

# **Advanced Programming**

## **Object-Oriented Programming in Java**

**Instructor:** Ali Najimi

**Author:** Hossein Masihi

**Department of Computer Engineering**

**Sharif University of Technology**

**Fall 2025**





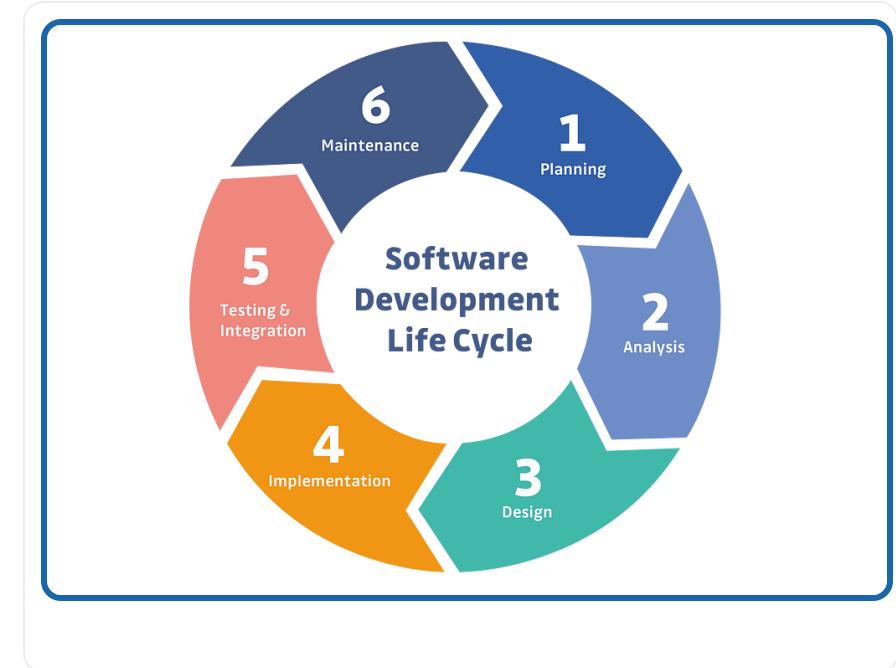
# Table of Contents

- 1. SDLC – Software Development Life Cycle
- 2. SSDLC – Secure Software Development Life Cycle
- 3. UML Diagrams
- 4. Creating Classes
- 5. Objects in Memory
- 6. Memory Management
- 7. Class Loading
- 8. Garbage Collection
- 9. Parameter Passing
- 10. Constructor & this
- 11. `static` Keyword
- 12. Packaging in Java



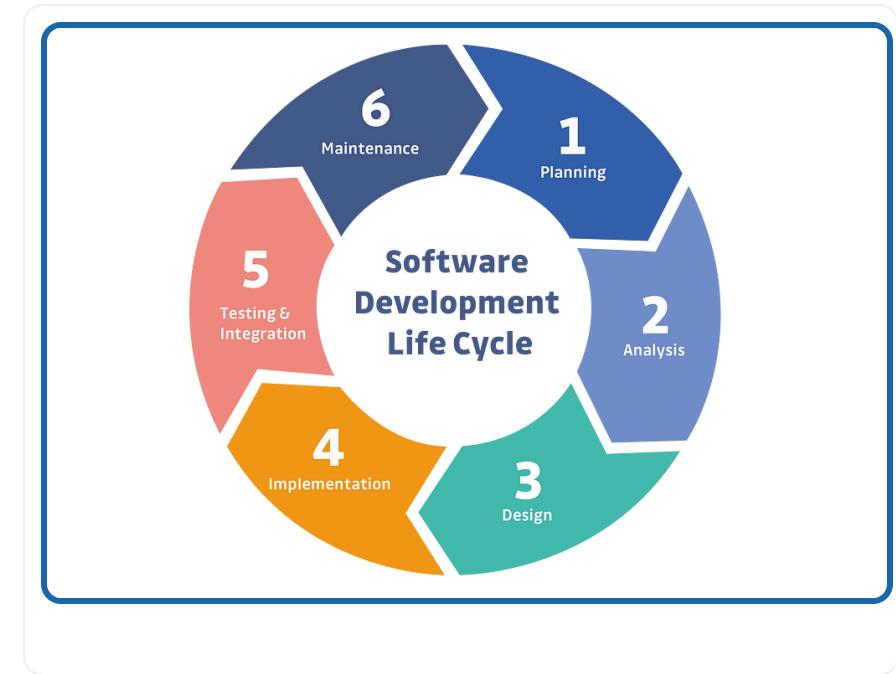
# SDLC — Software Development Life Cycle

- **SDLC** defines structured steps for building software.
- Common phases:
  - Requirement Analysis**
  - Design**
  - Implementation (Coding)**
  - Testing**
  - Deployment**
  - Maintenance**





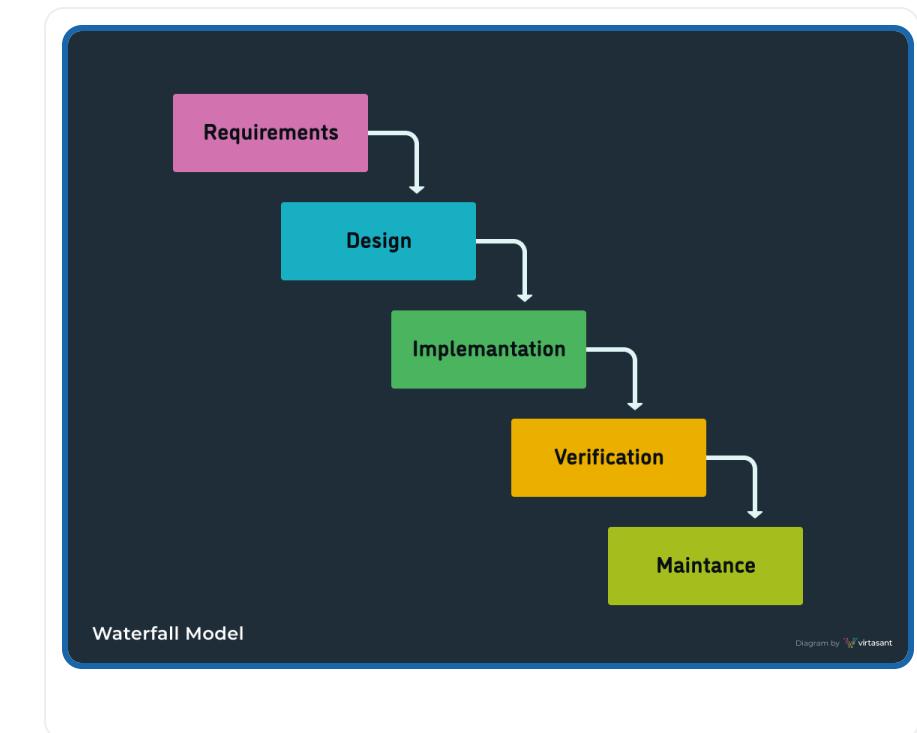
- Ensures:
  - Predictable delivery
  - Quality assurance
  - Cost & time control





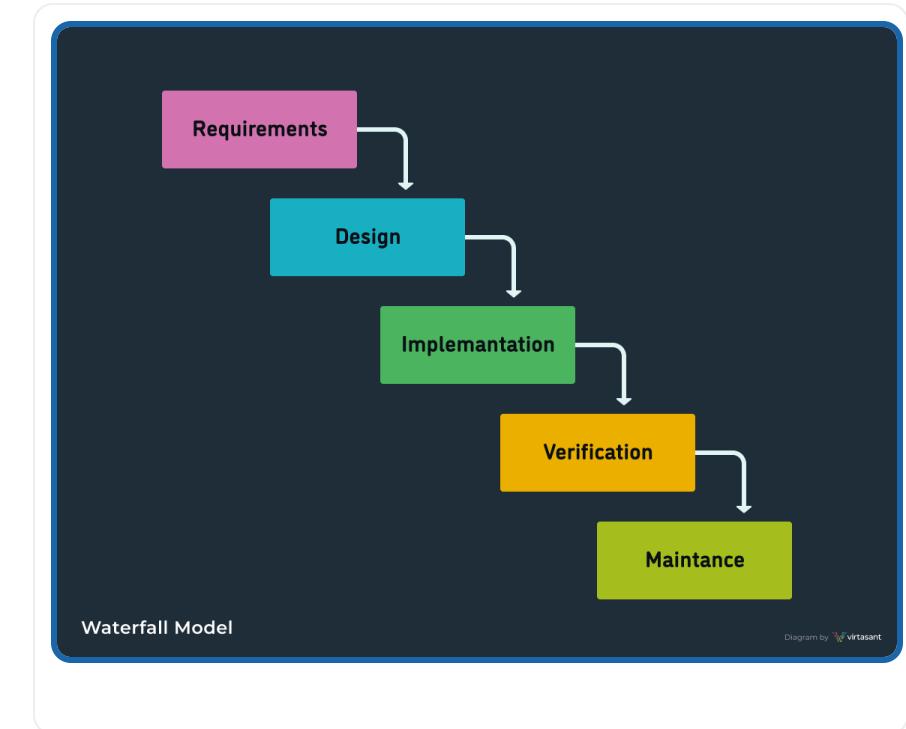
# SSDLC – Secure Software Development Life Cycle

- **SSDLC** = SDLC + integrated **security at every stage**.
- Adds security practices such as:
  - Threat modeling
  - Secure coding guidelines
  - Security testing & auditing
  - Vulnerability management





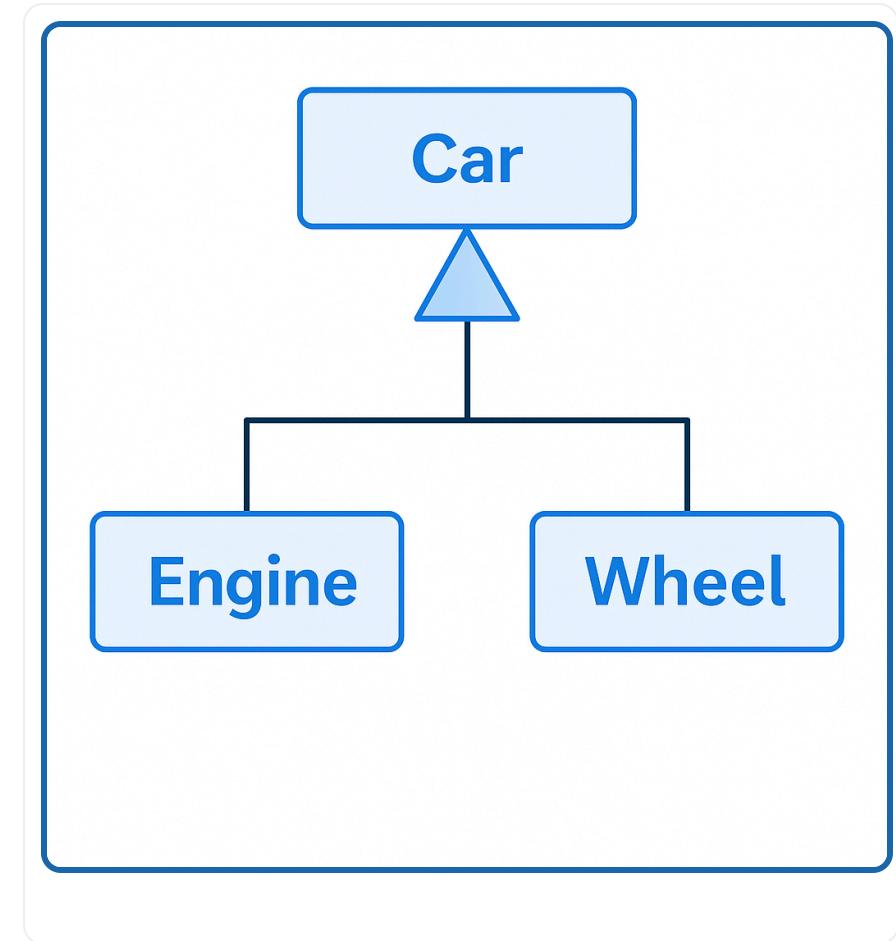
- Goal: build software that is **secure by design**.
- Common frameworks:
  - Microsoft SDL
  - OWASP SAMM
  - NIST SSDF





# UML Diagrams

- Visualize system structure.
- Relationships:
  - Association ( $\rightarrow$ )
  - Inheritance ( $\triangleright$ )
  - Composition / Aggregation ( $\diamond$ )





# Creating a Class in Java

```
public class Car {  
    String color;  
  
    void drive() {  
        System.out.println("Driving");  
    }  
}  
  
Car c = new Car();  
c.  
  
drive();
```

## Creating a Class in Java

```
public class Car {  
    String color;  
    void drive() {  
        System.out.println  
            "Driving";  
    }  
}  
Car c = neew Car();  
c.drive();
```

Console  
Driving

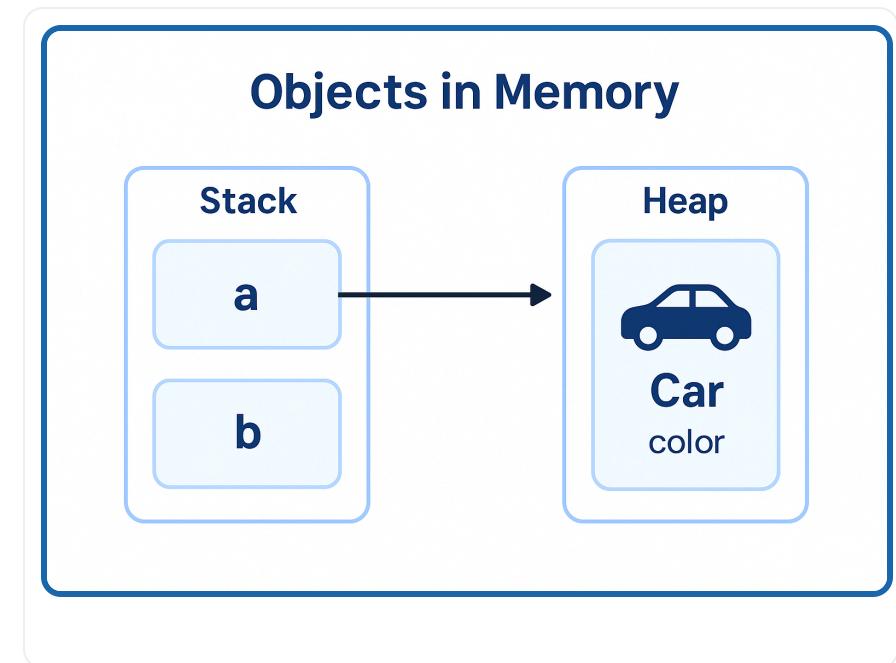


# Objects in Memory

- Objects live in **Heap**, references on **Stack**.
- Example:

```
Car a = new Car();  
Car b = a;
```

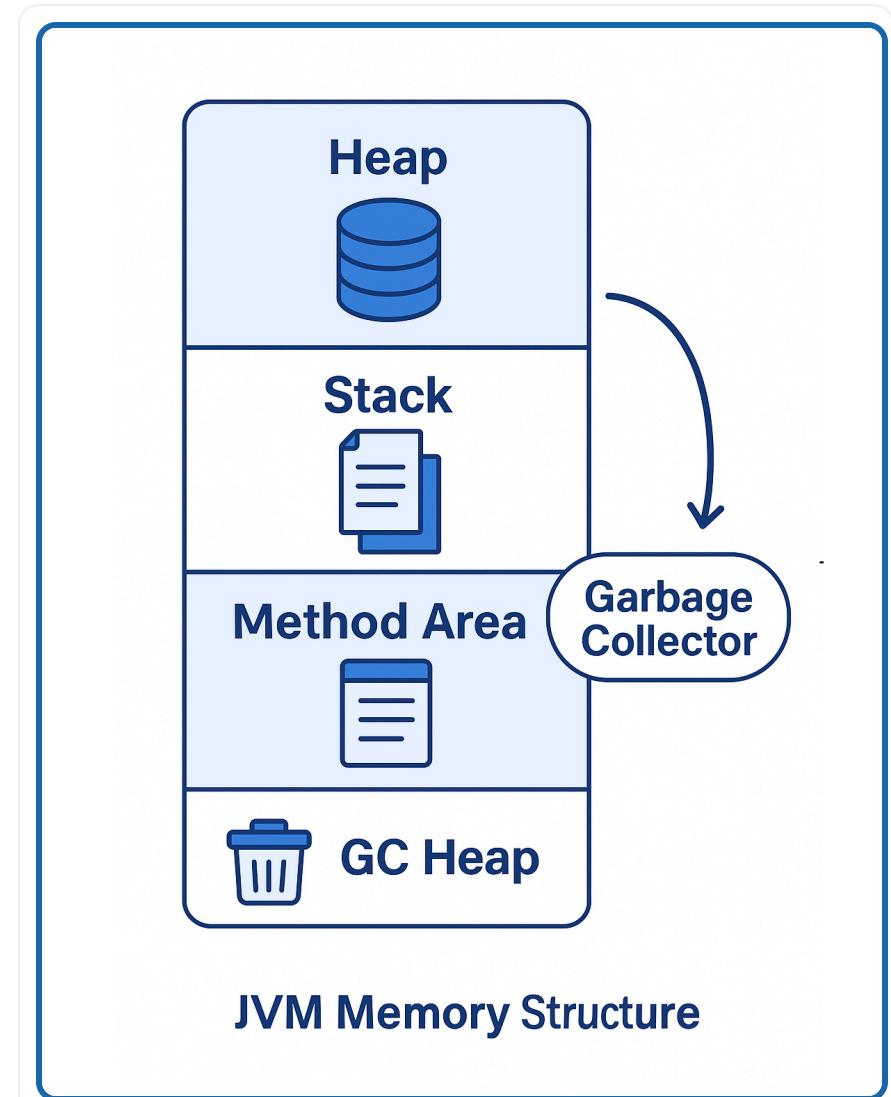
→ both point to the same object.





# Java Memory Management

- Heap → objects
- Stack → local vars
- GC → automatic cleanup
- JVM manages lifecycle





# Constructors and this Keyword

```
class Student {  
    String name;  
  
    Student(String name) {  
        this.name = name;  
    }  
}
```

- Called automatically at object creation.
- `this` → current object reference.

```
class Student {  
    String name;  
    Student(String name)  
        this.name = name;  
}
```

`this`





## static Keyword

- Belongs to the **class**, not objects.
- Shared between all instances.

```
class Counter {  
    static int count = 0;  
    Counter() { count++; }  
}
```



# Packaging in Java

```
package ir.sharif.course;  
  
import java.util.*;
```

- Groups related classes.
- Prevents naming conflicts.

# Thank You

AP – OOP in Java

