

# Subject Name (Gujarati)

4341602 -- Summer 2023

Semester 1 Study Material

*Detailed Solutions and Explanations*

## પ્રશ્ન 1(અ) [3 ગુણ]

પ્રોસિજર-ઓરિએન્ટેડ પ્રોગ્રામિંગ (POP) અને ઓફ્જેક્ટ-ઓરિએન્ટેડ પ્રોગ્રામિંગ (OOP) વચ્ચે તફાવત કરો.

### જવાબ

#### ટેબલ:

પાસું	POP	OOP
દ્યાન	ફંક્શન્સ/પ્રોસિજર્સ	ઓફ્જેક્ટ્સ અને કલાસ્સ્ઝ
ડેટા સિક્યોરિટી	ઓછી સુરક્ષા, ગ્લોબલ ડેટા	વધુ સુરક્ષા, ડેટા encapsulation
સમસ્યા ઉકેલ	ટોપ-ડાઉન એપ્રોચ	બોટમ-અપ એપ્રોચ
કોડ પુનઃઉપયોગ	મર્યાદિત	inheritance દ્વારા વધુ
ઉદાહરણો	C, Pascal	Java, C++, Python

- POP: પ્રોગ્રામ ફંક્શન્સમાં વિભાજિત, ડેટા ફંક્શન્સ વચ્ચે વહે છે
- OOP: પ્રોગ્રામ ઓફ્જેક્ટ્સની આસપાસ ગોઠવાયેલું જેમાં ડેટા અને મેથ્ડ્સ બંને હોય છે

### મેમરી ટ્રીક

"POP ફંક્શન્સ, OOP ઓફ્જેક્ટ્સ"

## પ્રશ્ન 1(બ) [4 ગુણ]

ઘોંય ઉદાહરણ સાથે inheritance માં Super કીવર્ડ સમજાવો.

### જવાબ

Super કીવર્ડ નો ઉપયોગ child class માંથી parent class ના members ને access કરવા માટે થાય છે.  
ટેબલ: Super કીવર્ડના ઉપયોગો

ઉપયોગ	હેતુ	ઉદાહરણ
super()	parent constructor ને કોલ કરે	super(name, age)
super.method()	parent method ને કોલ કરે	super.display()
super.variable	parent variable ને access કરે	super.name

### કોડ લોક:

```
class Animal {
    String name = "Animal";
    void eat() { System.out.println("Animal eats"); }
}

class Dog extends Animal {
    String name = "Dog";
    void eat() {
        super.eat(); // parent method
        System.out.println("Dog eats bones");
    }
    void display() {
        System.out.println(super.name); // "Animal"
    }
}
```

### મેમરી ટ્રીક

“Super Parent ને કોલ કરે”

### પ્રશ્ન 1(ક) [7 ગુણ]

વ્યાખ્યાપિત કરો: મેથડ ઓવરરાઇટિંગ. મેથડ ઓવરરાઇટિંગ માટેના નિયમોની યાદી બનાવો. એક જાવા પ્રોગ્રામ લખો જે મેથડ ઓવરરાઇટિંગને implement કરે છે.

#### જવાબ

**મેથડ ઓવરરાઇટિંગ:** Child class પોતાની parent class ના method નું specific implementation આપે છે સમાન signature સાથે.

**ટેબલ:** મેથડ ઓવરરાઇટિંગના નિયમો

નિયમ	વર્ણન
સમાન નામ	મેથડનું નામ સમાન હોવું જોઈએ
સમાન parameters	Parameter list બરાબર મેચ થવી જોઈએ
IS-A સંબંધ	inheritance હોવું જરૂરી
Access modifier	visibility ઘટાડી શકાતી નથી
Return type	સમાન અથવા covariant હોવું જોઈએ

### કોડ લોક:

```
class Shape {
    void draw() {
        System.out.println("Drawing a shape");
    }
}

class Circle extends Shape {
    @Override
    void draw() {
        System.out.println("Drawing a circle");
    }
}

class Main {
    public static void main(String[] args) {
        Shape s = new Circle();
        s.draw(); //      : Drawing a circle
    }
}
```

### મેમરી ટ્રીક

"Override સમાન Signature"

## પ્રશ્ન 1(ક OR) [7 ગુણ]

વાર્ષિકો: ઇન્સરફેસ. મલિટિપલ inheritance ને દર્શાવવા માટે interface નો ઉપયોગ કરીને જાવા પ્રોગ્રામ લખો.

### જવાબ

**Interface:** blueprint જેમાં abstract methods અને constants હોય છે. Classes implement કરીને multiple inheritance પ્રાપ્ત કરે છે.

**ટેબલ:** Interface ની વિશેષતાઓ

વિશેષતા	વર્ણન
Abstract methods	કોઈ implementation નથી (Java 8 પહેલાં)
Constants	અધાર variables public static final છે
Multiple inheritance	Class અનેક interfaces implement કરી શકે
Default methods	Concrete methods (Java 8+)

### કોડ લોક:

```
interface Flyable {
    void fly();
}

interface Swimmable {
    void swim();
}

class Duck implements Flyable, Swimmable {
    public void fly() {
        System.out.println("Duck flies");
    }

    public void swim() {
        System.out.println("Duck swims");
    }
}

class Main {
    public static void main(String[] args) {
        Duck d = new Duck();
        d.fly();
        d.swim();
    }
}
```

### મેમરી ટ્રીક

“Interface Multiple Implementation”

## પ્રશ્ન 2(અ) [3 ગુણ]

ઉદાહરણ સાથે Java પ્રોગ્રામ સ્ટ્રક્ચર સમજાવો.

### જવાબ

Java પ્રોગ્રામ સ્ટ્રક્ચર માં package, imports, class declaration અને main method હોય છે.  
આફ્ટિટિંગ:

```
+{--}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}+}
| Package statement |
+{--}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}+
| Import statements |
+{--}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}+
| Class declaration |
| +{--}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}+   |
| | Variables   |   |
| +{--}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}+   |
| | Methods   |   |
| +{--}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}+   |
+{--}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}+
```

### કોડ લોક:

```
package com.example;           // Package
import java.util.*;            // Import

public class HelloWorld {     // Class
```

```

static int count = 0; // Variable

public static void main(String[] args) {
    System.out.println("Hello World");
}

```

### મેમરી ટ્રીક

“Package Import Class Main”

## પ્રશ્ન 2(બ) [4 ગુણ]

યોગ્ય ઉદાહરણ સાથે static કીવર્ડ સમજાવો.

### જવાબ

Static કીવર્ડ class નું છે instance નું નહીં. memory એક વાર allocate થાય છે.  
ટેબલ: Static ના ઉપયોગો

પ્રકાર	વર્ણન	ઉદાહરણ
<b>Static variable</b>	બધા objects દ્વારા shared	static int count
<b>Static method</b>	object વિના કોણ થાય	static void display()
<b>Static block</b>	main પહેલાં execute થાય	static {}

### કોડ ફોટો:

```

class Student {
    static String college = "GTU"; // static variable
    String name;

    static void showCollege() { // static method
        System.out.println(college);
    }

    static { // static block
        System.out.println("Static block executed");
    }
}

class Main {
    public static void main(String[] args) {
        Student.showCollege(); // object
    }
}

```

### મેમરી ટ્રીક

“Static Class દ્વારા Shared”

## પ્રશ્ન 2(ક) [7 ગુણ]

વ્યાખ્યાપિત કરો: કન્સ્ટ્રક્ટર. તેના પ્રકારોની યારી બનાવો. પેરામીટરાઇઝડ અને કોપી કન્સ્ટ્રક્ટરને યોગ્ય ઉદાહરણ સાથે સમજાવો.

## જવાબ

**કન્સ્ટ્રક્ટર:** objects ને initialize કરવા માટેની special method, class જેવું જ નામ, કોઈ return type નથી.  
**ટેબલ:** કન્સ્ટ્રક્ટરના પ્રકારો

પ્રકાર	વર્ણન	ઉદાહરણ
Default	કોઈ parameters નથી	Student()
Parameterized	parameters સાથે	Student(String name)
Copy	object ની copy બનાવે	Student(Student s)

## કોડ ફોક:

```
class Student {
    String name;
    int age;

    // Parameterized constructor
    Student(String n, int a) {
        name = n;
        age = a;
    }

    // Copy constructor
    Student(Student s) {
        name = s.name;
        age = s.age;
    }

    void display() {
        System.out.println(name + " " + age);
    }
}

class Main {
    public static void main(String[] args) {
        Student s1 = new Student("John", 20); // Parameterized
        Student s2 = new Student(s1); // Copy
        s1.display();
        s2.display();
    }
}
```

## મેમરી ટ્રીક

"Constructor Objects ને Initialize કરો"

## પ્રશ્ન 2(અ OR) [3 ગુણ]

જવામાં પ્રિમિટિવ ડેટા પ્રકારો અને યુગર ડિફાઇન્ડ ડેટા પ્રકારો સમજાવો.

## જવાબ

**પ્રિમિટિવ ડેટા ટાઇપ્સ:** Java language દ્વારા આપવામાં આવેલા built-in types. **યુગર ડિફાઇન્ડ ટાઇપ્સ:** programmer દ્વારા classes વાપરીને બનાવવામાં આવેલા custom types.

**ટેબલ:** ડેટા ટાઇપ્સ

ક્રટેગારી	પ્રકારો	સાઇઝ	ઉદાહરણ
Primitive	byte, short, int, long	1,2,4,8 bytes	int x = 10;
Primitive	float, double	4,8 bytes	double d = 3.14;
Primitive	char, boolean	2,1 bytes	char c = 'A';
User Defined	Class, Interface, Array	Variable	Student s;

- **Primitive:** stack માં store થાય, જડાઈ access
- **User Defined:** heap માં store થાય, જટિલ operations

### મેમરી ટ્રીક

“Primitive Built-in, User Custom”

## પ્રશ્ન 2(બ OR) [4 ગુણ]

ચોંચ ઉદાહરણ સાથે this કીવર્ડ સમજાવો.

### જવાબ

This કીવર્ડ વર્તમાન object instance ને refer કરે છે, instance અને local variables વચ્ચે લેદ પાડવા માટે વાપરાય છે.  
ટેબલ: This કીવર્ડના ઉપયોગો

ઉપયોગ	હેતુ	ઉદાહરણ
this.variable	instance variable ને access કરે	this.name = name;
this.method()	instance method ને કોલ કરે	this.display();
this()	constructor ને કોલ કરે	this(name, age);

### કોડ એલોક:

```
class Student {
    String name;
    int age;

    Student(String name, int age) {
        this.name = name; // this instance parameter
        this.age = age; // 
    }

    void setData(String name) {
        this.name = name; // this object refer
    }

    void display() {
        System.out.println(this.name + " " + this.age);
    }
}
```

### મેમરી ટ્રીક

“This વર્તમાન Object”

## પ્રશ્ન 2(ક) OR) [7 ગુણ]

ઇનહેરિટન્સ વ્યાખ્યાયિત કરો. તેના પ્રકારોની ચાદી બનાવો. multilevel અને hierarchical ઇનહેરિટન્સને ચોગ્ય ઉદાહરણ સાથે સમજાવો.

### જવાબ

**Inheritance:** mechanism જેમાં child class parent class ના properties અને methods મેળવે છે.  
**ટેબલ:** Inheritance ના પ્રકારો

પ્રકાર	વર્ણન	સ્ટ્રક્ચર
<b>Single</b>	એક parent, એક child	$A \rightarrow B$
<b>Multilevel</b>	inheritance ની chain	$A \rightarrow B \rightarrow C$
<b>Hierarchical</b>	એક parent, અનેક children	$A \rightarrow B, A \rightarrow C$
<b>Multiple</b>	અનેક parents (interfaces દ્વારા)	$B, C \rightarrow A$

### આફ્ટિ - Multilevel:

Mermaid Diagram (Code)

```
{Shaded}
{Highlighting} []
graph LR
    A[Animal] --> B[Mammal]
    B --> C[Dog]
{Highlighting}
{Shaded}
```

### કોડ વ્યૂહ - Multilevel:

```
class Animal {
    void eat() {
        System.out.println("Animal eats");
    }
}

class Mammal extends Animal {
    void breathe() {
        System.out.println("Mammal breathes");
    }
}

class Dog extends Mammal {
    void bark() {
        System.out.println("Dog barks");
    }
}
```

### આફ્ટિ - Hierarchical:

Mermaid Diagram (Code)

```
{Shaded}
{Highlighting} []
graph TD
    A[Shape] --> B[Circle]
    A --> C[Rectangle]
    A --> D[Triangle]
{Highlighting}
{Shaded}
```

### કોડ વ્યૂહ - Hierarchical:

```
class Shape {
    void draw() {
        System.out.println("Drawing shape");
    }
}

class Circle extends Shape {
    void drawCircle() {
        System.out.println("Drawing circle");
    }
}

class Rectangle extends Shape {
    void drawRectangle() {
        System.out.println("Drawing rectangle");
    }
}
```

## પ્રશ્ન 3(અ) [3 ગુણ]

જાવામાં ટાઇપ કન્વર્ઝન અને કાસ્ટિંગ સમજાવો.

## જવાબ

**Type Conversion:** એક data type ને બીજામાં બદલવું. **Casting:** programmer દ્વારા explicit type conversion.  
**ટેબલ:** Type Conversion

Type	Description	Example
Implicit (Widening)	Automatic, smaller to larger	int to double
Explicit (Narrowing)	Manual, larger to smaller	double to int

## કોડ લોક:

```
// Implicit conversion
int i = 10;
double d = i;           // int   double (automatic)

// Explicit casting
double x = 10.5;
int y = (int) x;         // double   int (manual)

// String conversion
String s = String.valueOf(i);    // int   String
int z = Integer.parseInt("123"); // String   int
```

## મેમરી ટ્રીક

## પ્રશ્ન 3(બ) [4 ગુણ]

Java માં ઉપયોગમાં લેવાતા વિવિધ visibility controls સમજાવો.

## જવાબ

**Visibility Controls (Access Modifiers):** classes, methods અને variables ની access ને control કરે છે.  
**ટેબલ:** Access Modifiers

Modifier	Same Class	Same Package	Subclass	Different Package
private	□	□	□	□
default	□	□	□	□
protected	□	□	□	□
public	□	□	□	□

### કોડ લોક:

```
class Example \{
    private int x = 10;          // class
    int y = 20;                 // Package level
    protected int z = 30;        // Package + subclass
    public int w = 40;           //

    private void method1() \{ \}   // Private method
    public void method2() \{ \}   // Public method
\}
```

### મેમરી ટ્રીક

“Private Package Protected Public”

### પ્રશ્ન 3(ક) [7 ગુણ]

વ્યાખ્યાપિત કરો: શ્રેડ. શ્રેડ બનાવવા માટે ઉપયોગમાં લેવાતી વિવિધ પદ્ધતિઓની સૂચિ બનાવો. શ્રેડની લાઇફ સાઇકલ વિગતવાર સમજાવો.

#### જવાબ

**Thread:** lightweight subprocess જે program ના અનેક ભાગોને concurrent execution ની મંજૂરી આપે છે.

**ટેબલ:** Thread બનાવવાની પદ્ધતિઓ

પદ્ધતિ	વર્ણન	ઉદાહરણ
<b>Thread વિસ્તરિત</b>	Thread class ને inherit કરવું	class MyThread extends Thread
<b>Runnable implement કરવું</b>	Runnable interface implement કરવું	class MyTask implements Runnable

## આફ્ટિ: Thread Life Cycle

```
stateDiagram{-v2}
    direction LR
    [*] {-{->} NEW}
    NEW {-{->} RUNNABLE : start()}
    RUNNABLE {-{->} RUNNING : CPU allocation}
    RUNNING {-{->} RUNNABLE : yield()}
    RUNNING {-{->} BLOCKED : wait()/sleep()}
    BLOCKED {-{->} RUNNABLE : notify()/interrupt()}
    RUNNABLE {-{->} TERMINATED : completion}
    TERMINATED {-{->} [*]}
```

## ટેબલ: Thread States

સ્થિતિ	વર્ણન
<b>NEW</b>	Thread બન્યું પણ શરૂ નથી થયું
<b>RUNNABLE</b>	ચાલવા તૈયાર, CPU ની રાહમાં
<b>RUNNING</b>	હાલમાં execute થઈ રહ્યું છે
<b>BLOCKED</b>	resource અથવા sleep ની રાહમાં
<b>TERMINATED</b>	execution પૂર્ણ થયું

## કોડ એલોક:

```
//      1: Thread
class MyThread extends Thread \{
    public void run() \{
        System.out.println("Thread running");
    }
}

//      2: Runnable implement
class MyTask implements Runnable \{
    public void run() \{
        System.out.println("Task running");
    }
}

class Main \{
    public static void main(String[] args) \{
        MyThread t1 = new MyThread();
        Thread t2 = new Thread(new MyTask());
        t1.start();
        t2.start();
    }
}
```

## મેમરી ટ્રીક

“Thread Concurrent Execution”

## પ્રશ્ન 3(અ OR) [3 ગુણ]

java માં JVM નો હેતુ સમજાવો.

જવાબ

**JVM (Java Virtual Machine):** runtime environment જે Java bytecode execute કરે છે અને platform independence પ્રદાન કરે છે.

## ટેબલ: JVM Components

Component	ફુલ
Class Loader	.class files ને memory માં લોડ કરે
Execution Engine	bytecode execute કરે
Memory Area	heap અને stack memory ને manage કરે
Garbage Collector	automatic memory management

## આકૃતિ:

```
+{--}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}+  
| Java Source      |  
| (.java)          |  
+{--}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}+  
|  
| v  
+{--}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}+  
| Java Compiler    |  
| (javac)          |  
+{--}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}+  
|  
| v  
+{--}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}+  
| Bytecode          |  
| (.class)         |  
+{--}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}+  
|  
| v  
+{--}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}+  
| JVM               |  
| (Platform)       |  
| Specific          |  
+{--}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}+
```

- **Platform Independence:** “એકવાર લખો, દરેક જગ્યાએ ચલાવો”
  - **Memory Management:** automatic garbage collection
  - **Security:** Bytecode verification

ਮੇਮਰੀ ਟੀਕ

## “JVM Java Virtual Machine”

### પ્રશ્ન 3(બ OR) [4 ગુણ]

વ्याख्यापित कરો: પેકેજ. ચોંગ ઉદાહરણ સાથે પેકેજ બનાવવા માટેના પગલાંઓ લખો.

ଜୟାମ୍

**Package:** સંબંધિત classes અને interfaces નો સંગ્રહ, namespace અને access control પ્રદાન કરે છે.

ટેબલ: Package ના ફાયદા

ફાયદો	વર્ણન
<b>Namespace</b>	નામની ટકરાર ટાળે

<b>Access Control</b>	બહેતર encapsulation
<b>Organization</b>	logical grouping
<b>Reusability</b>	maintain કરવું સરળ

#### Package બનાવવાના પગલાં:

1. Package declare કરો file ની ટોચે
2. Directory structure બનાવો package name મુજબ
3. Compile કરો package structure સાથે
4. Import કરો અન્ય classes માં

#### કોડ ટ્લોક:

```
// File: com/company/utilities/Calculator.java
package com.company.utilities;

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

// File: Main.java
import com.company.utilities.Calculator;

class Main {
    public static void main(String[] args) {
        Calculator calc = new Calculator();
        System.out.println(calc.add(5, 3));
    }
}
```

#### Directory Structure:

```
com/
  company/
    utilities/
      Calculator.class
  Main.class
```

#### મેમરી ટ્રીક

“Package Classes ને Group કરો”

## પ્રશ્ન 3(ક OR) [7 ગુણ]

શેડમાં Synchronization ને યોગ્ય ઉદાહરણ સાથે સમજાવો.

#### જવાબ

**Synchronization:** shared resources ના access ને multiple threads દ્વારા control કરવાની mechanism, data inconsistency ટાજવા માટે.

**ટેબલ:** Synchronization ના પ્રકારો

પ્રકાર	વર્ણન	ઉપયોગ
Synchronized method	આપી method lock થાય	synchronized void method()
Synchronized block	ચોક્કસ code block lock થાય	synchronized(object) { }
Static synchronization	Class level locking	synchronized static void method()

### કોડ ફોલોક:

```
class Counter \{
    private int count = 0;

    // Synchronized method
    public synchronized void increment() \{
        count++;
    \}

    // Synchronized block
    public void decrement() \{
        synchronized(this) \{
            count{-{-};}
        \}
    \}

    public int getCount() \{
        return count;
    \}
\}

class CounterThread extends Thread \{
    Counter counter;

    CounterThread(Counter c) \{
        counter = c;
    \}

    public void run() \{
        for(int i = 0; i {} 1000; i++) \{
            counter.increment();
        \}
    \}
\}

class Main \{
    public static void main(String[] args) throws InterruptedException \{
        Counter c = new Counter();
        CounterThread t1 = new CounterThread(c);
        CounterThread t2 = new CounterThread(c);

        t1.start();
        t2.start();

        t1.join();
        t2.join();

        System.out.println("Final count: " + c.getCount());
    \}
\}
```

### મેમરી ટ્રીક

“Synchronization Race Conditions અટકાવે”

### પ્રશ્ન 4(અ) [3 ગુણ]

સિંગ્ટ્રેન્ચ કલાસ અને સિંગ્ટ્રેન્ચબફર કલાસ વચ્ચે તફાવત કરો.

## જવાબ

### ટેબલ: String vs StringBuffer

પાસું	String	StringBuffer
<b>Mutability</b>	Immutable (બદલતું નથી)	Mutable (બદલાય છે)
<b>Performance</b>	concatenation માટે ધીમું	concatenation માટે જડ્યું
<b>Memory</b>	દર વખતે નવું object બનાવે	હાલનું object modify કરે
<b>Thread Safety</b>	Thread safe	Thread safe
<b>Methods</b>	concat(), substring()	append(), insert(), delete()

#### કોડ અલોક:

```
// String {- Immutable}
String s1 = "Hello";
s1 = s1 + " World"; // String object

// StringBuffer {- Mutable}
StringBuffer sb = new StringBuffer("Hello");
sb.append(" World"); // object modify
```

- **String:** જ્યારે content વારંવાર બદલતું નથી ત્યારે વાપરો
- **StringBuffer:** જ્યારે વારંવાર modifications જરૂરી હોય ત્યારે વાપરો

## મેમરી ટ્રીક

``String Immutable, StringBuffer Mutable''

## પ્રશ્ન 4(બ) [4 ગુણ]

એસેની 10 સંખ્યાઓનો સરવાળો અને સરેરાશ મેળવવા માટે જાવા પ્રોગ્રામ લખો.

## જવાબ

#### કોડ અલોક:

```
class ArraySum {
    public static void main(String[] args) {
        // 10 numbers array initialize
        int[] numbers = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};

        int sum = 0;

        // calculate
        for(int i = 0; i < numbers.length; i++) {
            sum += numbers[i];
        }

        // calculate
        double average = (double) sum / numbers.length;

        // display
        System.out.println("Array elements: ");
        for(int num : numbers) {
            System.out.print(num + " ");
        }

        System.out.println("\nSum: " + sum);
        System.out.println("Average: " + average);
    }
}
```

```
\}\n\\}
```

### આઉટપુદ:

```
Array elements: 10 20 30 40 50 60 70 80 90 100\nSum: 550\nAverage: 55.0
```

### લોજિક પગલાં:

1. Initialize 10 numbers સાથે array
2. Loop array માં સરવાળો calculate કરવા માટે
3. Calculate સરેરાશ = સરવાળો / length
4. Display પરિણામો

### મેમરી ટ્રીક

“Loop Sum Divide Average”

## પ્રશ્ન 4(ક) [7 ગુણ]

I) હોય ઉદાહરણ સાથે abstract class સમજાવો. II) હોય ઉદાહરણ સાથે final class સમજાવો.

### જવાબ

I) **Abstract Class:** class જેનું instantiation થઈ શકતું નથી, abstract methods હોય છે જેનું implementation subclasses માં જરૂરી છે.

ટેબલ: Abstract Class ની વિશેષતાઓ

વિશેષતા	વર્ણન
Cannot instantiate	object બનાવી શકતું નથી
Abstract methods	implementation લિનાની methods
Concrete methods	implementation સાથેની methods
Inheritance	Subclasses માં abstract methods implement કરવા જરૂરી

### કોડ એલોક - Abstract Class:

```
abstract class Shape \{
    String color;

    // Abstract method
    abstract void draw();

    // Concrete method
    void setColor(String c) \{
        color = c;
    \}
\}

class Circle extends Shape \{
    void draw() \{
        System.out.println("Drawing Circle");
    \}
\}

class Main \{
    public static void main(String[] args) \{
        // Shape s = new Shape(); // : Instantiate
        Circle c = new Circle();
        c.draw();
    \}
\}
```

II) Final Class: class જેનું extension થઈ શકતું નથી (inheritance નથી).  
દેખાયાં: Final Class ની વિશેષતાઓ

વિશેષતા	વર્ણન
No inheritance	extend કરી શકતું નથી
Security	modification અટકાવે છે
Performance	બહેતર optimization
ઉદાહરણો	String, Integer, System

### કોડ એલોક - Final Class:

```
final class FinalClass \{
    void display() \{
        System.out.println("This is final class");
    \}
\}

// class SubClass extends FinalClass \{ \// : Extend \}

class Main \{
    public static void main(String[] args) \{
        FinalClass obj = new FinalClass();
        obj.display();
    \}
\}
```

### મેમરી ટ્રીક

“Abstract અધૂરું, Final પૂરું”

#### પ્રશ્ન 4(અ OR) [3 ગુણ]

## જાવામાં Garbage Collection સમજાવો.

જવાબ

**Garbage Collection:** automatic memory management process જે heap memory માંથી unused objects ને remove કરે છે.

ટેબલ: GC ના ફાયદા

ફાયદો	વર્ણન
<b>Automatic Memory leak prevention</b>	manual memory management નથી unreferenced objects ને દૂર કરે
<b>Performance</b>	memory usage optimize કરે
<b>Safety</b>	memory errors અટકાવે

અક્તિ:

```
+{--}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}+  
| Object created      |  
| (new keyword)       |  
+{--}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}+  
|  
| v  
+{--}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}+  
| Object in use       |  
| (has references)   |  
+{--}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}+  
|  
| v  
+{--}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}+  
| No references       |  
| (eligible for GC) |  
+{--}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}+  
|  
| v  
+{--}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}+  
| Garbage Collector |  
| removes object    |  
+{--}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}+
```

- ક્યારે થાય: જ્યારે heap memory ઓછી હોય અથવા System.gc() કોલ થાય
  - Process: Mark and Sweep algorithm
  - Guarantee કરી શકતું નથી: garbage collection નો exact timing

ਮੇਮਰੀ ਟੀਕ

## “GC Automatic Memory Cleanup”

#### પ્રશ્ન 4(બ) OR) [4 ગુણ]

'Divide by Zero' એરર માટે ચુક્કર ડિફાઇન એક્સેપ્શન હેન્ડલ કરવા માટે જાવા પ્રોગ્રામ લખો.

ଜ୍ଵାବୁ

କ୍ଷେତ୍ର ପାଇଁ

```
//      exception class
class DivideByZeroException extends Exception {\
```

```

public DivideByZeroException(String message) \{
    super(message);
\}

class Calculator \{
    public static double divide(int a, int b) throws DivideByZeroException \{
        if(b == 0) \{
            throw new DivideByZeroException("           !");
        \}
        return (double) a / b;
\}
\}

class Main \{
    public static void main(String[] args) \{
        try \{
            int num1 = 10;
            int num2 = 0;

            double result = Calculator.divide(num1, num2);
            System.out.println("   : " + result);

        \} catch(DivideByZeroException e) \{
            System.out.println("   : " + e.getMessage());
        \}
    \}
\}

```

**આઉટપુટ:**

: !

**પગલાં:**

1. Custom exception class બનાવો Exception extend કરીને
2. Exception throw કરો જ્યારે condition આવે
3. Exception handle કરો try-catch block સાથે

### મેમરી ટ્રીક

“Custom Exception Error Handle કરો”

## પ્રશ્ન 4(ક OR) [7 ગુણ]

Multiple try block અને multiple catch block exception દર્શાવવા માટે જાવા પ્રોગ્રામ લખો.

### જવાબ

**કોડ વ્લોક:**

```

class MultipleExceptionDemo \{
    public static void main(String[] args) \{
        //      try block
        try \{
            int[] arr = \{1, 2, 3\;};
            System.out.println("Array element: " + arr[5]); // ArrayIndexOutOfBoundsException
        \}
        catch(ArrayIndexOutOfBoundsException e) \{
            System.out.println("Array index   : " + e.getMessage());
        \}
    \}
\}

```

```

    \}
    catch(Exception e) \{
        System.out.println("      exception: " + e.getMessage());
    \}

    //    try block
    try \{
        String str = null;
        System.out.println("String length: " + str.length()); // NullPointerException
    \}
    catch(NullPointerException e) \{
        System.out.println("Null pointer : " + e.getMessage());
    \}

    //    try block multiple catch
    try \{
        int a = 10;
        int b = 0;
        int result = a / b; // ArithmeticException

        String s = "abc";
        int num = Integer.parseInt(s); // NumberFormatException
    \}
    catch(ArithmaticException e) \{
        System.out.println("Arithmatic : " + e.getMessage());
    \}
    catch(NumberFormatException e) \{
        System.out.println("Number format : " + e.getMessage());
    \}
    catch(Exception e) \{
        System.out.println("      : " + e.getMessage());
    \}
    finally \{
        System.out.println("          ");
    \}
\}
\}

```

### આઉટપુટ:

```

Array index : Index 5 out of bounds for length 3
Null pointer : null
Arithmatic : / by zero

```

### દર્શાવેલી વિશેષતાઓ:

- Multiple try blocks:** દરેક અલગ operations handle કરે
- Multiple catch blocks:** દરેક specific exception type handle કરે
- Exception hierarchy:** સામાન્ય Exception બધાને catch કરે
- Finally block:** હંમેશા execute થાય

### મેમરી ટ્રીક

“Multiple Try Multiple Catch”

### પ્રશ્ન 5(અ) [૩ ગુણ]

ફાઇલ બનાવવા અને આ ફાઇલ પર write operation કરવા માટે જાવામાં પ્રોગ્રામ લખો.

## જવાબ

### કોડ અલોક:

```

import java.io.*;

class FileWriteDemo ^{
    public static void main(String[] args) ^{
        try ^{
            //
            File file = new File("demo.txt");

            // FileWriter object
            FileWriter writer = new FileWriter(file);

            //
            writer.write("      !{n}");
            writer.write(" Java      .{n}");
            writer.write("      .");

            // writer
            writer.close();

            System.out.println("      !");
        } catch(IOException e) ^{
            System.out.println(" : " + e.getMessage());
        }
    }
}

```

### પગલાં:

1. Import java.io package
2. File object બનાવો filename સાથે
3. FileWriter object બનાવો
4. કેરા લખો write() method વાપરીને
5. Writer બંધ કરો changes save કરવા માટે

## મેમરી ટ્રીક

“File Writer Write Close”

## પ્રશ્ન 5(બ) [4 ગુણા]

Throw અને finally ને Exception Handling માં ઉદાહરણ સાથે સમજાવો.

## જવાબ

**Throw:** keyword જેનો ઉપયોગ explicitly exception throw કરવા માટે થાય. **Finally:** block જે exception આવે કે ન આવે હુમેશા execute થાય છે.

**ટેબલ: Throw vs Finally**

કોર્ડ	હેતુ	ઉપયોગ
throw	Explicitly exception throw કરે	throw new Exception()
finally	હુમેશા cleanup code execute કરે	finally {}

### કોડ લોક:

```
class ThrowFinallyDemo {\n    public static void checkAge(int age) throws Exception {\n        if(age < 18) {\n            throw new Exception(" 18 ");\n        }\n        System.out.println(" : " + age);\n    }\n\n    public static void main(String[] args) {\n        try {\n            checkAge(15); // Exception throw\n        }\n        catch(Exception e) {\n            System.out.println(" : " + e.getMessage());\n        }\n        finally {\n            System.out.println("Finally block execute ");\n        }\n    }\n}
```

### આઉટપુટ:

```
: 18\nFinally block execute
```

- **Throw:** Exception બજ્યૂર્વક બનાવે
- **Finally:** Cleanup code, resources બંધ કરે

### મેમરી ટ્રીક

“Throw Exception, Finally હંમેશા”

## પ્રશ્ન 5(ક) [7 ગુણ]

વાર્ષિકો: પોલીમોર્ફિઝમ. જાવામાં ચોગ્ય ઉદાહરણ સાથે રન ટાઇમ પોલીમોર્ફિઝમ સમજાવો.

### જવાબ

**Polymorphism:** એક interface, અનેક implementations. Object તેના actual type પર આધાર રાખીને અલગ રીતે વર્તે છે.  
**ટેબલ:** Polymorphism ના પ્રકારો

પ્રકાર	વર્ણન	ક્યારે નક્કી થાય
Compile-time	Method overloading	Compilation વખતે
Run-time	Method overriding	Execution વખતે

**Run-time Polymorphism:** Method call runtime पर actual object type आधारे resolve थाय.  
આકૃતિ:

#### Mermaid Diagram (Code)

```
{Shaded}
{Highlighting} []
graph LR
    A[Animal reference] --> B[Dog object]
    A --> C[Cat object]
    B --> D[Dog sound]
    C --> E[Cat sound]
{Highlighting}
{Shaded}
```

#### કોડ વીક:

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

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

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

class Main {
    public static void main(String[] args) {
        Animal animal1 = new Dog(); // Upcasting
        Animal animal2 = new Cat(); // Upcasting

        animal1.makeSound(); //      :
        animal2.makeSound(); //      :

        // Animals array
        Animal[] animals = {new Dog(), new Cat(), new Dog()};
        for(Animal a : animals) {
            a.makeSound(); // Dynamic method dispatch
        }
    }
}
```

#### આઉટપુટ:

#### વિશેષતાઓ:

- **Dynamic Method Dispatch:** JVM runtime पर નક્કી કરે છે કઈ method કોલ કરવી
- **Upcasting:** Child object ને parent reference દ્વારા reference કરવું
- **Flexibility:** સમાન કોડ વિવિધ object types સાથે કામ કરે

**પ્રશ્ન 5(અ OR) [3 ગુણ]**

જાવામાં એક પ્રોગ્રામ લખો જે બાઇટ બાય બાઇટ ફાઇલના કન્ટેન્ટ વાંચો અને તેને બીજું ફાઇલમાં કોપી કરો.

**જવાબ****કોડ લાલોક:**

```
import java.io.*;

class FileCopyDemo {
    public static void main(String[] args) {
        try {
            // source file           input stream
            FileInputStream input = new FileInputStream("source.txt");

            // destination file      output stream
            FileOutputStream output = new FileOutputStream("destination.txt");

            int byteData;

            //
            while((byteData = input.read()) != -1) {
                output.write(byteData);
            }

            // streams
            input.close();
            output.close();

            System.out.println("!");
        } catch(IOException e) {
            System.out.println(" : " + e.getMessage());
        }
    }
}
```

**પગલાં:**

1. `FileInputStream` બનાવો વાંચવા માટે
2. `FileOutputStream` બનાવો લખવા માટે
3. બાઇટ બાય બાઇટ વાંચો `read()` વાપરીને
4. દરેક બાઇટ લખો `write()` વાપરીને
5. બંને streams બંધ કરો

**પ્રશ્ન 5(બ OR) [4 ગુણ]**

Java સાથે ઉપલબ્ધ વિવિધ I/O વર્ગો સમજાવો.

## ટેબલ: Java I/O Classes

Class Type	Class Name	હેતુ
<b>Byte Stream</b>	FileInputStream	ફાઇલમાંથી bytes વાંચે
<b>Byte Stream</b>	FileOutputStream	ફાઇલમાં bytes લખે
<b>Character Stream</b>	FileReader	ફાઇલમાંથી characters વાંચે
<b>Character Stream</b>	FileWriter	ફાઇલમાં characters લખે
<b>Buffered</b>	BufferedReader	કાર્યક્ષમ character reading
<b>Buffered</b>	BufferedWriter	કાર્યક્ષમ character writing

## આફ્ટિ: I/O Class Hierarchy

```
+{--{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}+}
|   InputStream   |
+{--{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}+}
|           |
+{--{-} FileInputStream}
+{--{-} BufferedInputStream}

+{--{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}+}
|   OutputStream   |
+{--{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}+}
|           |
+{--{-} FileOutputStream}
+{--{-} BufferedOutputStream}

+{--{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}+}
|   Reader   |
+{--{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}+}
|           |
+{--{-} FileReader}
+{--{-} BufferedReader}

+{--{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}+}
|   Writer   |
+{--{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}+}
|           |
+{--{-} FileWriter}
+{--{-} BufferedWriter}
```

## કોડ ઉદાહરણ:

```
// Byte streams
FileInputStream fis = new FileInputStream("file.txt");
FileOutputStream fos = new FileOutputStream("output.txt");

// Character streams
FileReader fr = new FileReader("file.txt");
FileWriter fw = new FileWriter("output.txt");

// Buffered streams
BufferedReader br = new BufferedReader(new FileReader("file.txt"));
BufferedWriter bw = new BufferedWriter(new FileWriter("output.txt"));
```

## મેમરી ટ્રીક

``Byte Character Buffered Streams''

## પ્રશ્ન 5(ક) OR) [૭ ગુણ]

જાવા પ્રોગ્રામ લખો જે બે થ્રેડોને એક્ઝિક્યુટ કરે છે. એક થ્રેડ દર 3 સેકન્ડ્સ "Java Programming" દર્શાવે છે, અને બીજો દર 6 સેકન્ડ્સ "Semester - 4th IT" દર્શાવે છે. (થ્રેડને extending the Thread class દ્વારા બનાવો)

### જવાબ

#### કોડ વ્લોક:

```
class JavaThread extends Thread \{
    public void run() \{
        try \{
            while(true) \{
                System.out.println("Java Programming");
                Thread.sleep(3000); // 3
            \}
        \} catch(InterruptedException e) \{
            System.out.println("JavaThread");
        \}
    \}
}

class SemesterThread extends Thread \{
    public void run() \{
        try \{
            while(true) \{
                System.out.println("Semester {- 4th IT}");
                Thread.sleep(6000); // 6
            \}
        \} catch(InterruptedException e) \{
            System.out.println("SemesterThread");
        \}
    \}
}

class Main \{
    public static void main(String[] args) \{
        // thread objects
        JavaThread javaThread = new JavaThread();
        SemesterThread semesterThread = new SemesterThread();

        // threads
        javaThread.start();
        semesterThread.start();

        // threads 20
        try \{
            Thread.sleep(20000);
            javaThread.interrupt();
            semesterThread.interrupt();
        \} catch(InterruptedException e) \{
            System.out.println("Main thread");
        \}
    \}
}
```

#### નમૂના આઉટપુટ:

```
Java Programming
Semester - 4th IT
Java Programming
Java Programming
Semester - 4th IT
Java Programming
```

...

**વિશેષતાઓ:**

- બે અલગ threads: દરેક અલગ timing સાથે
- Thread.sleep(): ચોક્કસ milliseconds માટે execution પોર્ઝ કરે
- Concurrent execution: બંને threads એકસાથે ચાલે છે
- Thread class વિસ્તાર્યું: run() method ને override કરવું

**Execution Pattern:**

- JavaThread: દર 3 સેકન્ડે દર્શાવે
- SemesterThread: દર 6 સેકન્ડે દર્શાવે
- બંને concurrent રીતે અલગ timing બતાવે છે

**મેમરી ટ્રીક**

“બે Threads અલગ Timing”