

Chapter - 2

Introducing Data Types and Operators

- Java's Primitive Types,
- Literals
- A Closer Look at Variables
- The Scope and Lifetime of Variables
- operators
- Shorthand Assignments
- Type conversion in Assignments
- Using Cast
- Operator Precedence
- Expressions.

Data types

Primitive types

Integer

- byte
- short
- int
- long

Float

- float
- double

Logical

- boolean

Character

- char

Non-primitive Reference / object type

class

interface

Array

Data types

Primitive types

Integer

- byte
- short
- int
- long

No . of bytes
occupied

1 bytes

2 bytes

4 bytes

8 bytes

Default Value

0

0

0

0L

Float

- float
- double

4 bytes

8 bytes

0.0f

0.0d

Character

- char

2 bytes

\0

Logical

- boolean

1 bit

false

Program

Problem Description: The finance department of a company wants to calculate the monthly pay of one of its employee. Monthly pay should be calculated as mentioned in the below formula and display all the employee details.

Monthly pay =

*Number of hrs worked in a week * pay rate per hour * no of weeks in a month*

Note:

*Program Name: **EmployeePay.java***

*consider Number of hrs worked by employee in a week = **40***

*pay rate per hour = **Rs 400.***

*number of weeks in a month = **4.***

Lab Program - 20 min

```
public class EmployeePay {  
    public static void main(String[] args) {  
  
        float payRate = 400;  
        int week = 4;  
        int hrs = 40;  
  
        float monthlyPay = hrs * payRate * week;  
  
        System.out.println( " Monthly pay = " + monthlyPay);  
    }  
}
```

Reading User input: Inputting Character

- Syntax:

System.in.read();

- Waits till user supplies input character
- Character is returned as number, Need to typecast to character

Example:

```
class charInput {  
    public static void main() throws java.io.Exception {  
        char ch;  
        System.out.println("Input a character: ");  
        ch = (char) System.in.read();  
        System.out.println(" character is: " + ch);  
    }  
}
```

4. Reading User input: other Datatypes

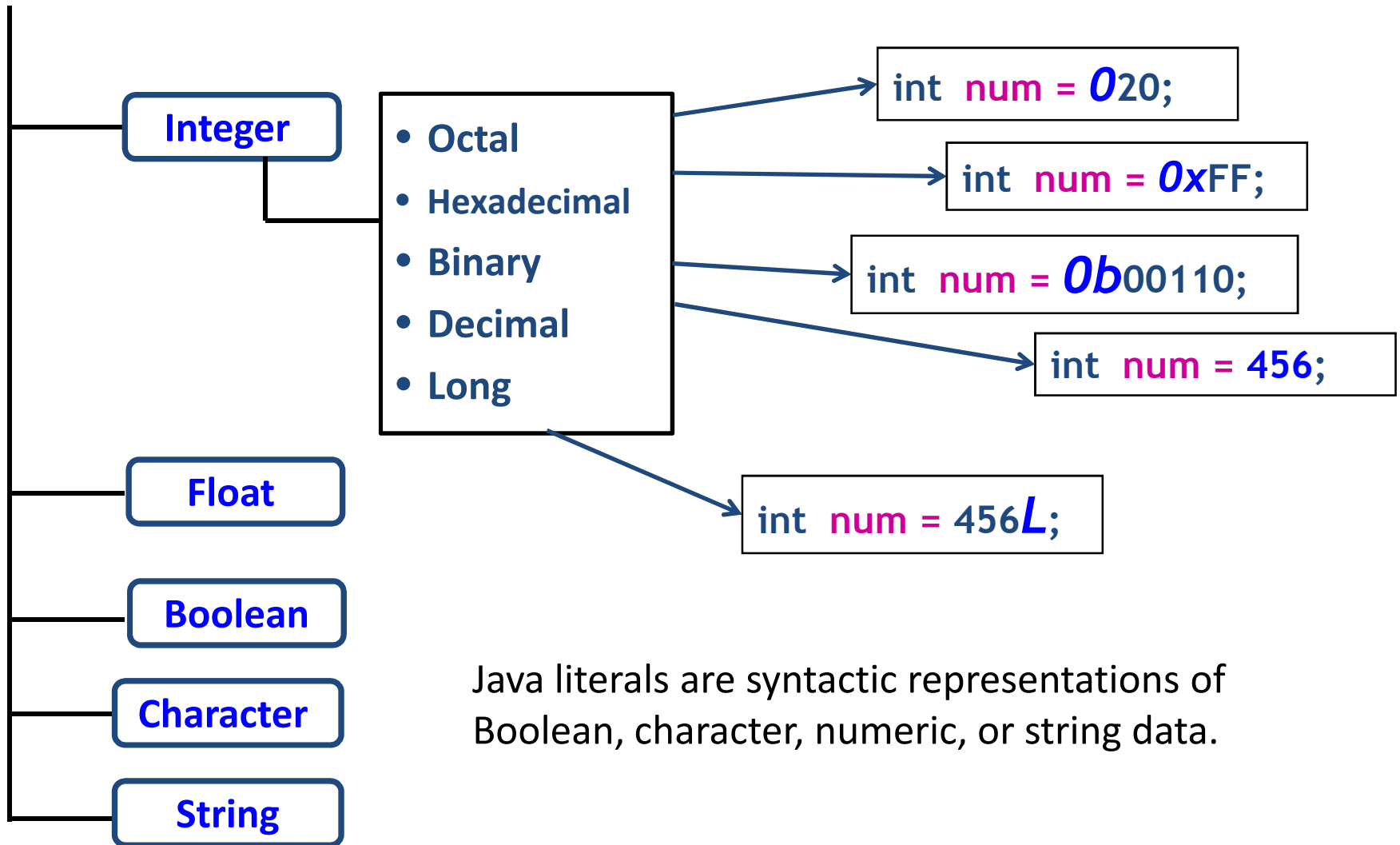
```
InputStreamReader input = new InputStreamReader(System.in);  
    // Convert Byte Stream into Character stream
```

```
BufferedReader buffer = new BufferedReader(input);  
    // Stores input into buffer
```

Combining above 2 lines:

```
BufferedReader buffer = new BufferedReader ( new InputStreamReader ( System.in) );
```


5. Literals



. Literals

Integer

Float

- float

Representation

F or f

Size

32 bits

Default

-

- double

D or d

64 bits

It is default type

Boolean

Character

String

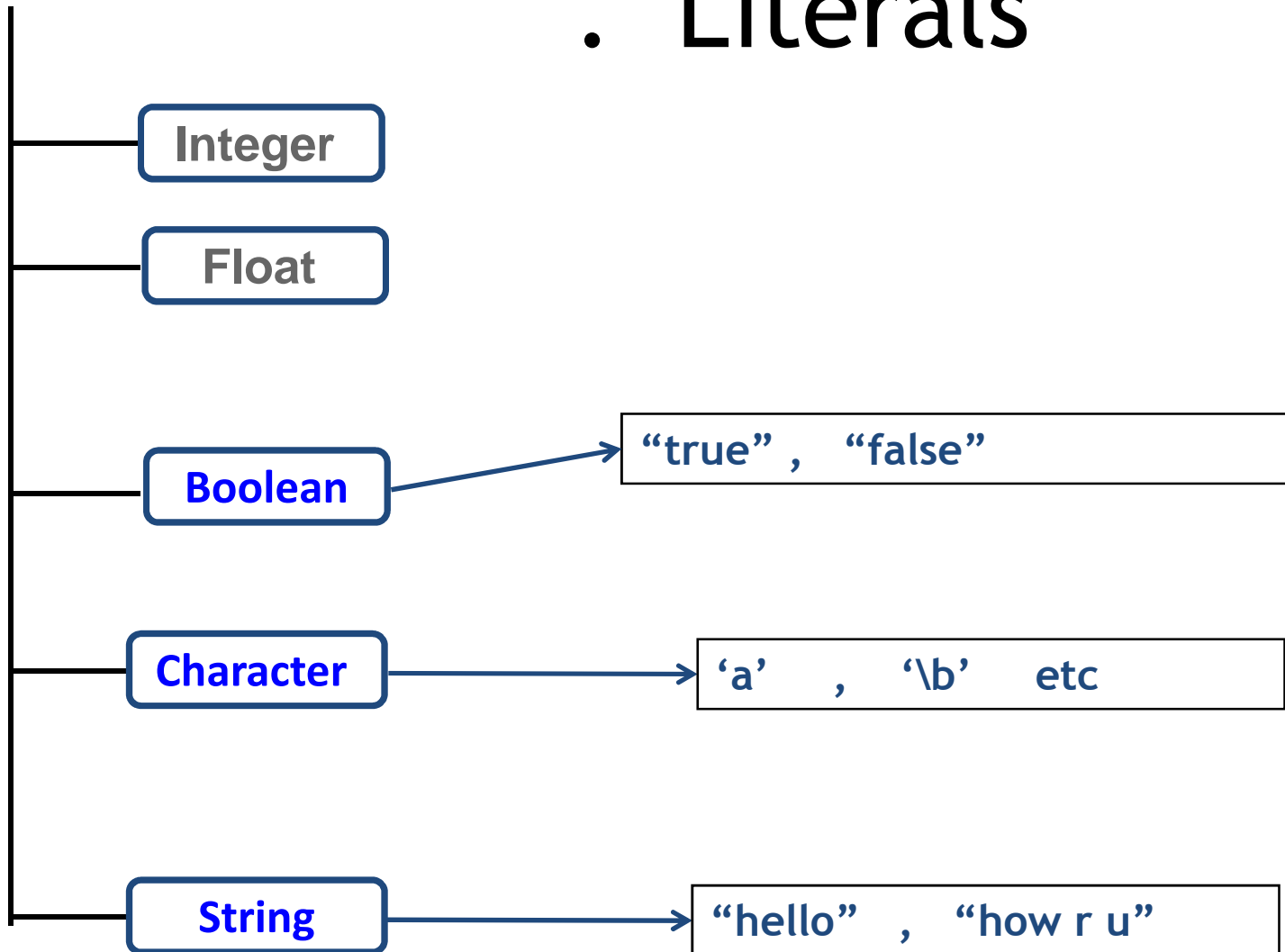
Example:

4.20 // default double *literal*

6.430**D**; // double *literal*

6.430**F**; // float *literal*

. Literals



Program

Problem Description: The finance department of a company wants to calculate the monthly pay of one of its employee. Monthly pay should be calculated as mentioned in the below formula and display all the employee details.

Monthly pay =

*Number of hrs worked in a week * pay rate per hour * no of weeks in a month*

Note:

Rewrite the program by reading data from user

```
public class EmployeePay {
```

```
    public static void main(String[] args) throws IOException {  
        InputStreamReader in = new InputStreamReader(System.in);  
        BufferedReader buffer = new BufferedReader(in);  
        String str;
```

```
        // Read the employee name  
        str = buffer.readLine();
```

```
        // Read the employee salary  
        str = buffer.readLine();
```

```
        // Calculate the employee pay  
        double salary = Double.parseDouble(str);
```

```
        // Print the employee pay  
        System.out.println("Employee Pay: " + salary);
```

```
    }
```

```
}
```

Variables

1. All variables must be declared before their use.
2. It enables java to perform strict type checking

Variables

- Initialization

```
int  count = 10;  
char ch = 'x';
```

- Dynamic Initialization

```
int  IT = getTax( basic, Da, PF);
```

Variables

❑ Difference between **declaring a variable** and **defining a variable**?

❑ **Declaration:**

➤ *String s;*

➤ **Definition:**

➤ *String s = new String ("abcd");*

➤ *String s = "abcd";*

Variables : Scope and lifetime

```
class Demo
{
    public static void main ( String [ ] args ) {
        int x =10;

        if (true) { // creates new scope
            int y = 20;

            System.out.println ( y ); // OK
        }

        System.out.println ( y ); // ERROR, not in scope
        System.out.println ( x ); // OK
    }
}
```

Operators

1. Arithmetic operators
2. Increment decrement
3. Relational operators
4. Logical operators
5. Short-Circuit logical operators
6. Bitwise operators
7. Assignment operators
8. ShorthHand operators

Operators

❑ Arithmetic operators

+	add
-	subtract
*	multiply
/	divide
%	find remainder

❑ Increment decrement

Operator	Name	Example expression	Meaning
++	Postfix increment	x++	add 1 to x and return the old value
++	Prefix increment	++x	add 1 to x and return the new value
--	Postfix decrement	x--	take 1 from x and return the old value
--	Prefix decrement	--x	take 1 from x and return the new value

. Operators

□ Relational operators

<i>Operator</i>	<i>Use</i>	<i>Description</i>
>	op1 > op2	op1 is greater than op2
>=	op1 >= op2	op1 is greater than or equal to op2
<	op1 < op2	op1 is less than op2
<=	op1 <= op2	op1 is less than or equal to op2
==	op1 == op2	op1 and op2 are equal
!=	op1 != op2	op1 and op2 are not equal

*Outcome of **relational** and **logical** operator is **boolean***

. Operators : Logical operators

Operator	Meaning of Operator	Example
&&	Logial AND. True only if all operands are true	If c = 5 and d = 2 then, expression <code>((c == 5) && (d > 5))</code> equals to 0.
	Logical OR. True only if either one operand is true	If c = 5 and d = 2 then, expression <code>((c == 5) (d > 5))</code> equals to 1.
!	Logical NOT. True only if the operand is 0	If c = 5 then, expression <code>! (c == 5)</code> equals to 0.

Outcome of relational and logical operator is boolean

. Operators

□ Short-circuit operators

Operator	Meaning	Description
&&	Short-circuit AND	Evaluate second operand only when necessary
	Short-circuit OR	

- false && (*anything*) is short-circuit evaluated to false.
- true || (*anything*) is short-circuit evaluated to true.

□ Short-circuit operators

1. $(1 \neq 1) \ \&\& \ (2 == 2)$

- **$2 == 2$ will never get evaluated**
- **$1 \neq 1$ that the result will be false**

2. $(1 == 1) \ || \ (2 \neq 2)$

- **$2 \neq 2$ will never get evaluated**
- **$1 == 1$ that the result will be true**

```
class ShortCircuitAnd
{
    public static void main(String arg[])
    {
        int c = 0, d = 100, e = 50;
        if(c == 1 && e++ < 100)
        {
            d = 150;
        }
        System.out.println("e = " + e );
    }
}
```

Output: 50

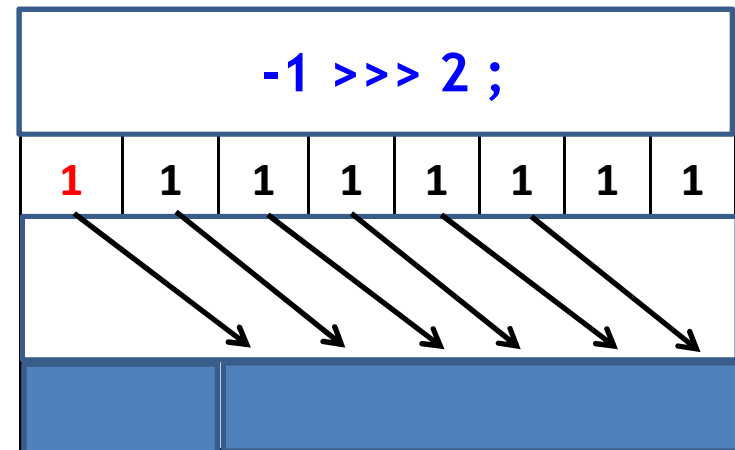
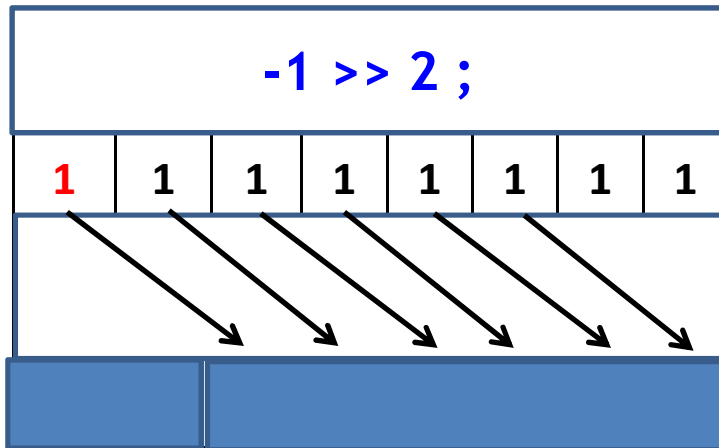
Operators

□ Bitwise operators

Operator	Meaning	Use	Description
&	Bitwise AND	b = 5 & 6	Bit by bit evaluated
	Bitwise OR	b = 5 6	Bit by bit evaluated
^	Exclusive OR	c = 5^6	Bit by bit evaluated
>>	Signed Shift right	b = -5 >> 2	sign bit is shifted in the high-order positions.
>>>	Unsigned Shift right	b = -5 >>> 2	Zero's are shifted in the high-order positions.
<<	Left shift	b = 5 << 2	Zero's are shifted in the Low-order positions.
~	One's complement	b = ~5;	One's complement of 5

. Operators

Operator	Meaning	Use	Description
<code>>></code>	Signed Shift right	<code>b = -1 >> 2</code>	sign bit is filled in the high-order positions.
<code>>>></code>	Unsigned Shift right	<code>b = -1 >>> 2</code>	Zero's are filled in the high-order positions.



Operators

□ Assignment operators

```
int x, y, z;  
x = y = z = 10;
```



TABLE 2.3 Shorthand Operators

<i>Operator</i>	<i>Name</i>	<i>Example</i>	<i>Equivalent</i>
+=	Addition assignment	i += 8	i = i + 8
-=	Subtraction assignment	f -= 8.0	f = f - 8.0
*=	Multiplication assignment	i *= 8	i = i * 8
/=	Division assignment	i /= 8	i = i / 8
%=	Remainder assignment	i %= 8	i = i % 8

6. Naming convention

- *Packages :*

- All small letters (Example: *java.io*)

- *Class / interface :*

- Each word Start with capital (1st letter)

Example: *FixedStackDemo { -- }*

- *Methods / variables:*

- First word all small letters
- Latter each new word start with capital

Example: *getArea()*

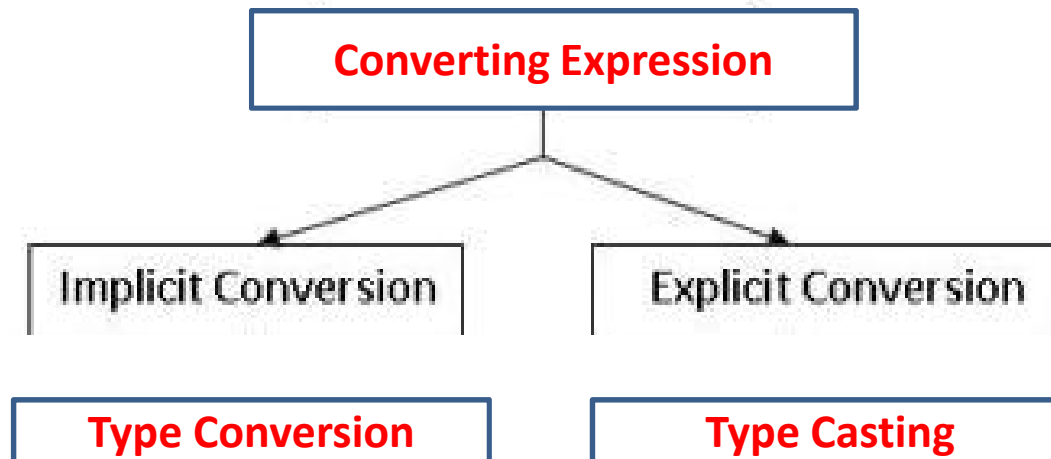
- *Constants*

- All capitals

Example: *MAX = 20*

Type Conversion in expression

- Type conversion and Type casting



Type conversion

- Boolean cannot be converted to other types.
- It is common to assign one type of variable to another.
- **Example:** assign an *int* value to a *float* variable

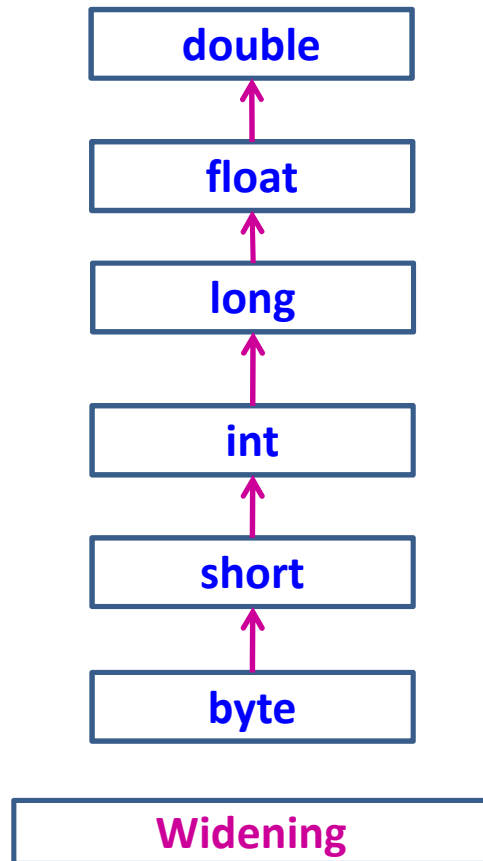
```
int x = 10;           // occupies 4 bytes  
double y = x;        // occupies 8 bytes  
System.out.println( y );    // prints 10.0
```

In the above code **4 bytes** integer value is assigned to **8 bytes** double value.

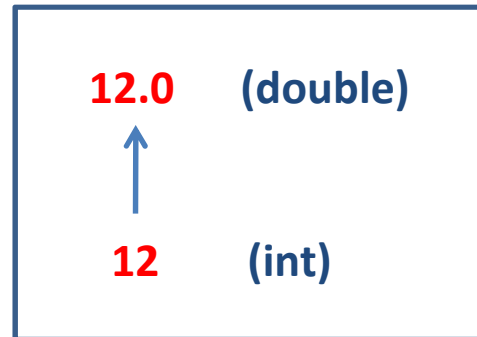
- Implicit conversions:
 - An implicit conversion means that a *value* of one type is *changed* to a *value* of another type without any special directive from the programmer.
 - A data type of lower size (occupying less memory) is assigned to a data type of higher size. This is done implicitly by the JVM.
 - The lower size is widened to higher size. This is also named as automatic type conversion

Type Conversion in expression - Type casting

- Type conversion and Type casting

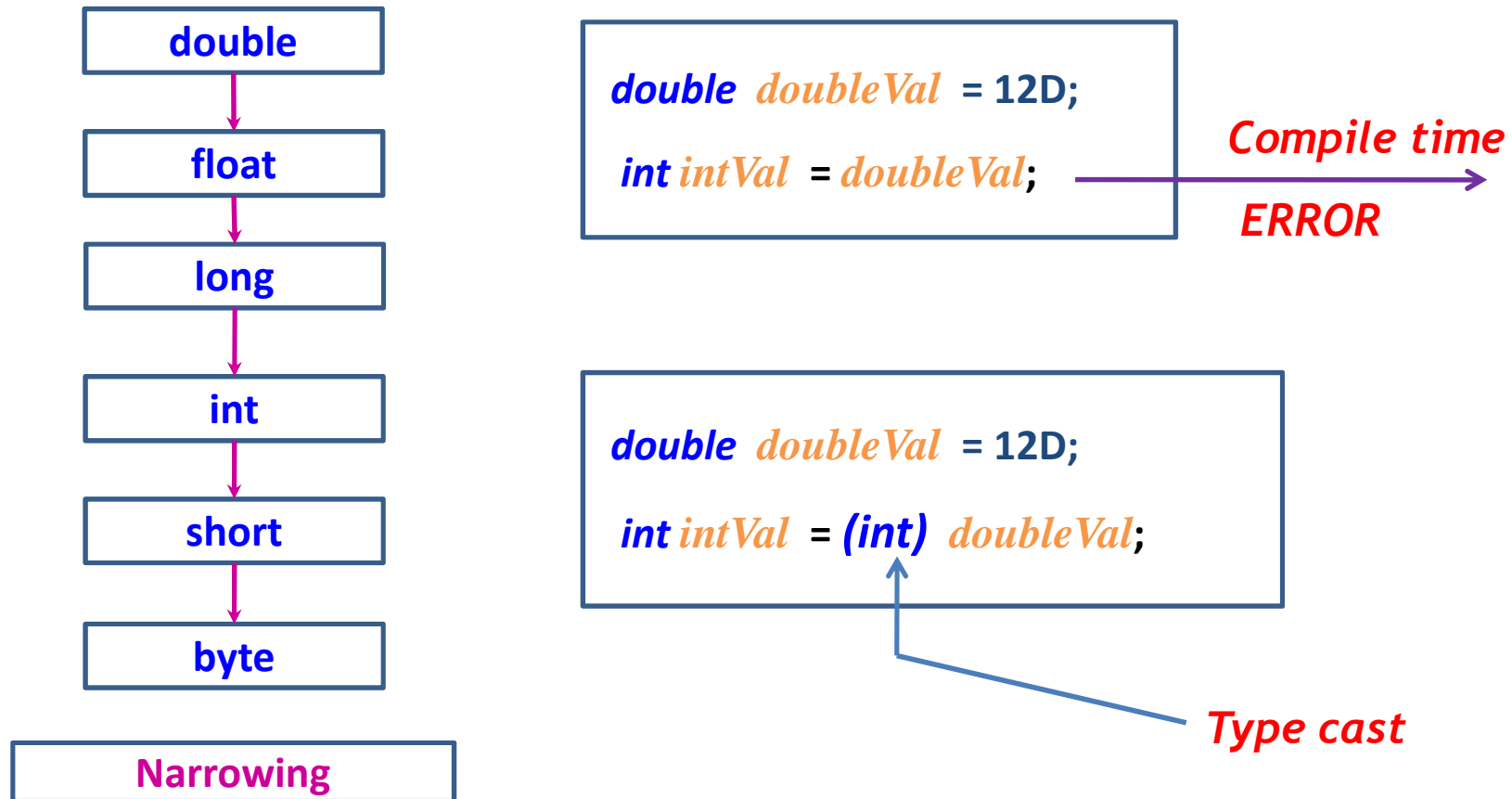


```
int intVal = 12;  
double dVal = intVal;
```



Type Conversion in expression - Type casting

- Why need type caste



Type Conversion and Type casting

- **Explicit conversions:**

- Explicit conversions are done via **casting**. The name of the type to which you want a value converted is given, in parentheses, in front of the value.

- The general form of cast is
(target-type)expression

- Here, target-type specifies the desired type to convert the specified expression to.

EX:

```
double x = 10.5;
```

```
int y = (int) x;
```

The double **x** is explicitly converted to int **y**. The thumb rule is, on both sides, the same data type should exist.

Conversion in expression

❑ Using wrapper class (Helper class)

Primitive Type	Wrapper Class	Primitive Type	Wrapper Class
boolean	Boolean	float	Float
byte	Byte	int	Integer
char	Character	long	Long
double	Double	short	Short

- ✓ *defined in `java.lang`*
- ✓ *holds fundamental data type values within an Object*

String => value

❑ *Wrapper.parse* *Type* ()

Type is a *DataType* like
int, float, double, etc

➤ parse a string & **return** the literal value.

```
String str = "42" ;
```

```
int intVal = Integer.parseInt(str);    // returns 42
```

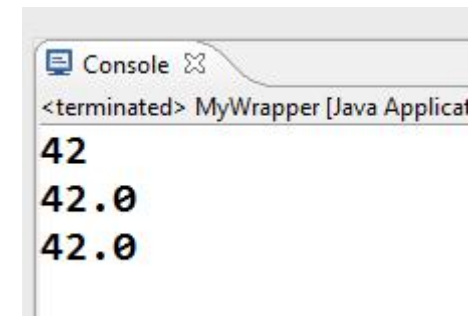
```
double dVal = Double.parseDouble(str); // returns 42.0
```

```
String str = "true" ;
```

```
boolean bVal = Boolean.parseBoolean(str); // returns true
```

WrapperDemo.java

```
class WrapperDemo {  
  
    public static void main (String[] args) {  
  
        String str = "42";  
  
        int iVal = Integer.parseInt(str);  
  
        double dVal = Double.parseDouble(str);  
  
        float fVal = Float.parseFloat(str);  
  
        System.out.println ( iVal );  
        System.out.println ( dVal );  
        System.out.println ( fVal );  
  
    }  
}
```



The screenshot shows a console window titled "Console" with a close button. Below the title bar, it says "<terminated> MyWrapper [Java Applicat". The output of the program is displayed on three lines: "42", "42.0", and "42.0".