JAVA OOP

By. aabid

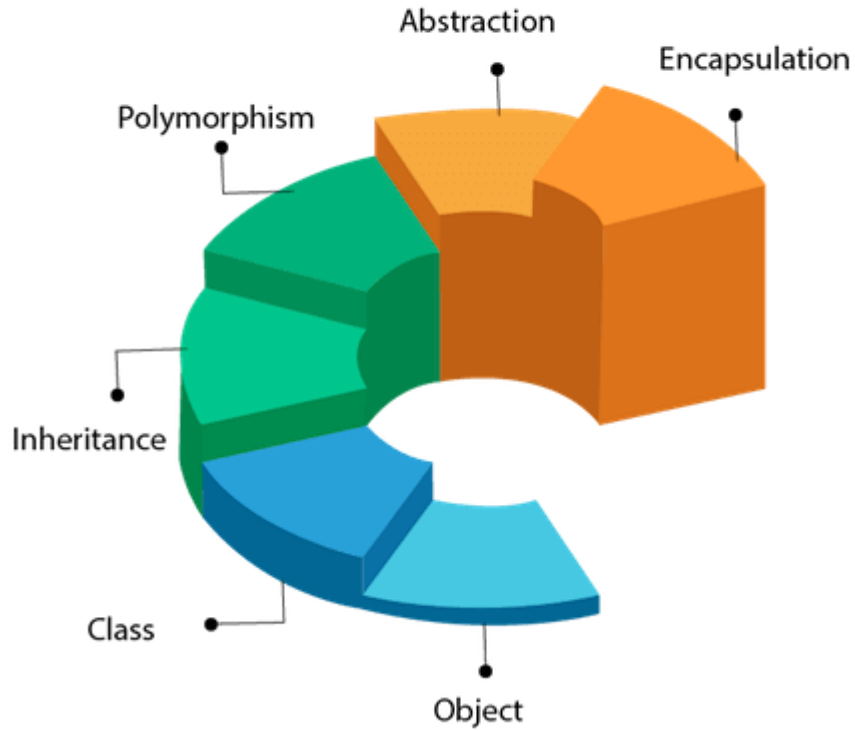
What is Object-Oriented Programming (OOP)?

Object-Oriented Programming (OOP) is a programming paradigm (style) that organizes code around **objects** rather than functions and logic. These objects represent real-world entities, and each object contains **data** (attributes) and **behaviors** (methods) that define its characteristics and actions.

Key Terminologies

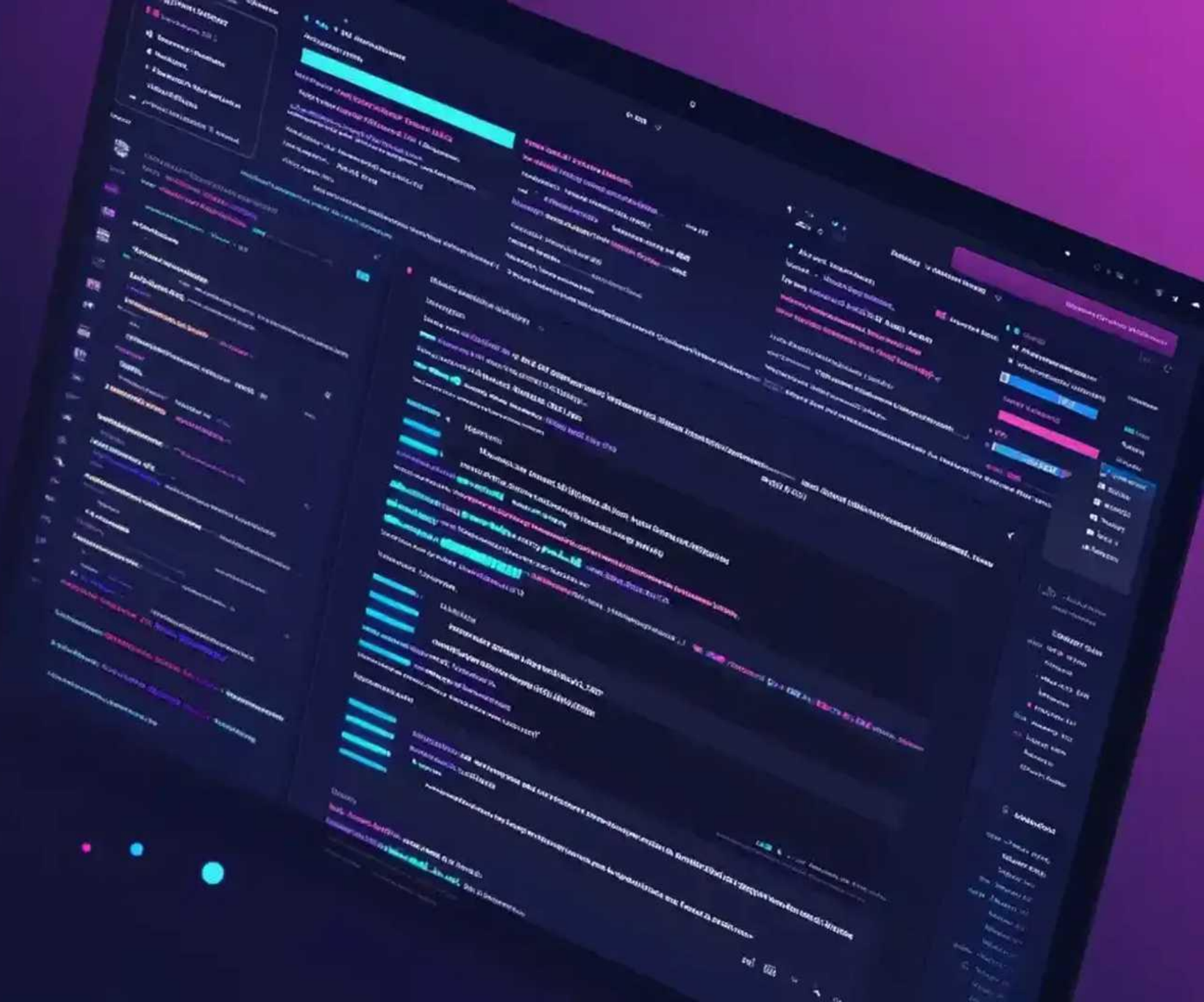
- Class
- Object
- Attributes
- Method
- Constructor
- Encapsulation
- Inheritance
- Polymorphism
- Abstraction
- Interface
- Static Members
- This Keyword
- Super Keyword
- Super Keyword
- Final Keyword
- Inner Classes
- Access Modifiers

OOPs (Object-Oriented Programming System)



Module – 1

- Class
- Object
- Attributes
- Method
- Constructor



Definitions

1. Class

A **class** is a blueprint or template used to create objects. It defines the properties (attributes) and actions (methods) that the objects created from the class can have.

2. Object

An **object** is an instance of a class. It is a real-world entity created using the class blueprint.

3. Attributes

Attributes are the properties or variables of a class. They define the characteristics of an object.

4. Method

A **method** is a function inside a class that defines the behavior of the objects.

5. Constructor

A **constructor** is a special method used to initialize an object. It is called automatically when an object is created.

Code

Class | Object | Method

```java
// class
class Car{

    // Method
    void drive() {
        System.out.println("I am driving the fastest car");
    }

}

public class Main{
    public static void main(String args[]){
        Car c1 = new Car(); // creating instance / object
        c1.drive(); // calling method
    }
}
```

```java
// class
class Car{

    // Method
    void drive() {
        System.out.println(x:"I am driving the fastest car");
    }

}

public class Main{
    Run | Debug
    public static void main(String args[]){
        Run main | Debug main
        Car c1 = new Car(); // creating instance / object
        c1.drive(); // calling method
    }
}
```

Code

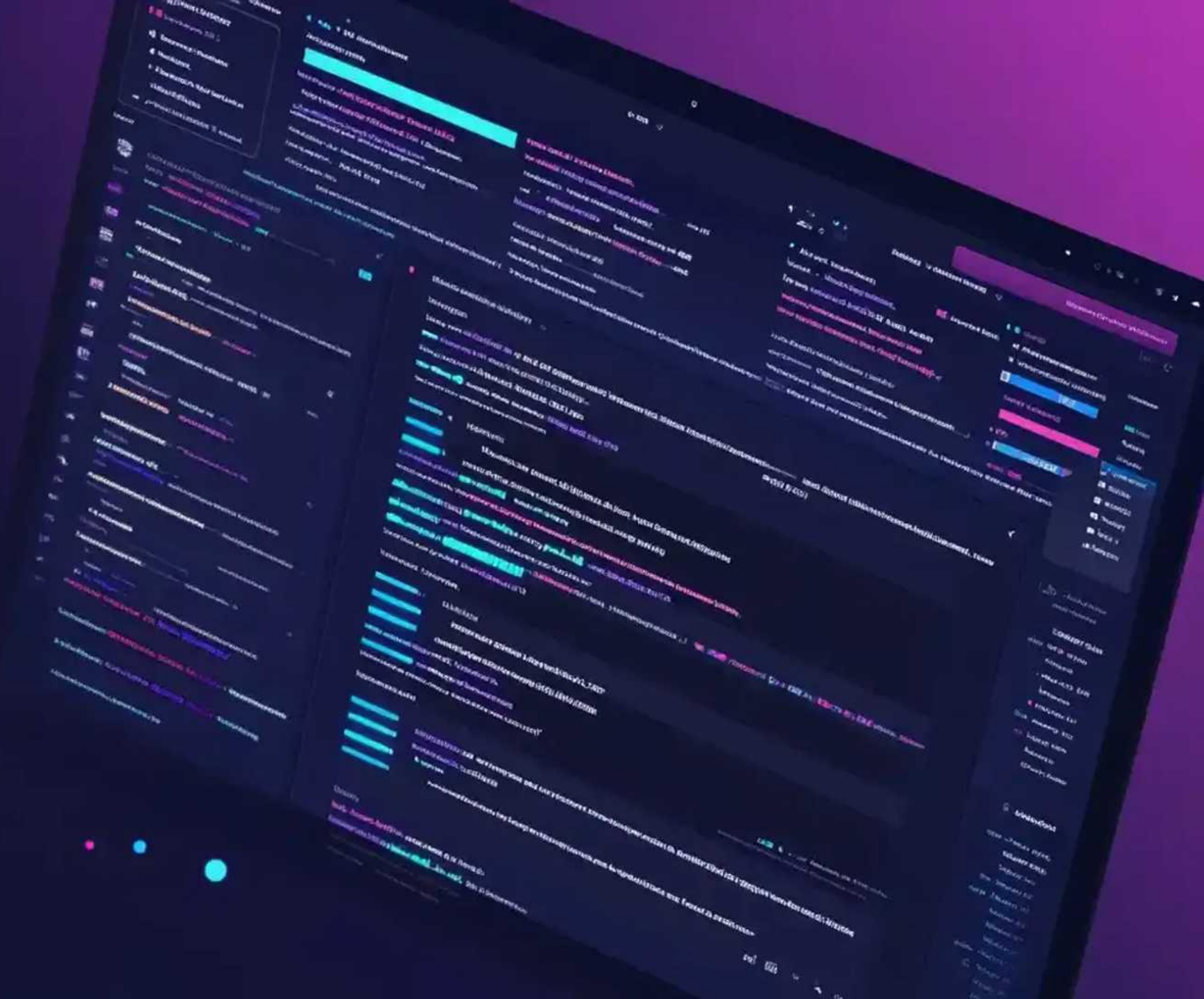
- Class
- Attributes
- Constructor
- Method
- Object

```
// class
class Car{
    // Attributes / Variables
    int seats;
    String color;

    // Constructor
    Car(String color, int seats){
        this.seats = seats;
        this.color = color;
    }

    // Method
    void drive() {
        System.out.println("I am driving the fastest car");
        System.out.println("I am driving a "+ color+ " car having "+seats+ "  seats" );
    }
}

public class Main{
    public static void main(String args[]){
        Car c1 = new Car("black", 2); // creating object

        c1.drive(); // calling method

    }
}
```

Constructor isn't called ?

Module – 2

- Encapsulation
- Inheritance
- Polymorphism
- Abstraction



1. Inheritance

```java
// parent class
class Animal {
    void eat() {
        System.out.println(x:"This animal eats food.");
    }
}

// child class
class Dog extends Animal { // Inheriting from Animal
    void bark() {
        System.out.println(x:"The dog barks.");
    }
}

// Main class
public class Ani{
    Run main | Debug main | Run | Debug
    public static void main(String args[]){

        Dog d1 = new Dog();

        d1.eat();
        d1.bark();

    }
}
```

Inheritance allows one class (child) to inherit the properties and methods of another class (parent). It promotes code reuse.

What is inherited in the program ?

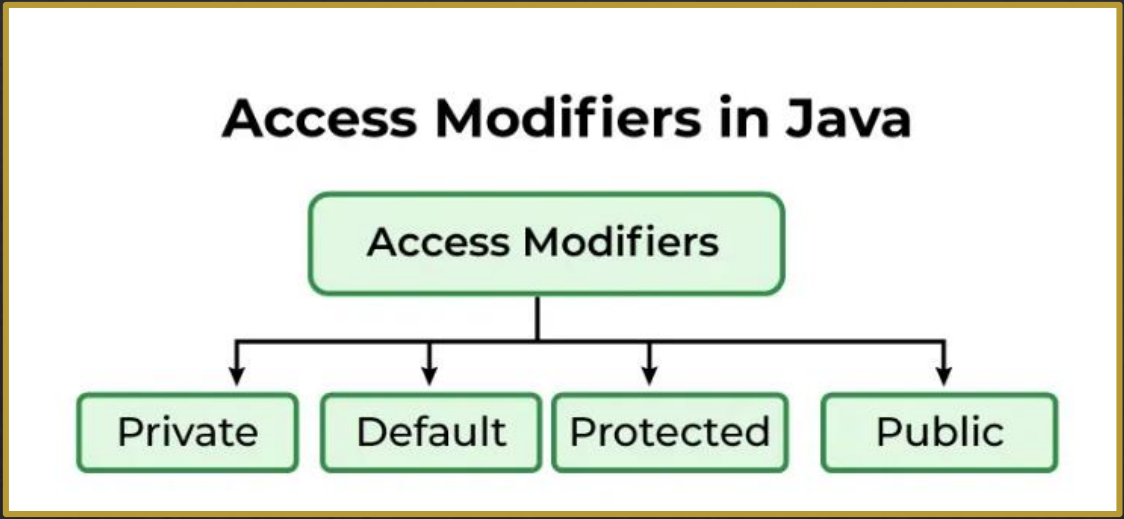
2. Encapsulation

```java
// parent class
class Animal {
    void eat() {
        System.out.println(x:"This animal eats food.");
    }
}

// child class
class Dog extends Animal { // Inheriting from Animal
    void bark() {
        System.out.println(x:"The dog barks.");
    }
}

// Main class
public class Ani{
    Run main | Debug main | Run | Debug
    public static void main(String args[]){

        Dog d1 = new Dog();

        d1.eat();
        d1.bark();

    }
}
```

Encapsulation is the concept of bundling data (attributes) and methods together and restricting direct access to some components (using access modifiers like private, protected, and public).

- Public
- Private
- Protected

Encapsulation



- ✓ Getters
- ✓ Setters

Access Modifiers

Modifier	Class	Package	Subclass	Global
Public	✓	✓	✓	✓
Protected	✓	✓	✓	✗
Default	✓	✓	✗	✗
Private	✓	✗	✗	✗

3. Polymorphism

Polymorphism is a fundamental concept in Object-Oriented Programming (OOP) that allows objects to take on many forms. In simple terms, it means that a single function, method, or operator can work differently based on the context, such as the type of object or data.

There are two main types of polymorphism in programming:

1. **Compile-time polymorphism** (Method Overloading)
2. **Runtime polymorphism** (Method Overriding)

4. Abstraction

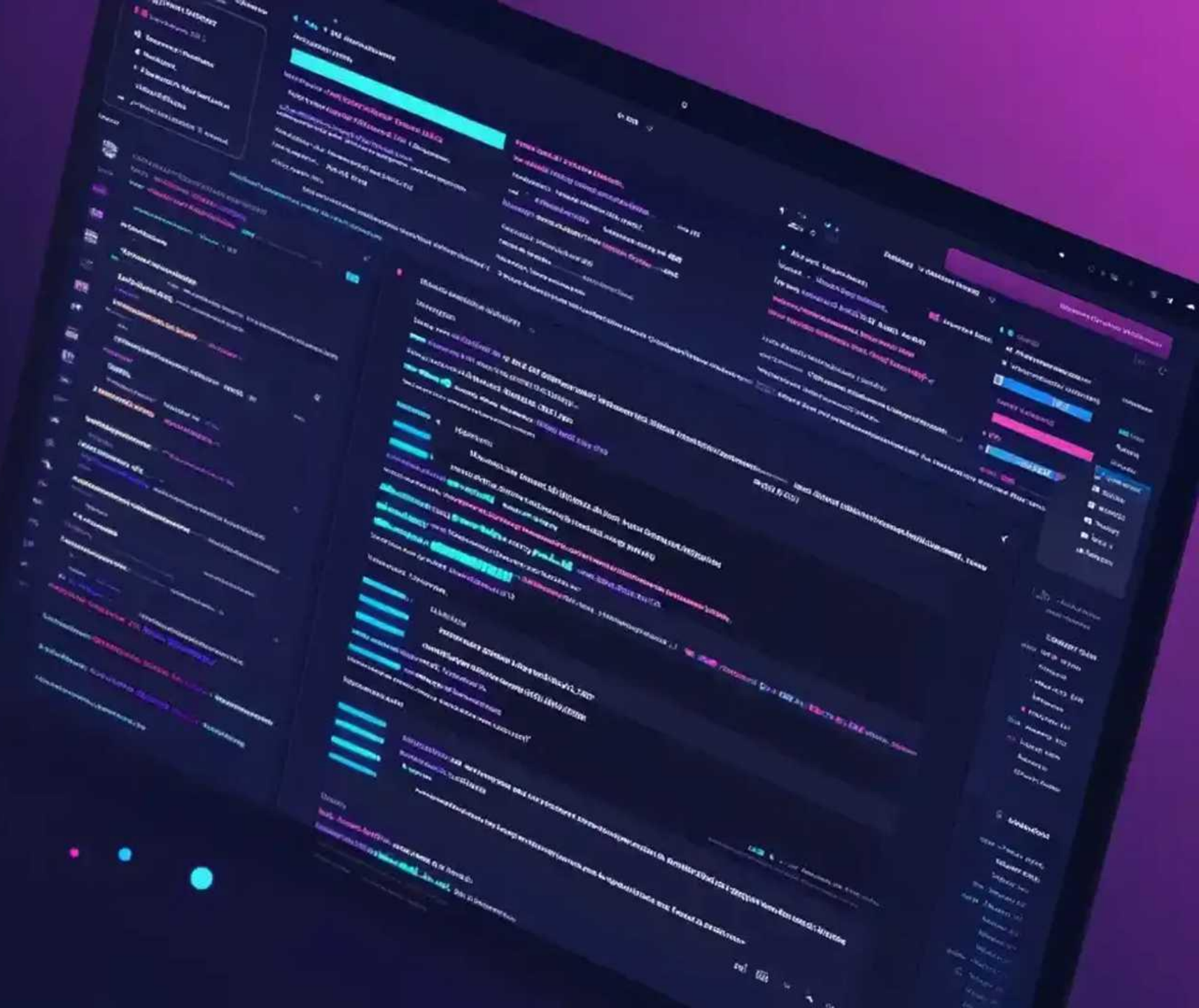
Abstraction is a core concept in object-oriented programming (OOP). It is the process of hiding the internal implementation details of a system and exposing only the necessary features to the user. This makes complex systems easier to understand and work with, by focusing on "what it does" rather than "how it does it."

Key Points of Abstraction:

1. It hides the implementation details and shows only essential information.
2. It reduces complexity and increases efficiency.
3. Achieved in OOP using **abstract classes** and **interfaces**.

Module – 3

- Interface (implement Keyword)
- Method Signature
- Class & Interface are blueprint (diff)



Interface

An **interface** is a blueprint for a class. It defines a set of methods that a class must implement. Interfaces are used to achieve abstraction and multiple inheritance in Java.

Key Points About Interfaces:

1. An interface contains only method signatures (no method body) and constants (final variables).
2. A class implements an interface by using the implements keyword and providing the method bodies.

Interface Code

```java
// Define an interface
interface Animal {
    // Abstract methods (no body)
    void eat();
    void sleep();
}
```

```java
// A class that implements the Animal interface
class Dog implements Animal {
    // Providing implementation for the methods
    @Override
    public void eat() {
        System.out.println("The dog eats bones.");
    }

    @Override
    public void sleep() {
        System.out.println("The dog sleeps in a kennel.");
    }
}
```

```java
class Cat implements Animal {
    // Providing implementation for the methods
    @Override
    public void eat() {
        System.out.println("The cat eats fish.");
    }

    @Override
    public void sleep() {
        System.out.println("The cat sleeps on the couch.");
    }
}
```

```java
// Main class to test the code
public class Main {
    public static void main(String[] args) {
        // Create objects of Dog and Cat
        Animal dog = new Dog();
        Animal cat = new Cat();

        // Call the methods
        dog.eat();
        dog.sleep();

        cat.eat();
        cat.sleep();
    }
}
```