Java Programming (4343203) – Winter 2024 Solution

Milav Dabgar

November 22, 2024

Question 1(a) [3 marks]

List out various Primitive data types in Java.

Solution

Java offers eight primitive data types for storing simple values directly in memory.

Table 1. Java Primitive Data Types

Data Type	Size	Description	Range
byte	8 bits	Integer type	-128 to 127
short	16 bits	Integer type	-32,768 to 32,767
int	32 bits	Integer type	-2^{31} to $2^{31} - 1$
long	64 bits	Integer type	-2^{63} to $2^{63} - 1$
float	32 bits	Floating-point	Single precision
double	64 bits	Floating-point	Double precision
char	16 bits	Character	Unicode characters
boolean	1 bit	Logical	true or false

Mnemonic

"BILFDC-B: Byte Int Long Float Double Char Boolean types"

Question 1(b) [4 marks]

Explain Structure of Java Program with suitable example.

Solution

Java program structure follows a specific organization with package declarations, imports, class definitions, and methods.

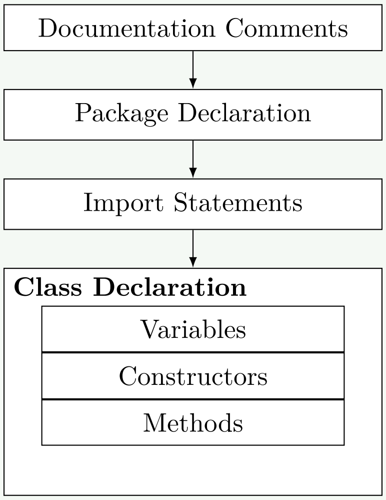

Figure 1. Java Program Structure

Listing 1. Java Program Structure Example

```java
// Documentation comment
/**
 * Simple program to demonstrate Java structure
 * @author GTU Student
 */

// Package declaration
package com.example;

// Import statements
import java.util.Scanner;

// Class declaration
public class HelloWorld {
    // Variable declaration
    private String message;

    // Constructor
    public HelloWorld() {
        message = "Hello, World!";
    }

    // Method
    public void displayMessage() {
        System.out.println(message);
    }

    // Main method
    public static void main(String[] args) {
        HelloWorld obj = new HelloWorld();
        obj.displayMessage();
    }
}
```

Mnemonic

"PICOM: Package Import Class Objects Methods in order"

Question 1(c) [7 marks]

List arithmetic operators in Java. Develop a Java program using any three arithmetic operators and show the output of program.

Solution

Arithmetic operators in Java perform mathematical operations on numeric values.

Table 2. Java Arithmetic Operators

Operator	**Description**	**Example**
+	Addition	a + b
-	Subtraction	a - b
*	Multiplication	a * b
/	Division	a / b
%	Modulus (Remainder)	a % b
++	Increment	a++ or ++a
–	Decrement	a– or –a

Listing 2. Arithmetic Operators Demo

```java
public class ArithmeticDemo {
    public static void main(String[] args) {
        int a = 10;
        int b = 3;

        // Addition
        int sum = a + b;

        // Multiplication
        int product = a * b;

        // Modulus
        int remainder = a % b;

        // Display results
        System.out.println("Values: a = " + a + ", b = " + b);
        System.out.println("Addition (a + b): " + sum);
        System.out.println("Multiplication (a * b): " + product);
        System.out.println("Modulus (a % b): " + remainder);
    }
}
```

Mnemonic

"SAME: Sum Addition Multiply Exponentiation basic operations"

Question 1(c OR) [7 marks]

Write syntax of Java for loop statement. Develop a Java program to find out prime number between 1 to 10.

Solution

The for loop in Java provides a compact way to iterate over a range of values.
Syntax:

Listing 3. For Loop Syntax

```java
for (initialization; condition; increment/decrement) {
    // statements to be executed
}
```

Listing 4. Prime Numbers Program

```java
public class PrimeNumbers {
    public static void main(String[] args) {
        System.out.println("Prime numbers between 1 and 10:");

        // Check each number from 1 to 10
        for (int num = 1; num <= 10; num++) {
            boolean isPrime = true;

            // Check if num is divisible by any number from 2 to num-1
            if (num > 1) {
                for (int i = 2; i < num; i++) {
                    if (num % i == 0) {
                        isPrime = false;
                        break;
                    }
                }

                // Print if prime
                if (isPrime) {
                    System.out.print(num + " ");
                }
            }
        }
    }
}
```

Mnemonic

"ICE: Initialize, Check, Execute steps of for loop"

Question 2(a) [3 marks]

List the differences between Procedure-Oriented Programming (POP) and Object-Oriented Programming (OOP).

Solution

Procedure-Oriented and Object-Oriented Programming represent fundamentally different programming paradigms.

Table 3. POP vs OOP

Feature	Procedure-Oriented	Object-Oriented
Focus	Functions/Procedures	Objects
Data	Separate from functions	Encapsulated in objects
Security	Less secure	More secure with access control
Inheritance	Not supported	Supported
Reusability	Less reusable	Highly reusable
Complexity	Simpler for small programs	Better for complex systems

- **Organization**: POP divides into functions; OOP groups into objects
- **Approach**: POP follows top-down; OOP follows bottom-up

Mnemonic

"FIOS: Functions In Objects Structure key difference"

Question 2(b) [4 marks]

Explain static keyword with example.

Solution

The `static` keyword in Java creates class-level members shared across all objects of that class.

Table 4. Uses of static Keyword

Use	Purpose	Example
static variable	Shared across all objects	`static int count;`
static method	Can be called without object	`static void display()`
static block	Executed when class loads	`static { // code }`
static nested class	Associated with outer class	`static class Inner {}`

Listing 5. Static Keyword Example

```java
public class Counter {
    // Static variable shared by all objects
    static int count = 0;

    // Instance variable unique to each object
    int instanceCount = 0;

    // Constructor
    Counter() {
        count++;         // Increments the shared count
        instanceCount++; // Increments this object's count
    }

    public static void main(String[] args) {
        Counter c1 = new Counter();
        Counter c2 = new Counter();
        Counter c3 = new Counter();

        System.out.println("Static count: " + Counter.count);
        System.out.println("c1's instance count: " + c1.instanceCount);
        System.out.println("c2's instance count: " + c2.instanceCount);
        System.out.println("c3's instance count: " + c3.instanceCount);
    }
}
```

Mnemonic

"CBMS: Class-level, Before objects, Memory single, Shared by all"

Question 2(c) [7 marks]

Define Constructor. List types of Constructors. Develop a java code to explain Parameterized constructor.

Solution

A constructor is a special method with the same name as its class, used to initialize objects when created.
Types of Constructors:

Table 5. Constructor Types in Java

Type	Description	Example
Default	No parameters, created by compiler	Student() {}
No-arg	Explicitly defined, no parameters	Student() { name = ``Unk''; }
Parameterized	Accepts parameters	Student(String n) { name = n; }
Copy	Creates object from another object	Student(Student s) { …}

Listing 6. Parameterized Constructor Example

```java
public class Student {
    // Instance variables
    private String name;
    private int age;
    private String course;

    // Parameterized constructor
    public Student(String name, int age, String course) {
        this.name = name;
        this.age = age;
        this.course = course;
    }

    // Method to display student details
    public void displayDetails() {
        System.out.println("Student Details:");
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("Course: " + course);
    }

    // Main method for demonstration
    public static void main(String[] args) {
        // Creating object using parameterized constructor
        Student student1 = new Student("John", 20, "Computer Science");
        student1.displayDetails();

        // Another student
        Student student2 = new Student("Lisa", 22, "Engineering");
        student2.displayDetails();
    }
}
```

Mnemonic

"IDCR: Initialize Data Create Ready objects"

Question 2(a OR) [3 marks]

List the basic OOP concepts in Java and explain any one.

Solution

Java implements Object-Oriented Programming through several fundamental concepts.

Table 6. Basic OOP Concepts in Java

Concept	Description
Encapsulation	Binding data and methods together
Inheritance	Creating new classes from existing ones
Polymorphism	One interface, multiple implementations
Abstraction	Hiding implementation details
Association	Relationship between objects

Encapsulation Example:

Listing 7. Encapsulation Example

```java
public class Person {
    // Private data - hidden from outside
    private String name;
    private int age;

    // Public methods - interface to access data
    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setAge(int age) {
        // Validation ensures data integrity
        if (age > 0 && age < 120) {
            this.age = age;
        } else {
            System.out.println("Invalid age");
        }
    }
}
```

- **Data Hiding**: Private variables inaccessible from outside
- **Controlled Access**: Through public methods (getters/setters)
- **Integrity**: Data validation ensures correct values

Mnemonic

"EIPA: Encapsulate Inherit Polymorphize Abstract"

Question 2(b OR) [4 marks]

Explain final keyword with example.

Solution

The `final` keyword in Java restricts changes to entities, creating constants, unchangeable methods, and non-inheritable classes.

Table 7. Uses of final Keyword

Use	**Effect**	**Example**
final variable	Cannot be modified	`final int MAX = 100;`
final method	Cannot be overridden	`final void display() {}`
final class	Cannot be extended	`final class Math {}`
final parameter	Cannot be changed	`void m(final int x) {}`

Listing 8. Final Keyword Usage

```java
public class FinalDemo {
    // Final variable (constant)
    final int MAX_SPEED = 120;

    // Final method cannot be overridden
    final void showLimit() {
        System.out.println("Speed limit: " + MAX_SPEED);
    }

    public static void main(String[] args) {
        FinalDemo car = new FinalDemo();
        car.showLimit();

        // car.MAX_SPEED = 150; // Compile error
    }
}

// Final class cannot be extended
final class MathUtil {
    public int square(int num) { return num * num; }
}
```

Mnemonic

"VMP: Variables Methods Permanence with final"

Question 2(c OR) [7 marks]

Write scope of java access modifier. Develop a java code to explain public modifier.

Solution

Access modifiers in Java control visibility and accessibility of classes, methods, and variables.

Table 8. Java Access Modifier Scope

Modifier	Class	Package	Subclass	World
private	✓	×	×	×
default	✓	✓	×	×
protected	✓	✓	✓	×
public	✓	✓	✓	✓

Listing 9. Public Modifier Example

```java
// File: PublicDemo.java
package com.example;

public class PublicDemo {
    public String message = "Hello, World!";

    public void displayMessage() {
        System.out.println(message);
    }
}

// File: Main.java
package com.test;
import com.example.PublicDemo;

public class Main {
    public static void main(String[] args) {
        // Creating object of class from different package
        PublicDemo demo = new PublicDemo();

        // Accessing public variable and method
        System.out.println("Message: " + demo.message);
        demo.displayMessage();
    }
}
```

Mnemonic

"CEPM: Class Everywhere Public Most accessible"

Question 3(a) [3 marks]

List out different types of inheritance and explain any one with example.

Solution

Inheritance enables a class to inherit attributes and behaviors from another class.

Table 9. Types of Inheritance in Java

Type	Description
Single	One class extends one class
Multilevel	Chain of inheritance (A→B→C)
Hierarchical	Multiple classes extend one class
Multiple	One class inherits from multiple classes (via interfaces)
Hybrid	Combination of multiple inheritance types

Single Inheritance Example:

Listing 10. Single Inheritance

```java
// Parent class
class Animal {
    protected String name;
    public Animal(String name) { this.name = name; }
    public void eat() { System.out.println(name + " is eating"); }
}

// Child class
class Dog extends Animal {
    private String breed;
    public Dog(String name, String breed) {
        super(name);
        this.breed = breed;
    }
    public void bark() { System.out.println(name + " is barking"); }
}

public class InheritanceDemo {
    public static void main(String[] args) {
        Dog dog = new Dog("Max", "Labrador");
        dog.eat();     // Inherited method
        dog.bark();    // Own method
    }
}
```

Mnemonic

"SMHMH: Single Multilevel Hierarchical Multiple Hybrid types"

Question 3(b) [4 marks]

Explain any two String buffer class methods with suitable example.

Solution

StringBuffer is a mutable sequence of characters used for modifying strings, offering various manipulation methods.

Table 10. Two StringBuffer Methods

Method	Purpose	Syntax
append()	Adds string at the end	`sb.append(String str)`
insert()	Adds string at specified position	`sb.insert(int offset, String str)`

Listing 11. StringBuffer Methods

```java
public class StringBufferMethodsDemo {
    public static void main(String[] args) {
        StringBuffer sb = new StringBuffer("Hello");

        // append() method
        sb.append(" World");
        System.out.println("After append: " + sb);

        // insert() method
        sb.insert(0, "Java ");
        System.out.println("After insert: " + sb);
    }
}
```

Mnemonic

"AIMS: Append Insert Modify StringBuffer"

Question 3(c) [7 marks]

Define Interface. Write a java program to demonstrate multiple inheritance using interface.

Solution

An interface is a contract that declares methods a class must implement, enabling multiple inheritance in Java. It contains only constants, method signatures, default methods, and static methods.

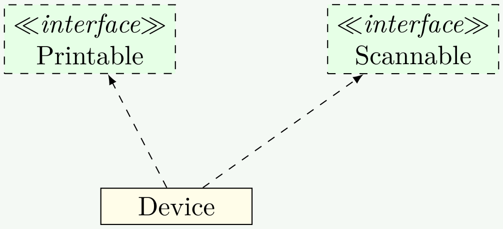

Figure 2. Multiple Inheritance using Interfaces

Listing 12. Multiple Inheritance Example

```java
interface Printable {
    void print();
}

interface Scannable {
    void scan();
}

class Device implements Printable, Scannable {
    private String model;

    public Device(String model) { this.model = model; }

    @Override
    public void print() {
        System.out.println(model + " is printing");
    }

    @Override
    public void scan() {
        System.out.println(model + " is scanning");
    }
}

public class MultipleInheritanceDemo {
    public static void main(String[] args) {
        Device device = new Device("HP LaserJet");
        device.print();
        device.scan();
    }
}
```

Mnemonic

"IMAC: Interface Multiple Abstract Contract"

Question 3(a OR) [3 marks]

Give differences between Abstract class and Interface.

Solution

Abstract classes and interfaces are both used for abstraction but differ in several key aspects.

Table 11. Abstract Class vs Interface

Feature	Abstract Class	Interface
Keyword	`abstract`	`interface`
Methods	Both abstract and concrete	Abstract (and default since Java 8)
Variables	Any type	Only public static final
Constructor	Has	Doesn't have
Inheritance	Single	Multiple
Access Modifiers	Any	Only public
Purpose	Partial implementation	Complete abstraction

- **Implementation**: Abstract classes can provide partial implementation; interfaces traditionally provide none
- **Relationship**: Abstract class says "is-a"; interface says "can-do-this"

Mnemonic

"MAPS: Methods Access Purpose Single vs multiple"

Question 3(b OR) [4 marks]

Explain any two String class methods with suitable example.

Solution

The String class offers various methods for string manipulation, comparison, and transformation.

Table 12. Two String Methods

Method	Purpose	Syntax
substring()	Extracts portion of string	`str.substring(int begin, int end)`
equals()	Compares string content	`str1.equals(str2)`

Listing 13. String Methods Example

```java
public class StringMethodsDemo {
    public static void main(String[] args) {
        String message = "Java Programming";

        // substring() method
        String sub1 = message.substring(0, 4); // "Java"
        System.out.println("Substring: " + sub1);

        // equals() method
        String str1 = "Hello";
        String str2 = new String("Hello");

        System.out.println("equals(): " + str1.equals(str2)); // true
        System.out.println("== operator: " + (str1 == str2)); // false
    }
}
```

Mnemonic

"SEC: Substring Equals Compare string content"

Question 3(c OR) [7 marks]

Explain package and list out steps to create package with suitable example.

Solution

A package in Java is a namespace that organizes related classes and interfaces, preventing naming conflicts.
Steps to Create a Package:

Table 13. Package Creation Steps

Step	Action
1	Declare package name at the top of source files
2	Create proper directory structure matching package name
3	Save Java file in the appropriate directory
4	Compile with `javac -d` option to create package directory
5	Run the program with fully qualified name

Listing 14. Package Example

```java
// File: Calculator.java
package com.example.math;

public class Calculator {
    public int add(int a, int b) {
        return a + b;
    }
}
```

```

// File: CalculatorApp.java
package com.example.app;
import com.example.math.Calculator;

public class CalculatorApp {
    public static void main(String[] args) {
        Calculator calc = new Calculator();
        System.out.println("Addition: " + calc.add(10, 5));
    }
}
```

Compilation Commands:

Listing 15. Terminal Commands

```
javac -d . Calculator.java
javac -d . CalculatorApp.java
java com.example.app.CalculatorApp
```

Mnemonic

"DISCO: Declare Import Save Compile Organize"

Question 4(a) [3 marks]

List types of errors in Java.

Solution

Java programs can encounter various errors during development and execution.

Table 14. Types of Errors in Java

Error Type	When Occurs	Example
Compile-time	During compilation	Syntax errors, type errors
Runtime	During execution	NullPointerException
Logical	During execution	Incorrect calculation
Linkage	During class loading	NoClassDefFoundError

Mnemonic

"CRLLT: Compile Runtime Logical Linkage Thread errors"

Question 4(b) [4 marks]

Explain try catch block with example.

Solution

The try-catch block in Java handles exceptions, allowing programs to continue executing despite errors.

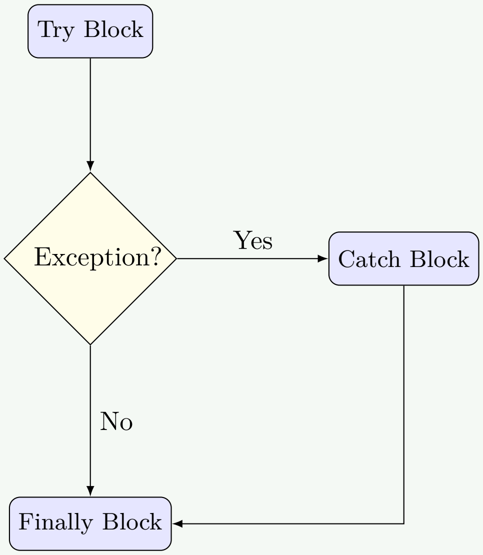

Figure 3. Try-Catch Flow

Listing 16. Try-Catch Example

```java
public class TryCatchDemo {
    public static void main(String[] args) {
        try {
            int[] numbers = {1, 2, 3};
            System.out.println(numbers[10]); // IndexOutOfBounds
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Exception: " + e.getMessage());
        } finally {
            System.out.println("Finally always executes");
        }
        System.out.println("Program continues");
    }
}
```

Mnemonic

"TCFE: Try Catch Finally Execute despite errors"

Question 4(c) [7 marks]

List out any four differences between method overloading and overriding. Write a java code to explain method overriding.

Solution

Method overloading and overriding are both forms of polymorphism but differ in functionality and implementation.

Table 15. Method Overloading vs Overriding

Feature	Overloading	Overriding
Occurrence	Same class	Parent & child classes
Parameters	Different parameters	Same parameters
Return Type	Can be different	Same or covariant
Binding	Compile-time (static)	Runtime (dynamic)
Purpose	Multiple behaviors	Specific implementation

Listing 17. Method Overriding Example

```java
class Animal {
    public void makeSound() {
        System.out.println("Animal makes a sound");
    }
}

class Dog extends Animal {
    @Override
    public void makeSound() {
        System.out.println("Dog barks");
    }
}

public class OverridingDemo {
    public static void main(String[] args) {
        Animal myDog = new Dog();
        myDog.makeSound(); // Outputs: Dog barks
    }
}
```

Mnemonic

"SBRE: Same-name, Base-derived, Runtime-resolution, Extend functionality"

Question 4(a OR) [3 marks]

List any four inbuilt exceptions.

Solution

Java provides many built-in exception classes that represent various error conditions.

Table 16. Four Common Inbuilt Exceptions

Exception	Cause	Package
NullPointerException	Accessing null reference	java.lang
ArrayIndexOutOfBounds	Invalid array index	java.lang
ArithmeticException	Division by zero	java.lang
ClassCastException	Invalid type casting	java.lang

Mnemonic

"NAAC: Null Array Arithmetic Cast common exceptions"

Question 4(b OR) [4 marks]

Explain “throw” keyword with suitable example.

Solution

The `throw` keyword in Java manually generates exceptions for exceptional conditions in programs.

Table 17. throw Keyword Usage

Usage	Purpose
throw new Exception()	Create and throw exception
throw new Exception(msg)	Create with custom message
Inside method	Used to explicitly throw

Listing 18. Throw Example

```java
public class ThrowDemo {
    static void validateAge(int age) {
        if (age < 18) {
            throw new ArithmeticException("Not eligible");
        }
        System.out.println("Eligible to vote");
    }

    public static void main(String[] args) {
        try {
            validateAge(15);
        } catch (ArithmeticException e) {
            System.out.println("Exception: " + e.getMessage());
        }
    }
}
```

Mnemonic

“CET: Create Exception Throw for error handling”

Question 4(c OR) [7 marks]

Compare ‘this’ keyword Vs ‘Super’ keyword. Explain super keyword with suitable Example.

Solution

The `this` and `super` keywords are used for referencing in Java, with distinct purposes and behaviors.

Table 18. this vs super Comparison

Feature	this Keyword	super Keyword
Reference	Current class object	Parent class object
Usage	Access current members	Access parent members
Constructor	`this()` calls own	`super()` calls parent
Variables	`this.var`	`super.var`

Listing 19. Super Keyword Example

```java
class Vehicle {
    String color = "White";
    Vehicle() { System.out.println("Vehicle created"); }
}

class Car extends Vehicle {
    String color = "Black";

    Car() {
        super(); // Call parent constructor
        System.out.println("Car created");
    }

    void displayColor() {
        System.out.println("Car color: " + this.color);
        System.out.println("Vehicle color: " + super.color);
    }
}

public class SuperDemo {
    public static void main(String[] args) {
        Car c = new Car();
        c.displayColor();
    }
}
```

Mnemonic

"PCIM: Parent Class Inheritance Members with super"

Question 5(a) [3 marks]

List Different Stream Classes.

Solution

Java I/O provides various stream classes for handling input and output operations.

Table 19. Java Stream Classes

Category	**Stream Classes**
Byte Streams	FileInputStream, FileOutputStream
Character Streams	FileReader, FileWriter
Buffered Streams	BufferedInputStream, BufferedReader
Data Streams	DataInputStream, DataOutputStream
Object Streams	ObjectInputStream, ObjectOutputStream

Mnemonic

"BCDOP: Byte Character Data Object Print streams"

Question 5(b) [4 marks]

Write a java program to develop user defined exception for "Divide by zero" error.

Solution

User-defined exceptions allow creating custom exception types for application-specific error conditions.

Listing 20. Custom Exception Example

```java
class DivideByZeroException extends Exception {
    public DivideByZeroException(String s) {
        super(s);
    }
}

public class CustomExceptionDemo {
    static int divide(int a, int b) throws DivideByZeroException {
        if (b == 0) throw new DivideByZeroException("Div by 0");
        return a / b;
    }

    public static void main(String[] args) {
        try {
            divide(10, 0);
        } catch (DivideByZeroException e) {
            System.out.println("Caught: " + e.getMessage());
        }
    }
}
```

Mnemonic

"ETC: Extend Throw Catch custom exceptions"

Question 5(c) [7 marks]

Write a program in Java that reads the content of a file byte by byte and copy it into another file.

Solution

File I/O operations in Java allow reading from and writing to files, with byte streams handling binary data.

Listing 21. File Copy Byte by Byte

```java
import java.io.*;

public class FileCopyDemo {
    public static void main(String[] args) {
        FileInputStream in = null;
        FileOutputStream out = null;

        try {
            in = new FileInputStream("input.txt");
            out = new FileOutputStream("output.txt");

            int c;
            while ((c = in.read()) != -1) {
                out.write(c);
            }
            System.out.println("File copied.");
        } catch(IOException e) {
            System.out.println("Error: " + e);
        } finally {
            try {
                if (in != null) in.close();
                if (out != null) out.close();
            } catch(IOException e) {}
        }
    }
}
```

Mnemonic

"CROW: Create Read Open Write file operations"

Question 5(a OR) [3 marks]

List different file operations in Java.

Solution

Java provides comprehensive file handling capabilities through various file operations.

Table 20. File Operations in Java

Operation	Classes Used
File Creation	File, FileOutputStream, FileWriter
File Reading	FileInputStream, FileReader
File Writing	FileOutputStream, FileWriter
File Deletion	File.delete()
File Rename	File.renameTo()

Mnemonic

"CRWD: Create Read Write Delete basic operations"

Question 5(b OR) [4 marks]

Write a java program to explain finally block in exception handling.

Solution

The finally block in exception handling ensures code execution regardless of whether an exception occurs.

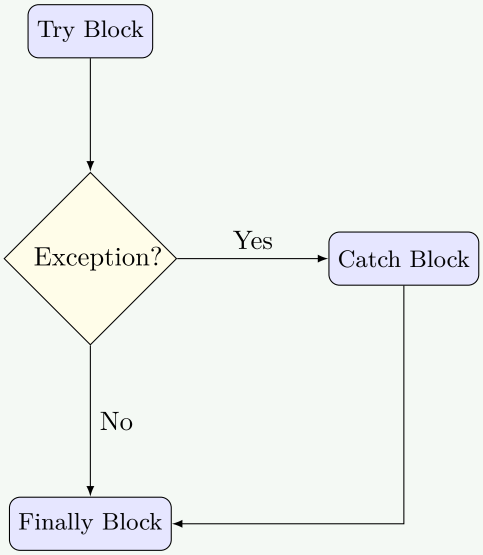

Figure 4. Finally Block Flow

Listing 22. Finally Block Example

```java
public class FinallyDemo {
    public static void main(String[] args) {
        try {
            int data = 25 / 0;
            System.out.println(data);
        } catch (ArithmeticException e) {
            System.out.println(e);
        } finally {
            System.out.println("finally block is always executed");
        }
        System.out.println("rest of the code...");
    }
}
```

Mnemonic

"ACRE: Always Cleanup Resources Executes"

Question 5(c OR) [7 marks]

Write a java program to create a file and perform write operation on this file.

Solution

Java provides several ways to create files and write data to them using character or byte streams.

Listing 23. File Write Example

```java
import java.io.FileWriter;
import java.io.IOException;

public class FileWriteDemo {
    public static void main(String[] args) {
        try {
            FileWriter writer = new FileWriter("output.txt");
            writer.write("Hello World!\n");
            writer.write("This is a Java file write operation.");
            writer.close();
            System.out.println("Successfully wrote to the file.");
        } catch (IOException e) {
            System.out.println("An error occurred.");
            e.printStackTrace();
        }
    }
}
```

Mnemonic

"COWS: Create Open Write Save file operations"