PROGRAMMING IN JAVA

Lecture 6

Classes, Objects, and the Java Object Model

Classes, Objects, References, Fields, Methods, Constructors, `this`, Object Arrays, and Memory Concepts

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:

- explain the relationship between a class and an object;
- distinguish an object from a reference variable;
- define fields, constructors, and instance methods;
- explain object state, behavior, and identity;
- use the `this` reference;
- create multiple objects from one class;
- explain reference assignment and aliasing;
- identify and avoid `null`-reference errors;
- explain Java's conceptual stack/heap model carefully;
- pass object references to methods using Java's pass-by-value rule;
- create arrays of object references;
- model simple relationships between objects;
- explain object reachability and the role of garbage collection.

2. From Procedures to Objects

Earlier programs concentrated on variables, arrays, control structures, and static methods. Object-oriented programming groups related data and behavior inside classes.

For example, a student has data such as:

- ID;
- name;
- GPA.

A student can also perform or participate in behavior such as displaying information or updating a GPA.

A class defines this structure; an object is a concrete instance created from that class.

3. Class, Object, and Reference Variable

Consider:

```
Student s = new Student();
```

Three ideas appear in this statement:

- `Student` is a class/type;
- `new Student()` creates an object;
- `s` is a reference variable containing a reference value that designates that object.

> **Precise wording:** The reference variable is not the object itself. It stores a reference value that can designate an object or be `null`.

4. Anatomy of a Simple Class

Example 1. A minimal `Student` class

```
public class Student {

    String name;
    double gpa;

    void displayInfo() {
        System.out.println(name + " " + gpa);
    }

    public static void main(String[] args) {
        Student student = new Student();

        student.name = "Aruzhan";
        student.gpa = 3.8;

        student.displayInfo();
    }
}
```

Expected output

```
Aruzhan 3.8
```

The fields store object state. The instance method operates on an object.

5. Fields and Local Variables

A field is declared in a class body and belongs to an object or to the class, depending on whether it is static.

A local variable is declared inside a method or block.

Property	Field	Local variable
declared in	class body	method/block
lifetime	associated with object/class lifetime	block/method execution
default initialization	yes for fields	no; must be definitely assigned before use
accessed through	object/class context	local name in scope

6. Object State, Behavior, and Identity

An object can be understood through three ideas:

- **state**: current field values;
- **behavior**: operations defined by methods;
- **identity**: the fact that two distinct objects can have equal field values but still be different objects.

Example 2. Two different objects with similar state

```
public class Book {

    String title;

    public static void main(String[] args) {
        Book a = new Book();
        Book b = new Book();

        a.title = "Java";
        b.title = "Java";

        System.out.println(a == b);
    }
}
```

Expected output

```
false
```

7. Constructors

A constructor initializes a newly created object.

A constructor:

- has the same name as the class;
- has no return type;
- is invoked during object creation.

Example 3. Initializing an object with a constructor

```java
public class Student {

    String name;
    double gpa;

    Student(String name, double gpa) {
        this.name = name;
        this.gpa = gpa;
    }

    void displayInfo() {
        System.out.println(name + " " + gpa);
    }

    public static void main(String[] args) {
        Student s = new Student("Aruzhan", 3.9);
        s.displayInfo();
    }
}
```

Expected output

```
Aruzhan 3.9
```

If a class declares no constructor at all, the compiler supplies a default no-argument constructor. Once any constructor is declared explicitly, that compiler-supplied constructor is not added automatically.

8. The `this` Reference

Inside an instance method or constructor, `this` refers to the current object.

A common use is distinguishing fields from parameters.

```
Student(String name, double gpa) {
    this.name = name;
    this.gpa = gpa;
}
```

Here, `this.name` means the field of the current object, while `name` means the constructor parameter.

9. Multiple Objects of the Same Class

One class can create many independent objects.

Example 4. Independent object state

```
public class Counter {

    int value;

    void increment() {
        value++;
    }

    public static void main(String[] args) {
        Counter a = new Counter();
        Counter b = new Counter();

        a.increment();
        a.increment();
        b.increment();

        System.out.println(a.value);
        System.out.println(b.value);
    }
}
```

Expected output

```
2
1
```

10. References and Aliasing

Assigning one reference variable to another copies the reference value; it does not clone the object.

Example 5. Two variables referring to one object

```
public class AliasingExample {

    static class Student {
        String name;
    }

    public static void main(String[] args) {
        Student first = new Student();
        first.name = "Aruzhan";

        Student second = first;
        second.name = "Dana";

        System.out.println(first.name);
        System.out.println(first == second);
    }
}
```

Expected output

```
Dana
true
```

This situation is called aliasing.

11. `null` References

A reference variable can hold `null`, meaning that it does not designate an object.

```
Student student = null;
```

Trying to use an instance member through `null` causes a `NullPointerException`.

> **Common error:** Creating a reference variable does not automatically create an object. `Student s;` and `Student s = new Student();` are not equivalent.

12. A Conceptual Memory Model

For learning purposes, it is useful to picture local variables and references separately from dynamically created objects.

Conceptual model

```
Method-call context                    Objects
-------------------                    -------
student  ----------------------------> Student object
                                        name = "Aruzhan"
                                        gpa  = 3.9
```

This is a conceptual model, not a complete guarantee of how every JVM physically lays out memory. JVM implementations may optimize storage in sophisticated ways.

13. Instance Methods Operate on the Current Object

When:

```
student.displayInfo();
```

is executed, the method operates on the object designated by `student`. Inside the method, `this` denotes that current object.

14. Passing Objects to Methods

Java still uses pass-by-value. For an object variable, the copied value is a reference value.

Example 6. A method can modify the same object

```
public class ObjectParameter {

    static class Student {
        double gpa;
    }

    static void increaseGpa(Student s) {
        s.gpa += 0.1;
    }

    public static void main(String[] args) {
        Student student = new Student();
        student.gpa = 3.5;

        increaseGpa(student);

        System.out.println(student.gpa);
    }
}
```

Expected output

```
3.6
```

The parameter receives a copy of the reference value. The copied reference still designates the same object.

15. Arrays of Objects

Creating an array of a reference type creates reference slots, not the objects themselves.

```
Student[] students = new Student[3];
```

Initially, the three elements are `null`.

Objects must be created separately:

```
students[0] = new Student("Aruzhan", 3.9);
students[1] = new Student("Dias", 3.5);
students[2] = new Student("Ali", 3.7);
```

16. Objects Can Refer to Other Objects

Object relationships are created by storing references in fields.

```
class Course {
    String title;
    Teacher teacher;
}
```

A `Course` object can therefore designate a `Teacher` object.

This is the foundation of composition and association in object-oriented design.

17. Object Reachability and Garbage Collection

Java automatically manages memory for objects. An object may become eligible for garbage collection when it is no longer reachable through live references.

Reference removed

```
Before:
student  ----------> Student object

After:
student = null

No remaining live reference
          X-------> Student object
```

Garbage collection is automatic, but its exact timing is not normally controlled by application code.

> Automatic memory management does not replace explicit resource management for files, sockets, or database connections. Those resources should be closed using the appropriate APIs.

18. Comprehensive Example: Small Student Registry

Example 7. Objects, constructors, methods, and an object array

```
public class StudentRegistry {

    static class Student {
        int id;
        String name;
        double gpa;

        Student(int id, String name, double gpa) {
            this.id = id;
            this.name = name;
            this.gpa = gpa;
        }

        void display() {
            System.out.println(
                    id + " " + name + " " + gpa
            );
        }
    }

    static Student bestStudent(Student[] students) {
        Student best = students[0];

        for (Student student : students) {
            if (student.gpa > best.gpa) {
                best = student;
            }
        }

        return best;
    }

    public static void main(String[] args) {
        Student[] students = {
            new Student(1, "Aruzhan", 3.9),
            new Student(2, "Dias", 3.5),
            new Student(3, "Ali", 3.7)
        };

        for (Student student : students) {
            student.display();
```

Example 7 (continued)

```
        }

        Student best = bestStudent(students);

        System.out.println(
```

19. Common Errors and Good Practices

- Do not confuse a class with an object.
- Do not confuse an object with a reference variable.
- Remember that reference assignment can create aliases.
- Check whether a reference can be `null` before dereferencing it.
- Use constructors to establish valid initial object state.
- Keep fields and behavior conceptually related.
- Remember that object arguments are still passed by value.
- When creating an object array, create the individual objects as well.

20. Mental Model to Remember

Object-oriented mental model

```
Class
  |
  +-- defines fields and methods
  |
  +-- constructor initializes new objects
  |
  v
Object
  |
  +-- has its own state
  +-- responds to instance methods
  ^
  |
Reference variable stores a reference value
that can designate the object
```

21. Lecture Summary

- A class defines a type; an object is an instance of that class.
- A reference variable stores a reference value, not the full object.
- Fields represent object state; instance methods represent behavior.
- Constructors initialize newly created objects.
- `this` refers to the current object.
- Reference assignment may create aliasing.
- `null` means a reference currently designates no object.
- Java passes reference values by value when object variables are arguments.
- An object array initially contains reference slots, normally `null`.
- Object relationships are represented by references stored in fields.
- Unreachable objects may be reclaimed automatically by garbage collection.

22. Review Questions

1. What is a class?
2. What is an object?
3. What is a reference variable?
4. What is the difference between a field and a local variable?
5. What are object state and behavior?
6. What is object identity?
7. What is a constructor?
8. When does the compiler supply a default no-argument constructor?
9. What does `this` mean?
10. What is aliasing?
11. What does `null` mean?
12. What causes a `NullPointerException`?
13. Is Java pass-by-value when objects are involved?
14. What value is copied when an object variable is passed to a method?
15. What does `new Student[3]` create?
16. Why are the elements of a newly created `Student[]` initially `null`?
17. How can one object refer to another object?
18. What does object reachability mean?

23. Programming Exercises

1. Create a `Book` class with title and price fields.
2. Add a constructor to initialize a `Book`.
3. Add an instance method that displays book information.
4. Create three `Book` objects.
5. Demonstrate two independent objects with different field values.
6. Demonstrate aliasing using two references to one object.
7. Demonstrate a `null` reference safely.
8. Create a method that receives an object and modifies one field.
9. Create an array of five `Student` references and initialize all objects.
10. Find the object with the largest numeric field in an object array.
11. Create a `Course` object that contains a reference to a `Teacher`.
12. Build a small registry using objects and object arrays.

24. Homework

Develop a small University Course Registry using at least these classes:

- `Student`;
- `Teacher`;
- `Course`.

Requirements:

- use constructors to initialize objects;
- create arrays of objects;
- give `Course` a reference to a `Teacher`;
- store several student references;
- write a method that finds the highest-GPA student;
- demonstrate the difference between aliasing and creating a separate object;
- print a clear final report.

25. Key Terms

Term	Meaning
class	definition of a Java type containing fields, methods, and constructors
object	runtime instance of a class
reference variable	variable containing a reference value that can designate an object
field	variable declared as part of a class
instance method	method that operates in the context of a current object
constructor	special member used to initialize new objects
`this`	reference to the current object
aliasing	situation where multiple reference variables designate the same object
`null`	special reference value designating no object
reachability	whether an object can be reached through live references
garbage collection	automatic reclamation of storage associated with unreachable objects