false;
   System.out.println("a && b = " + (a&&b) );
   System.out.println("a || b = " + (a||b) );
   System.out.println("!(a && b) = " + !(a && b) );
   }
}
```

Output:

```
a && b = false
a || b = true
!(a && b) = true
```

Simple Assignment Operator

= Simple assignment operator

Which assigns right hand side value to left hand side variable

Ex:

```
int a;
a = 10;
```

Bitwise Operators

The bitwise operators take two bit numbers, use OR/AND to determine the result on a bit by bit basis.

The 3 bitwise operators are :

- & (which is the bitwise AND)
- | (which is the bitwise inclusive OR)
- ^ (which is the bitwise exclusive OR)

Bitwise & (AND) operator

The & operator compares corresponding bits between two numbers and if both the bits are 1, only then the resultant bit is 1. If either one of the bits is 0, then the resultant bit is 0.

Example :

```
    int x = 7;
    int y = 9;
```

What will be x & y ?

```
    7 - >  0 1 1 1
    9 - >  1 0 0 1
    ----------------
           0 0 0 1
    ----------------
```

x & y results in 1.

Bitwise & Operator Demo

```
class BitwiseExample1 {
  public static void main(String[] args) {
    int x = 7;
    int y = 9;
    int z = x & y;
    System.out.println("z = "+z);
  }
}
```

Output :
z = 1

Bitwise | (inclusive OR) operator

The | operator will set the resulting bit to 1 if *either* one of them is 1. It will return 0 only if both the bits are 0.

Example :

```
    int x = 5;
    int y = 9;
```

What will be x | y ?

```
    5 - >  0 1 0 1
    9 - >  1 0 0 1
    -----------------
           1 1 0 1
    -----------------
```

x | y results in 13.

Bitwise | Operator Demo

```
class BitwiseExample2 {
  public static void main(String[] args) {
    int x = 5;
    int y = 9;
    int z = x | y;
    System.out.println("z = "+z);
  }
}
```

Output :
z = 13

Shift Operators << and >>

The shift operators(<< and >>) shift the bits of a number to the left or right, resulting in a new number.

They are used only on integral numbers(and not on floating point numbers, i.e. decimals).

The right shift operator(>>) is used to divide a number in the multiples of 2, while the left shift operator(<<) is used to multiply a number in the multiples of 2.

Ternary Operator

? :

Any expression that evaluates to a boolean value.

boolean_expression **?** expression_1 **:** expression_2

If **true** this expression is evaluated and becomes the value entire expression.

If **false** this expression is evaluated and becomes the value entire expression.

Ternary (? :) Operator Examples

```java
public class Example {
  public static void main(String[] args) {
      boolean t = true;
      boolean f = false;

      System.out.println("t?true:false "+(t ? true : false ));
      System.out.println("t?1:2 "+(t ? 1 : 2 ));
      System.out.println("f?true:false "+(f ? true : false ));
      System.out.println("f?1:2 "+(f ? 1 : 2 ));
  }
}
```

```
> java Example
t?true:false true
t?1:2 1
f?true:false false
f?1:2 2
>
```

Right Shift Operator >>

Let us understand the use of right shift operator with the following example :

```
int x = 16;
x = x >> 3;
```

When we apply the right shift operator >>, the value gets divided by 2 to the power of number specified after the operator. In this case, we have 3 as the value after the right shift operator. So, 16 will be divided by the value 2 to the power of 3, which is 8.

The result is 2.

Right Shift Operator >> (Contd.).

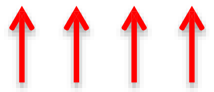
When we represent 16 in binary form, we will get the following binary value :

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

When we apply >> which is the right shift operator, the bit represented by 1 moves by 3 positions to the right(represented by the number after the right shift operator).

After shifting the binary digit 1, we will get :

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0



x = 2

Right Shift Operator >> Demo

```
class ShiftExample1 {
  public static void main(String[] args) {
    int x = 16;
    System.out.println("The original value of x is "+x);
    x = x >> 3;
    System.out.println("After using >> 3, the new value is
  "+x);
  }
}
```

```
The original value of x is 16
After using >> 3, the new value is 2
```

Left Shift Operator <<

Let us understand the use of left shift operator with the following example :

```
int x = 8;
x = x << 4;
```

When we apply the left shift operator <<, the value gets multiplied by 2 to the power of number specified after the operator. In this case, we have 4 as the value after the left shift operator. So, 8 will be multiplied by the value 2 to the power of 4, which is 16.

The result is 128.

Left Shift Operator << (Contd.).

When we represent 8 in binary form, we will get the following binary value :

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

When we apply << which is the left shift operator, the bit represented by 1 moves by 4 positions to the left (represented by the number after the right shift operator).

After shifting the binary digit 1, we will get :

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

X=128

Left Shift Operator << Demo

```
class ShiftExample2
{
   public static void main(String[] A)     {
    int x =8;
  System.out.println("The original value of x is "+x);
   x = x << 4;
   System.out.println("After using << 4, the new value is  "+x);
 }
}
```

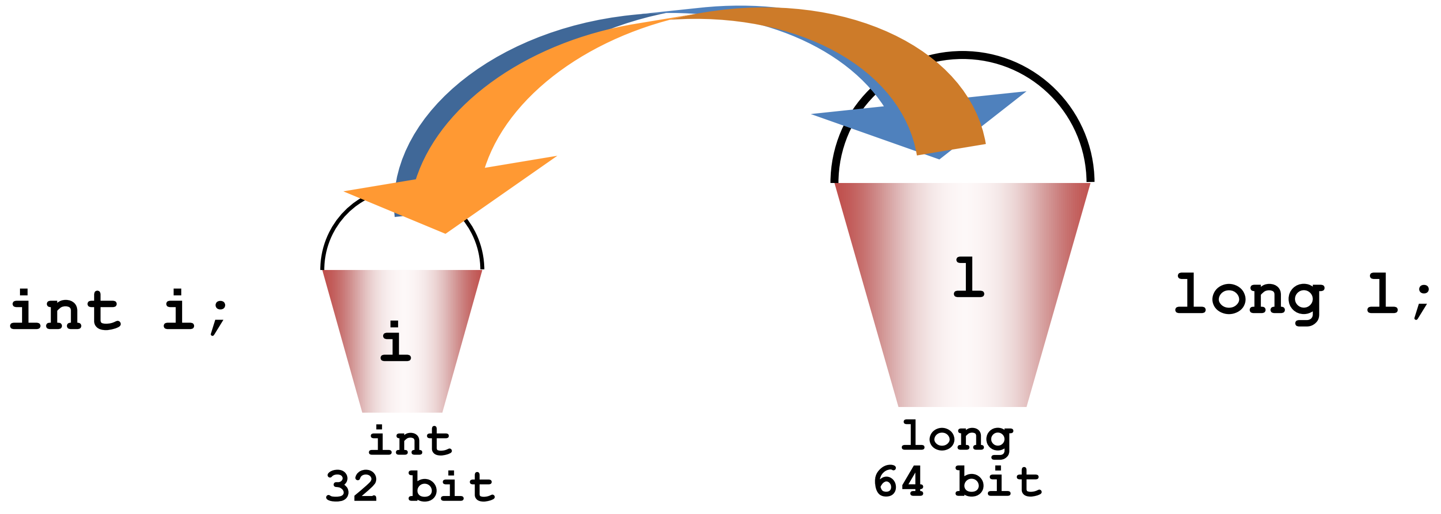
```
The original value of x is 8
After using << 4, the new value is 128
```

Typecasting or Type conversion

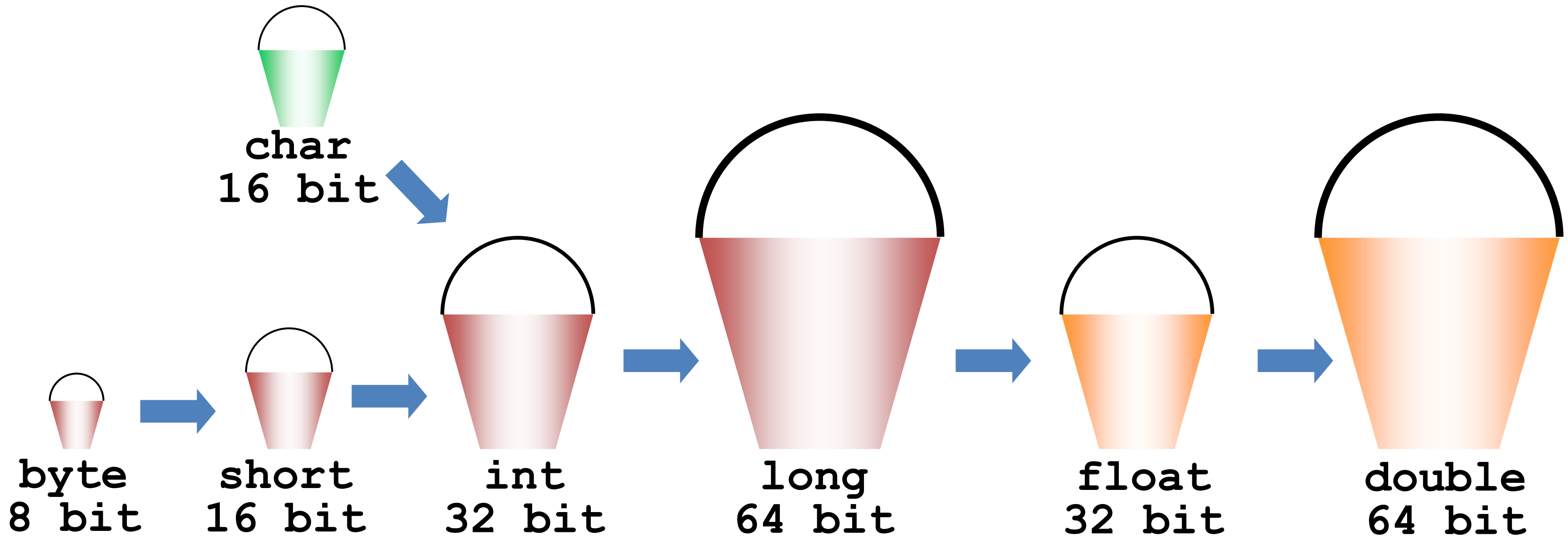
Moving Between Buckets "Casting"

```
i = (int)l;
```
Narrowing

```
l = i;
```
Widening



Acceptable Implicit Casts



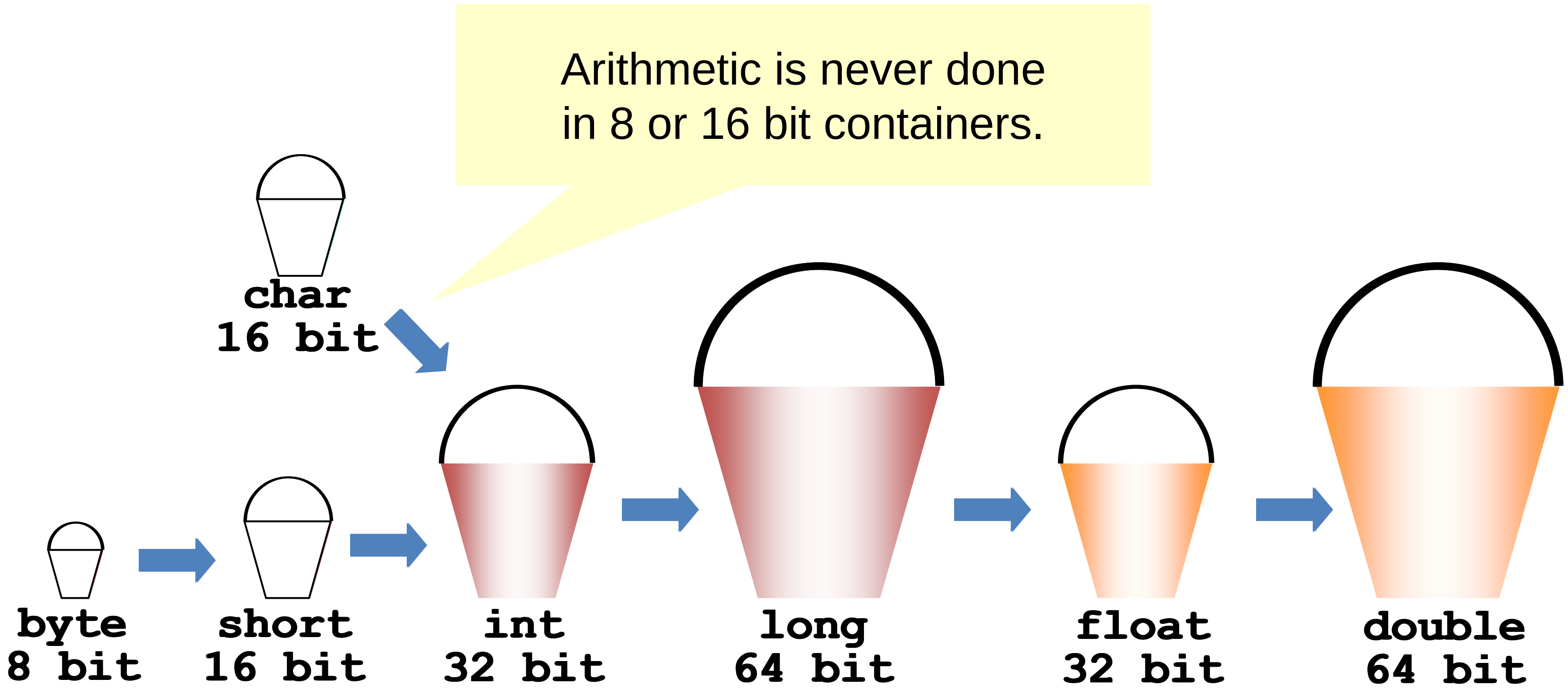
Illegal

```
b = l;
l = f;
c = s;
```

OK

```
l = b;
i = c;
f = l
```

Automatic Promotion with Arithmetic

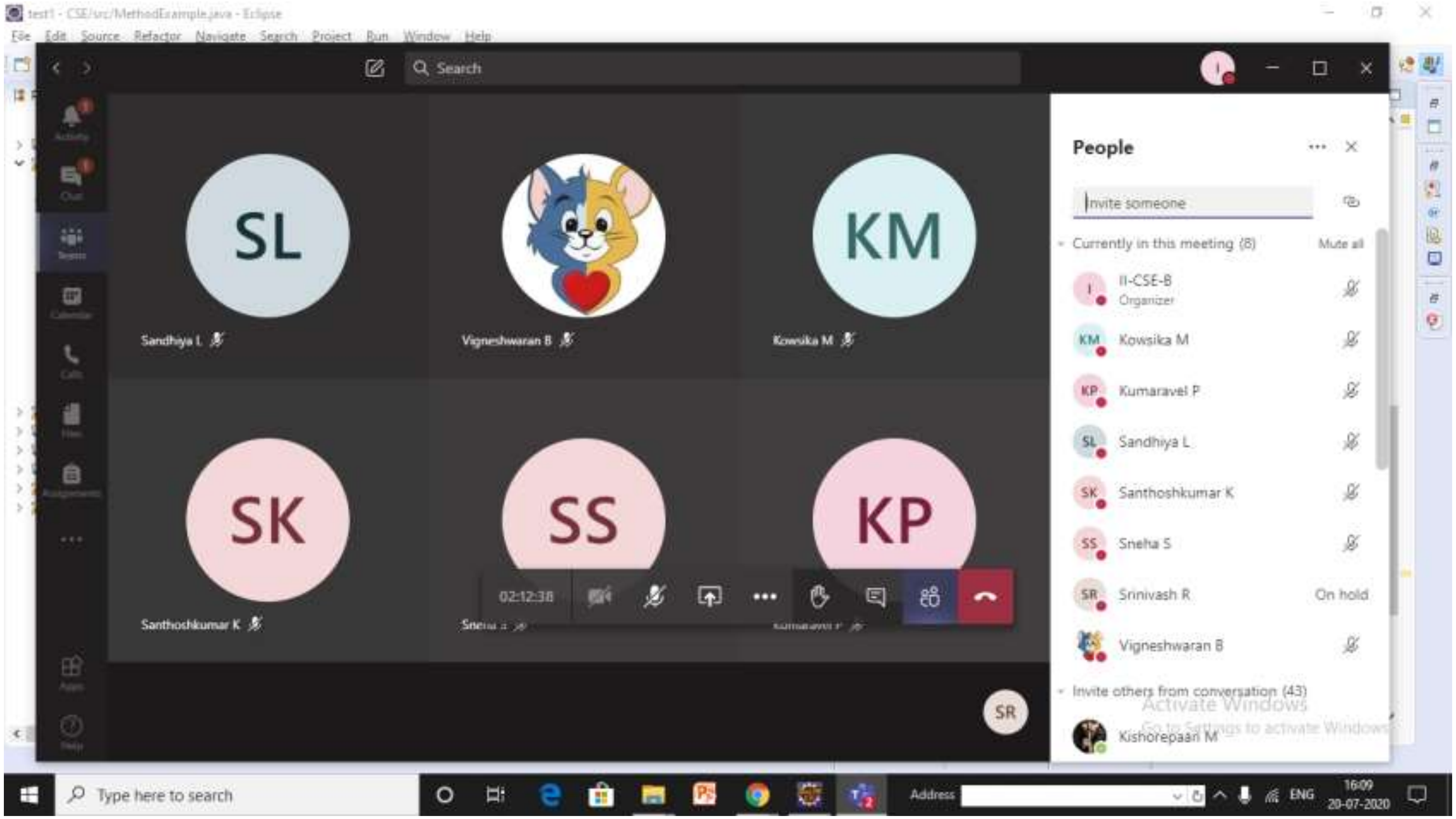


<u>Illegal Casts</u>

```
s = s + b;
s = s + s;
```

<u>OK</u>

```
s = (short)(s + b);
s = (short)(s + s);
```


test1 - CSE/src/MethodExample.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Search
SL
Sandhiya L
Vigneshwaran B
KM
Kowsika M
SK
Santhoshkumar K
SS
Sneha S
KP
Kumaravel P
02:12:38
SR
People
Invite someone
Currently in this meeting (8)
Mute all
II-CSE-B
Organizer
Kowsika M
Kumaravel P
Sandhiya L
Santhoshkumar K
Sneha S
Srinivash R
On hold
Vigneshwaran B
Invite others from conversation (43)
Kishorepaan M
Activate Windows
Go to Settings to activate Windows.
Type here to search
Address
ENG
16:09
20-07-2020