☕ Java and OOP Review

For High School IT Students

> Duration: 90 minutes
>
> Language: Java Programming

🎯 Learning Objectives

By the end of this session, you will be able to:

- Understand **OOP (Object-Oriented Programming)** concepts
- Create and use **classes** and **objects** in Java
- Apply **Encapsulation, Inheritance, Polymorphism**
- Use **Interfaces, Packages, and Exception Handling**
- Compare Java OOP with other languages (like Python or C++)

🧠 What is OOP?

OOP = *Object-Oriented Programming*

It's a way of organizing programs using **objects**.

Concept	**Meaning**
Class	The "blueprint" or design
Object	An actual thing created from the class
Encapsulation	Hiding internal details
Inheritance	Reusing and extending code
Polymorphism	Many forms of behavior

☕ Quick Java Recap

- Java is a **Strongly Typed**, fully **Object-Oriented** language.
- Every program starts with a `main()` method.
- Compile first, then run:

```
javac Main.java
java Main
```

📦 Class and Object Example

👉 `Student` is a **class**, and `s` is an **object**.

```
class Student {
    String name;
    int score;
    void showInfo() {
        System.out.println(name + " got " + score + " points");
    }
}

public class Main {
    public static void main(String[] args) {
        Student s = new Student();
        s.name = "Alice";
        s.score = 90;
        s.showInfo();
    }
}
```

🔒 Encapsulation: Protect Your Data

- Keep data **safe** and **controlled**
- Use **private** fields + **getter/setter** methods

```
class Student {
    private int score;

    public void setScore(int s) {
        if (s >= 0 && s <= 100)
            score = s;
        else
            System.out.println("Invalid score!");
    }

    public int getScore() {
        return score;
    }
}
```

🧬 Inheritance: Reuse and Extend

- One class can **inherit** another's features
- Use the keyword `extends`

```java
class Animal {
    void speak() {
        System.out.println("Some sound...");
    }
}

class Dog extends Animal {
    void speak() {
        System.out.println("Woof!");
    }
}
```

🐶 Dog inherits from Animal.

🧠 Polymorphism: Many Forms

- Same method, different behavior
- Achieved through **method overriding**

```java
public class Main {
    public static void main(String[] args) {
        Animal a = new Dog(); // Reference type: Animal
        a.speak();            // Output depends on the object
    }
}
```

Output:

```
Woof!
```

⚙ Interface: The Contract

- Defines **what to do**, not **how to do it**
- Classes must **implement** all interface methods

```
interface Playable {
    void play();
}

class Dog implements Playable {
    public void play() {
        System.out.println("Dog plays fetch!");
    }
}
```

🔗 Interfaces connect unrelated classes by common behavior.

🧱 Package: Organize Your Code

Folder structure example:

```
src/
 └── animals/
      ├── Animal.java
      └── Dog.java
```

In code:

```java
package animals;

public class Dog extends Animal { ... }
```

And to use it:

```java
import animals.Dog;
```

⚠️ Exception Handling: Handle Errors Safely

- Prevent your program from **crashing**
- Use `try`, `catch`, and `finally`

```java
public class Demo {
    public static void main(String[] args) {
        try {
            int result = 10 / 0;
        } catch (ArithmeticException e) {
            System.out.println("Cannot divide by zero!");
        } finally {
            System.out.println("Done.");
        }
    }
}
```

🔄 Two Types of Exceptions

Type	Example	Must Handle?
Checked	IOException, SQLException	✅ Yes
Unchecked	NullPointerException, ArithmeticException	❌ No (but recommended)

🧰 Full Example

```java
interface Playable {
    void play();
}

class Animal {
    public void speak() { System.out.println("..."); }
}

class Dog extends Animal implements Playable {
    private String name;

    Dog(String name) { this.name = name; }

    @Override
    public void speak() { System.out.println(name + " says Woof!"); }

    public void play() { System.out.println(name + " plays fetch!"); }
}

public class Main {
    public static void main(String[] args) {
        try {
            Dog d = new Dog("Bobby");
            d.speak();
            d.play();
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

📚 OOP Key Concepts Recap

Concept	Description
Class & Object	Blueprint and instance
Encapsulation	Hide internal data
Inheritance	Reuse and extend classes
Polymorphism	One interface, many behaviors
Interface	Shared behavior across classes
Exception Handling	Manage runtime errors

🧩 Mini Challenge

> Design a small “School Registration System”
>
> - Create `Student`, `Course`, and `Registration` classes
> - Use **Encapsulation** and **Inheritance**
> - Throw an Exception if a class is full

🧠 Think in **objects** — who interacts with whom?

❓ Quick Quiz

1. What keyword allows one class to inherit another?
2. What is the difference between a **class** and an **object**?
3. Why do we use **getter** and **setter** methods?
4. What is the role of an **interface**?
5. What happens when we divide by zero in Java?

🎓 Final Thought

> “Object-Oriented Programming isn’t just code —
> it’s a way to **think** about problems and solutions.”