PROGRAMMING IN JAVA

Lecture 2

Java Data Types

Primitive and Reference Types, Variables, Constants, Literals, and Type Conversion

Lecturer: Arshyn Altybay

Department of Computer Engineering,
Almaty Technological University, Almaty, Kazakhstan

1. Learning Objectives

By the end of this lecture, students should be able to:

- distinguish primitive and reference types;
- identify the eight primitive Java data types;
- choose suitable integer and floating-point types;
- declare, initialize, and update variables;
- declare constants using `final`;
- write decimal, binary, octal, hexadecimal, floating-point, character, string, and Boolean literals;
- explain widening and narrowing primitive conversions;
- use explicit casting when narrowing values;
- explain numeric promotion in expressions;
- convert between text and numeric values using parsing methods;
- explain wrapper classes and basic autoboxing/unboxing.

2. Data and Types in Java

Every value manipulated by a Java program has a type. A type determines what kind of value can be stored, how much information it represents, and which operations are meaningful.

Java types are commonly divided into two broad categories:

High-level type classification

```
Java Types
 |
 +-- Primitive types
 |    byte, short, int, long
 |    float, double
 |    char
 |    boolean
 |
 +-- Reference types
      String
      arrays
      classes
      interfaces
      other object types
```

3. Primitive Types

Java defines eight primitive types.

Type	Category	Typical use
`byte`	`integer`	small integer values, binary-oriented data
`short`	`integer`	relatively small integer values
`int`	`integer`	normal whole-number arithmetic; common default integer type
`long`	`integer`	large whole-number values
`float`	`floating point`	lower-precision decimal calculations
`double`	`floating point`	normal floating-point calculations; common default
`char`	`character`	one UTF-16 code unit
`boolean`	`logical`	`true` or `false`

4. Integer Types

Java's signed integer types have fixed sizes.

Type	Size	Approximate range
`byte`	`8 bits`	-128 to 127
`short`	`16 bits`	$-32\,768$ to $32\,767$
`int`	`32 bits`	about -2.1×10^{9} to 2.1×10^{9}
`long`	`64 bits`	about -9.22×10^{18} to 9.22×10^{18}

Example 1. Declaring integer variables

```
public class IntegerTypes {
    public static void main(String[] args) {
        byte small = 100;
        short count = 20_000;
        int population = 1_500_000;
        long worldValue = 8_000_000_000L;

        System.out.println(small);
        System.out.println(count);
        System.out.println(population);
        System.out.println(worldValue);
    }
}
```

Expected output

```
100
20000
1500000
8000000000
```

A `long` literal that does not fit into `int` should normally use the suffix `L`.

5. Floating-Point Types

Java provides `float` and `double` for approximate real-number calculation.

Example 2. Floating-point values

```
public class FloatingPoint {
    public static void main(String[] args) {
        float temperature = 23.5F;
        double pi = 3.141592653589793;
        double scientific = 6.02e23;

        System.out.println(temperature);
        System.out.println(pi);
        System.out.println(scientific);
    }
}
```

Expected output

```
23.5
3.141592653589793
6.02E23
```

Important: Binary floating-point types cannot represent every decimal fraction exactly. For example, values such as `0.1` are usually approximations in memory.

6. The char Type

A `char` stores one UTF-16 code unit and uses single quotation marks.

Example 3. Characters and numeric values

```
public class CharExample {
    public static void main(String[] args) {
        char letter = 'A';
        char digit = '7';
        char newline = '\n';

        System.out.println(letter);
        System.out.println((int) letter);
        System.out.println(digit);
        System.out.print("Java" + newline + "Programming");
    }
}
```

Expected output

```
A
65
7
Java
Programming
```

A `char` is not the same as a `String`. A character literal uses single quotes, while a string literal uses double quotes.

7. The boolean Type

A Boolean value is either `true` or `false`.

Example 4. Boolean variables

```
public class BooleanExample {
    public static void main(String[] args) {
        boolean javaIsPortable = true;
        boolean finished = false;

        System.out.println(javaIsPortable);
        System.out.println(finished);
    }
}
```

Expected output

```
true
false
```

Java does not use integers as Boolean values. Conditions require a Boolean expression.

8. Variables and Initialization

A variable declaration introduces a named storage location with a type.

```
int age;
double gpa;
String name;
```

Initialization supplies an initial value.

```
int age = 19;
double gpa = 3.8;
String name = "Aruzhan";
```

Local variables must be definitely assigned before they are read.

> **Common mistake:** Do not assume that an uninitialized local variable automatically contains zero. Java requires a local variable to receive a value before use.

9. Constants with `final`

A variable declared `final` can be assigned only once.

Example 5. Declaring constants

```
public class Constants {
    public static void main(String[] args) {
        final double PI = 3.141592653589793;
        final int MAX_STUDENTS = 30;

        System.out.println(PI);
        System.out.println(MAX_STUDENTS);
    }
}
```

Expected output

```
3.141592653589793
30
```

By convention, constant names are often written in uppercase with underscores.

10. Literals

A literal is a value written directly in source code.

10.1 Integer literals

```
int decimal = 42;
int binary = 0b101010;
int octal = 052;
int hexadecimal = 0x2A;
int readable = 1_000_000;
```

All five values above represent integers; the first four examples can represent the same numeric value using different bases.

10.2 Floating-point literals

```
double a = 12.5;
double b = 1.25e2;
float c = 12.5F;
```

10.3 Character, string, and Boolean literals

```
char grade = 'A';
String course = "Programming in Java";
boolean passed = true;
```

11. Reference Types

Reference variables do not directly contain the full object in the same sense as primitive variables contain primitive values. They contain reference values that can designate objects or be `null`.

```
String name = "Aruzhan";
int[] scores = {90, 80, 95};
```

> The distinction between an object and a reference variable becomes especially important in Lecture 6.

12. Type Inference with var

For local variables, Java can infer the declared type from the initializer.

```
var count = 10;          // inferred as int
var name = "Aruzhan";    // inferred as String
```

`var` does not make Java dynamically typed. The compiler still assigns one specific static type to the variable.

13. Widening Primitive Conversion

A widening conversion moves a value to a type that can represent a broader numeric range or precision category.

Example 6. Automatic widening

```
public class Widening {
    public static void main(String[] args) {
        int count = 25;
        long larger = count;
        double value = larger;

        System.out.println(larger);
        System.out.println(value);
    }
}
```

Expected output

```
25
25.0
```

Typical widening chains include:

Common widening directions

```
byte -> short -> int -> long -> float -> double
             char -> int -> long -> float -> double
```

14. Narrowing Conversion and Casting

A narrowing conversion may lose information and therefore normally requires an explicit cast.

Example 7. Explicit narrowing conversion

```
public class Narrowing {
    public static void main(String[] args) {
        double value = 19.75;
        int whole = (int) value;

        System.out.println(value);
        System.out.println(whole);
    }
}
```

Expected output

```
19.75
19
```

The fractional part is discarded when a floating-point value is cast to an integer type.

15. Numeric Promotion in Expressions

Small integer types such as `byte`, `short`, and `char` are typically promoted to `int` during arithmetic.

Example 8. Integer and floating-point division

```
public class PromotionExample {
    public static void main(String[] args) {
        int a = 5;
        int b = 2;

        System.out.println(a / b);
        System.out.println((double) a / b);
    }
}
```

Expected output

```
2
2.5
```

The operands determine the arithmetic. If both operands are integers, integer division occurs before the result is assigned to another type.

16. Parsing Text into Numbers

Input often arrives as text and must be converted.

Example 9. Parsing numeric text

```
public class ParsingExample {
    public static void main(String[] args) {
        String ageText = "19";
        String gpaText = "3.85";

        int age = Integer.parseInt(ageText);
        double gpa = Double.parseDouble(gpaText);

        System.out.println(age + 1);
        System.out.println(gpa);
    }
}
```

Expected output

```
20
3.85
```

Invalid numeric text causes a `NumberFormatException`, which will be studied in detail in the exception-handling lecture.

17. Wrapper Classes and Autoboxing

Each primitive type has a corresponding wrapper class.

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

Autoboxing converts a primitive value to a wrapper object automatically in an appropriate context; unboxing performs the reverse conversion.

Example 10. Autoboxing and unboxing

```
public class BoxingExample {
    public static void main(String[] args) {
        Integer boxed = 100;   // autoboxing
        int value = boxed;     // unboxing

        System.out.println(boxed);
        System.out.println(value + 1);
    }
}
```

Expected output

```
100
101
```

18. Comprehensive Example: Java Data Types

Example 11. Using several data types together

```
1  public class JavaDataTypesExample {
2      public static void main(String[] args) {
3          byte level = 4;
4          short year = 2026;
5          int students = 30;
6          long population = 20_000_000L;
```

Example 11 (continued)

```
7
8          float temperature = 24.5F;
9          double gpa = 3.875;
10
11         char grade = 'A';
12         boolean active = true;
```

Example 11 (continued)

```
13
14         String course = "Programming in Java";
15         final int MAX_SCORE = 100;
16
17         int binary = 0b1010;
18         int hex = 0xFF;
```

Example 11 (continued)

```
19
20         double widened = students;
21         int narrowed = (int) gpa;
22
23         String numberText = "42";
24         int parsed = Integer.parseInt(numberText);
```

Example 11 (continued)

```
25
26         Integer boxed = parsed;
27         int unboxed = boxed;
28
29         System.out.println(course);
30         System.out.println("Level = " + level);
```

Example 11 (continued)

```
31         System.out.println("Year = " + year);
32         System.out.println("Students = " + students);
33         System.out.println("Population = " + population);
34         System.out.println("Temperature = " + temperature);
35         System.out.println("GPA = " + gpa);
36         System.out.println("Grade = " + grade);
```

Example 11 (continued)

19. Common Errors and Good Practices

- Do not use an uninitialized local variable.
- Use `L` for large `long` literals and `F` for `float` literals where required.
- Remember that integer division discards the fractional part.
- Narrow only when the possible information loss is understood.
- Use underscores to improve readability of long numeric literals.
- Use `final` for values that should not change.
- Choose `double` rather than `float` for general floating-point work unless there is a specific reason not to.
- Do not confuse `'A'` with `"A"`.

20. Lecture Summary

- Java has eight primitive types and many reference types.
- `int` and `double` are common default numeric choices.
- Variables have types; constants can be declared with `final`.
- Literals can be written in several numeric bases and formats.
- Widening conversions are often automatic; narrowing conversions normally require casting.
- Arithmetic expressions follow numeric promotion rules.
- Parsing converts numeric text to primitive values.
- Wrapper classes allow primitive-like values to participate in object-oriented APIs.

21. Review Questions

1. What are the two broad categories of Java types?
2. Name all eight primitive types.
3. What is the difference between `int` and `long`?
4. Why is `double` often preferred to `float`?
5. What does a `char` store?
6. What values can a `boolean` hold?
7. What is initialization?
8. What does `final` mean for a variable?
9. What is a literal?
10. How is a binary integer literal written?
11. What is widening conversion?
12. What is narrowing conversion?
13. Why can casting lose information?
14. What is numeric promotion?
15. What is the result of `5 / 2` when both operands are integers?
16. What is parsing?
17. What is a wrapper class?
18. What are autoboxing and unboxing?

22. Programming Exercises

1. Declare one variable of each primitive type and print all values.
2. Create constants for the number of months in a year and the value of π.
3. Write the decimal number 31 in binary, octal, and hexadecimal literal form.
4. Convert an `int` value into `double`.
5. Cast a `double` value into `int` and observe the result.
6. Demonstrate integer division and floating-point division.
7. Convert a `char` to its integer code value.
8. Parse an integer from a `String`.
9. Parse a `double` from a `String`.
10. Demonstrate autoboxing and unboxing with `Integer`.
11. Calculate the area of a circle using a `final` constant.
12. Store a student's name, age, GPA, and active status using suitable types.

23. Homework

Create a program called `StudentDataCard`. It must:

- declare variables for student ID, name, age, GPA, grade, and active status;
- use at least five different Java data types;
- declare at least two constants using `final`;
- demonstrate one widening conversion and one narrowing conversion;
- parse at least one number from text;
- print a clearly formatted student-information report.

24. Key Terms

Term	Meaning
primitive type	built-in Java type whose variable stores a primitive value
reference type	type whose variable stores a reference value designating an object or `null`
variable	named storage location with a declared or inferred type
initialization	assignment of an initial value to a variable
constant	value represented by a variable that is not reassigned after initialization
literal	value written directly in source code
widening conversion	conversion to a broader primitive numeric type
narrowing conversion	conversion to a narrower primitive numeric type, often requiring a cast
casting	explicit request to convert a value to another compatible type
wrapper class	object type corresponding to a primitive type
autoboxing	automatic primitive-to-wrapper conversion
unboxing	automatic wrapper-to-primitive conversion