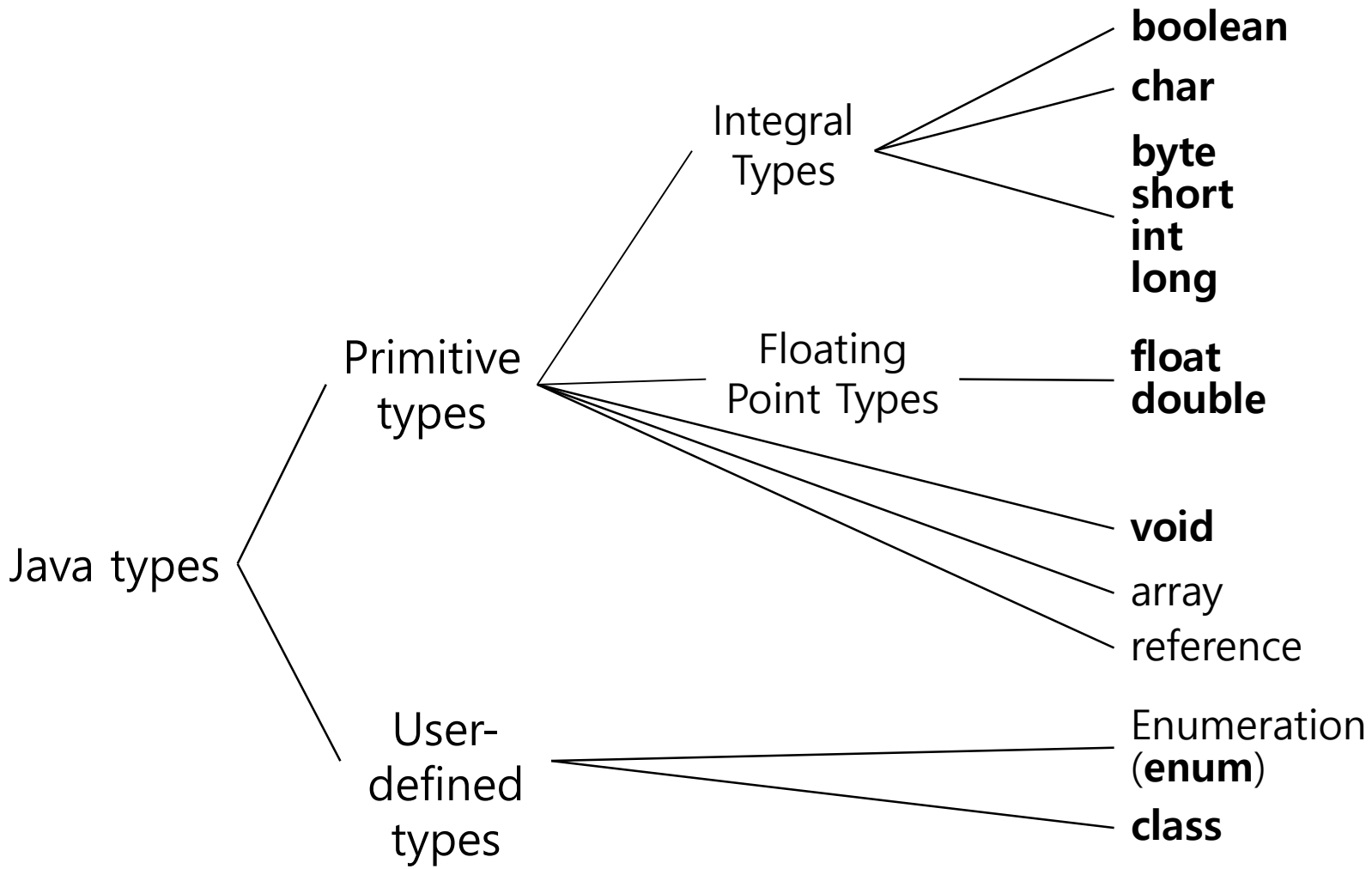
Java Types

- ❖Primitive Types
- ❖Operators
- ❖Wrapper Classes

- ❖String
- ❖Array
- ❖Constants

- ❖Inputs and Outputs
- ❖Date & Time
- ❖var

Java Types: Overview



Operators in Java

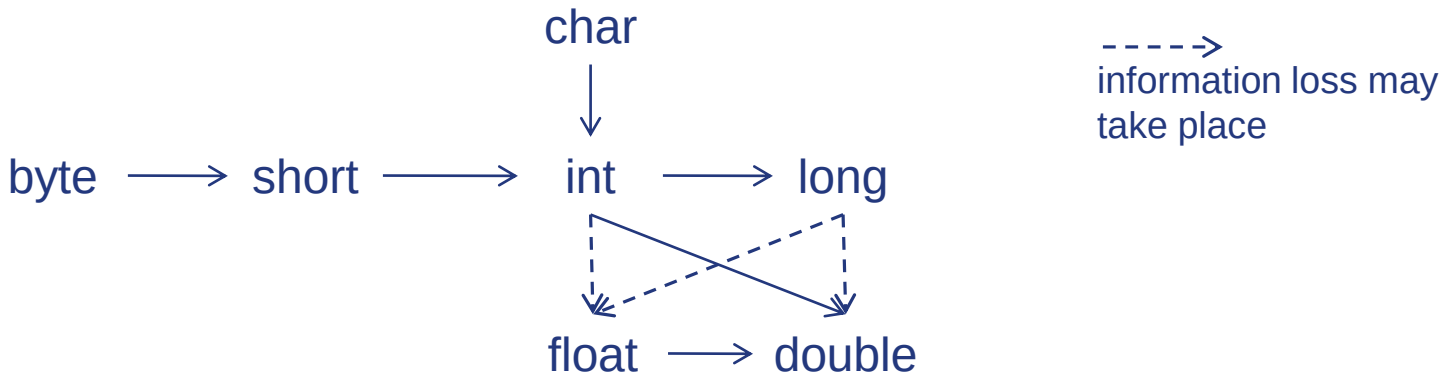
Categories	Operators
Assignment	=, +=, -=, *=, /=
Arithmetic	+, -, *, /, %, ++,, --
Equality and Relational Operators	==, !=, >, >=, <, <=
Conditional	&& ||, ? : (= if then else)
Type Comparison	instanceof
Bitwise and Bit Shift	&, ^, |, ~ <<, >>, >>>

Primitive Types

Category	Type	Bytes	Range(inclusive)	Literals
Integer	int	4	-2^{31} (=-2,147,483,648) to 2^{31}-1 (=2,147,483,647)	26 **0**32 **0x**1a
	short	2	-2^{15} (=-32768) to 2^{15}-1 (=32767)	
	long	8	-2^{63} to 2^{63}-1	26**L**
	byte	1	-2^{7}(=-128) to 2^{7}-1(=127)	
Floating-Point	float	4	approximately 3.403E+38 (6-7 significant decimal digits)	123.4**f** 123.4**F**
	double	8	approximately 1.198E+308 (15 significant decimal digits)	123.4
Character	char	2	Any character supported by Unicode	'A' '한' '₩u203B'
Truth value	boolean	1(?)		true, false

Type Conversion

- Impicit conversion: legal conversions between numeric types



- Explicit cast

```
double x = 9.997 ;
int nx1 = (int) x ;
Int nx2 = (int) Math.round(x) ;
```

Unicode

- Unicode is an encoding scheme for representing various characters including Alphabet, Chinese, Koreans, and Japanese.
- Java supports Unicode. That is, we can use all the characters supported by the Unicode.

```
public class Unicode {
    public static void main(String[] args) {
        // Korean
        System.out.print("안녕하세요! ") ;
        char[] korean = {'\uC790', '\uBC14'} ;
        System.out.println(korean) ;

        // Japanese
        char[] japanese = {'\u3051', '\u304F'} ;
        System.out.println(japanese) ;

        // Symbols
        char[] symbol = {'\u2020', '\u203B'} ;
        System.out.println(symbol) ;
    }
}
```

```
안녕하세요! 자바
け く
†※
```

- For the Unicode, visit http://unicode.org/charts/

Beyond Basic Arithmetic: Math Class

- The Math class provides methods and constants for doing more advanced mathematical computation.

```
public class MathExample {
   public static void main(String[] args) {
      System.out.println(Math.abs(-10)) ;
      System.out.println(Math.PI) ;
   }
}
```

Categories	methods
Basic math methods	abs, ceil, floor, round, min, max
Exponential and Logarithmic Methods	exp, log, pow, sqrt
Trigonometric methods	sin, cos, tan, asin, acos, atan
Random numbers (0.0 – 1.0)	random

WRAPPER CLASSES

Wrapper Classes

- Java supports wrapper classes for primitive numeric types
  - **int** intValue = 10;
  - **Integer** integerValue = intValue;

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

Conversion: Primitive Type → Wrapper

```
public class Conversion2Wrapper {
    public static void main(String[] args) {
        // 1. using constructor
        // The constructor Integer(int) is deprecated since version 9
        Integer integer1 = new Integer(10);
        System.out.println(integer1);

        // 2. using static factory method: valueOf()
        // The static factory valueOf(int) is generally a better choice,
        // as it is likely to yield significantly better space and time performance
        Integer integer2 = Integer.valueOf(20);
        System.out.println(integer2);

        System.out.println(integer1 + integer2);
    }
}
```

```
10
20
30
```

Conversion: Wrapper → Primitive Type

- Use xxxValue()

```
public class Conversion2PrimitiveType {

    public static void main(String[] args) {
        Integer integer1 = new Integer(10);
        int intValue = integer1.intValue();

        Integer integer2 = Integer.valueOf(118);
        short shortIntValue = integer2.shortValue();

        Integer integer3 = integer1 + integer2;
        long longIntValue = integer3.longValue();

        byte byteValue = integer3.byteValue(); // overflow
        System.out.println(byteValue); // -128, not 128
    }
}
```

Wrapper Classes: Autoboxing and Unboxing

- **Boxing**
  - Automatic conversion: primitive type ➔ wrapper class
  - Character characterValue = ‘A’;
  - Integer integerValue = 10;

- **Unboxing**
  - Automatic conversion: wrapper class ➔ primitive type
  - int intValue1 = integerValue;
  - char charValue = characterValue;

Auto Boxing

```
import java.util.ArrayList;
import java.util.List;

public class AutoBoxing {
    public static void main(String[] args) {
        List<Integer> integerList = new ArrayList<>();
        for (int i = 1; i < 10; i ++) {
            integerList.add(i);     // int to Integer; add(Integer.valueOf(i))
        }
        System.out.println(integerList); // [1, 2, 3, 4, 5, 6, 7, 8, 9]
    }
}
```

Auto Unboxing

```
public class AutoUnboxing {
    public static void main(String[] args) {
        List<Integer> integerList = new ArrayList<>();
        for ( int i = 1; i <= 5; i ++ )
          integerList.add(i); // auto boxing
       System.out.println(integerList);                // [1, 2, 3, 4, 5]

       int sumOfEven = 0;
       for ( Integer i: integerList ) {
            if ( i % 2 == 0 )                          // Integer to int
              sumOfEven += i ;                         // Integer to int
        }
       System.out.println(sumOfEven);                  // 6
    }
}
```

Wrapper Classes: Useful Features

- Wrapper classes provide useful variables.
  - MIN_VALUE, MAX_VALUE, SIZE for integer types
  - NEGATIVE_INFINITY, POSITIVE_INFINITY for floating-point types

```
…
byte b ;
if ( integerValue <= Byte.MAX_VALUE )
    b = integerValue.byteValue() ;
else
    b = 0 ;

Double d ;
if ( Double.isInfinite(d) ) …
if ( d.isInfinite() ) …

if ( Double.isNaN(d) ) …
if ( d.isNaN()) …
```

When to Use Wrapper Classes

- **Collections** in Java deal only with objects; to store a primitive type in one of these classes, you need to wrap the primitive type in a class.

  - `List<int> ints = new ArrayList<>();` // X
  - `List<Integer> integers = new ArrayList<>();` // O