Object-Oriented Programming in Java Training Module

A Comprehensive Guide for Beginners

Prepared by xAI Training Team May 07, 2025

Downloadable PDF version available for training purposes.

**Contents**

1. [Introduction to OOP in Java](#_bookmark0) 2
   1. [Why Use OOP in Java?](#_bookmark1) 2
   2. [Setting Up the Environment](#_bookmark2) 2
2. [Classes and Objects](#_bookmark3) 2
   1. [Creating a Class](#_bookmark4) 2
3. [Encapsulation](#_bookmark5) 3
   1. [Using Getters and Setters](#_bookmark6) 3
4. [Inheritance](#_bookmark7) 4
   1. [Single Inheritance Example](#_bookmark8) 4
5. [Polymorphism](#_bookmark9) 4
   1. [Method Overriding](#_bookmark10) 4
6. [Abstraction](#_bookmark11) 5
   1. [Using Abstract Classes](#_bookmark12) 5
7. [Interfaces](#_bookmark13) 6
   1. [Interface Example](#_bookmark14) 6
8. [Exception Handling in OOP](#_bookmark15) 7
   1. [Custom Exception Example](#_bookmark16) 7
9. [Conclusion](#_bookmark17) 8
10. [References](#_bookmark18) 8

1

# Introduction to OOP in Java

Object-Oriented Programming (OOP) is a paradigm that organizes code into ob- jects, promoting modularity and reusability. Java, a fully object-oriented lan- guage, implements OOP through classes and objects. This module covers core OOP concepts—encapsulation, inheritance, polymorphism, and abstraction—with practical Java examples.

## Why Use OOP in Java?

* + - **Modularity**: Encapsulates data and behavior in objects.
    - **Reusability**: Inheritance and polymorphism reduce code duplication.
    - **Scalability**: Facilitates large-scale application development.

## Setting Up the Environment

Install the Java Development Kit (JDK) from Oracle’s website. Use an IDE like IntelliJ IDEA or Eclipse. Verify installation with:

java -version

1

# Classes and Objects

Classes are blueprints for objects, which are instances of classes.

## Creating a Class

Below is a simple class representing a Person.

**public class** Person { String name;

**int** age;

**public** Person(String name, **int** age) {

**this**.name = name;

**this**.age = age;

}

**public void** displayInfo() {

System.out.println(”Name: ” + name + ”, Age: ” + age);

}

**public static void** main(String[] args) { Person person = **new** Person(”Alice”, 25); person.displayInfo();

}

}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

**Explanation**:

* class Person: Defines the class.
* this.name: Refers to the instance variable.
* new Person: Creates an object.

# Encapsulation

Encapsulation hides data and exposes it through methods, ensuring data in- tegrity.

## Using Getters and Setters

1

**public class** BankAccount { **private double** balance; **private** String accountHolder;

**public** BankAccount(String accountHolder, **double** initialBalance)

{

**this**.accountHolder = accountHolder;

**this**.balance = initialBalance;

}

**public double** getBalance() {

**return** balance;

}

**public void** setBalance(**double** balance) {

**if** (balance >= 0) {

**this**.balance = balance;

}

}

**public** String getAccountHolder() {

**return** accountHolder;

}

**public static void** main(String[] args) {

BankAccount account = **new** BankAccount(”Bob”, 1000.0); account.setBalance(1500.0); System.out.println(”Account Holder: ” + account.

getAccountHolder());

System.out.println(”Balance: $” + account.getBalance());

}

}

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

# Inheritance

Inheritance allows a class to inherit properties and methods from another.

## Single Inheritance Example

1

**class** Vehicle { String brand;

Vehicle(String brand) {

**this**.brand = brand;

}

**void** honk() {

System.out.println(”Beep!”);

}

}

**class** Car **extends** Vehicle {

**int** wheels;

Car(String brand, **int** wheels) {

**super**(brand); **this**.wheels = wheels;

}

**void** displayInfo() {

System.out.println(”Brand: ” + brand + ”, Wheels: ” + wheels

);

}

}

**public class** InheritanceExample {

**public static void** main(String[] args) { Car car = **new** Car(”Toyota”, 4); car.honk();

car.displayInfo();

}

}

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

# Polymorphism

Polymorphism allows objects to be treated as instances of their parent class, with method overriding for specific behavior.

## Method Overriding

1

**class** Animal {

**void** makeSound() {

System.out.println(”Some generic sound”);

}

}

**class** Dog **extends** Animal { @Override

**void** makeSound() { System.out.println(”Woof!”);

}

}

**class** Cat **extends** Animal { @Override

**void** makeSound() { System.out.println(”Meow!”);

}

}

**public class** PolymorphismExample {

**public static void** main(String[] args) { Animal dog = **new** Dog();

Animal cat = **new** Cat(); dog.makeSound(); // Woof! cat.makeSound(); // Meow!

}

}

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

# Abstraction

Abstraction hides implementation details, exposing only essential features.

## Using Abstract Classes

1

**abstract class** Shape {

**abstract double** calculateArea();

}

**class** Circle **extends** Shape {

**double** radius;

Circle(**double** radius) {

**this**.radius = radius;

}

@Override

**double** calculateArea() {

2

3

4

5

6

7

8

9

10

11

12

13

14

**return** Math.PI \* radius \* radius;

}

}

**class** Rectangle **extends** Shape {

**double** width, height;

Rectangle(**double** width, **double** height) {

**this**.width = width;

**this**.height = height;

}

@Override

**double** calculateArea() {

**return** width \* height;

}

}

**public class** AbstractionExample {

**public static void** main(String[] args) { Shape circle = **new** Circle(5);

Shape rectangle = **new** Rectangle(4, 6); System.out.println(”Circle Area: ” + circle.calculateArea())

;

System.out.println(”Rectangle Area: ” + rectangle. calculateArea());

}

}

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

# Interfaces

Interfaces define contracts for classes to implement.

## Interface Example

1

**interface** Printable {

**void** print();

}

**class** Document **implements** Printable { String content;

Document(String content) {

**this**.content = content;

}

@Override

**public void** print() { System.out.println(”Printing: ” + content);

2

3

4

5

6

7

8

9

10

11

12

13

14

15

}

}

**public class** InterfaceExample {

**public static void** main(String[] args) {

Printable doc = **new** Document(”Sample Document”); doc.print();

}

}

16

17

18

19

20

21

22

23

# Exception Handling in OOP

Handle errors in OOP applications using try-catch.

## Custom Exception Example

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6 | **class** InsufficientFundsException **extends** Exception {  **public** InsufficientFundsException(String message) {  **super**(message);  }  } | |
| 7 | **class** Bank { |  |
| 8 | **private double** balance; |  |
| 9 |  |  |
| 10 | **public** Bank(**double** balance) { |  |
| 11 | **this**.balance = balance; |  |
| 12 | } |  |
| 13 |  |  |
| 14 | **public void** withdraw(**double** amount) **throws** |  |
|  | InsufficientFundsException { |  |
| 15 | **if** (amount > balance) { |  |
| 16 | **throw new** InsufficientFundsException(”Insufficient | funds |
|  | !”); | |
| 17 | } | |
| 18 | balance -= amount; | |
| 19 | System.out.println(”Withdrawal successful. New balance: $” + | |
|  | balance); | |
| 20 | } | |
| 21 | } | |
| 22 |  | |
| 23 | **public class** ExceptionExample { | |
| 24 | **public static void** main(String[] args) { | |
| 25 | Bank bank = **new** Bank(1000); | |
| 26 | **try** { | |
| 27 | bank.withdraw(1500); | |
| 28  29 | } **catch** (InsufficientFundsException e) { System.out.println(”Error: ” + e.getMessage()); | |
| 30 | } | |

31

}

}

32

# Conclusion

This module covers OOP principles in Java—classes, encapsulation, inheritance, polymorphism, abstraction, and interfaces. Practice these examples and explore advanced topics like design patterns and Java frameworks.

# References

* Oracle Java Documentation: <https://docs.oracle.com/javase>
* Java Tutorials by Oracle: <https://docs.oracle.com/javase/tutorial>