FARABI UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

Lecture 2.
Introduction to Java

Language fundamentals, objects, classes and input/output

Dr. Bazargul Matkerim, Professor-Researcher

Lecture goals and plan

Learning outcomes

Explain how Java source code runs.

Use types, control flow and methods.

Connect a UML class to Java code.

Handle text, arrays and basic I/O.

Lecture plan

1. Java execution and syntax

2. Values, methods and references

3. Strings and class design

4. Arrays, input and files

5. Review and reading

From Java source to execution



What makes Java portable?

A compatible JVM loads and executes bytecode on the target platform.
The runtime may interpret bytecode and compile frequently executed code to native instructions.

A first class and the main method

```
public class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello, Java!");
  }
}

// Compile, then run in a terminal:
// javac HelloWorld.java
// java HelloWorld
```

Reading the code

public: accessible entry point

static: no object needed

void: no return value

args: command-line strings

Packages organize types; imports let you refer to types by their simple names.

Primitive types and initialization

Type	Representation	Example
byte / short	8-bit / 16-bit signed integer	byte b = 12;
int / long	32-bit / 64-bit signed integer	long count = 42L;
float / double	32-bit / 64-bit floating point	double rate = 0.25;
char	16-bit UTF-16 code unit	char grade = 'A';
boolean	true or false	boolean valid = true;

Fields and array elements receive default values: 0, false or null.
Local variables must be definitely assigned before use.

Operators and numeric conversions

```
int whole = 5 / 2;       // 2
double fraction = 5.0 / 2; // 2.5

double widened = 7;      // 7.0
int truncated = (int) 3.9; // 3

boolean ready = age >= 18 && active;
int result = (a + b) * c;
```

Key rules

Use parentheses to make intent clear.

&& and || short-circuit.

Widening can lose precision.

Narrowing needs a cast and may lose data.

Control flow: selection and repetition

```
if (score >= 50) {
  System.out.println("Pass");
} else {
  System.out.println("Try again");
}

for (int i = 0; i < 3; i++) {
  System.out.println(i);
}
```

Choose the structure

if / else: a Boolean decision

switch: choose among cases

while: test before repeating

do / while: run at least once

for: initialization, test, update

Methods and the Math API

```
static double square(double x) {
  return x * x;
}

double area = square(4.0); // 16.0
double root = Math.sqrt(81.0); // 9.0
int larger = Math.max(7, 12); // 12
double circle = Math.PI * square(3.0);
```

Method contract

Parameters describe inputs.

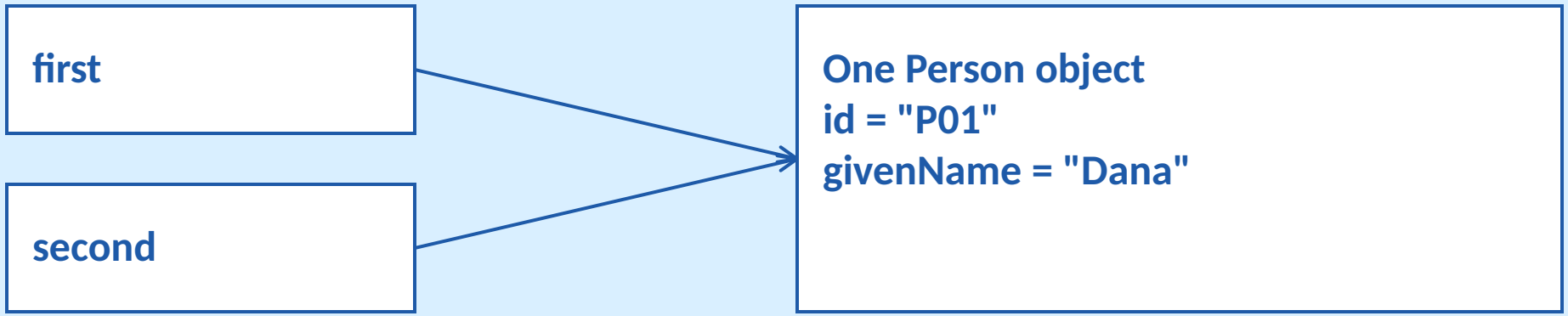
The return type describes the result.

static methods belong to the class.

Instance methods act through an object.

References and pass-by-value

```
Person first = new Person("P01", "Aida");
Person second = first;
second.setGivenName("Dana");
// first.getGivenName() is now "Dana"
```



Java copies argument values. For an object argument, that value is a reference.

Strings: values, equality and escapes

```
String a = new String("Java");
String b = new String("Java");

boolean same = a == b; // false: identity
boolean equal = a.equals(b); // true: content
int order = a.compareTo("Kotlin"); // negative

String upper = a.toUpperCase();
// a remains "Java"
```

String is immutable

Operations return values; they do not change the original String.

Common escapes:
\n new line
\t tab
\" quotation mark
\\ backslash

Building and splitting text

```
StringBuilder report = new StringBuilder();
report.append("Student: ").append("Aida");
report.append(" | Score: ").append(95);
String result = report.toString();

String line = "Java   UML  design";
String[] words = line.trim().split("\\s+");
```

Choose the API

StringBuilder: mutable text for local use.

StringBuffer: synchronized methods.

split: regex delimiters.

StringTokenizer: legacy API.

From a UML class to Java

Person

-id: String {readOnly}
-givenName: String

+Person(id: String, name: String)
+getGivenName(): String
+setGivenName(name: String): void
+equals(other: Object): boolean
+hashCode(): int

Visibility and responsibility

– private: internal state

+ public: accessible operations

protected: package/subclass access

~ package: within the package

Constructors and encapsulated state

```
public final class Person {
  private final String id;
  private String givenName;

  public Person(String id, String name) {
    this.id = java.util.Objects.requireNonNull(id);
    setGivenName(name);
  }
  public String getGivenName() { return givenName; }
  public void setGivenName(String name) {
    givenName = java.util.Objects.requireNonNull(name);
  }
}
```

The constructor establishes valid state; methods preserve it.

Equality, hashing and readable output

```
@Override
public boolean equals(Object other) {
  if (this == other) return true;
  if (!(other instanceof Person)) return false;
  Person p = (Person) other;
  return id.equals(p.id);
}
@Override
public int hashCode() { return id.hashCode(); }
@Override
public String toString() {
  return id + ": " + givenName;
}
```

Object contract

Equality uses the immutable id.

Equal objects must have equal hash codes.

equals(Object) overrides Object.

equals(Person) only overloads it.

Arrays and safe traversal

```
double[] grades = {78.0, 91.5, 85.0};
double total = 0.0;
for (double grade : grades) {
  total += grade;
}
if (grades.length > 0) {
  double average = total / grades.length;
  System.out.printf("Average: %.2f%n", average);
}
```

Array essentials

Fixed length.

Indexes: 0 … length − 1.

Need the index? Use a counted loop.

Check for empty input.

Wrappers, parsing and formatted output

```
int count = Integer.parseInt("42");
double price = Double.parseDouble("12.50");

Integer boxed = count;   // autoboxing
int unboxed = boxed;    // unboxing

System.out.println("Count: " + count);
System.out.printf("Price: %.2f%n", price);
```

Value conversion

Wrappers represent primitive values as objects.

Reject invalid numeric input.

Never unbox a null wrapper.

Dialog input: cancellation and errors

```
String input = JOptionPane.showInputDialog("Age?");
if (input != null) {
  try {
    int age = Integer.parseInt(input.trim());
    JOptionPane.showMessageDialog(null, "Age: " + age);
  } catch (NumberFormatException ex) {
    JOptionPane.showMessageDialog(null,
        "Enter a whole number.");
  }
}
```

Import javax.swing.JOptionPane.
showInputDialog may return null; showMessageDialog returns void.

Text files and resource management

```
Path file = Path.of("students.txt");
try (BufferedReader reader = Files.newBufferedReader(
        file, StandardCharsets.UTF_8)) {
  String line;
  while ((line = reader.readLine()) != null) {
    if (!line.isBlank()) {
      String[] words = line.trim().split("\\s+");
      System.out.println(words[0]);
    }
  }
}
```

Byte data: InputStream / OutputStream. Text: Reader / Writer.
try-with-resources closes the reader, including when an exception occurs.

Review and classroom practice

Check your understanding

1. What does the JVM execute?
2. Why can a local variable be uninitialized?
3. How do == and equals differ?
4. What changes when a reference is passed?
5. Why override hashCode with equals?

Apply the lecture

Implement Person and create two objects with the same id.

Compare identity and logical equality.

Read names from a UTF-8 text file.

Test blank input, null references and an empty array.

Reading and references

Core reading

Horstmann, C. S. (2025). Core Java, Volume I: Fundamentals, 14th ed. Oracle Press. Chapters 3–5. ISBN 9780135558577.

Specifications and API reference

Oracle (2025). The Java Language Specification, Java SE 25.
Oracle (2025). Java SE 25 API Specification.
Object Management Group (2017). UML, Version 2.5.1.

Full links and source-slide mapping are included in the speaker notes.