Classes and Objects in Java

Object-Oriented Programming – Introduction

Zbigniew Dendzik

2026

Lecture Plan

- Basic concepts: classes and objects
- Class definition in Java
- Class fields (member variables)
- Constructors
- Methods (member functions)
- Creating and using objects
- Object composition
- Practical examples

What is a Class?

A **class** is a template (blueprint) for creating objects.

- Defines data structure (fields)
- Specifies behaviors (methods)
- Constitutes a complex data type
- Is the foundation of object-oriented programming

What is an Object?

An **object** is a concrete instance of a class.

- Has state (field values)
- Has identity (unique reference in memory)
- Responds to method invocations
- Created using the `new` operator

Example: Rectangle Class

```
class Rectangle
{
    double length;
    double width;

    Rectangle() {
        this.length = 0.0;
        this.width = 0.0;
    }
}
```

Class Fields (Member Variables)

- Store the state of an object
- Have a specified type (e.g., `double`, `int`, `String`)
- Accessible to all methods of the class
- Example: `double length;`
- Each object has its own copies of fields

Default Constructor

A constructor without parameters initializes fields with default values.

```
Rectangle()
{
    this.length = 0.0;
    this.width = 0.0;
}
```

- Called when creating an object
- Same name as the class name
- No return type

The "this" Keyword

- Refers to the current object
- Distinguishes class fields from parameters with the same name
- Used in assignments within constructors
- Example: `this.length = length;`

```
Rectangle(double length, double width) {
    this.length = length;  // field = parameter
    this.width = width;
}
```

Parameterized Constructor

```
Rectangle(double length, double width)
{
    this.length = length;
    this.width = width;
}
```

Allows initialization of an object with specific values upon creation.

Constructor Overloading

- A class can have multiple constructors
- They differ in number or types of parameters
- Provides flexibility in object creation
- Java automatically selects the appropriate constructor based on arguments

Example:

- `Rectangle()` – parameterless constructor
- `Rectangle(double l, double w)` – with parameters

Class Methods (Member Functions)

```
double area()
{
    return length * width;
}
```

- Define behaviors and operations on object data
- Have a return type (or `void`)
- Can accept parameters
- Have access to object fields

Example: perimeter() Method

```
double perimeter()
{
    return 2 * (length + width);
}
```

A method returning a `double` value – the perimeter of the rectangle.

Creating an Object – Syntax

```
// Variable declaration
Rectangle obj;

// Object creation
obj = new Rectangle(3, 4);

// Or shorter in one line:
Rectangle obj = new Rectangle(3, 4);
```

The "new" Operator

- Allocates memory for a new object on the heap
- Calls the appropriate constructor
- Returns a reference to the created object
- Example: `new Rectangle(3, 4)`

Note: The variable stores a reference (address), not the object itself.

Invoking a Method on an Object

```
Rectangle obj = new Rectangle(3, 4);
double x = obj.area();
System.out.println("Rectangle area: " + x);

// Result: Rectangle area: 12.0
```

We use the dot operator (`.`) to invoke a method on an object.

The toString() Method

```
public String toString()
{
    return "[length: " + length +
           ", width: " + width + "]";
}
```

- Automatically called when printing an object
- Returns a textual representation of the object
- Useful for debugging

Example: Point Class

```
class Point
{
    double x;
    double y;

    Point(double x, double y)
    {
        this.x = x;
        this.y = y;
    }

    public String toString() {
        return "[x: " + x + ", y: " + y + "]";
    }
}
```

Object as a Class Field

```
class Rectangle
{
    double length;
    double width;
    Point vertex;  // Object as a field!

    Rectangle(double l, double w) {
        this.length = l;
        this.width = w;
        this.vertex = new Point(0, 0);
    }
}
```

Object Composition

- Objects can contain other objects as fields
- Building complex data structures
- Example: `Rectangle` contains `Point`
- Modeling real-world relationships between objects
- A fundamental technique in object-oriented programming

Constructor with Object as Parameter

```
Rectangle(double length,
          double width,
          Point vertex)
{
    this.length = length;
    this.width = width;
    this.vertex = vertex;
}
```

We can pass an existing object as a constructor parameter.

Methods with Parameters

```
void translate(double dx, double dy)
{
    this.x += dx;
    this.y += dy;
}
```

- A method that modifies the object's state
- Return type: `void` (returns nothing)
- Accepts two parameters of type `double`

Example Usage of translate() Method

```
Point obj = new Point(0, 0);
System.out.println(obj);
// Result: [x: 0.0, y: 0.0]

obj.translate(-1, 3);
System.out.println(obj);
// Result: [x: -1.0, y: 3.0]
```

Method Call Delegation

```
// In the Rectangle class:
void translate(double u, double v)
{
    this.vertex.translate(u, v);
}
```

- Using a method of another object
- `Rectangle` delegates translation to the `Point` object
- Reusing existing code

Methods Returning boolean

```
boolean contains(Point obj)
{
    return (obj.x >= vertex.x &&
            obj.x <= vertex.x + length &&
            obj.y >= vertex.y &&
            obj.y <= vertex.y + width);
}
```

Methods testing conditions – return `true` or `false`.

Example: Circle Class

```
class Circle
{
    double radius;
    Point center;

    Circle(double r, Point c) {
        this.radius = r;
        this.center = c;
    }

    double area() {
        return Math.PI * radius * radius;
    }
}
```

Testing Relationships Between Objects

Example methods:

- `boolean contains(Point obj)`
  – Is the point inside the rectangle?
- `boolean intersects(Circle obj)`
  – Do the circles intersect?
- `boolean intersects(Rectangle obj)`
  – Does the rectangle intersect the circle?

We apply geometric calculations and conditional logic.

Program Class with main() Method

```
public class Program
{
    public static void main(String[] args)
    {
        Point obj1 = new Point(-1, 1);
        System.out.println("point: " + obj1);

        Rectangle obj2 = new Rectangle(3, 4, obj1);
        System.out.println("rectangle: " + obj2);

        double a = obj2.area();
        System.out.println("area: " + a);
    }
}
```

Compilation Unit

- Multiple classes can be saved in one `.java` file
- Only one class can be `public` per file
- The file name must match the `public` class name
- Compilation: `javac Program.java`
- Each class generates a separate `.class` file

Compilation and Execution

Process:

1. The `javac` compiler (part of JDK) creates bytecode
2. Result: `.class` files for each class
3. The `java` interpreter (part of JRE) runs the program
4. Command: `java Program`

JDK – Java SE Development Kit
JRE – Java SE Runtime Environment

javadoc Documentation

- The `javadoc` tool generates HTML documentation
- Uses special comments `/** ... */`
- Documents classes, fields, and methods
- Standard documentation format in the Java ecosystem
- Command: `javadoc Program.java`

Summary of Key Concepts

- **Class** = template, **object** = instance
- **Fields** store object state
- **Constructors** initialize new objects
- **Methods** define object behaviors
- **The new operator** creates objects in memory
- **Composition** – objects contain other objects
- **toString()** – textual representation of object