PT821: Object-Oriented Programming

Lecture 3: Classes and Objects in Java

Masoud Hamad

State University of Zanzibar
BITA Second Year - Semester 1

Lecture 3

Outline

Quick Recap: From Design to Implementation

What We Learned in OOD:

- Identifying objects from problem statements
- Noun identification technique
- CRC Cards (Class-Responsibility-Collaboration)
- Relationships: Association, Composition, Aggregation, Inheritance

Today's Focus:

- Implementing classes in Java
- Creating and using objects
- Constructors
- Methods and parameters
- Access modifiers

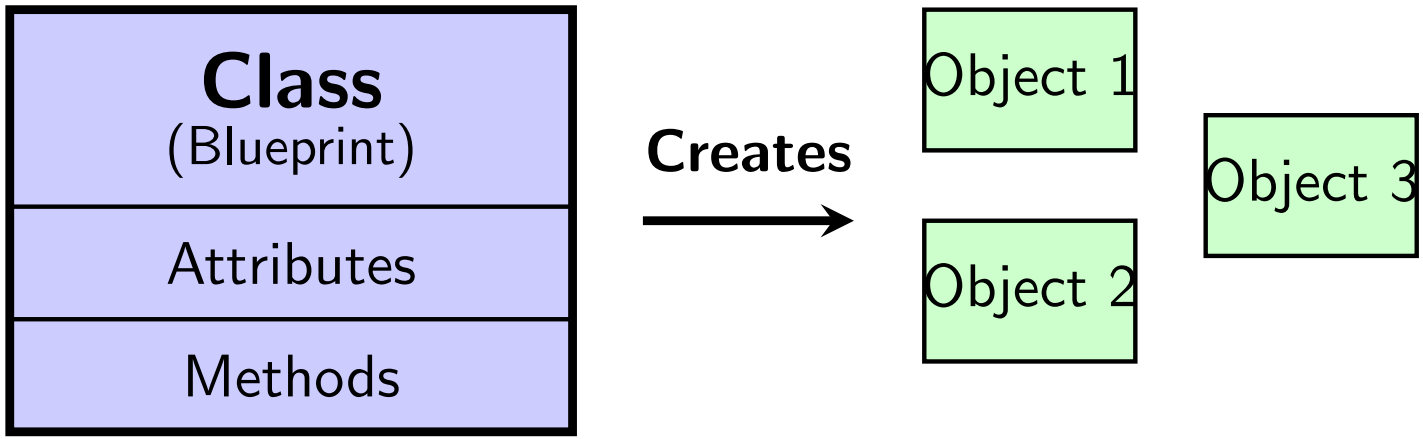
From Design to Code

Today we translate our OOD blueprints into working Java code!

What is a Class?

Definition

A **class** is a blueprint or template that defines the structure and behavior of objects. It specifies what data (attributes) and what operations (methods) objects of that type will have.

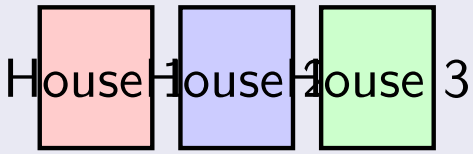


Real-World Analogy: Class vs Object

Class = Blueprint

- Defines structure
- Not a real house
- Template for building
- One plan, many houses

Object = Actual Instance



- Real, tangible
- Has actual values
- Occupies memory
- Each is unique

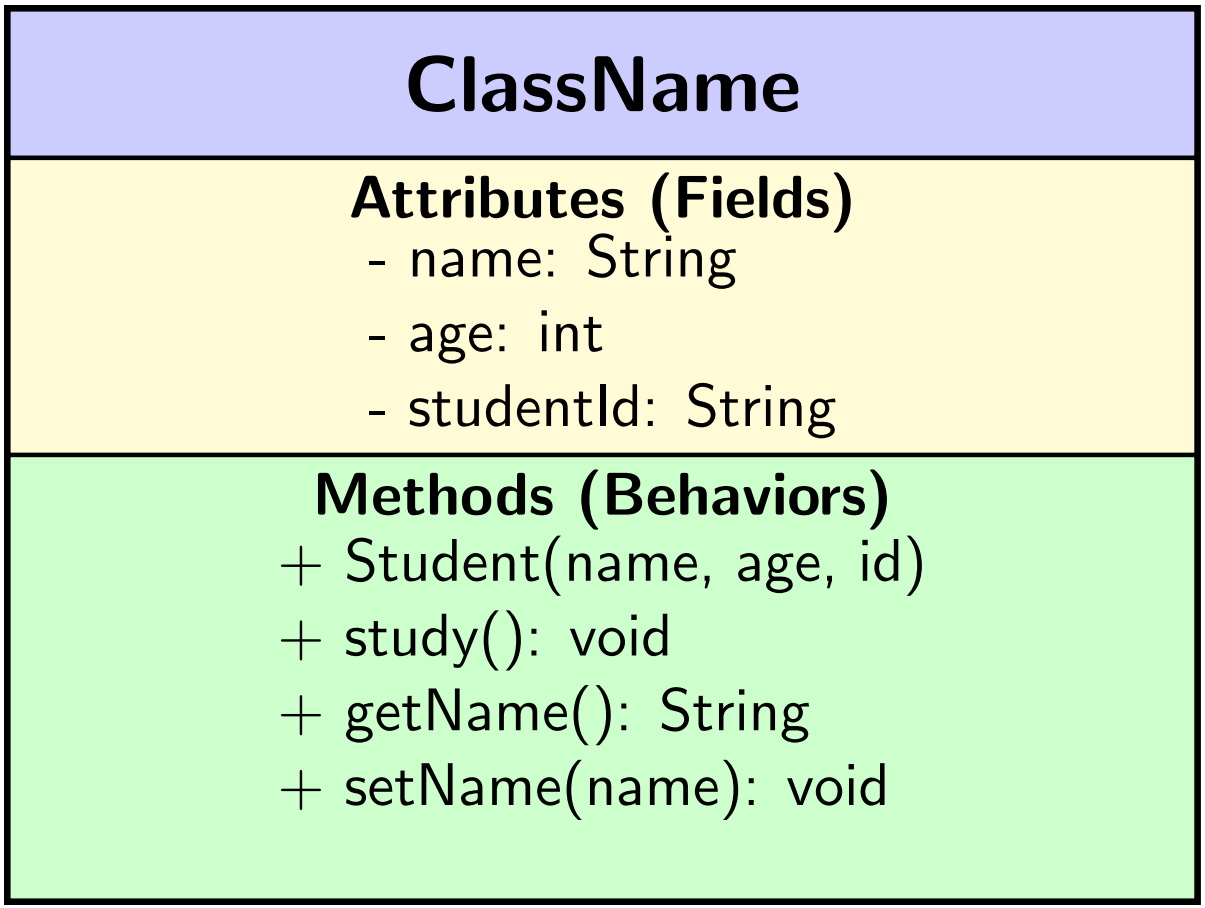
Example

Class: `Student` (defines name, age, study())
Objects: Ali (name="Ali", age=20), Fatma (name="Fatma", age=21)

Anatomy of a Java Class

```
public class Student {
    // 1. ATTRIBUTES (Instance Variables/Fields)
    private String name;
    private int age;
    private String studentId;

    // 2. CONSTRUCTOR(S)
    public Student(String name, int age, String
        studentId) {
        this.name = name;
        this.age = age;
        this.studentId = studentId;
    }

    // 3. METHODS (Behaviors)
    public void study() {
        System.out.println(name + " is studying.");
    }
```

Components of a Class



What are Attributes?

Definition

Attributes (also called instance variables or fields) are variables declared inside a class but outside any method. They represent the state or data of an object.

Characteristics:

- Declared at class level
- Each object has its own copy
- Can have different values per object
- Should be `private` (encapsulation)
- Accessed via getters/setters

Common Data Types:

- `int`, `double`, `boolean`
- `String`
- `char`
- Arrays
- Other objects (composition)

Declaring Attributes

```
public class BankAccount {
    // Attributes with different data types
    private String accountNumber;     // Text
    private String ownerName;         // Text
    private double balance;           // Decimal number
    private boolean isActive;         // True/False
    private int transactionCount;     // Whole number

    // Default values if not initialized:
    // String -> null
    // int -> 0
    // double -> 0.0
    // boolean -> false
}
```

Best Practice

Always declare attributes as `private` and provide public getter/setter methods to access them. This is called **encapsulation**.

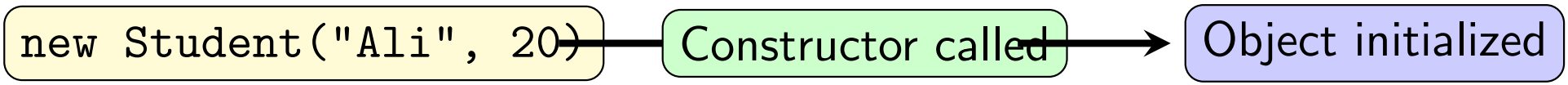
What is a Constructor?

Definition

A **constructor** is a special method that is called automatically when an object is created. It initializes the object's attributes with initial values.

Key Characteristics:

- Same name as the class
- No return type (not even `void`)
- Called automatically with `new` keyword
- Can be overloaded (multiple constructors)
- If not defined, Java provides a default constructor



Types of Constructors

```
public class Student {
    private String name;
    private int age;

    // 1. DEFAULT CONSTRUCTOR (no parameters)
    public Student() {
        name = "Unknown";
        age = 0;
    }

    // 2. PARAMETERIZED CONSTRUCTOR
    public Student(String name, int age) {
        this.name = name;  // 'this' refers to current
            object
        this.age = age;
    }

    // 3. COPY CONSTRUCTOR (creates copy of another
        object)
```

The 'this' Keyword

Definition

The `this` keyword refers to the **current object** - the object whose method or constructor is being called.

```
public class Student {
    private String name;

    public Student(String name) {
        // 'this.name' = attribute of this object
        // 'name' = parameter passed to constructor
        this.name = name;
    }

    public void setName(String name) {
        this.name = name;  // Distinguish attribute
            from parameter
    }
}
```

Constructor Overloading

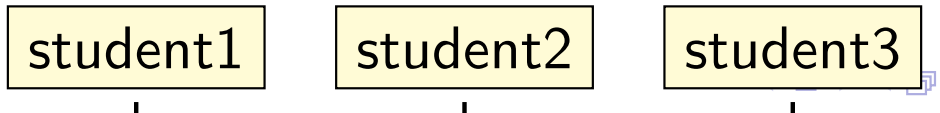
Multiple constructors with different parameters:

```
public class Book {
    private String title;
    private String author;
    private int pages;

    // Constructor 1: All parameters
    public Book(String title, String author, int pages
        ) {
        this.title = title;
        this.author = author;
        this.pages = pages;
    }

    // Constructor 2: Title and author only
    public Book(String title, String author) {
        this(title, author, 0);  // Calls constructor
            1
    }
```

Creating Objects

Syntax: `ClassName objectName = new ClassName(arguments);`

```
public class Main {
    public static void main(String[] args) {
        // Creating objects using different
            constructors
        Student student1 = new Student();  // Default
            constructor
        Student student2 = new Student("Ali", 20);  //
            Parameterized
        Student student3 = new Student(student2);   //
            Copy constructor

        // Each object is independent
        // Changes to student2 don't affect student3
    }
}
```

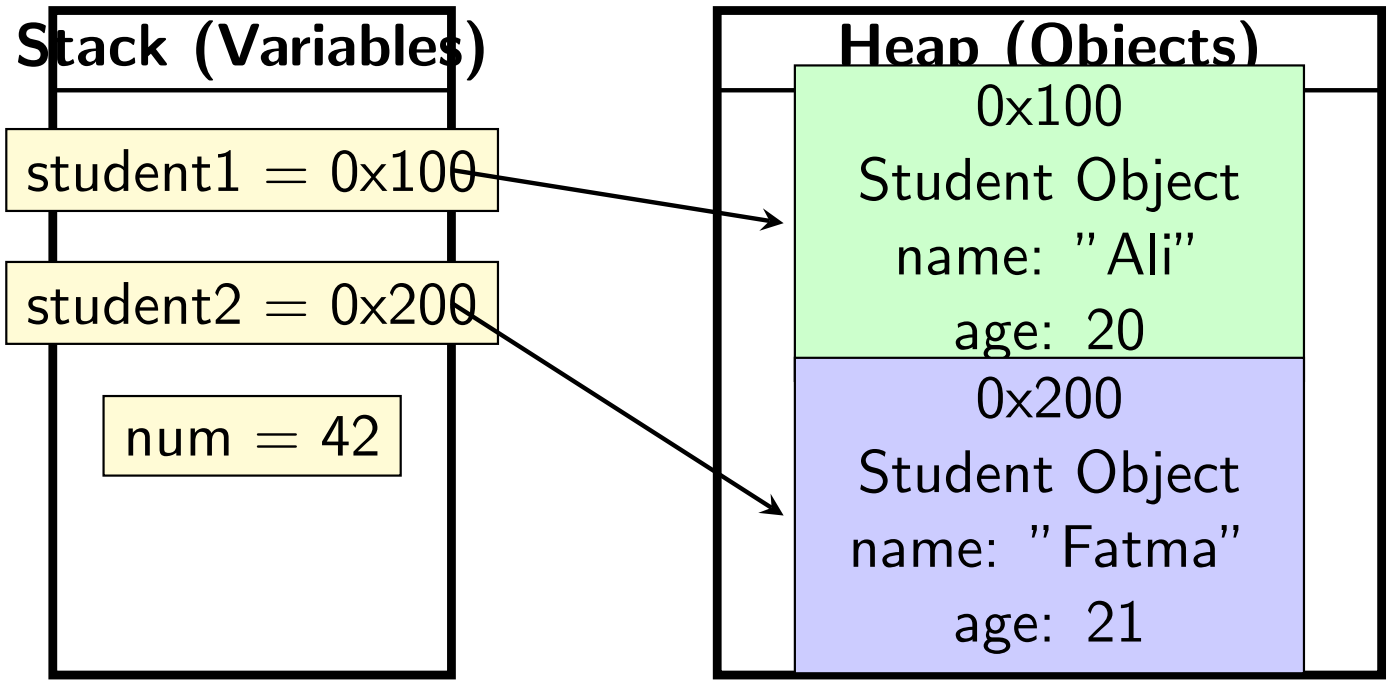


Accessing Object Members

Using the dot (.) operator:

```
public class Main {
    public static void main(String[] args) {
        Student student = new Student("Fatma", 21);

        // Calling methods on the object
        student.study();                    // Fatma is
            studying.

        String name = student.getName();   // Get name
        System.out.println(name);           // Fatma

        student.setAge(22);                 // Set new age

        // Multiple objects
        Student s1 = new Student("Ali", 20);
        Student s2 = new Student("Hassan", 19);

        s1.study();   // Ali is studying.
```

Object Memory Model



- **Stack**: Stores references (addresses) to objects
- **Heap**: Stores actual object data
- Primitive types (int, double, etc.) are stored directly in stack

What are Methods?

Definition

Methods define the behaviors or actions that objects of a class can perform. They are functions defined inside a class.

Method Signature:

```
accessModifier returnType methodName(parameters)
```

Types of Methods:

- Instance methods
- Static methods
- Getter methods
- Setter methods
- Constructor methods

Return Types:

- `void` - returns nothing
- `int`, `double`, etc. - primitives
- `String` - returns text
- `ClassName` - returns object
- `boolean` - returns true/false

Defining Methods

```
public class Calculator {
    // Method with no parameters, no return
    public void displayWelcome() {
        System.out.println("Welcome to Calculator!");
    }

    // Method with parameters and return value
    public int add(int a, int b) {
        return a + b;
    }

    // Method with multiple parameters
    public double calculateAverage(double[] numbers) {
        double sum = 0;
        for (double num : numbers) {
            sum += num;
        }
        return sum / numbers.length;
    }
```

Getters and Setters

Purpose: Control access to private attributes (encapsulation)

```
public class Person {
    private String name;
    private int age;

    // GETTER - retrieves the value
    public String getName() {
        return name;
    }

    // SETTER - modifies the value (with validation)
    public void setName(String name) {
        if (name != null && !name.isEmpty()) {
            this.name = name;
        }
    }

    public int getAge() {
        return age;
```

Method Overloading

Definition

Method Overloading: Multiple methods with the same name but different parameters (different number, type, or order of parameters).

```
public class Printer {
    // Overloaded print methods
    public void print(String message) {
        System.out.println("String: " + message);
    }

    public void print(int number) {
        System.out.println("Integer: " + number);
    }

    public void print(String message, int times) {
        for (int i = 0; i < times; i++) {
            System.out.println(message);
        }
```

Access Modifiers Overview

Definition

Access modifiers control the visibility and accessibility of classes, attributes, and methods.

Modifier	Class	Package	Subclass	World
`public`	✓	✓	✓	✓
`protected`	✓	✓	✓	×
default	✓	✓	×	×
`private`	✓	×	×	×

Rule of Thumb

- Make attributes `private`
- Make getters/setters `public`
- Make helper methods `private`

Access Modifiers in Practice

```
public class BankAccount {
    // Private - only this class can access
    private double balance;
    private String pin;

    // Public - anyone can access
    public String accountNumber;

    // Protected - this class and subclasses
    protected String ownerName;

    // Default (no modifier) - same package only
    int transactionCount;

    // Private helper method
    private boolean validatePin(String inputPin) {
        return this.pin.equals(inputPin);
    }
```

Complete Example: Student Class (Part 1)

```
public class Student {
    // Attributes (private for encapsulation)
    private String studentId;
    private String name;
    private String email;
    private int age;
    private double gpa;

    // Default Constructor
    public Student() {
        this.studentId = "UNKNOWN";
        this.name = "Unknown Student";
        this.email = "";
        this.age = 0;
        this.gpa = 0.0;
    }

    // Parameterized Constructor
    public Student(String studentId, String name, int
```

Complete Example: Student Class (Part 2)

```
    // Getters
    public String getStudentId() { return studentId; }
    public String getName() { return name; }
    public int getAge() { return age; }
    public double getGpa() { return gpa; }

    // Setters with validation
    public void setName(String name) {
        if (name != null && name.length() > 0) {
            this.name = name;
        }
    }

    public void setAge(int age) {
        if (age >= 16 && age <= 100) {
            this.age = age;
        }
    }
```

Complete Example: Student Class (Part 3)

```
    // Behavior Methods
    public void study(String subject) {
        System.out.println(name + " is studying " +
            subject);
    }

    public void attendClass(String courseName) {
        System.out.println(name + " is attending " +
            courseName);
    }

    // Method to display student info
    public void displayInfo() {
        System.out.println("=== Student Information
            ===");
        System.out.println("ID: " + studentId);
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("GPA: " + gpa);
```

Using the Student Class

```
public class StudentDemo {
    public static void main(String[] args) {
        // Create students
        Student s1 = new Student("BITA001", "Ali
            Hassan", 20);
        Student s2 = new Student("BITA002", "Fatma
            Said", 21);
        Student s3 = new Student();  // Default
            constructor

        // Set additional information
        s1.setGpa(3.5);
        s2.setGpa(3.8);

        // Call methods
        s1.study("Object-Oriented Programming");
        s2.attendClass("PT821");

        // Display information
```

Common Mistakes

Mistake 1: Public Attributes

```
// BAD
public class Student {
    public String name;
    public int age;
}
// Anyone can set invalid values:
student.age = -50;
```

Correct

```
// GOOD
public class Student {
    private int age;

    public void setAge(int age) {
        if (age > 0) {
            this.age = age;
        }
    }
}
```

Mistake 2: Forgetting 'this'

```
// BAD
public Student(String name) {
    name = name; // Does nothing!
}
```

Correct

```
// GOOD
public Student(String name) {
    this.name = name;
}
```

Mistake 3: No Constructor

```
// Forgetting to initialize
Student s = new Student();
s.getName(); // Returns null!
```

Lab Exercise: Create a Book Class

Task: Create a complete `Book` class with the following requirements:

Attributes:

- ISBN (String)
- Title (String)
- Author (String)
- Price (double)
- Pages (int)
- Available (boolean)

Requirements:

1. Create default and parameterized constructors
2. Add getters and setters with validation:
   - Price must be positive
   - Pages must be greater than 0
3. Add methods: `borrow()`, `returnBook()`, `displayInfo()`
4. Create a `BookDemo` class to test your Book class

Time: 20 minutes

Work individually or in pairs. Ask questions if stuck!

Lab Exercise: Sample Solution Structure

```
public class Book {
    private String isbn;
    private String title;
    private String author;
    private double price;
    private int pages;
    private boolean available;

    // TODO: Add constructors

    // TODO: Add getters and setters

    public void borrow() {
        if (available) {
            available = false;
            System.out.println(title + " has been
                borrowed.");
        } else {
            System.out.println("Sorry, " + title + "
```

Key Takeaways

Classes:

- Blueprint for objects
- Contains attributes and methods
- Defines structure and behavior

Objects:

- Instances of classes
- Created using `new`
- Have unique state

Constructors:

- Initialize objects
- Same name as class
- Can be overloaded

Access Modifiers:

- `private`: class only
- `public`: everywhere
- `protected`: inheritance
- Use encapsulation!

Next Lecture

Inheritance and Polymorphism - Creating class hierarchies and achieving code reuse.

Homework Assignment

Create a BankAccount Management System:

Design and implement the following classes:

1. **BankAccount** class with:
   - Attributes: accountNumber, ownerName, balance, accountType
   - Constructors (default and parameterized)
   - Methods: deposit(), withdraw(), getBalance(), transfer()
2. **Customer** class with:
   - Attributes: customerId, name, email, phone
   - A BankAccount object (composition)
3. **BankDemo** class to test your implementation

Bonus: Add validation to prevent negative balances and invalid withdrawals.

Due: Before Next Class

Submit your .java files via the course LMS.

Any Questions?

Topics Covered:

- Classes and Objects
- Attributes and Methods
- Constructors and Overloading
- Access Modifiers
- Encapsulation

"In OOP, we don't just write code, we model the world."

Thank You!

Next Class: Inheritance and Polymorphism

Contact Information:
Instructor: Masoud Hamad
Email: massoud.hamad@suza.ac.tz
Office Hours: Thursday 08:00AM-12:00PM