Overview

ADVANCED JAVA PROGRAMMING

Level (3) - SW & IT

mohmd798380@gmail.com

Eng. Mohammed Muthanna

Advanced Java Programming Course

Prerequisites

- Basic Java knowledge
- OOP concepts understanding
- Knowledge of Java GUI (e.g. Swing or JavaFX)
- GitHub Knowledge

Course Structure

- 12 Theory lectures
- 10 Practical sessions
- Total of 44 hours

Learning Objectives

01

Concurrent Programming

Build robust concurrent applications that can handle multiple tasks simultaneously.
Real-world example: Banking System

02

File Systems

Develop applications that efficiently manage file operations.
Real-world example: Document management Systems (e.g. Google Drive)

03

Networking

Create networked applications that can communicate effectively.
Implement client-server applications using sockets
Real-world example: Chat System

04

Distributed Systems

Build scalable distributed applications using RMI or RPC.
Enabling communication between processes or software components running on different devices across a network.
Real-world example: Cloud Services

Course Outline :

1. Java & OOP Review

- OOP Concepts
  - Inheritance
  - Interface
  - Abstract
- Exception Handling
  - Custom exceptions
  - Exception hierarchies
- Packages
- JDBC Programming
  - Database connections
  - CRUD operations

2. Concurrent Programming

- Thread Fundamentals.
  - Thread lifecycle
  - Creating and managing threads
  - Thread states and priorities
- Synchronization
  - synchronized keyword
  - wait/notify mechanism

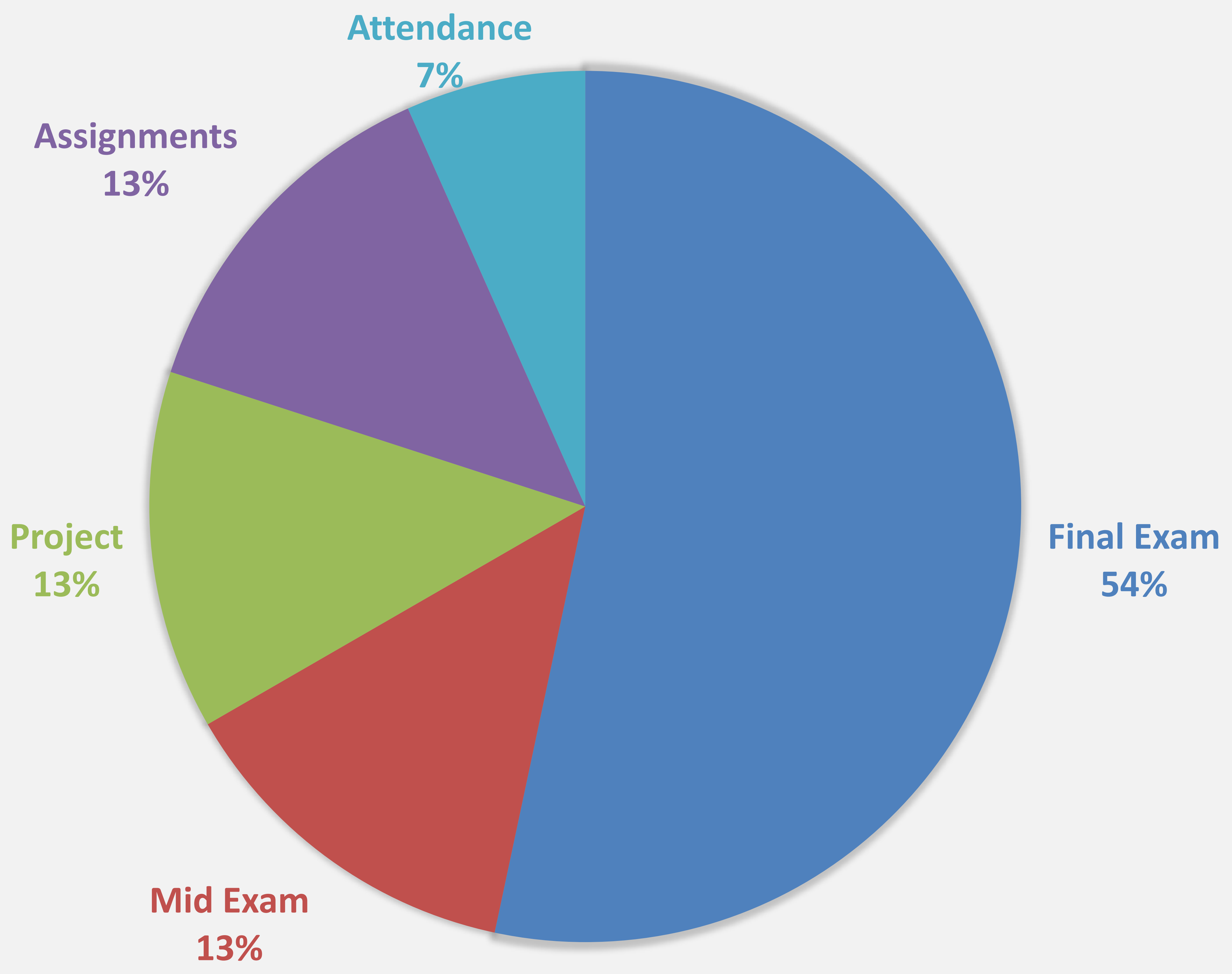
3. File System Operations

- File class operations
- FileReader/FileWriter
- BufferedReader/ BufferedWriter
- Exception handling in I/O

4. Network Programming

- Socket Programming
  - TCP/UDP implementation
  - Client-server architecture
- Distributed Systems
  - RMI
  - RPC

COURSE GRADES



Lecture 1

REVIEW OF OBJECT-ORINTEND PROGRAMMING IN JAVA OOP

mohmd798380@gmail.com

Object-Oriented Programming in Java

Introduction to OOP :

- What is OOP?
- Why OOP?
- The benefits of OOP

Core Concepts :

- Classes and Objects
- Constructors and this keyword
- Static vs Instance members
- Encapsulation and Abstraction
- Inheritance and Polymorphism
- Method Overloading/Overriding
- Inner Classes

What is OOP ?

Object-Oriented Programming (OOP): is a programming paradigm that uses objects and classes to structure and organize code.

Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism, etc. in programming. The main aim of OOPs is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

Why OOP ?

- Organization
- Reusability
- Maintenance
- Security

The benefits of OOP :

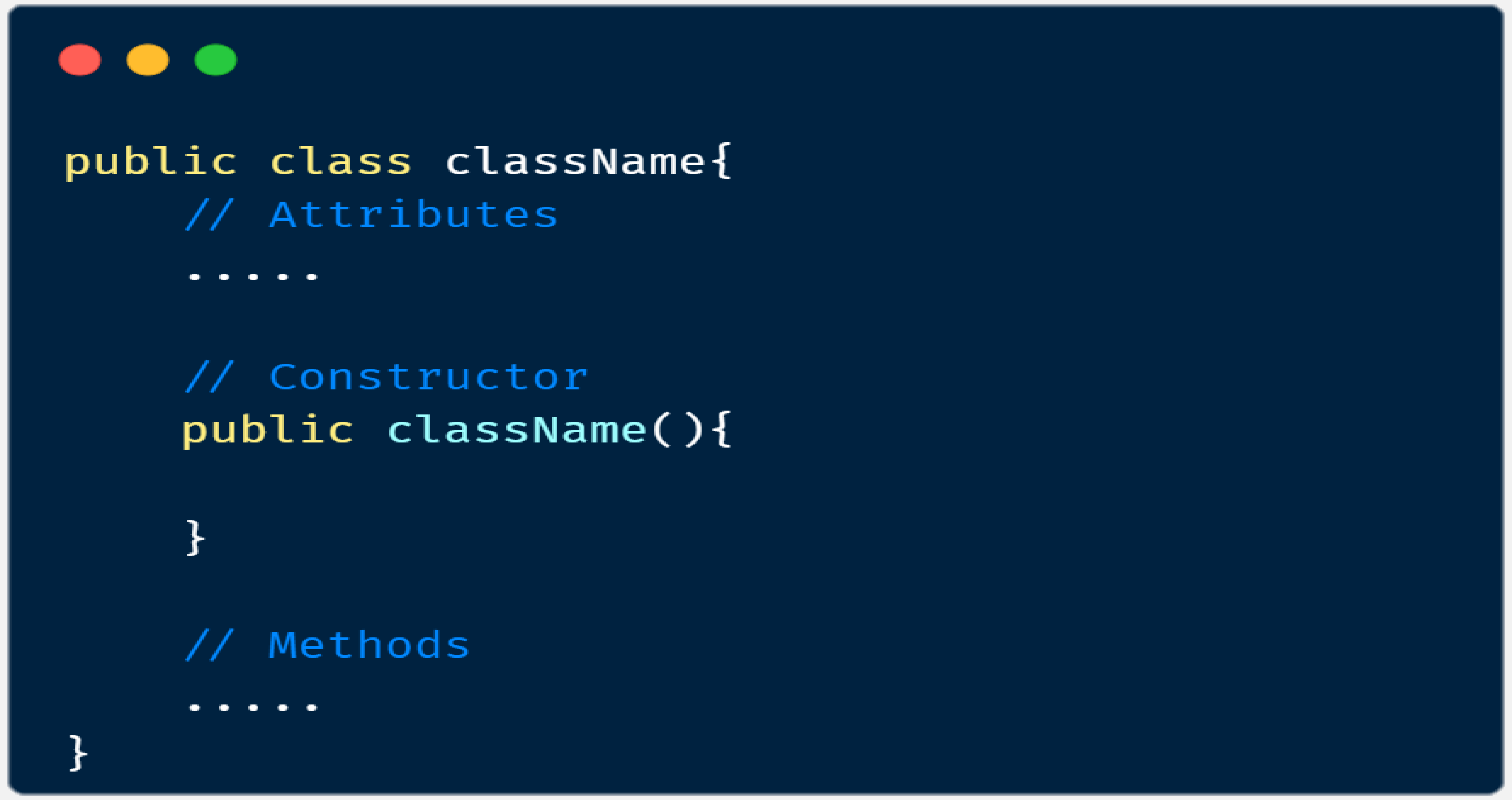
- Better Code Organization
- Code Reusability
- Easier Maintenance
- Enhanced Security
- Flexibility

❑ Classes & Objects

What is a Class ?

A Class: is a blueprint or template for creating objects. It represents the set of properties or methods that are common to all objects of one type.

Class Structure:



```
public class className{
    // Attributes
    .....

    // Constructor
    public className(){

    }

    // Methods
    .....
}
```

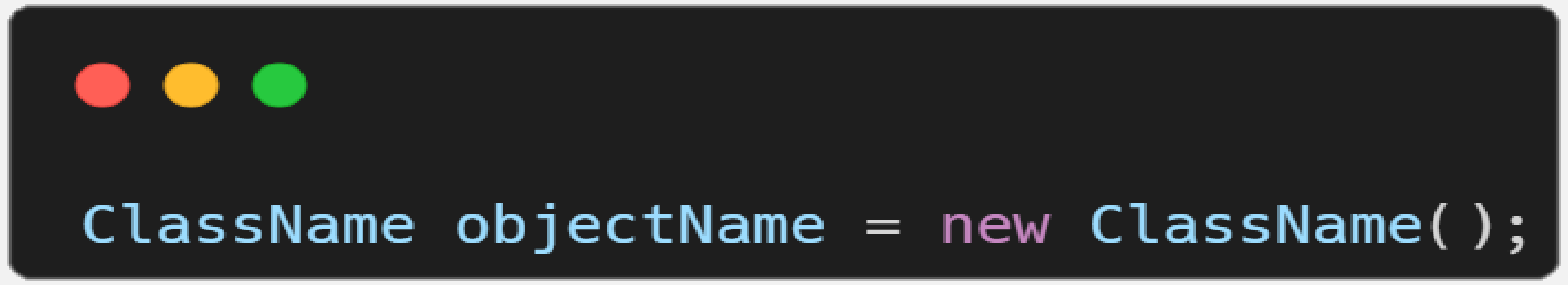
❑ Classes & Objects

What is an Object?

An Object: is an instance of a class. It represents a specific entity with its own set of data and behaviors defined by the class.

- Objects are created using the **new** keyword.
- The object provides access to the methods and fields of the class.

Creating Objects:



```
ClassName objectName = new ClassName();
```

❑ Constructors & this Keyword

What is a Constructor ?

A Constructor: is a special method that is used to initialize objects.

The constructor is called when an object of a class is created. It can be used to set initial values for object attributes.

Key Characteristics:

- The name of the constructor must match the name of the class.
- It does not have a return type.
- It is invoked automatically when an object is created.
- If no constructor is defined, Java provides a default no-argument constructor.
- A constructor in Java can not be abstract, final, static, or synchronized.

❑ Constructors & this Keyword

Types of Constructors:

- Default Constructor
- Parameterized Constructor
- Copy Constructor

```
public className(){}
```

```
public className(dataType ParameterName){}
```

```
public className(className objectName){}
```

❑ Constructor Chaining

Constructor chaining: is the process of calling one constructor from another constructor with respect to the current object.

Types of Constructor Chaining:

- Within the same class using this()
- From super class using super()

Can a Java constructor be private?

❑ Constructors & this Keyword

What is the this keyword ?

The this keyword: is a reference to the current object.

It can be used to:

- Access fields, methods, or constructors of the current object.
- Differentiate between instance variables and parameters.
- Call another constructor.
- Pass or return the current object.

❑ Static vs Instance members

Static members :

Static members: are shared among all instances of a class. They belong to the class itself rather than any specific object.

Key Characteristics:

- Declared using the static keyword
- Can be accessed directly using the class name without creating an instance.
- Only one copy of a static member exists for all objects of the class.

instance members :

Instance members: are unique to each object of a class. They represent the state and behavior specific to that object.

Key Characteristics:

- Each object has its own copy of instance fields.
- Accessed through an object reference.

❑ Encapsulation & Abstraction

What is an Encapsulation ?

An Encapsulation: It's the mechanism that binds together code and the data it manipulates.

Encapsulation in Java is achieved by:

- Declaring fields (attributes) as private
- Providing public getter and setter methods to access and modify private fields.

Key Concepts:

- Data Hiding (Protect sensitive data from unauthorized access)
- Access Modifiers
  - public
  - private
  - protected
  - default

Modifier	Within Class	Within Package	Outside Package (Subclass)	Outside Package (Non-Subclass)
Public	✓	✓	✓	✓
Protected	✓	✓	✓	✕
Default	✓	✓	✕	✕
Private	✓	✕	✕	✕

❑ Encapsulation & Abstraction

What is an Abstraction ?

An Abstraction: is the process of hiding the implementation details and only showing the essential functionality or features to the user.

Abstraction in Java is achieved by:

- abstract classes
- interfaces achieve 100% abstraction

▪ Data Abstraction may also be defined as the process of identifying only the required characteristics of an object ignoring the irrelevant details.

▪ It helps in reducing programming complexity.

❑ Abstract Class

An abstract class: is a class that is declared with an abstract keyword. It may contain both abstract and concrete methods. Abstract classes cannot be instantiated but can be subclassed.

- There can be no object of an abstract class.
- Any class that contains one or more abstract methods must also be declared with an abstract keyword.
- Override the abstract methods in the concrete classes to provide their specific implementations.
- **When to Use :**
  - Use abstract classes when a class represents a clear inheritance hierarchy.
  - When You Need Both Abstract and Concrete Methods.
  - When You Need to define instance fields.
  - When You Need Constructors

❑ Interface

An interfaces: are another method of implementing abstraction in Java. Interface is a collection of abstract methods.

- It supports multiple inheritance.
- It can not have constructors.
- Methods in interfaces are implicitly public and abstract.
- Fields in interfaces are implicitly public, static and final.
- **When to Use :**
  - To Define a Contract for Unrelated Classes.
  - When Multiple Inheritance Is Needed.
  - To Define Behavior Without Implementation.

Can a method body be implemented within an interface in Java?

What is the difference between an Abstract class and an Interface in Java?

❑ Inheritance and Polymorphism

What is an Inheritance ?

An Inheritance: is a mechanism that allows a class to inherit attributes and methods from another class.

- The class that inherits is called the subclass (or derived class),
- And the class being inherited from is called the superclass (or base class).
- Using the extends keyword indicates you are derived from an existing class.

Types of Inheritance:

- Single Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance
- Multiple Inheritance *
- Hybrid Inheritance (Multilevel Inheritance and Hierarchical Inheritance)

Note: In Java, we can achieve multiple inheritances only through Interfaces.

❑ Inheritance and Polymorphism

What is a Polymorphism ?

In Java, polymorphism refers to the ability of a method or object to take multiple forms.

Types of Java Polymorphism :

- **Compile-Time Polymorphism :**
  - Achieved through method overloading.
- **Run-Time Polymorphism :**
  - Achieved through method overriding.

Note: Polymorphism enhances flexibility and scalability.

❑ Method Overloading/Overriding

Method Overloading :

Method overloading in Java means when there are multiple functions with the same name but different parameters then these functions are said to be overloaded.

- Methods can be overloaded by changes in the number of arguments or/and a change in the type of arguments.

Method Overriding :

Method overriding in Java occurs when a subclass implements a method which is already defined in the superclass

- The method in the subclass must have the same signature as in the superclass.
- If we don't want a method to be overridden, we declare it as final.

Can we Overload or Override static methods in java ?

❑ Inner Classes

An Inner Classes: are classes defined within another class. They are associated with their outer class and can access its members, including private fields and methods.

Types of Inner Classes:

- Nested Inner Class
- Static Nested Classes
- Method Local Inner Classes
- Anonymous Inner Classes

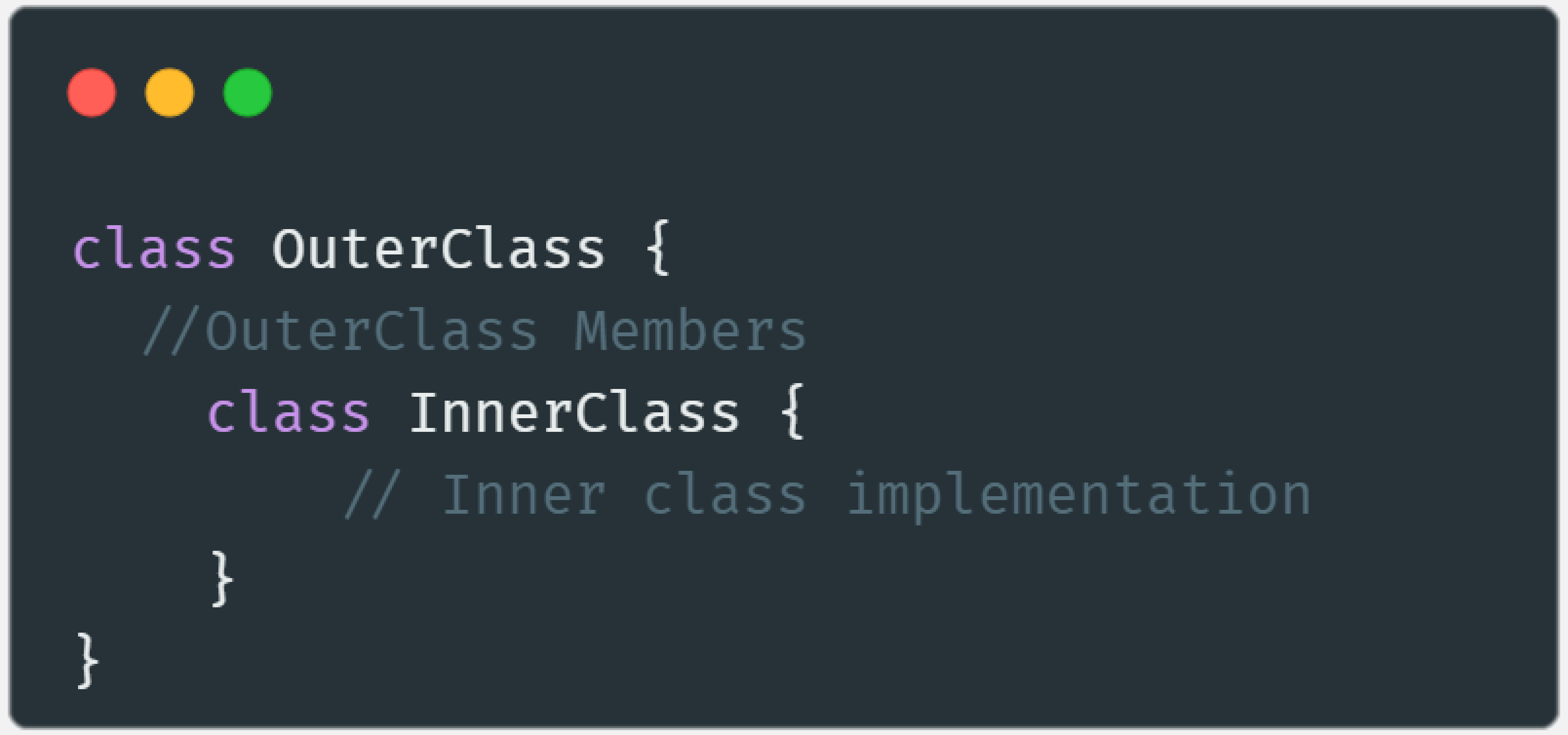
Key Characteristics:

- Group classes logically that are only used in one place.
- Improve encapsulation by hiding the inner class from other classes.
- Facilitate access to members of the outer class.

❑ Nested Inner Class

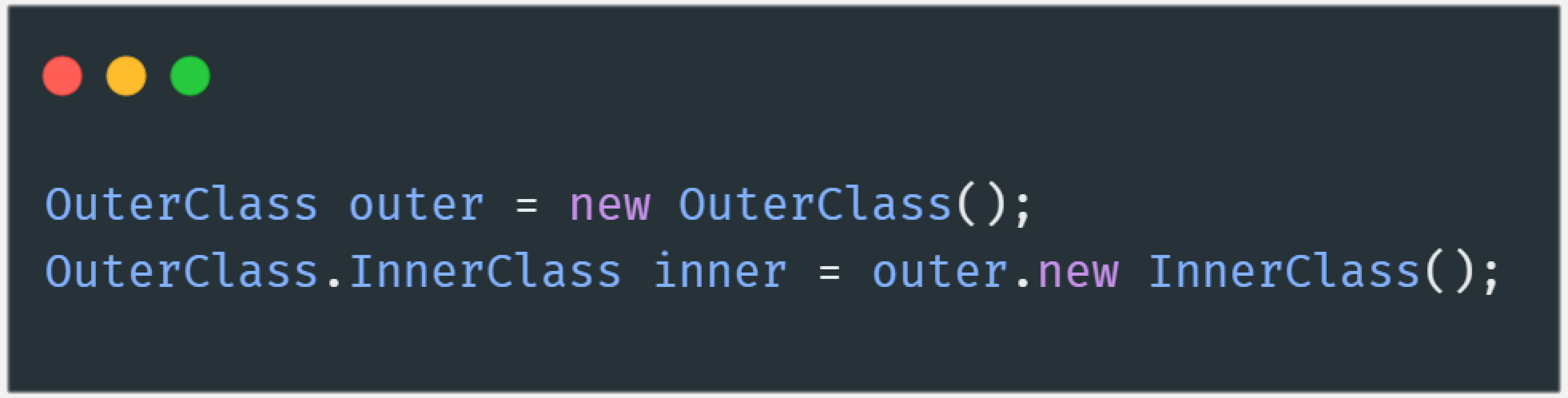
- It can access any private instance variable of the outer class.
- Like any other instance variable, we can have access modifier private, protected, public, and default modifier.
- Like class, an interface can also be nested and can have access specifiers.

❖ Syntax



```
class OuterClass {
  //OuterClass Members
    class InnerClass {
        // Inner class implementation
    }
}
```

❖ Creating Objects

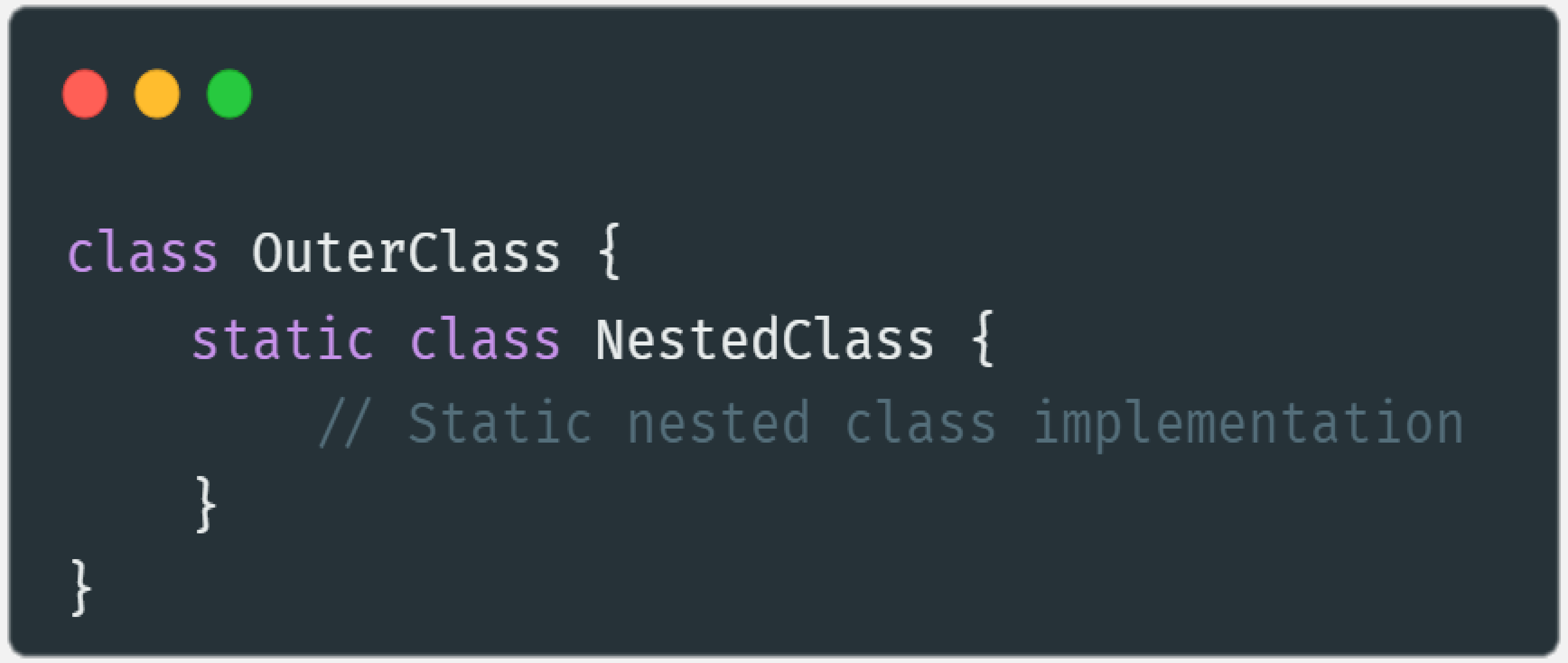


```
OuterClass outer = new OuterClass();
OuterClass.InnerClass inner = outer.new InnerClass();
```

❑ Static Nested Classes

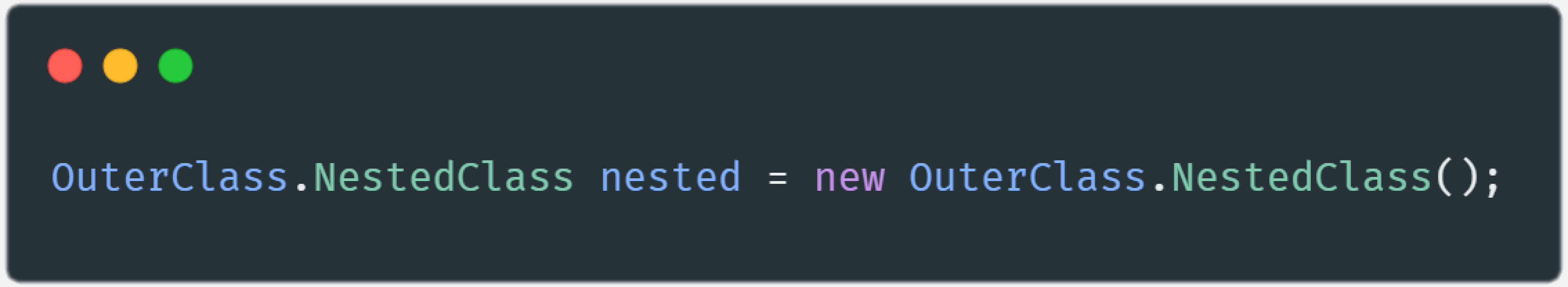
- Cannot access non-static members of the outer class.
- Doesn't need an instance of the outer class.
- A static nested class is associated with the outer class itself rather than an instance.

❖ Syntax



```java
class OuterClass {
    static class NestedClass {
        // Static nested class implementation
    }
}
```

❖ Creating Objects

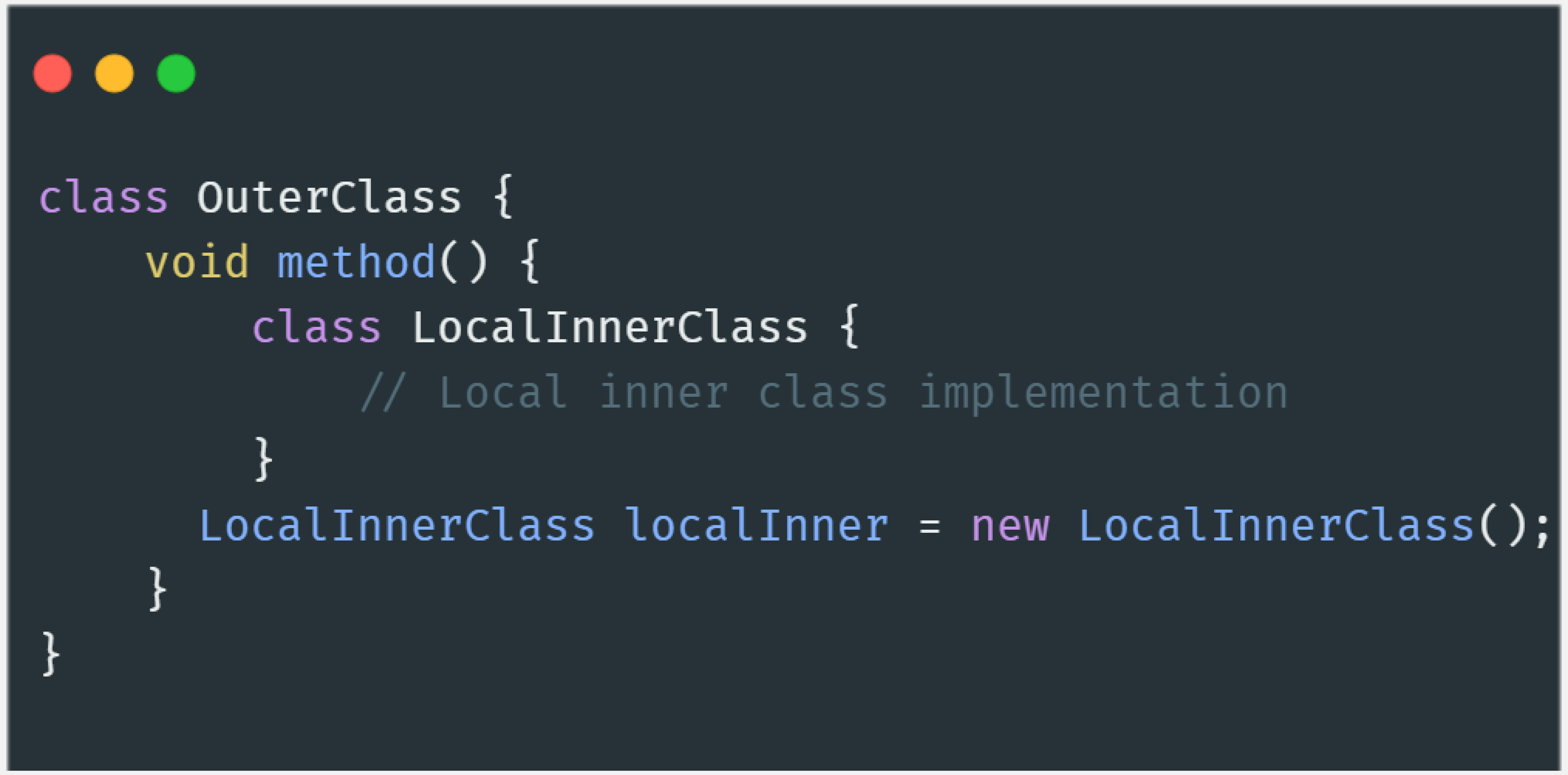


```java
OuterClass.NestedClass nested = new OuterClass.NestedClass();
```

❑ Local Inner Classes

- Inner class can be declared within a method, constructor, or block of an outer class.
- It can be used when you need a class only once and it is limited to a specific scope.

❖ **Syntax**



```
class OuterClass {
    void method() {
        class LocalInnerClass {
            // Local inner class implementation
        }
       LocalInnerClass localInner = new LocalInnerClass();
    }
}
```

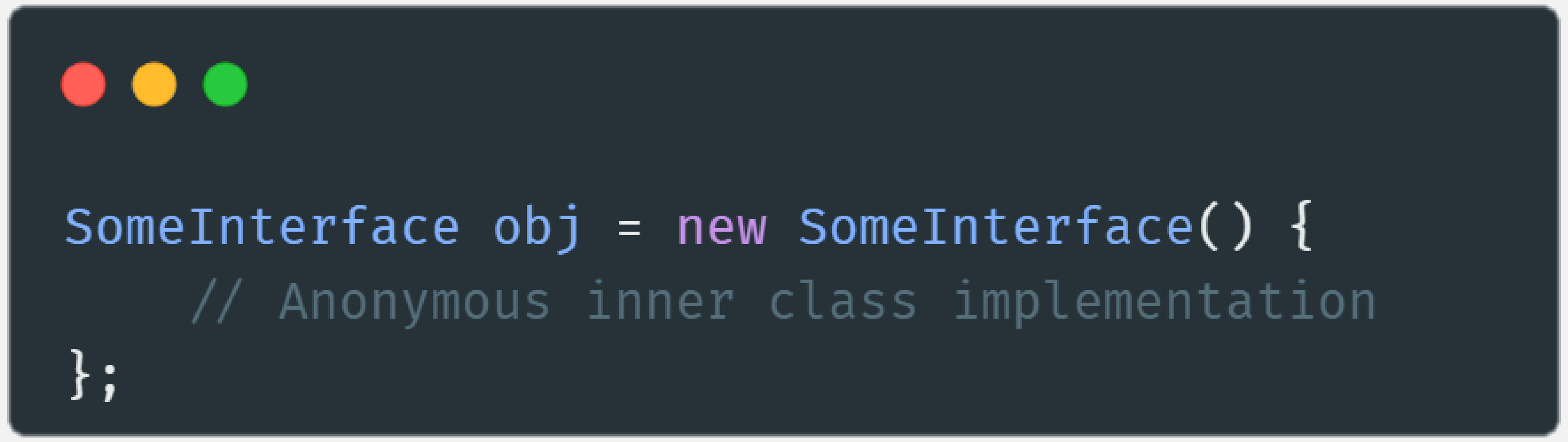
❑ Anonymous Inner Classes

- **An anonymous class:** is a class without a name that is defined and instantiated at the same time.
- It is used when you need to create a one-time-use class.

They are created in two ways:

- As a subclass of the specified type
- As an implementer of the specified interface

❖ **Syntax**



```java
SomeInterface obj = new SomeInterface() {
    // Anonymous inner class implementation
};
```

❑ Others Topics

- ❑ Generics
- ❑ Exception Handling
- ❑ JDBC
- ❑ Object Class
- ❑ Packages
- ❑ Relationships in Java OOP
- ❑ Functional Interface
- ❑ Marker Interface
- ❑ Clean Code
- ❑ SOLID Principles
- ❑ Design Patterns

References:

- Trail: Learning the Java Language (The Java™ Tutorials)
- Java Tutorial - GeeksforGeeks
- Java A Beginner’s Guide - Eighth Edition
- Java Network Programming- 4th Edition "*Elliotte Rusty Harold*"

THE END

mohmd798380@gmail.com