

# 1. Person Class

---

## Aim

The goal is to create a simple Java class named **Person** with name and age attributes. We'll then create two instances of this class using a constructor to set their attributes and print the values to the console.

## Algorithm

1. Define a class called Person.
2. Declare two instance variables within the class: a String for the name and an int for the age.
3. Create a **constructor** for the Person class that accepts name and age as parameters and initializes the instance variables.
4. In the main method, create the first Person object, let's call it person1, and pass "Alice" and 30 to its constructor.
5. Create a second Person object, person2, and pass "Bob" and 25 to its constructor.
6. Print the name and age of both person1 and person2 to the console.

## Pseudocode

```
CLASS Person
    name: String
    age: Integer
```

```
    CONSTRUCTOR(name, age)
```

```
this.name = name  
this.age = age
```

END CLASS

MAIN

```
// Create first person  
person1 = NEW Person("Alice", 30)  
  
// Create second person  
person2 = NEW Person("Bob", 25)  
  
// Print details  
PRINT "Person 1: " + person1.name + ", " + person1.age  
PRINT "Person 2: " + person2.name + ", " + person2.age  
END MAIN
```

## Code

Java

```
class Person {  
    String name;  
    int age;  
  
    public Person(String name, int age) {  
        this.name = name;  
        this.age = age;  
    }  
}  
  
public class Main {  
    public static void main(String[] args) {  
        // Create two instances of the Person class  
        Person person1 = new Person("Alice", 30);  
        Person person2 = new Person("Bob", 25);  
    }  
}
```

```
// Print their name and age
System.out.println("Person 1: " + person1.name + ", " + person1.age);
System.out.println("Person 2: " + person2.name + ", " + person2.age);
}
}
```

## Output

```
Person 1: Alice, 30
Person 2: Bob, 25
```

---

## 2. Rectangle Class

---

### Aim

The aim is to design a Java class named **Rectangle** with attributes for width and height. The class should have methods to calculate and return the area and perimeter of the rectangle.

### Algorithm

1. Create a class named Rectangle.
2. Declare two double instance variables: width and height.
3. Implement a constructor that takes width and height as arguments to initialize the object.
4. Define a public method `getArea()` that calculates the area using the formula  $\text{width} * \text{height}$  and returns the result.
5. Define a public method `getPerimeter()` that calculates the perimeter using the formula  $2 * (\text{width} + \text{height})$ .

- (width + height) and returns the result.
6. In the main method, create a Rectangle object with specific dimensions.
  7. Call the `getArea()` and `getPerimeter()` methods on the object and print the results.

## Pseudocode

```
CLASS Rectangle
```

```
    width: Double
```

```
    height: Double
```

```
    CONSTRUCTOR(width, height)
```

```
        this.width = width
```

```
        this.height = height
```

```
    METHOD getArea()
```

```
        RETURN width * height
```

```
    METHOD getPerimeter()
```

```
        RETURN 2 * (width + height)
```

```
END CLASS
```

```
MAIN
```

```
    rect = NEW Rectangle(5.0, 10.0)
```

```
    area = rect.getArea()
```

```
    perimeter = rect.getPerimeter()
```

```
    PRINT "Rectangle Area: " + area
```

```
    PRINT "Rectangle Perimeter: " + perimeter
```

```
END MAIN
```

## Code

Java

```
class Rectangle {  
    double width;  
    double height;  
  
    public Rectangle(double width, double height) {  
        this.width = width;  
        this.height = height;  
    }  
  
    public double getArea() {  
        return width * height;  
    }  
  
    public double getPerimeter() {  
        return 2 * (width + height);  
    }  
}  
  
public class Main {  
    public static void main(String[] args) {  
        Rectangle rect = new Rectangle(5.0, 10.0);  
  
        System.out.println("Rectangle Area: " + rect.getArea());  
        System.out.println("Rectangle Perimeter: " + rect.getPerimeter());  
    }  
}
```

**Output**

Rectangle Area: 50.0  
Rectangle Perimeter: 30.0

---

### 3. Book Class

---

#### Aim

The objective is to create a **Book** class with title, author, and ISBN attributes. We'll also create a **BookCollection** class that uses a collection (like an ArrayList) to store Book objects and provides methods to add and remove them.

#### Algorithm

1. Define a class Book with String attributes for title, author, and ISBN. Include a constructor to initialize them.
2. Define a second class BookCollection.
3. Inside BookCollection, declare a private List<Book> to hold the book objects.
4. Implement a public method addBook(Book book) that adds the provided book to the list.
5. Implement a public method removeBook(String isbn) that iterates through the list, finds the book with the matching ISBN, and removes it.
6. In the main method, create a BookCollection object.
7. Create a few Book objects and add them to the collection using the addBook method.
8. Call the removeBook method with an ISBN to remove a specific book.

#### Pseudocode

CLASS Book

```
title: String
author: String
isbn: String
```

```
CONSTRUCTOR(title, author, isbn)
```

```
...
```

```
CLASS BookCollection
```

```
books: List of Book
```

```
METHOD addBook(book)
```

```
books.add(book)
```

```
METHOD removeBook(isbn)
```

```
FOR each book in books
```

```
IF book.isbn == isbn
```

```
books.remove(book)
```

```
RETURN
```

```
MAIN
```

```
library = NEW BookCollection()
```

```
book1 = NEW Book("The Great Gatsby", "F. Scott Fitzgerald", "123")
```

```
book2 = NEW Book("1984", "George Orwell", "456")
```

```
library.addBook(book1)
```

```
library.addBook(book2)
```

```
library.removeBook("456")
```

## Code

```
Java
```

```
import java.util.ArrayList;
```

```
import java.util.List;
```

```
class Book {
```

```
String title;  
String author;  
String isbn;
```

```
public Book(String title, String author, String isbn) {  
    this.title = title;  
    this.author = author;  
    this.isbn = isbn;  
}  
}
```

```
class BookCollection {  
    private List<Book> books;
```

```
    public BookCollection() {  
        this.books = new ArrayList<>();  
    }
```

```
    public void addBook(Book book) {  
        books.add(book);  
        System.out.println("Book " + book.title + " added.");  
    }
```

```
    public void removeBook(String isbn) {  
        books.removeIf(book -> book.isbn.equals(isbn));  
        System.out.println("Book with ISBN " + isbn + " removed.");  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        BookCollection library = new BookCollection();  
        Book book1 = new Book("The Great Gatsby", "F. Scott Fitzgerald", "978-0743273565");  
        Book book2 = new Book("1984", "George Orwell", "978-0451524935");  
  
        library.addBook(book1);  
        library.addBook(book2);  
  
        System.out.println("\nRemoving a book...");  
        library.removeBook("978-0451524935");  
    }  
}
```

## Output

Book 'The Great Gatsby' added.

Book '1984' added.

Removing a book...

Book with ISBN '978-0451524935' removed.

---

## 4. Bank and Account Classes

---

### Aim

The aim is to create a **Bank** class that manages a collection of **Account** objects. The Account class will handle individual account details, including depositing and withdrawing money, while the Bank class will handle adding, removing, and finding accounts.

### Algorithm

1. Define an Account class with attributes: accountNumber, customerName, and balance.
2. In the Account class, create methods deposit(amount) and withdraw(amount). The withdraw method should check for sufficient balance.
3. Define a Bank class with a List<Account> to store accounts.
4. Implement addAccount(Account account), removeAccount(accountNumber), and a helper method findAccount(accountNumber) in the Bank class.
5. In the main method, create a Bank object, add a new Account, find it, and then perform deposit and withdrawal operations.

## Pseudocode

CLASS Account

    accountNumber: String

    balance: Double

    customerName: String

METHOD deposit(amount)

    IF amount > 0

        balance = balance + amount

METHOD withdraw(amount)

    IF amount > 0 AND amount <= balance

        balance = balance - amount

CLASS Bank

    accounts: List of Account

METHOD addAccount(account)

    accounts.add(account)

METHOD removeAccount(accountNumber)

    // Find and remove account

METHOD findAccount(accountNumber)

    // Find and return account by number

MAIN

    bank = NEW Bank()

    acc = NEW Account("001", "John Doe")

    bank.addAccount(acc)

    foundAcc = bank.findAccount("001")

    IF foundAcc IS NOT NULL

        foundAcc.deposit(100)

```
foundAcc.withdraw(50)
PRINT "Current balance: " + foundAcc.balance
```

## Code

Java

```
import java.util.ArrayList;
import java.util.List;

class Account {
    String accountNumber;
    double balance;
    String customerName;

    public Account(String accountNumber, String customerName) {
        this.accountNumber = accountNumber;
        this.customerName = customerName;
        this.balance = 0.0;
    }

    public void deposit(double amount) {
        if (amount > 0) {
            this.balance += amount;
            System.out.println("Deposited: $" + amount);
        } else {
            System.out.println("Deposit amount must be positive.");
        }
    }

    public void withdraw(double amount) {
        if (amount > 0 && amount <= this.balance) {
            this.balance -= amount;
            System.out.println("Withdrew: $" + amount);
        } else if (amount > this.balance) {
            System.out.println("Insufficient balance.");
        }
    }
}
```

```

    } else {
        System.out.println("Withdrawal amount must be positive.");
    }
}
}

class Bank {
    List<Account> accounts;

    public Bank() {
        this.accounts = new ArrayList<>();
    }

    public void addAccount(Account account) {
        accounts.add(account);
        System.out.println("Account for " + account.customerName + " added.");
    }

    public Account findAccount(String accountNumber) {
        for (Account account : accounts) {
            if (account.accountNumber.equals(accountNumber)) {
                return account;
            }
        }
        return null;
    }
}

public class Main {
    public static void main(String[] args) {
        Bank myBank = new Bank();
        Account johnsAccount = new Account("ACC001", "John Doe");
        myBank.addAccount(johnsAccount);

        Account foundAccount = myBank.findAccount("ACC001");
        if (foundAccount != null) {
            System.out.println("\nInitial balance for " + foundAccount.customerName + ": $" +
foundAccount.balance);
            foundAccount.deposit(500.00);
            foundAccount.withdraw(150.50);
            System.out.println("Final balance for " + foundAccount.customerName + ": $" +
foundAccount.balance);
        }
    }
}

```

```
}  
}
```

## Output

Account for John Doe added.

Initial balance for John Doe: \$0.0

Deposited: \$500.0

Withdrew: \$150.5

Final balance for John Doe: \$349.5

---

## 5. Employee Class

---

### Aim

The goal is to create an **Employee** class with attributes for name, salary, and hire date. The class will also include a method to calculate the number of years the employee has been with the company.

### Algorithm

1. Define an Employee class with a String for name, a double for salary, and an int for hireYear.
2. Implement a constructor to set these attributes.
3. Create a public method getYearsOfService(int currentYear) that takes the current year as an argument.

4. Inside the method, calculate the years of service by subtracting hireYear from currentYear.
5. In the main method, instantiate an Employee object.
6. Call the getYearsOfService method, passing in the current year, and print the result.

## Pseudocode

```
CLASS Employee
  name: String
  salary: Double
  hireYear: Integer

  CONSTRUCTOR(name, salary, hireYear)
  ...

  METHOD getYearsOfService(currentYear)
    RETURN currentYear - this.hireYear

END CLASS

MAIN
  employee = NEW Employee("Jane Doe", 75000.0, 2018)
  years = employee.getYearsOfService(2024)
  PRINT "Years of service: " + years
```

## Code

Java

```
class Employee {
```

```

String name;
double salary;
int hireYear;

public Employee(String name, double salary, int hireYear) {
    this.name = name;
    this.salary = salary;
    this.hireYear = hireYear;
}

public int getYearsOfService(int currentYear) {
    return currentYear - hireYear;
}
}

public class Main {
    public static void main(String[] args) {
        Employee emp = new Employee("Michael", 60000.00, 2015);
        int currentYear = 2025;
        int yearsOfService = emp.getYearsOfService(currentYear);

        System.out.println("Employee: " + emp.name);
        System.out.println("Hired in: " + emp.hireYear);
        System.out.println("Years of service: " + yearsOfService);
    }
}

```

## Output

```

Employee: Michael
Hired in: 2015
Years of service: 10

```

---

## 6. Read from File and Throw Exception

---

### Aim

The goal is to write a Java program that reads a list of numbers from a file and throws a **custom exception** if any of the numbers are positive.

### Algorithm

1. Define a custom exception class, `PositiveNumberException`, that extends `Exception`.
2. Create a helper method to write a file with a mix of positive and negative numbers for testing.
3. Create a main method `readAndCheckNumbers(String fileName)` that takes the file name as a parameter.
4. Inside this method, use a `Scanner` to read integers from the file.
5. Use a while loop to process each number.
6. Inside the loop, use an if statement to check if the number is greater than 0. If it is, throw a new `PositiveNumberException`.
7. In the main method, call `readAndCheckNumbers` inside a try-catch block to handle the potential exception and print an informative message.

### Pseudocode

```
CLASS PositiveNumberException EXTENDS Exception
```

```
METHOD writeTestFile(fileName, content)
```

```
    // Writes numbers to a file for testing
```

```
METHOD readAndCheckNumbers(fileName)
```

```

file = NEW File(fileName)
scanner = NEW Scanner(file)
WHILE scanner has next integer
    number = scanner.nextInt()
    IF number > 0
        THROW NEW PositiveNumberException("Found positive number: " + number)

```

```

MAIN
TRY
    writeTestFile("numbers.txt", "-10\n5\n-3")
    readAndCheckNumbers("numbers.txt")
CATCH PositiveNumberException e
    PRINT "Exception caught: " + e.getMessage()
CATCH FileNotFoundException e
    PRINT "Error: File not found."

```

## Code

Java

```

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;

class PositiveNumberException extends Exception {
    public PositiveNumberException(String message) {
        super(message);
    }
}

public class Main {
    public static void main(String[] args) {
        String fileName = "test_numbers.txt";
    }
}

```

```

try {
    // Helper to create a test file with a positive number
    createTestFile(fileName, "-10\n-5\n20\n-3");

    System.out.println("Attempting to read numbers from file...");
    readAndCheckNumbers(fileName);

} catch (FileNotFoundException e) {
    System.out.println("Error: The file was not found.");
} catch (PositiveNumberException e) {
    System.out.println("SUCCESSFULLY CAUGHT EXCEPTION: " + e.getMessage());
} catch (IOException e) {
    System.out.println("An error occurred creating the file.");
}
}

private static void createTestFile(String fileName, String content) throws IOException {
    FileWriter writer = new FileWriter(fileName);
    writer.write(content);
    writer.close();
}

private static void readAndCheckNumbers(String fileName) throws FileNotFoundException,
PositiveNumberException {
    File file = new File(fileName);
    Scanner fileScanner = new Scanner(file);

    while (fileScanner.hasNextInt()) {
        int number = fileScanner.nextInt();
        System.out.println("Checking number: " + number);
        if (number > 0) {
            fileScanner.close();
            throw new PositiveNumberException("Found a positive number: " + number);
        }
    }

    fileScanner.close();
    System.out.println("All numbers in the file are non-positive.");
}
}

```

## Output

```
Attempting to read numbers from file...
Checking number: -10
Checking number: -5
Checking number: 20
SUCCESSFULLY CAUGHT EXCEPTION: Found a positive number: 20
```

---

## 7. Read Integers and Throw Exception for Duplicates

---

### Aim

The aim is to read a list of integers from the user and throw a **custom exception** if a duplicate number is found.

### Algorithm

1. Define a custom exception class, DuplicateNumberException, that extends Exception.
2. Create a method, checkDuplicateNumbers(String input), that takes a string of space-separated numbers.
3. Inside this method, use a Scanner to parse the string and a HashSet<Integer> to store numbers we've already seen. HashSet is perfect because it only stores unique elements.
4. In a while loop, read each integer from the scanner.
5. Attempt to add the number to the HashSet. The add method returns false if the element already exists.
6. If add returns false, a duplicate has been found. throw a DuplicateNumberException with an informative message.
7. In the main method, simulate user input and call the method within a try-catch block.

## Pseudocode

```
CLASS DuplicateNumberException EXTENDS Exception
```

```
METHOD checkDuplicateNumbers(inputString)
```

```
    seenNumbers = NEW HashSet()
```

```
    scanner = NEW Scanner(inputString)
```

```
    WHILE scanner has next integer
```

```
        number = scanner.nextInt()
```

```
        IF NOT seenNumbers.add(number)
```

```
            THROW NEW DuplicateNumberException("Duplicate number found: " + number)
```

```
MAIN
```

```
    input = "1 2 3 5 2 4"
```

```
    TRY
```

```
        checkDuplicateNumbers(input)
```

```
    CATCH DuplicateNumberException e
```

```
        PRINT "Exception caught: " + e.getMessage()
```

## Code

```
Java
```

```
import java.util.HashSet;
```

```
import java.util.Scanner;
```

```
import java.util.Set;
```

```
class DuplicateNumberException extends Exception {  
    public DuplicateNumberException(String message) {
```

```

        super(message);
    }
}

public class Main {
    public static void main(String[] args) {
        String inputWithDuplicates = "10 20 30 20 50";
        String inputWithoutDuplicates = "1 2 3 4 5";

        System.out.println("Checking for duplicates in: " + inputWithDuplicates);
        try {
            checkDuplicateNumbers(inputWithDuplicates);
        } catch (DuplicateNumberException e) {
            System.out.println("SUCCESSFULLY CAUGHT EXCEPTION: " + e.getMessage());
        }

        System.out.println("\nChecking for duplicates in: " + inputWithoutDuplicates);
        try {
            checkDuplicateNumbers(inputWithoutDuplicates);
        } catch (DuplicateNumberException e) {
            System.out.println("SUCCESSFULLY CAUGHT EXCEPTION: " + e.getMessage());
        }
    }

    public static void checkDuplicateNumbers(String input) throws DuplicateNumberException {
        Scanner lineScanner = new Scanner(input);
        Set<Integer> numbers = new HashSet<>();
        while (lineScanner.hasNextInt()) {
            int number = lineScanner.nextInt();
            if (!numbers.add(number)) {
                lineScanner.close();
                throw new DuplicateNumberException("Duplicate number found: " + number);
            }
        }
        lineScanner.close();
        System.out.println("No duplicate numbers found.");
    }
}

```

## Output

Checking for duplicates in: 10 20 30 20 50  
SUCCESSFULLY CAUGHT EXCEPTION: Duplicate number found: 20

Checking for duplicates in: 1 2 3 4 5  
No duplicate numbers found.

---

## 8. Check String for Vowels and Throw Exception

---

### Aim

The aim is to create a Java method that takes a String as input and throws a **custom exception** if the string contains no vowels (a, e, i, o, u, case-insensitive).

### Algorithm

1. Define a custom exception class, NoVowelsException, that extends Exception.
2. Create a public static method, checkStringForVowels(String str).
3. Inside the method, convert the input string to lowercase to simplify the check.
4. Iterate through each character of the string.
5. Use an if condition to check if the character is one of 'a', 'e', 'i', 'o', or 'u'.
6. If a vowel is found, a flag can be set to true and the loop can be broken, as the condition has been met.
7. After the loop, if the flag is still false, it means no vowels were found, so throw a NoVowelsException.
8. In the main method, call the checkStringForVowels method with both a string that has vowels and one that doesn't, using try-catch blocks to handle the exception.

## Pseudocode

```
CLASS NoVowelsException EXTENDS Exception

METHOD checkStringForVowels(str)
    hasVowels = FALSE
    vowels = "aeiou"
    FOR each character in str
        IF vowels contains character (case-insensitive)
            hasVowels = TRUE
            BREAK

    IF NOT hasVowels
        THROW NEW NoVowelsException("String has no vowels.")

MAIN
    TRY
        checkStringForVowels("rhythm")
    CATCH NoVowelsException e
        PRINT "Exception caught: " + e.getMessage()

    TRY
        checkStringForVowels("hello")
    CATCH NoVowelsException e
        PRINT e.getMessage() // This block will not be reached
```

## Code

Java

```

class NoVowelsException extends Exception {
    public NoVowelsException(String message) {
        super(message);
    }
}

```

```

public class Main {
    public static void main(String[] args) {
        String strWithoutVowels = "rhythm";
        String strWithVowels = "hello world";

        System.out.println("Checking string: \"" + strWithoutVowels + "\"");
        try {
            checkStringForVowels(strWithoutVowels);
            System.out.println("The string contains vowels."); // This line will not be executed
        } catch (NoVowelsException e) {
            System.out.println("SUCCESSFULLY CAUGHT EXCEPTION: " + e.getMessage());
        }
    }
}

```

```

        System.out.println("\nChecking string: \"" + strWithVowels + "\"");
        try {
            checkStringForVowels(strWithVowels);
            System.out.println("The string contains vowels.");
        } catch (NoVowelsException e) {
            System.out.println("SUCCESSFULLY CAUGHT EXCEPTION: " + e.getMessage()); // This line
will not be executed
        }
    }
}

```

```

    public static void checkStringForVowels(String str) throws NoVowelsException {
        boolean hasVowels = false;
        String vowels = "aeiouAEIOU";

        for (int i = 0; i < str.length(); i++) {
            char ch = str.charAt(i);
            if (vowels.indexOf(ch) != -1) {
                hasVowels = true;
                break;
            }
        }
    }
}

```

```

    if (!hasVowels) {

```

```
        throw new NoVowelsException("The string does not contain any vowels.");  
    }  
}  
}
```

## Output

Checking string: "rhythm"

SUCCESSFULLY CAUGHT EXCEPTION: The string does not contain any vowels.

Checking string: "hello world"

The string contains vowels.