Object Oriented Programming With Java (4341602) - Summer 2024 Solution

Milav Dabgar

June 13, 2024

Question 1(a) [3 marks]

Explain the basic structure of Java program.

Solution

Basic Structure: A Java program consists of classes, methods, and statements organized in a specific hierarchy.

Table 1. Basic Structure Components

Component	Description
Package Declaration	Optional statement defining namespace membership
Import Statements	Bring in classes from other packages
Class Declaration	Main unit of code, blueprint for objects
Main Method	Entry point `public static void main(String[] args)`

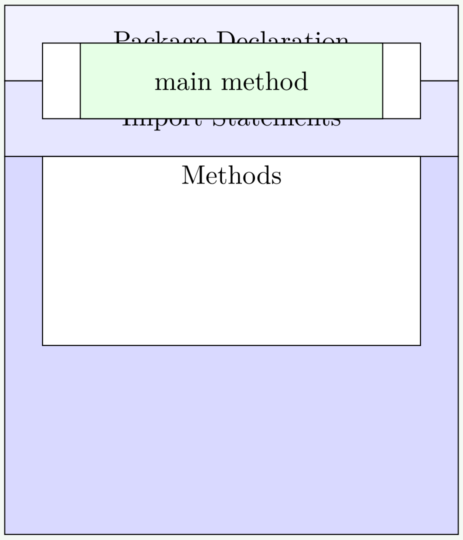

Figure 1. Structure of Java Program

- **Package**: Groups related classes (e.g., `package com.example;`)
- **Import**: Access libraries (e.g., `import java.util.*;`)
- **Class**: Contains all code
- **Main method**: Execution starts here

Mnemonic

"PICM - Package, Import, Class, Main"

Question 1(b) [4 marks]

List out different features of java. Explain any two.

Solution

Java Features:

1. **Platform Independent**: Write Once, Run Anywhere (WORA)
2. **Object Oriented**: Everything is an object
3. **Simple**: No complex features like pointers
4. **Secure**: Bytecode verification, no explicit memory access
5. **Robust**: Strong memory management, exception handling
6. **Multithreaded**: Concurrent execution support

1. Platform Independence: Java programs are compiled into bytecode, which is platform-neutral. This bytecode is interpreted by the Java Virtual Machine (JVM) specific to each operating system (Windows, Linux, Mac), allowing the same code to run everywhere without recompilation.

2. Object Oriented: Java models real-world entities using objects and classes. It supports key OOP principles:

- **Encapsulation**: bundling data and methods
- **Inheritance**: code reusability
- **Polymorphism**: same interface, multiple forms

Mnemonic

"POSRMM - Platform, Object, Simple, Robust, Multithreaded, Memory"

Question 1(c) [7 marks]

Write a program in java to find out sum of the digits of entered number. (Ex. Number is 123 output is 6).

Solution

Listing 1. Sum of Digits

```java
public class DigitSum {
    public static void main(String[] args) {
        if (args.length == 0) {
            System.out.println("Please provide a number.");
            return;
        }

        int number = Integer.parseInt(args[0]);
        int sum = 0;
        int temp = Math.abs(number);

        // Loop to extract and add digits
        while (temp > 0) {
            sum += temp % 10;  // Extract last digit
            temp /= 10;        // Remove last digit
        }

        System.out.println("Sum of digits: " + sum);
    }
}
```

Table 2. Algorithm Trace (Input: 123)

Step	Operation	Result
1	Extract digit	$123\%10 = 3$
2	Add to sum	$sum = 0 + 3 = 3$
3	Remove digit	$123/10 = 12$
4	Extract digit	$12\%10 = 2$
5	Add to sum	$sum = 3 + 2 = 5$
6	Remove digit	$12/10 = 1$
7	Extract digit	$1\%10 = 1$
8	Add to sum	$sum = 5 + 1 = 6$

Mnemonic

"EARD - Extract, Add, Remove, Done"

Question 1(c OR) [7 marks]

Write a program in java to find out maximum from any ten numbers using command line argument.

Solution

Listing 2. Find Maximum from Arguments

```java
public class FindMaximum {
    public static void main(String[] args) {
        if (args.length < 10) {
            System.out.println("Please enter 10 numbers");
            return;
        }

        // Initialize max with the first number
        int max = Integer.parseInt(args[0]);

        // Loop through remaining numbers
        for (int i = 1; i < 10; i++) {
            int current = Integer.parseInt(args[i]);
            if (current > max) {
                max = current;
            }
        }

        System.out.println("Maximum number: " + max);
    }
}
```

Table 3. Logic Steps

Step	Action
Validation	Check if at least 10 arguments are provided
Initialization	Assume first argument is `max`
Comparison	Loop through remaining 9 numbers
Update	If `current > max`, set `max = current`
Output	Print the final `max` value

Mnemonic

"VCIU - Validate, Compare, Initialize, Update"

Question 2(a) [3 marks]

List out different concept of oop. Explain anyone in detail.

Solution

OOP Concepts:
- Encapsulation
- Inheritance
- Polymorphism
- Abstraction

Encapsulation: Encapsulation is the mechanism of wrapping code (methods) and data (variables) together into a single unit (class). It protects data from outside interference and misuse.
- **Data Hiding**: Variables are declared `private` to restrict direct access.
- **Accessors/Mutators**: Public `getter` and `setter` methods are provided to access and modify data controllably.
- **Benefits**: Improves security, modularity, and maintainability.

Mnemonic

"EIPA - Encapsulation, Inheritance, Polymorphism, Abstraction"

Question 2(b) [4 marks]

Explain JVM in detail.

Solution

JVM (Java Virtual Machine) is the engine that executes Java bytecode. It provides a runtime environment for Java code.

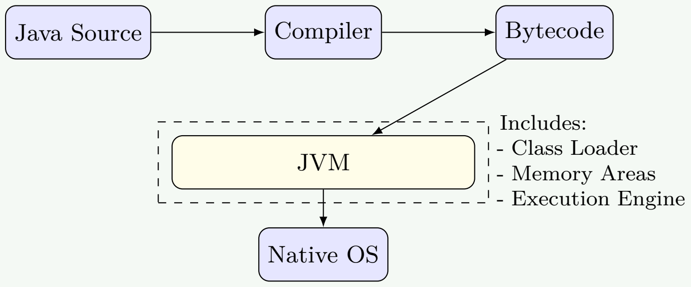

Figure 2. JVM Architecture

Table 4. JVM Components

Component	Function
Class Loader	Loads `.class` files into memory
Memory Areas	Allocates memory (Heap, Stack, Method Area)
Execution Engine	Interprets bytecode or uses JIT compiler
JIT Compiler	Compiles hot code to native machine code for speed

Key Features:

- **Platform Independence**: JVM makes Java portable.
- **Memory Management**: Handles allocation and garbage collection.

Mnemonic

"CEMJ - Class loader, Execution, Memory, JIT"

Question 2(c) [7 marks]

Explain constructor overloading with example.

Solution

Constructor Overloading: A technique of having more than one constructor in a class with different parameter lists.

Listing 3. Constructor Overloading

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

    // 1. Default constructor
    public Student() {
        this.name = "Unknown";
        this.age = 0;
        this.course = "Not Assigned";
    }

    // 2. Constructor with one parameter
    public Student(String name) {
        this.name = name;
        this.age = 0;
        this.course = "Not Assigned";
    }

    // 3. Constructor with two parameters
    public Student(String name, int age) {
        this.name = name;
        this.age = age;
        this.course = "Not Assigned";
    }

    // 4. Constructor with all parameters
    public Student(String name, int age, String course) {
        this.name = name;
        this.age = age;
        this.course = course;
    }
}
```

Table 5. Types of Constructors in Example

Constructor	Parameters	Purpose
Default	None	Initialize with default values
Single Param	Name	Register student with just name
Two Param	Name, Age	Register with basic details
Full Param	All fields	Complete initialization

Rules:

- Must have the same name as the class.
- Must have different parameter lists (number or type of args).
- Resolved at compile-time (Compile-time Polymorphism).

Mnemonic

"SNDF - Same Name, Different Parameters, Flexible"

Question 2(a OR) [3 marks]

What is wrapper class? Explain with example.

Solution

Wrapper Class: A class whose object wraps or contains a primitive data type. They convert primitive data types into objects.

Table 6. Primitive vs Wrapper

Primitive	Wrapper Class
byte	Byte
int	Integer
char	Character
double	Double
boolean	Boolean

Listing 4. Boxing and Unboxing

```java
// Boxing: Primitive to Object
int num = 10;
Integer obj = Integer.valueOf(num); // Manual
Integer autoBox = 10;               // Autoboxing

// Unboxing: Object to Primitive
Integer wrapper = new Integer(20);
int value = wrapper.intValue();     // Manual
int autoUnbox = wrapper;            // Auto-unboxing
```

Usage: Required in Collection Frameworks (ArrayList, Vector) where only objects are stored.

Mnemonic

"BUC - Boxing, Unboxing, Collections"

Question 2(b OR) [4 marks]

Explain static keyword with example.

Solution

Static Keyword: Indicates that a member belongs to the class type itself, rather than to an instance of that class.

Table 7. Static Uses

Member	Behavior
Variable	Shared memory among all instances (class variable)
Method	Can be called without creating an object
Block	Executed once when class is loaded

Listing 5. Static Example

```java
public class Counter {
    static int count = 0;  // Shared variable
    int id;                // Instance variable

    public Counter() {
        count++;           // Increments for every new object
        this.id = count;
    }

    public static void showCount() {
        // Can only access static data
        System.out.println("Total objects: " + count);
    }
}
```

- **Memory Efficiency**: Variable created only once.
- **Utility**: Used for utility methods (e.g., `Math.sqrt()`).

Mnemonic

"SCMA - Shared, Class-level, Memory, Access"

Question 2(c OR) [7 marks]

What is constructor? Explain copy constructor with example.

Solution

Constructor: A special block of code used to initialize an object. It is called when an instance of the class is created.
Copy Constructor: A constructor that initializes an object using another object of the same class. It creates a clone of the existing object.

Listing 6. Copy Constructor

```java
public class Book {
    private String title;
    private String author;

    // Parameterized constructor
    public Book(String title, String author) {
        this.title = title;
        this.author = author;
    }

    // Copy constructor
    public Book(Book other) {
        this.title = other.title;
        this.author = other.author;
    }

    public void display() {
        System.out.println(title + " by " + author);
    }
}

// Usage
class Main {
    public static void main(String[] args) {
        Book b1 = new Book("Java Guide", "James");
        Book b2 = new Book(b1); // Creates a copy of b1
        b1.display();
        b2.display();
    }
}
```

Table 8. Constructor Properties

Property	Description
Name	Must match class name
Return Type	None (not even void)
Invocation	Automatic at time of `new`

Mnemonic

"SNAC - Same Name, Automatic Call"

Question 3(a) [3 marks]

Explain any four-string function in java with example.

Solution

String Functions:

Table 9. Common String Functions

Function	Purpose	Example
`length()`	Returns number of characters	`"Hi".length()` → 2
`charAt(i)`	Returns char at index i	`"Hi".charAt(0)` → 'H'
`substring(i)`	Returns part of string from i	`"Code".substring(2)` → "de"
`toUpperCase()`	Converts to uppercase	`"java".toUpperCase()` → "JAVA"

Listing 7. String Example

```java
String str = "Java Programming";
```

```java
int len = str.length();           // 16
char ch = str.charAt(0);          // 'J'
String sub = str.substring(5);    // "Programming"
String upper = str.toUpperCase(); // "JAVA PROGRAMMING"
```

Mnemonic

"LCST - Length, Character, Substring, Transform"

Question 3(b) [4 marks]

List out different types of inheritance. Explain multilevel inheritance.

Solution

Types of Inheritance:

1. Single Inheritance
2. Multilevel Inheritance
3. Hierarchical Inheritance
4. Multiple Inheritance (via Interfaces)
5. Hybrid Inheritance

Multilevel Inheritance: A mechanism where a class inherits from a derived class, forming a chain of inheritance.

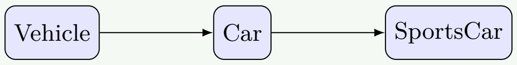

Figure 3. Multilevel Inheritance

Listing 8. Multilevel Inheritance

```java
class Vehicle { void start() {} }
class Car extends Vehicle { void drive() {} }
class SportsCar extends Car { void race() {} }
```

In this example, `SportsCar` inherits features from both `Car` and `Vehicle`.

Mnemonic

"SMHM - Single, Multilevel, Hierarchical, Multiple"

Question 3(c) [7 marks]

What is interface? Explain multiple inheritance with example.

Solution

Interface: An abstract reference type in Java that is similar to a class but contains only constants, method signatures (empty methods), default methods, and static methods. It is used to achieve total abstraction and multiple inheritance.

Multiple Inheritance: Java does not support multiple inheritance with classes to avoid ambiguity (Diamond Problem), but it supports it through interfaces. A class can implement multiple interfaces.

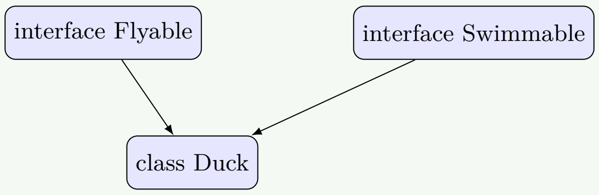

Figure 4. Multiple Inheritance via Interfaces

Listing 9. Multiple Inheritance Example

```java
interface Flyable {
    void fly();
}
interface Swimmable {
    void swim();
}

// Single class implementing multiple interfaces
class Duck implements Flyable, Swimmable {
    public void fly() {
        System.out.println("Duck is flying");
    }
    public void swim() {
        System.out.println("Duck is swimming");
    }
}
```

Table 10. Interface vs Class

Interface	**Class**
Can contain abstract methods	Contains concrete methods
Variables are `public static final`	Any variable type allowed
Supports multiple inheritance	Supports single inheritance

Mnemonic

"CMDS - Contract, Multiple, Diamond-solution"

Question 3(a OR) [3 marks]

Explain this keyword with example.

Solution

'this' Keyword: A reference variable in Java that refers to the current object.

Table 11. Uses of 'this'

Use Case	**Purpose**
Instance Variable	Distinguish field from parameter (`this.x = x`)
Method Call	Invoke current class method (`this.method()`)
Constructor Call	Chain constructors (`this()`)
Return Object	Return current instance (`return this`)

Listing 10. Using 'this'

```java
public class Person {
    String name;

    public Person(String name) {
        this.name = name; // Resolves ambiguity
    }

    public Person getInstance() {
        return this; // Returns current object
    }
}
```

Mnemonic

"CRPM - Current, Resolve, Parameter, Method"

Question 3(b OR) [4 marks]

Explain method overriding with example.

Solution

Method Overriding: Occurs when a subclass provides a specific implementation for a method that is already defined in its parent class. It is used for Runtime Polymorphism.

Listing 11. Method Overriding

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

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

class Main {
    public static void main(String[] args) {
        Animal a = new Dog(); // Upcasting
        a.makeSound(); // Output: Dog barks
    }
}
```

Table 12. Overriding Rules

Rule	Description
Signature	Method name and args must be identical
Inheritance	Must involve IS-A relationship
Access	Access level cannot be more restrictive
Binding	Resolved at runtime (Dynamic Binding)

Mnemonic

"SSRD - Same Signature, Runtime Decision"

Question 3(c OR) [7 marks]

What is package? Write steps to create a package and give example of it.

Solution

Package: A namespace that organizes a set of related classes and interfaces. It helps in:

- Preventing naming conflicts.
- Controlling access (protected/default access).
- Making searching/usage of classes easier.

Steps to Create and Use a Package:

1. **Directory**: Create a folder structure matching the package name (e.g., `com/utils`).
2. **Declaration**: Add `package com.utils;` at the top of the file.
3. **Compile**: Compile with `-d .` to generate folders automatically.
4. **Import**: Use `import com.utils.*;` in another file.

Listing 12. Creating a Package

```java
// File: src/com/company/utils/MathUtils.java
package com.company.utils;

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

Listing 13. Using a Package

```java
// File: src/Calculator.java
import com.company.utils.MathUtils;

public class Calculator {
    public static void main(String[] args) {
        int result = MathUtils.add(5, 10);
        System.out.println("Result: " + result);
    }
}
```

Table 13. Compiling and Running

Action	Command
Compile Package	`javac -d . MathUtils.java`
Compile Main	`javac Calculator.java`
Run	`java Calculator`

Mnemonic

"ONAM - Organization, Namespace, Access, Maintenance"

Question 4(a) [3 marks]

Explain thread priorities with suitable example.

Solution

Thread Priority: Java threads have priority values from 1 to 10 that help the Thread Scheduler decide which thread to execute.

Table 14. Priority Constants

Level	Constant	Value
Min	`Thread.MIN_PRIORITY`	1
Norm	`Thread.NORM_PRIORITY`	5 (Default)
Max	`Thread.MAX_PRIORITY`	10

Listing 14. Thread Priority

```java
class MyThread extends Thread {
    public void run() {
        System.out.println("Running: " + getName());
    }
}

public class PriorityDemo {
    public static void main(String[] args) {
        MyThread t1 = new MyThread();
        MyThread t2 = new MyThread();

        t1.setPriority(Thread.MAX_PRIORITY); // 10
        t2.setPriority(Thread.MIN_PRIORITY); // 1

        t1.start(); // Likely to run first
        t2.start();
    }
}
```

Mnemonic

"HNG - Higher priority, Not Guaranteed"

Question 4(b) [4 marks]

What is Thread? Explain Thread life cycle.

Solution

Thread: A lightweight subprocess that runs concurrently with other threads.
Thread Life Cycle: A thread goes through various states during its lifetime.

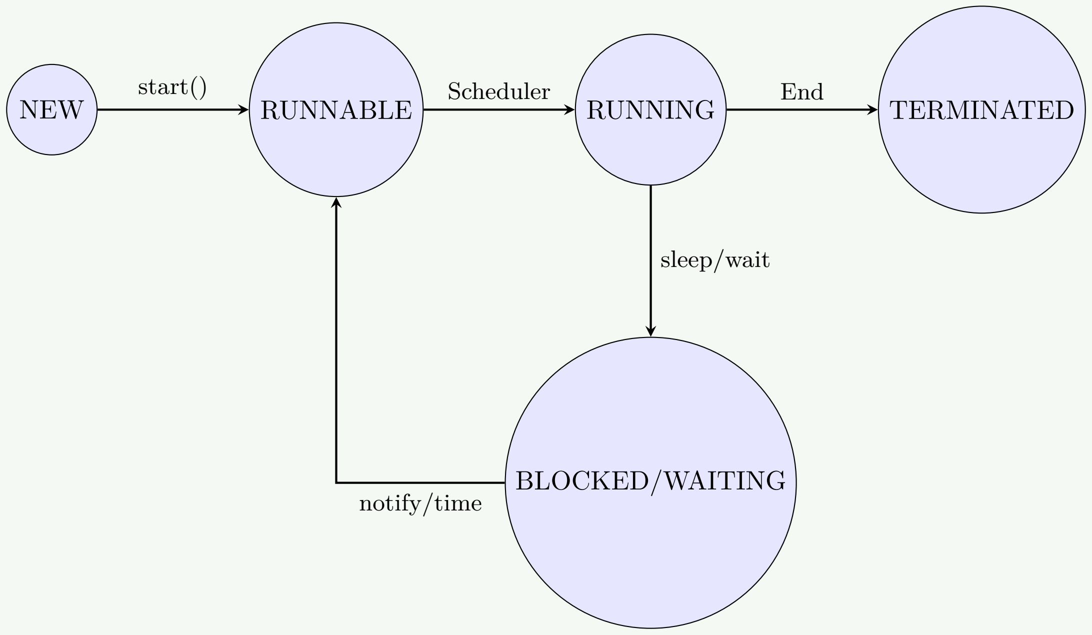

Figure 5. Thread Life Cycle

Table 15. Thread States

State	**Description**
NEW	Created instance, `start()` not called
RUNNABLE	Ready to run, waiting for CPU
RUNNING	Currently executing
BLOCKED	Waiting for resource explicitly
TERMINATED	Executed finished

Mnemonic

"NRBT - New, Runnable, Blocked, Terminated"

Question 4(c) [7 marks]

Write a program in java that create the multiple threads by implementing the Thread class.

Solution

Listing 15. Multiple Threads

```java
class NumberPrinter extends Thread {
    String name;

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

    public void run() {
        for(int i=1; i<=3; i++) {
            System.out.println(name + ": " + i);
            try {
                Thread.sleep(500); // Pause
            } catch(Exception e) {}
        }
    }
}

public class MultiThreadDemo {
    public static void main(String[] args) {
        NumberPrinter t1 = new NumberPrinter("Thread-1");
        NumberPrinter t2 = new NumberPrinter("Thread-2");
        NumberPrinter t3 = new NumberPrinter("Thread-3");

        // Start all threads concurrently
        t1.start();
        t2.start();
        t3.start();

        System.out.println("Main finished");
    }
}
```

Steps:

1. Extend `Thread` class.
2. Override `run()` method with logic.
3. Create instances of the class.
4. Call `start()` to begin execution.

Mnemonic

"EOCS - Extend, Override, Create, Start"

Question 4(a OR) [3 marks]

Explain basic concept of Exception Handling.

Solution

Exception Handling: A mechanism to handle runtime errors so that the normal flow of the application can be maintained.

Table 16. Key Blocks

Keyword	Function
try	Block of code to monitor for errors
catch	Block that handles the exception
finally	Block that executes regardless of outcome
throw	Used to explicitly throw an exception
throws	Declares exceptions a method can throw

Listing 16. Exception Syntax

```java
try {
    // Risky code
} catch (Exception e) {
    // Handling code
} finally {
    // Cleanup
}
```

Mnemonic

"TRCF - Try, Runtime error, Catch, Finally"

Question 4(b OR) [4 marks]

Explain multiple catch with suitable example.

Solution

Multiple Catch Blocks: A `try` block can be followed by multiple `catch` blocks to handle different types of exceptions separately.

Listing 17. Multiple Catch

```java
public class MultiCatch {
    public static void main(String[] args) {
        try {
            int a[] = new int[5];
            a[10] = 30 / 0; // Risky code
        } catch (ArithmeticException e) {
            System.out.println("Math Error: " + e);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Array Error: " + e);
        } catch (Exception e) {
            // Generic catch must be last
            System.out.println("General Error: " + e);
        }
    }
}
```

Rules:

- At a time only one exception occurs and only one catch block is executed.
- Specific exceptions must be caught before general exceptions (`Exception` parent class).

Mnemonic

"SOOF - Specific first, One executes, Order matters, Finally"

Question 4(c OR) [7 marks]

What is Exception? Write a program that show the use of Arithmetic Exception.

Solution

Exception: An unwanted or unexpected event that occurs during the execution of a program (at runtime) that disrupts the normal flow of instructions.
ArithmeticException: A runtime exception thrown when an exceptional arithmetic condition has occurred, such as division by zero.

Listing 18. Arithmetic Exception Demo

```java
public class DivisionDemo {
    public static void main(String[] args) {
        System.out.println("Start of program");

        try {
            int numerator = 100;
            int denominator = 0; // Division by zero

            // This line throws ArithmeticException
            int result = numerator / denominator;

            System.out.println("Result: " + result);
        } catch (ArithmeticException e) {
            System.out.println("Error detected: Division by zero is not allowed.");
            System.out.println("Exception: " + e.getMessage());
        } finally {
            System.out.println("Cleanup actions...");
        }

        System.out.println("Program continues normally...");
    }
}
```

Output:

```
Start of program
Error detected: Division by zero is not allowed.
Exception: / by zero
Cleanup actions...
Program continues normally...
```

Without the try-catch block, the program would crash immediately at the point of division.

Mnemonic

"DZMI - Division by Zero, Mathematical Invalid"

Question 5(a) [3 marks]

Explain ArrayIndexOutOfBound Exception in Java with example.

Solution

ArrayIndexOutOfBoundsException: A runtime exception thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.

Table 17. Causes

Cause	Description	Example
Negative Index	Index < 0	a[-1]
Size Exceeded	Index $\geq$ Length	a[length]
Empty Array	Accessing index 0 of empty	a[0]

Listing 19. Array Index Exception

```java
int[] numbers = {10, 20, 30}; // Size 3, indices 0-2
try {
    System.out.println(numbers[5]); // Index 5 is invalid
} catch (ArrayIndexOutOfBoundsException e) {
    System.out.println("Invalid Index: " + e.getMessage());
}
```

Mnemonic

"NIE - Negative, Index-exceed, Empty"

Question 5(b) [4 marks]

Explain basics of stream classes.

Solution

Stream Classes: A stream is a sequence of data. Java I/O is based on streams.

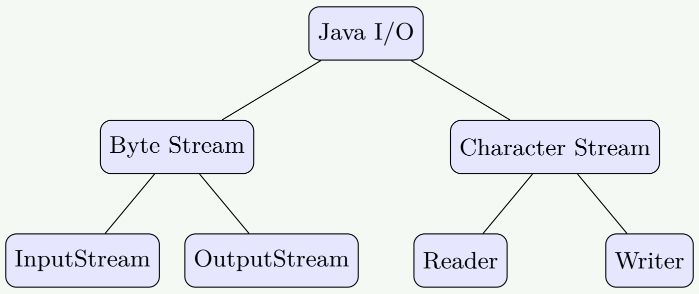

Figure 6. Stream Hierarchy

Table 18. Stream Categories

Stream Type	Data Type	Classes
Byte Stream	Binary Data (Images, etc)	InputStream, OutputStream
Char Stream	Text Data (Strings)	Reader, Writer

Subclasses:
- **File**: FileInputStream, FileWriter, etc.
- **Buffered**: BufferedReader, BufferedOutputStream (for efficiency).

Mnemonic

"BCIF - Byte, Character, Input/Output, File"

Question 5(c) [7 marks]

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

Solution

Listing 20. Write to File

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

public class FileWriteDemo {
    public static void main(String[] args) {
        // Data to write
        String data = "This is a sample text file created by Java program.\nWelcome to Summer 2024
        ↪ Solution.";

        // Using Try-with-resources to automatically close the writer
        try (FileWriter writer = new FileWriter("output.txt")) {

            // Writing data
            writer.write(data);

            System.out.println("Successfully wrote to the file.");

        } catch (IOException e) {
            System.out.println("An error occurred.");
            e.printStackTrace();
        }
    }
}
```

Explanation:

1. **Import**: `java.io.FileWriter` and `IOException`.
2. **FileWriter**: Creates a file writer object. Pass file name string.
3. **write()**: Method to write string content to the file.
4. **close()**: Automatically called by try-with-resources block to save data and free resources.

Table 19. Methods Used

Method	Description
FileWriter(String)	Creates new file, overwrites if exists
write(String)	Writes text to stream
close()	Flushes and closes stream

Mnemonic

"CWCH - Create, Write, Close, Handle"

Question 5(a OR) [3 marks]

Explain Divide by Zero Exception in Java with example.

Solution

Divide by Zero: Trying to divide an integer by zero is an illegal operation in Java.

Table 20. Behavior

Case	Result	Exception
Integer / 0	Illegal	ArithmeticException
Float / 0.0	Infinity	None
Modulo % 0	Illegal	ArithmeticException

Listing 21. Divide By Zero

```java
try {
    int a = 10 / 0; // Throws Exception
} catch (ArithmeticException e) {
    System.out.println("Cannot divide integer by zero");
}

double b = 10.0 / 0.0;
System.out.println(b); // Prints "Infinity"
```

Mnemonic

"IFM - Integer exception, Float infinity, Modulo error"

Question 5(b OR) [4 marks]

Explain try and catch block with example.

Solution

Try-Catch: The core mechanism for exception handling.

- **try**: Encloses the code that might generate an exception.
- **catch**: Defines how to handle the exception. It must follow a try block.

Listing 22. Try-Catch Example

```java
public class Example {
    public static void main(String args[]) {
        try {
            String s = null;
            System.out.println(s.length()); // NullPointer
        } catch(NullPointerException e) {
            System.out.println("Caught Null Pointer Exception");
        }

        System.out.println("Rest of the code...");
    }
}
```

Table 21. Program Flow

Scenario	Execution Path
No Exception	try executes -> catch skipped -> rest of code
Exception	try stops at error -> catch executes -> rest of code

Mnemonic

"TCF - Try risky, Catch exception, Finally cleanup"

Question 5(c OR) [7 marks]

Write a java program to display the content of a text file and perform append operation on the text file.

Solution

Listing 23. Read and Append

```java
import java.io.*;

public class FileReadAppend {
    public static void main(String[] args) {
        String filename = "log.txt";

        // 1. Append Operation
        try (FileWriter fw = new FileWriter(filename, true);
             BufferedWriter bw = new BufferedWriter(fw)) {

            bw.write("New Log Entry\n");
            System.out.println("Appended to file.");

        } catch (IOException e) {
            e.printStackTrace();
        }

        // 2. Read Operation
        System.out.println("--- Reading File ---");
        try (FileReader fr = new FileReader(filename);
             BufferedReader br = new BufferedReader(fr)) {

            String line;
            while ((line = br.readLine()) != null) {
                System.out.println(line);
            }

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Key Concepts:

- **Append Mode**: `new FileWriter(file, true)` opens file in append mode.
- **BufferedReader**: Reads text efficiently line by line using `readLine()`.

Mnemonic

"CDADS - Create, Display, Append, Display, Statistics"