GANPAT UNIVERSITY

U. V. PATEL COLLEGE OF ENGINEERING

2CEIT302

OBJECT ORIENTED PROGRAMMING

UNIT 2

LITERALS, DATA TYPES, VARIABLES, OPERATORS AND CONTROL STATEMENTS

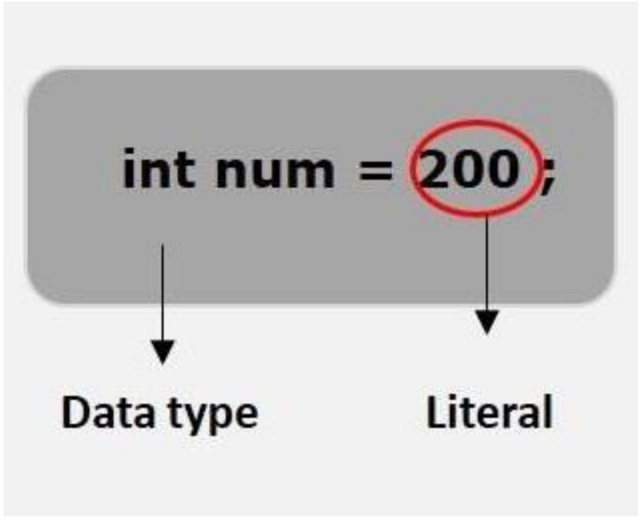
Prepared by: Prof. Y. J. Prajapati (Asst. Prof in IT Dept. , UVPCE)

Outline

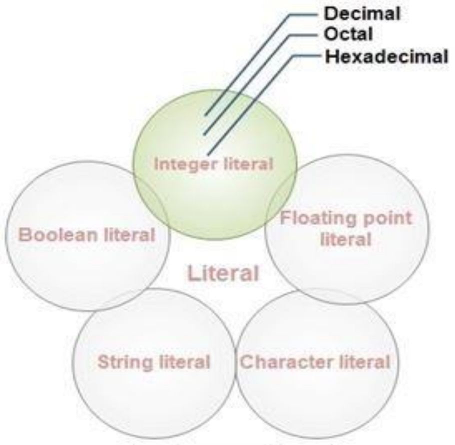
- Literals, Data Types (Integer, Float, Char, Boolean), Variables
- Operators, Operator Precedence
- if else Statement, switch Statement
- while, do while, for and enhanced for loop
- break, continue, comma Statement
- Implicit and Explicit Type Conversion

Literals

- Literals is an identifier whose value is fixed and does not change during the execution of the program.



Types of Literals



Literals

❖ Integer Literals

- Integer Literals are numbers that has no fractional pars or exponent. It refers to the whole numbers. Integers always begin with a digit or + or -.

We can specify integer constants in

- Decimal
- Octal
- Hexadecimal

Decimal Integer Literals

It consists of any combination of digits taken from the set 0 to 9.

Example:

```
int a = 100;   //Decimal Constant
int b = -145   // A negative decimal constant
```

Octal Integer Literals

It consists of any combination of digits taken from the set 0 to 7. However the first digit must be 0, in order to identify the constant as octal number.

Example:

```
int a = 0374; //Octal Constant
int b = 097;  // Error: 9 is not an octal digit.
```

Hexadecimal Integer Literals

A Sequence of digits begin the specification with 0X or 0X, shadowed by a system of digits in the range 0 to 9 and A (a) to F(f).

Example:

```
int  a = 0x34;
int b = -0XABF;
```

Literals

Floating-point Literals

- Floating-point Literals are also called as real constants. The Floating Point contains decimal points and can contain exponents. They are used to represent values that will have a fractional part and can be represented in two forms – fractional form and exponent form.
- In the fractional form, the fractional number contains the integer part and fractional part. A dot (.) is used to separate the integer part and fractional part.

Example:

float x = 2.7f;

Boolean literals

There are two Boolean literals

- true represents a true Boolean value
- false represents a false Boolean value

Example:

boolean b = true;

Character Literals

- Character Literals are specified as single character enclosed in pair of single quotation marks.

Example:

char ch = 'a';

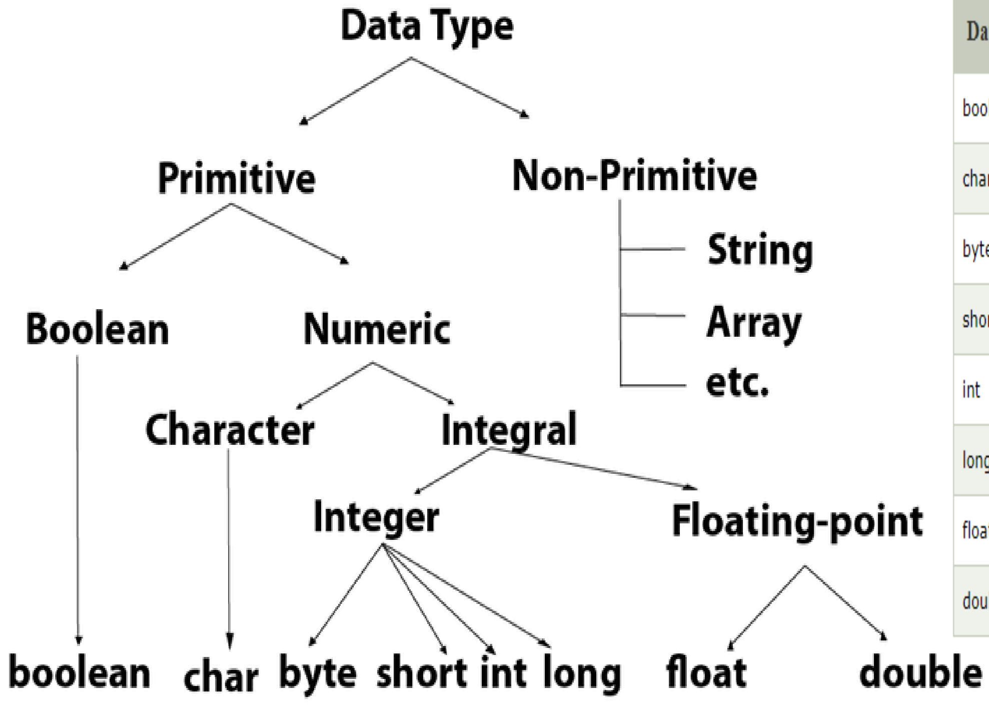
String Literals

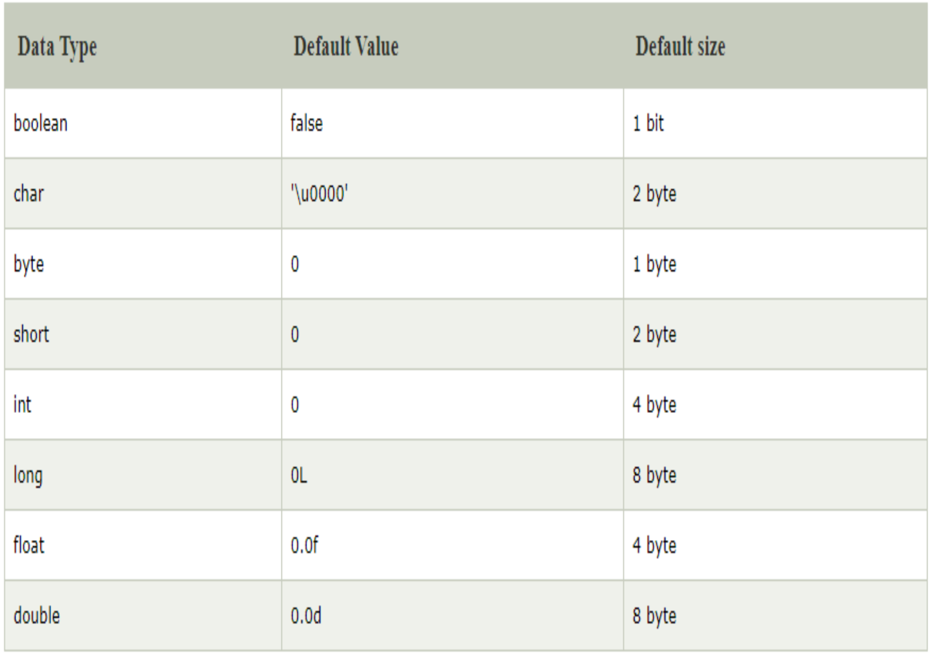
String Literals are treated as an array of char. By default, the compiler adds a special character called the ‘null character’ (‘\0’) at the end of the string to mark the end of the string.

Example:

String str = “good morning”;

Data Types (Integer, Float, Char, Boolean)



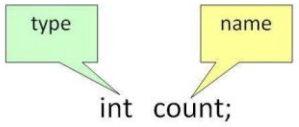


Data Type	Default Value	Default size
boolean	false	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

Variables

- A variable is a container which holds the value while the Java program is executed. A variable is assigned with a data type.

How to declare variables?



type: Type of data that can be stored in this variable.

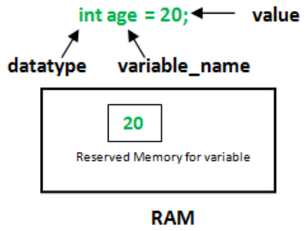
name: Name given to the variable.

How to initialize variables?

datatype: Type of data that can be stored in this variable.

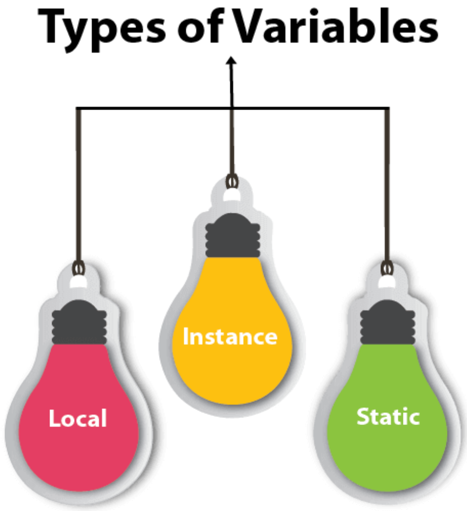
variable name: Name given to the variable.

value: It is the initial value stored in the variable.



There are three types of variables in Java:

- local variable
- instance variable
- static variable



Variables

Local Variable

- A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.
- A local variable cannot be defined with "static" keyword.

Instance Variable

- A variable declared inside the class but outside the body of the method, is called instance variable. It is not declared as static.
- It is called instance variable because its value is instance specific and is not shared among instances.

Static variable

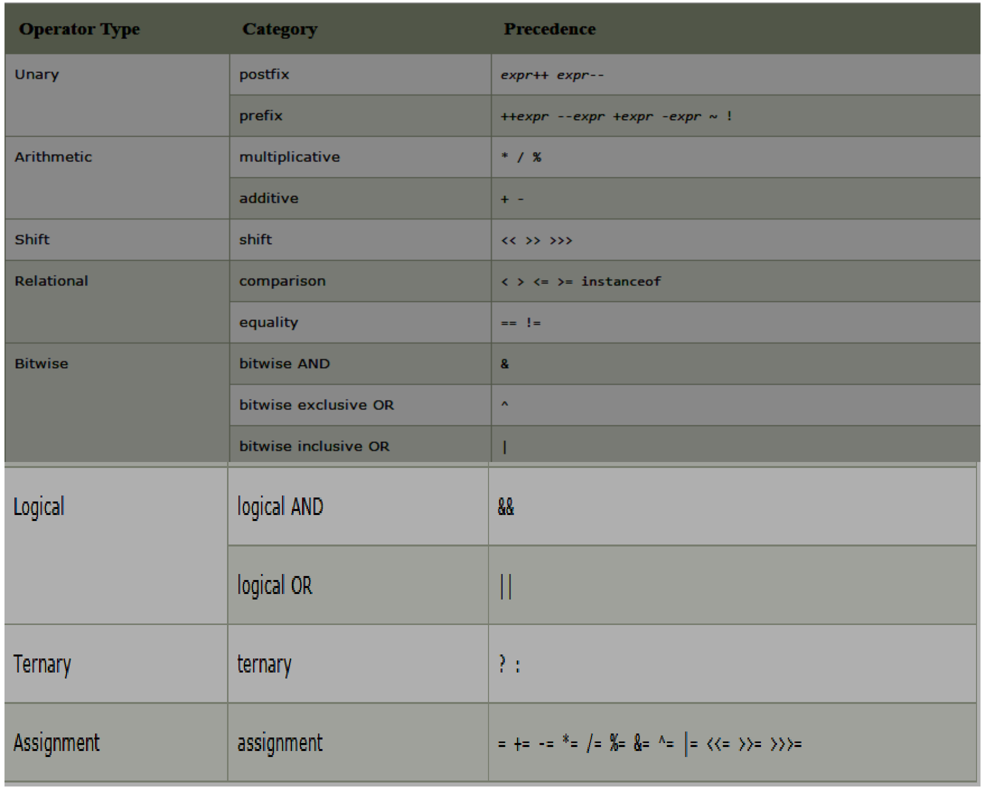
- A variable which is declared as static is called static variable. It cannot be local. You can create a single copy of static variable and share among all the instances of the class. Memory allocation for static variable happens only once when the class is loaded in the memory.

Example

```
class A
{
    int data=50;//instance variable
    static int m=100;//static variable
    void method()
    {
    int n=90;//local variable
    }
}//end of class
```

Operators

- Arithmetic Operators
- Relational Operators
- Bitwise Operators
- Logical Operators
- Assignment Operators
- Conditional Operator



Operator Type	Category	Precedence
Unary	postfix	*expr++ expr--*
	prefix	*++expr --expr +expr -expr* ~ !
Arithmetic	multiplicative	* / %
	additive	+ -
Shift	shift	<< >> >>>
Relational	comparison	< > <= >= instanceof
	equality	== !=
Bitwise	bitwise AND	&
	bitwise exclusive OR	^
	bitwise inclusive OR	|
Logical	logical AND	&&
	logical OR	||
Ternary	ternary	? :
Assignment	assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

if else Statement, switch Statement

The Java *if statement* is used to test the condition.

It checks boolean condition: *true* or *false*.

There are various types of if statement in Java.

- if statement
- if-else statement
- if-else-if ladder
- nested if statement

Java if Statement

- The Java if statement tests the condition. It executes the *if block* if condition is true.

Syntax:

```
if(condition){
//code to be executed
}
```

Java if-else Statement

The Java if-else statement also tests the condition. It executes the *if block* if condition is true otherwise *else block* is executed.

Syntax:

```
if(condition)
{
//code if condition is true
}else{
//code if condition is false
}
```

Java if-else-if ladder Statement

The if-else-if ladder statement executes one condition from multiple statements.

Syntax:

```
if(condition1){
//code to be executed if condition1 is true
}else if(condition2){
//code to be executed if condition2 is true
}
else if(condition3){
//code to be executed if condition3 is true
}
...
else{
//code to be executed if all the conditions are false
}
```

if else Statement, switch Statement

Java Nested if statement

The nested if statement represents the *if block within another if block*. Here, the inner if block condition executes only when outer if block condition is true.

Syntax:

```
if(condition){
    //code to be executed
        if(condition){
          //code to be executed
    }
}
```

Java Switch Statement

The switch statement is a multiway branch statement. It provides an easy way to dispatch execution to different parts of code based on the value of the expression.
Syntax:

```
switch(expression)
{
case value1:
 //code to be executed;
 break;  //optional
case value2:
 //code to be executed;
 break;  //optional
......
default:
 code to be executed if all cases are not matched;
}
```

Java While Loop

- **while loop:** A while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition. The while loop can be thought of as a repeating if statement.
- While loop is also called entry control loop because, in while loop, compiler will 1st check the condition, whether it is true or false, if condition is true then execute the statements.

Syntax :

```
while (boolean condition)
{
loop statements...
}
```

Example :

```
class Example1
 {
public static void main(String[] args)
 {
   int i=1;
   while(i<=10)
   {
      System.out.println(i);
       i++;
   }
}
}
```

Output:

```
1
2
3
4
5
6
7
8
9
10
```

While Loop

Java Infinitive While Loop

- If you pass **true** in the while loop, it will be infinitive while loop.

Syntax:

```
while(true){
//code to be executed
}
```

Example:

```
class Example2 {
public static void main(String[] args) {
    while(true){
        System.out.println("infinitive while loop");
    }
}
}
```

Output:

```
infinitive while loop
infinitive while loop
infinitive while loop
infinitive while loop
infinitive while loop
```

Java do-while Loop

- The Java *do-while loop* is used to iterate a part of the program several times. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use do-while loop.
- The Java *do-while loop* is executed at least once because condition is checked after loop body.
- do...while loop is also called exit control loop because, in do...while loop, compiler will 1st execute the statements, then check the condition, whether it is true or false.

Syntax:

```
do{
//code to be executed
}while(condition);
```

Example:

```
class Example3 {
public static void main(String[] args) {

    int i=1;
    do{
        System.out.println(i);
    i++;
    }while(i<=10);
}
}
```

Output:

```
1
2
3
4
5
6
7
8
9
10
```

Java For Loop

- The Java *for loop* is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop.
- In for loop we put initialization, condition and increment/decrement all together. Initialization will be done once at the beginning of loop. Then, the condition is checked by the compiler. If the condition is false, for loop is terminated. But, if condition is true then, the statements are executed until condition is false.

Syntax:

```
for(initialization;condition;incr/decr)
{
//statement or code to be executed
}
```

```
class Example {
public static void main(String[] args) {
    //Code of Java for loop
    for(int i=1;i<=10;i++){
        System.out.println(i);
    }
}
}
```

Output:

```
1
2
3
4
5
6
7
8
9
10
```

Java For-each Loop | Enhanced For Loop

- The Java for-each loop or enhanced for loop is introduced since J2SE 5.0.
- It provides an alternative approach to traverse the array or collection in Java.
- It is mainly used to traverse the array or collection elements.
- The advantage of the for-each loop is that it eliminates the possibility of bugs and makes the code more readable.
- It is known as the for-each loop because it traverses each element one by one.
- The drawback of the enhanced for loop is that it cannot traverse the elements in reverse order.
- Here, you do not have the option to skip any element because it does not work on an index basis.

How it works?

- The Java for-each loop traverses the array or collection until the last element.
- For each element, it stores the element in the variable and executes the body of the for-each loop.

For-each loop Examples

```
class Example1
{
  public static void main(String args[])
{
   //declaring an array
   int arr[]={12,13,14,44};
   //traversing the array with for-
each loop
   for(int i:arr)
   {
     System.out.println(i);
   }
 }
}
```

Output:

```
12
13
14
44
```

- **Java for-each loop where we are going to total the elements.**

```
class Example2
{
  public static void main(String args[])
{
   int arr[]={12,13,14,44};
   int total=0;
   for(int i:arr)
  {
     total=total+i;
   }
  System.out.println("Total: "+total);
 }
}
```

Output:

```
Total: 83
```

- **collection elements**

```
import java.util.*;
class Example3
{
  public static void main(String args[])
{
   //Creating a list of elements
   ArrayList<String> list=new ArrayList<String>();
   list.add("OOP");
   list.add("DS");
   list.add("DE");
   //traversing the list of elements using for-each loop
   for(String s:list)
  {
     System.out.println(s);
   }
 }
}
```

Output:

```
OOP
DS
DE
```

Java For Loop vs While Loop vs Do While Loop

Comparison	for loop	while loop	do while loop
Introduction	The Java for loop is a control flow statement that iterates a part of the programs multiple times.	The Java while loop is a control flow statement that executes a part of the programs repeatedly on the basis of given boolean condition.	The Java do while loop is a control flow statement that executes a part of the programs at least once and the further execution depends upon the given boolean condition.
When to use	If the number of iteration is fixed, it is recommended to use for loop.	If the number of iteration is not fixed, it is recommended to use while loop.	If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use the do-while loop.

Syntax	`for(init;condition;incr/decr)`
	`{`
	` // code to be executed`
	`}`
		`while(condition)`
		`{`
		`//code to be executed`
		`}`
			`do{`
			`//code to be executed`
			`}`
			`while(condition);`

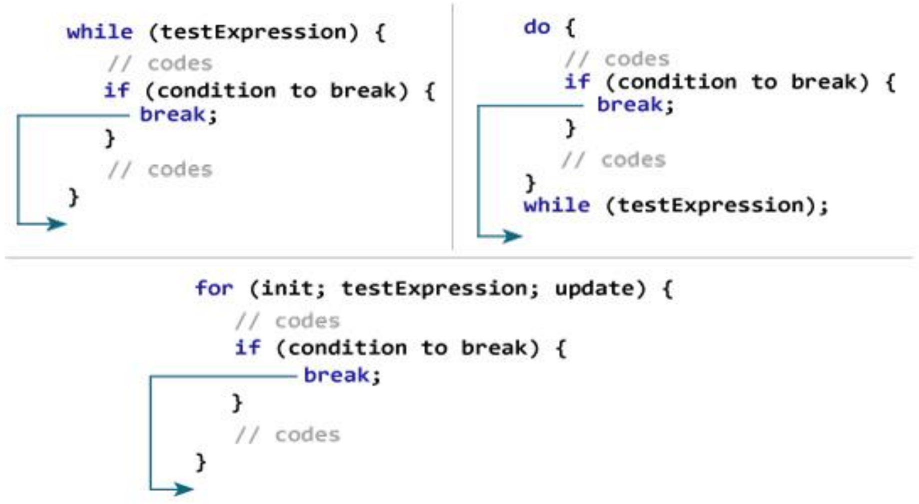
Example	`//for loop`
	`for(int i=1;i<=10;i++)`
	`{`
	`System.out.println(i);`
	`}`
		`//while loop`
		` int i=1;`
		`while(i<=10)`
		`{`
		`System.out.println(i); i++;`
		`}`
			`//do-while loop`
			`int i=1;`
			` do`
			`{`
			`System.out.println(i); i++;`
			`}while(i<=10);`

Syntax for infinitive loop	`for(;;)`
	`{`
	` //code to be executed`
	`}`
		`while(true)`
		`{`
		`//code to be executed`
		`}`
			`Do`
			`{`
			`//code to be executed`
			`}while(true);`

break Statement

- The break statement in Java terminates the loop immediately, and the control of the program moves to the next statement following the loop.
- It is almost always used with decision-making statements (Java if..else statements)

How break statement works?



```
while (testExpression) {
   // codes
   if (condition to break) {
      break;
   }
   // codes
}
```

```
do {
   // codes
   if (condition to break) {
      break;
   }
   // codes
}
while (testExpression);
```

```
for (init; testExpression; update) {
   // codes
   if (condition to break) {
      break;
   }
   // codes
}
```

- **Example :**

```
class Example
{
public static void main(String[] args)
 {
// for loop
for (int i = 1; i <= 10; ++i)
{
// if the value of i is 5 the loop terminates
if (i == 5)
{
break;
}
System.out.println(i);
} } }
```

Output:

```
1
2
3
4
```

Continue Statement

- The **continue** keyword can be used in any of the loop control structures. It causes the loop to immediately jump to the next iteration of the loop.

Syntax :

```
continue;
```

- Example

```
class Example
{
  public static void main(String[] args) {
    for (int i = 0; i < 10; i++) {
      if (i == 5) {
        continue;
      }
      System.out.println(i);
    }
  }

}
```

Output:

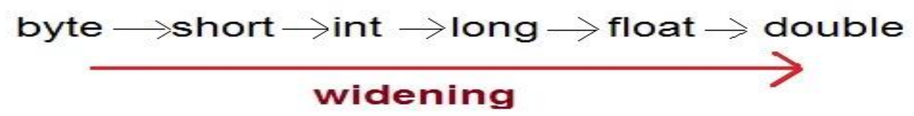
```
1
2
3
4
6
7
8
9
10
```

Implicit and Explicit Type Conversion

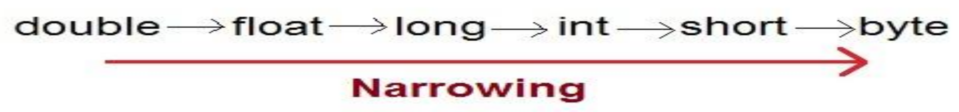
- When you assign value of one data type to another, the two types might not be compatible with each other. If the data types are compatible, then Java will perform the conversion automatically known as Automatic Type Conversion and if not then they need to be casted or converted explicitly.
- **Type Casting Types in Java**

Java Type Casting is classified into two types.

- Widening Casting (**Implicit**) – Automatic Type Conversion



- Narrowing Casting (**Explicit**) – Need Explicit Conversion



Example

- **Widening Casting**(**Implicit**) **(smaller to larger type)**
- **Widening Type Conversion** can happen if both types are compatible and the target type is larger than source type.

```
class ImplicitExample {
  public static void main(String args[]) {
    byte i = 40;
    // No casting needed for below conversion
    short j = i;
    int k = j;
    long l = k;
    float m = l;
    double n = m;
    System.out.println("byte value : "+i);
    System.out.println("short value : "+j);
    System.out.println("int value : "+k);
    System.out.println("long value : "+l);
    System.out.println("float value : "+m);
    System.out.println("double value : "+n);
  } }
```

- **Narrowing Casting(Explicit) (larger to smaller type)**
- When we are assigning a larger type to a smaller type, **Explicit Casting** is required.

```
class ExplicitExample {
  public static void main(String args[]) {
    double d = 30.0;
    // Explicit casting is needed for below conversion
    float f = (float) d;
    long l = (long) f;
    int i = (int) l;
    short s = (short) i;
    byte b = (byte) s;
    System.out.println("double value : "+d);
    System.out.println("float value : "+f);
    System.out.println("long value : "+l);
    System.out.println("int value : "+i);
    System.out.println("short value : "+s);
    System.out.println("byte value : "+b);
  }}
```

THANK YOU!
ANY QUESTIONS?