Object-Oriented Programming (CS F213)

Module I: Object-Oriented and Java Basics

CS F213 RL 2.2: Java Primitive Types

BITS Pilani

Dr. Pankaj Vyas
Department of Computer Science, BITS-Pilani, Pilani Campus

CS F213 RL 2.2 : Topics

- Java Type System (Only Introduction)
- Primitive Types in Java
- What is Type Promotion ?
- What is Type Casting ?
- Use of System.out.println() and System.out.print() statements

Java Type System

- A type in Java specifies a set of values and set of operations that can be applied over the values
- A type is used to declare the type of variables.
- For Example, ‘int’ type specifies all 32-bit integers (set of values : -2^{32} to $+2^{32}-1$, set of operations: All arithmetic operations)
- Every ‘Type’ is any one of the following

1. Primitive Types (boolean, byte, short, character, int, long, float and double)
2. A class Type [For Example: Box, Student, String Types]
3. An interface Type
4. An Array Type
5. ‘null’ Type

Java Primitive Types

- Eight Primitive Types

1. boolean (true / false)
2. byte
3. short
4. character
5. int
6. long
7. float
8. double

Java Primitive Types: boolean

- boolean type for variables is used to store only two values (true and false)
- Memory Requirement : 1 bit
- Java does not represent boolean values by 1 and 0 as used in ‘C’ Programming language
- Every logical and relational expression results in boolean type value

```
boolean b = true;
boolean x = a>b;
boolean z =  a > b && b > 10
```

Valid Statements

```
boolean b = 1;
booelan x = 0;
```

In-Valid Statements

Java Primitive Types: boolean

- Conditional ‘if’ statements in Java use boolean type
- Note that in ‘C’ programming language ‘if’ statements use ‘0’ to represent false and any other positive value (>=1) represents true

C-Programming Code

```
if(10)
        printf(“Hello”);
else
        printf(“Hi”);
```

Displays “Hello”

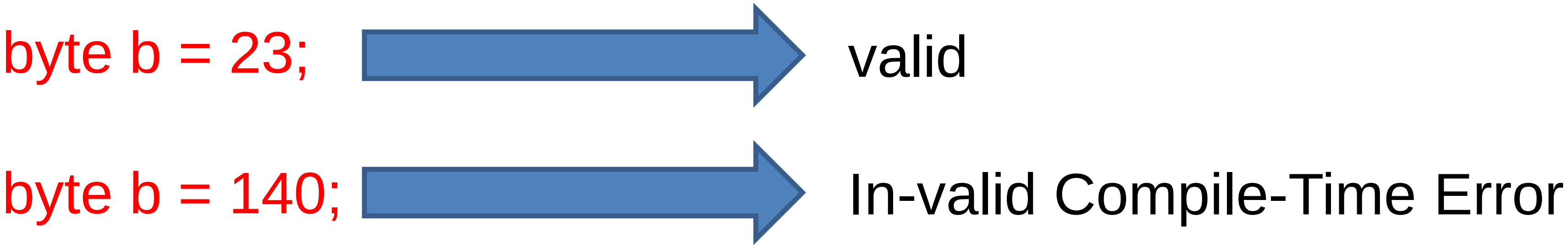
Equivalent Java-Programming Code

```
if(10)
        System.out.println(“Hello”);
else
        System.out.println(“Hi”);
```

Compile-Time Error

Java Primitive Types: byte

- Memory Requirement : 8 bits (1 Byte), Value Range: -128 to 127
- byte type variables can not store values outside their defined range

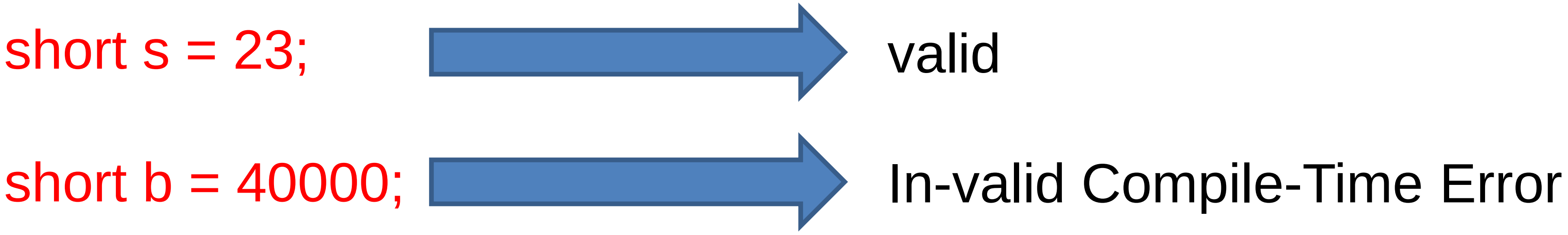


byte b = 23; → valid

byte b = 140; → In-valid Compile-Time Error

Java Primitive Types: short

- Memory Requirement : 16 bits (2 Bytes)
- Value Range: -32768 $(-2^{16})$to +32767 $(+2^{16}-1)$



short s = 23; → valid

short b = 40000; → In-valid Compile-Time Error

Java Primitive Types: char

- Memory Requirement : 16 bits (2 Bytes)
- Java Follows Unicode coding scheme and Each Unicode character is assigned a unique integer value (For Example: ‘65’ value denotes character ‘A’)
- Value Range: 0 to +65535 (+2^{16}-1)

char x = ‘a’; → valid

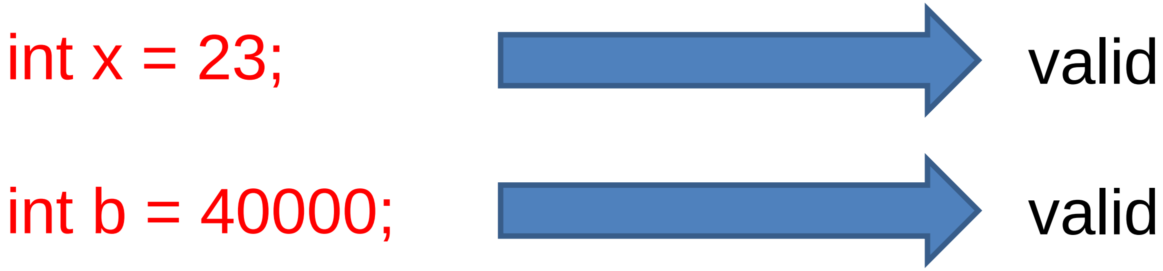
char y = 65; → valid

char z = ‘\n’; → valid

char a = 967; → valid

Java Primitive Types: int

- Specifies all 32-bit integer values
- Memory Requirement : 32 bits (4 Bytes)
- Size is Independent of the Platforms (Unlike in ‘C’ where size is platform dependent)
- Value Range: (-2^{31}) to $(+2^{31}-1)$

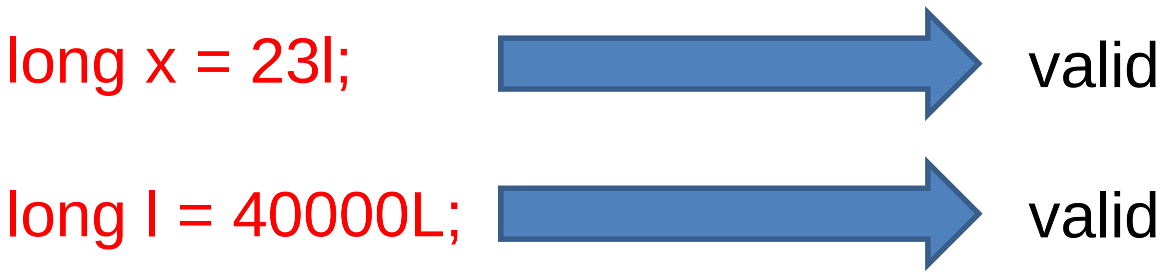


int x = 23; → valid

int b = 40000; → valid

Java Primitive Types: long

- Specifies all 64-bit integer values
- Memory Requirement : 64 bits (8 Bytes)
- Size is Independent of the Platforms
- To explicitly represent a long type value add letter ‘L’ or ‘l’ after the value. For Example: 20L, 4000l etc.
- Value Range: $(-2^{63})$to + $(+2^{63}-1)$



long x = 23l; → valid

long l = 40000L; → valid

Java Primitive Types: float

- Used for storing real values (Numbers with fractional parts)
- Memory Requirement : 32 bits (4 Bytes)
- Value Range: +3.40282347E+38F, -3.40282347E+38F
- To explicitly represent a real value of type float, insert a letter ‘f’ or ‘F’ after the value. For Example: 2.3f, 4.9f, 1.456f etc.
- Precision: 7 Significant Decimal Digits

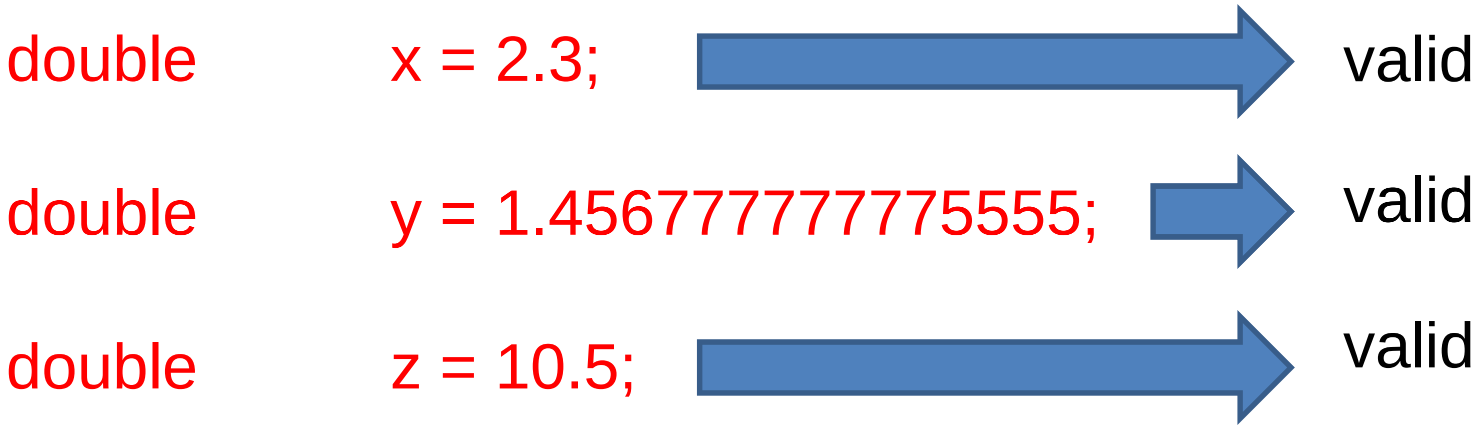
float x = 2.3F; → valid

float y = 1.4567777777775555f; → valid

float z = 10.5; → In-valid

Java Primitive Types: double

- Also Used for storing real values (Numbers with fractional parts)
- Memory Requirement : 64 bits (8 Bytes)
- Value Range: +1.7976931348623157OE+308F, -1.7976931348623157OE+308F
- Precision: 15 Significant Decimal Digits



double x = 2.3; valid

double y = 1.456777777775555; valid

double z = 10.5; valid

What is Type Promotion?

- Type Promotion : Lower Type value is automatically promoted to Higher Type in an arithmetic expression
- Rule 1: ‘byte’, ‘short’ and ‘char’ type values are automatically locally promoted to ‘int’ type and final result of the expression is ‘int’ type
- Rule 2: If any one operand is of ‘long’ type then whole expression is promoted to ‘long’
- Rule 3: If any one operand is of ‘float’ type then whole expression is promoted to ‘float’
- Rule 4: If any one operand is of ‘double’ type then whole expression is promoted to ‘double’.

Type Promotion Example 1

```
// File Name Demo.java
class X
{
    public static void main(String[] args)
    {
            byte        b           =           40;

            short       s           =           20;

            int         x           =           10;

            int         y           =           b * s + x;

            System.out.println(y);

    }// End of main() Method

}// End of class X
```

Type Promotion Example 2

```
// File Name TypePromotion.java
class TypePromotion
{
    public static void main(String[] args)
    {
            byte      b        =          42;
            char      c        =          ‘a’;
            short     s        =          1024;
            int       i        =          50000;
            float     f        =          5.67f;
            double    d        =          0.1234;

            double    result   =          ( f  * b )  + ( i / c) – ( d * s);

            System.out.println(result);

    }// End of main() Method

}// End of class TypePromotion
```

Result is : 626.7784146484375

Type Promotion Example 3

```
// File Name TypePromotion.java
class TypePromotion
{
    public static void main(String[] args)
    {
            byte b = 40;

            b = b + 1;

    }// End of main() Method

}// End of class TypePromotion
```

```
possible loss of precision
found   : int
required: byte
                b = b + 1;
                      ^
1 error
```

Type Promotion Example 4

```
// File Name TypePromotion.java
class TypePromotion
{
    public static void main(String[] args)
    {
        short       s = 40;
        s = s + 1;

        char        x = 65;
        x = x-1

    }// End of main() Method

}// End of class TypePromotion
```

```
possible loss of precision
found   : int
required: short
        s = s + 1;
            ^
possible loss of precision
found   : int
required: char
        x = x-1;
            ^
2 errors
```

What is Type Casting?

- Converting a value of one type (Generally Higher Type) to another type (Generally a Lower Type) only if the types are convertible
- Syntax :

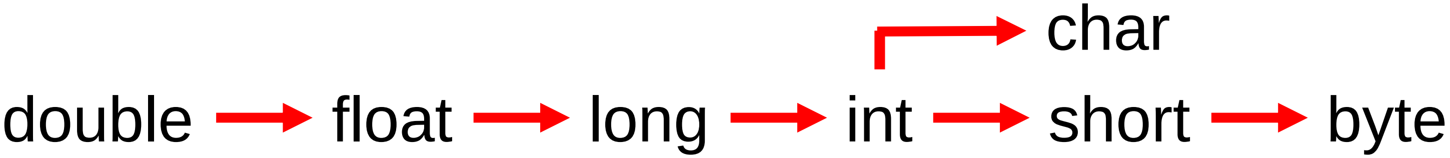
v = (T) value-or-variable-of-higher-type;

where 'v' is variable and 'T' represents the type of 'v'

- Examples

1.	int	x	=	(int) 3.56;
2.	float	y	=	(float) 4.56;
3.	byte	b	=	(byte) 400;

Convertible Types



char

double → float → long → int → short → byte

int → char

Type Casting Example

```
// File Name TypeCasting.java
class TypeCasting
{
        public static void main(String[] args)
        {
                double  d = 65.56;
                char    x = (char) d;
                System.out.println(x);

                char    y = 'A';
                byte    b = (byte) y;
                System.out.println(b);

                float   f = (float) 4.56;
                System.out.println(f);

        }// End of main() Method

}// End of class TypeCasting
```

```
javac TypeCasting.java

java TypeCasting

A
65
4.56
```

Inconvertible Types

- Inconvertible Types : Types that can not be converted to each other
- No numeric type can be type casted to ‘boolean’ type and vice versa

```
// File Name TypeCasting.java
class TypeCasting
{
            public static void main(String[] args)
            {
                        int x = 0;
                        boolean b = (boolean) x;

                        boolean b1 = false;
                        byte        b2 = (byte) b1;
            }// End of main() Method
}// End of class TypeCasting
```

```
inconvertible types
found   : int
required: boolean
          boolean b = (boolean) x;
                                  ^
inconvertible types
found   : boolean
required: byte
          byte   b2 = (byte) b1;
                               ^
2 errors
```

System.out.println()

- Prints/Displays output on console and shifts the print control to a new line (Similar to printf(“\n”) in C)
- Displays output only in String form
- If parameter to it is not in String form then it will be converted to string form
- + operator can be used to concatenate values of from different types
- + operator in Java is used for numeric addition as well as string concatenation
- Tabs can given between values of various fields using tab character ‘\t’
- New line character ‘\n’ can also be used for insering new lines

System.out.println() : Example

- System.out.println(“Hello”+10); → Hello10
- System.out.println(10+20); → 30
- System.out.println(10 + 20 + “Object”); → 30Object
- System.out.println(“10”+20); → 1020
- System.out.println(10 + “20”); → 1020
- System.out.println(“Hello: ” + 20 + “ is my age”); → Hello 20 is my age
- System.out.println(“10” + (20 + 40 * 3) + 60); → 1014060
- System.out.println(10 + (“20” + 40 / 4) + 50); → 10201050

System.out.print()

- Prints/Displays output starting from the same line (Similar printf() without newline)
- Displays output only in String form
- If parameter to it is not in String form then it will be converted to string form by internally calling toString()
- + operator can be used to concatenate data from different types

System.out.print() : Example

```
class Test
{
        public static void main(String args[])
        {
                System.out.print("Hello ");
                System.out.print(“ I am fine");
                System.out.println(" It is OK");
                System.out.print(“Welcome”);
        }// End of Method
}// End of class Test
```

```
Hello I am fine It is OK
Welcome
```

Thank You