

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:

```
1 class Animal {  
2     String name = "Animal";  
3     void eat() { System.out.println("Animal eats"); }  
4 }  
5  
6 class Dog extends Animal {  
7     String name = "Dog";  
8     void eat() {  
9         super.eat(); // parent method  
10        System.out.println("Dog eats bones");  
11    }  
12    void display() {  
13        System.out.println(super.name); // "Animal"  
14    }  
15 }
```

મેમરી ટ્રીક

"Super Parent ને કોલ કરો"

પ્રશ્ન 1(ક) [7 ગુણ]

વાખ્યાપિત 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 શકાતો નથી
Return type	સમાન અથવા covariant હોવું જોઈએ

કોડ વ્લોક:

```
1 class Shape {
2     void draw() {
3         System.out.println("Drawing a shape");
4     }
5 }
6
7 class Circle extends Shape {
8     @Override
9     void draw() {
10         System.out.println("Drawing a circle");
11     }
12 }
13
14 class Main {
15     public static void main(String[] args) {
16         Shape s = new Circle();
17         s.draw(); // : Drawing a circle
18     }
19 }
```

મેમરી ટ્રીક

"Override સમાન Signature"

પ્રશ્ન 1(ક OR) [7 ગુણ]

વાણિય: ઇન્ટરફેસ, મલિટેપલ 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 ફોલોક:

```
1 interface Flyable {
2     void fly();
3 }
4
5 interface Swimmable {
6     void swim();
7 }
8
9 class Duck implements Flyable, Swimmable {
10     public void fly() {
11         System.out.println("Duck flies");
12     }
13
14     public void swim() {
15         System.out.println("Duck swims");
16     }
17 }
18
19 class Main {
20     public static void main(String[] args) {
21         Duck d = new Duck();
22         d.fly();
23         d.swim();
24     }
25 }
```

મેમરી ટ્રીક

“Interface Multiple Implementation”

પ્રશ્ન 2(અ) [૩ ગુણ]

ઉદાહરણ સાથે Java પ્રોગ્રામ સ્ટ્રક્ચર સમજાવો.

જવાબ

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

```
1 +-----+
2 | Package statement|
3 +-----+
4 | Import statements|
5 +-----+
6 | Class declaration|
7 | +-----+ |
8 | | Variables | |
9 | +-----+ |
10| | Methods | |
11| +-----+ |
12+-----+
```

કોડ ફોલોક:

```
1 package com.example;           // Package
2 import java.util.*;             // Import
3
4 public class HelloWorld {      // Class
5     static int count = 0;        // Variable
6
7     public static void main(String[] args) { // Main method
```

```

8     System.out.println("Hello World");
9 }
0 }
```

મેમરી ટ્રીક

“Package Import Class Main”

પ્રશ્ન 2(બ) [4 ગુણ]

ચોગ્ય 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
Static variable	બધા objects દ્વારા shared	static int count
Static method	object વિના કોલ થાય	static void display()
Static block	main પહેલાં execute થાય	static { }

કોડ વ્લોક:

```

1 class Student {
2     static String college = "GTU"; // static variable
3     String name;
4
5     static void showCollege() { // static method
6         System.out.println(college);
7     }
8
9     static { // static block
10        System.out.println("Static block executed");
11    }
12 }
13
14 class Main {
15     public static void main(String[] args) {
16         Student.showCollege(); // object
17     }
18 }
```

મેમરી ટ્રીક

“Static Class દ્વારા Shared”

પ્રશ્ન 2(ક) [7 ગુણ]

વ્યાખ્યાયિત કરો: કન્સ્ટ્રક્ટર. તેના પ્રકારોની યાદી બનાવો. પેરામીટ્રાઇઝ અને કોપી કન્સ્ટ્રક્ટરને ચોગ્ય 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 ફોલોક:

```

1 class Student {
2     String name;
3     int age;
4
5     // Parameterized constructor
6     Student(String n, int a) {
7         name = n;
8         age = a;
9     }
10
11    // Copy constructor
12    Student(Student s) {
13        name = s.name;
14        age = s.age;
15    }
16
17    void display() {
18        System.out.println(name + " " + age);
19    }
20
21
22 class Main {
23     public static void main(String[] args) {
24         Student s1 = new Student("John", 20); // Parameterized
25         Student s2 = new Student(s1);          // Copy
26         s1.display();
27         s2.display();
28     }
29 }
```

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

કેટેગરી	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 मां store थाय, जडपी access
- **User Defined:** heap मां store थाय, जटिल operations

મેમરી ટ્રીક

“Primitive Built-in, User Custom”

પ્રશ્ન 2(બ) OR) [4 ગુણ]

યોગ્ય ઉદાહરણ સાથે this કીવર્ડ સમજાવો.

જવાબ

This કીવર્ડ વર્તમાન object instance ને refer કરે છે, instance અને local variables વર્ચે ભેદ 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 વ્લોક:

```

1 class Student {
2     String name;
3     int age;
4
5     Student(String name, int age) {
6         this.name = name;      // this instance parameter
7         this.age = age;        //
8     }
9
10    void setData(String name) {
11        this.name = name;    // this object refer
12    }
13
14    void display() {
15        System.out.println(this.name + " " + this.age);
16    }
17 }
```

મેમરી ટ્રીક

“This વર્તમાન Object”

પ્રશ્ન 2(ક) OR) [7 ગુણ]

ઇનહેરિટન્સ વ્યાખ્યાયિત કરો. તેના પ્રકારોની યાદી બનાવો. multilevel અને hierarchical ઇનહેરિટન્સને યોગ્ય ઉદાહરણ સાથે સમજાવો.

જવાબ

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

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

આફ્ટિ - Multilevel:

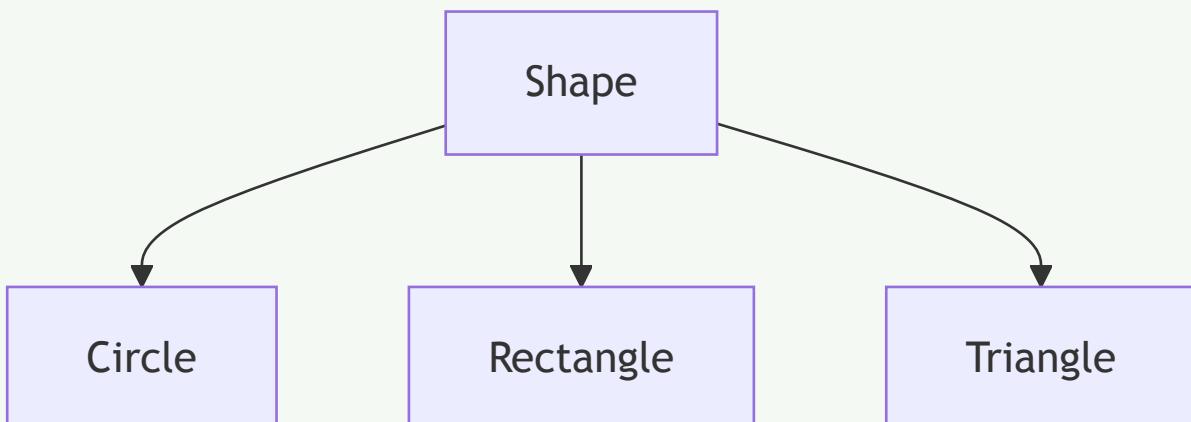


કોડ વ્યૂહ - Multilevel:

```

1 class Animal {
2     void eat() { System.out.println("Animal eats"); }
3 }
4
5 class Mammal extends Animal {
6     void breathe() { System.out.println("Mammal breathes"); }
7 }
8
9 class Dog extends Mammal {
10    void bark() { System.out.println("Dog barks"); }
11 }
  
```

આફ્ટિ - Hierarchical:



કોડ વ્યૂહ - Hierarchical:

```

1 class Shape {
2     void draw() { System.out.println("Drawing shape"); }
3 }
4
5 class Circle extends Shape {
6     void drawCircle() { System.out.println("Drawing circle"); }
7 }
8
9 class Rectangle extends Shape {
10    void drawRectangle() { System.out.println("Drawing rectangle"); }
11 }
  
```

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:

```

1 // Implicit conversion
2 int i = 10;
3 double d = i;           // int   double (automatic)
4
5 // Explicit casting
6 double x = 10.5;
7 int y = (int) x;        // double   int (manual)
8
9 // String conversion
10 String s = String.valueOf(i);    // int   String
11 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:

```
1 class Example {  
2     private int x = 10;           //    class  
3     int y = 20;                 // Package level  
4     protected int z = 30;        // Package + subclass  
5     public int w = 40;           //  
6  
7     private void method1() { }    // Private method  
8     public void method2() { }    // Public method  
9 }
```

મેમરી ટ્રીક

“Private Package Protected Public”

પદ્ધતિ [7 ગુણ]

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

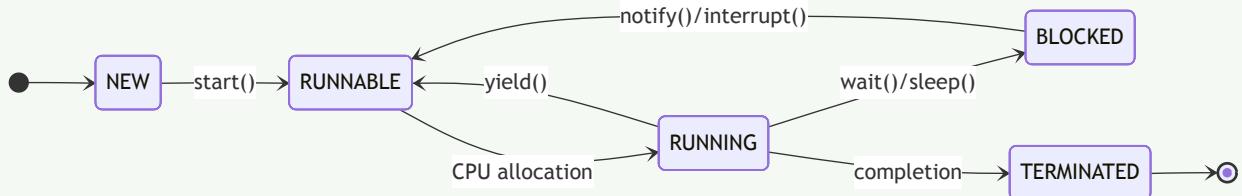
જવાબ

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

ટેબલ: Thread બનાવવાની પદ્ધતિઓ

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

આફ્ટિ: Thread Life Cycle



ટેબલ: Thread States

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

કોડ વ્લોક:

```

1 // 1: Thread
2 class MyThread extends Thread {
3     public void run() {
4         System.out.println("Thread running");
5     }
6 }
7
8 // 2: Runnable implement
9 class MyTask implements Runnable {
10    public void run() {
11        System.out.println("Task running");
12    }
13 }
14
15 class Main {
16     public static void main(String[] args) {
17         MyThread t1 = new MyThread();
18         Thread t2 = new Thread(new MyTask());
19         t1.start();
20         t2.start();
21     }
22 }
```

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	હેતુ
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:

```

1 +-----+
2 | Java Source   |
3 | (.java)        |
4 +-----+
5   |
6   v
7 +-----+
8 | Java Compiler  |
9 | (javac)        |
10 +-----+
11   |
12   v
13 +-----+
14 | Bytecode      |
15 | (.class)       |
16 +-----+
17   |
18   v
19 +-----+
20 | JVM           |
21 | (Platform     |
22 | Specific)     |
23 +-----+

```

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

મેમરી ટ્રીક

"JVM Java Virtual Machine"

પ્રશ્ન 3(બ OR) [4 ગુણ]

વાખ્યાયિત કરો: પેકેજ. ચોંચ ઉદાહરણ સાથે પેકેજ બનાવવા માટેના પગલાંઓ લખો.

જવાબ

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

ફાયદો	વર્ણન
Namespace	નામની ટકરાર ટાળો
Access Control	બહેતર encapsulation
Organization	logical grouping
Reusability	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 ટોક:

```
1 // File: com/company/utilities/Calculator.java
2 package com.company.utilities;
3
4 public class Calculator {
5     public int add(int a, int b) {
6         return a + b;
7     }
8 }
9
10 // File: Main.java
11 import com.company.utilities.Calculator;
12
13 class Main {
14     public static void main(String[] args) {
15         Calculator calc = new Calculator();
16         System.out.println(calc.add(5, 3));
17     }
18 }
```

Directory Structure:

```
1 com/
2   company/
3     utilities/
4       Calculator.class
5 Main.class
```

મેમરી ટ્રીક

“Package Classes ને Group કરો”

પ્રશ્ન 3(ક OR) [7 ગુણ]

શેડમાં Synchronization ને ચોખ્ય ઉદાહરણ સાથે સમજાવો.

જવાબ

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

ટેબલ: Synchronization ના પ્રકારો

પ્રકાર	વર્ણન	ઉપયોગ
Synchronized method	આખી 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 વેલોક:

```
1 class Counter {
2     private int count = 0;
3
4     // Synchronized method
5     public synchronized void increment() {
6         count++;
7     }
8
9     // Synchronized block
10    public void decrement() {
11        synchronized(this) {
12            count--;
13        }
14    }
15
16    public int getCount() {
17        return count;
18    }
19 }
20
21 class CounterThread extends Thread {
22     Counter counter;
23
24     CounterThread(Counter c) {
25         counter = c;
26     }
27
28     public void run() {
29         for(int i = 0; i < 1000; i++) {
30             counter.increment();
31         }
32     }
33 }
34
35 class Main {
36     public static void main(String[] args) throws InterruptedException {
37         Counter c = new Counter();
38         CounterThread t1 = new CounterThread(c);
39         CounterThread t2 = new CounterThread(c);
40
41         t1.start();
42         t2.start();
43
44         t1.join();
45         t2.join();
46
47         System.out.println("Final count: " + c.getCount());
48     }
49 }
```

મેમરી ટ્રીક

“Synchronization Race Conditions અટકાવે”

પ્રશ્ન 4(અ) [3 ગુણ]

સિંગ્ર કલાસ અને સિંગ્રબફર કલાસ વચ્ચે તફાવત કરો.

ટેબલ: String vs StringBuffer

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

કોડ ફોક:

```

1 // String - Immutable
2 String s1 = "Hello";
3 s1 = s1 + " World"; // String object
4
5 // StringBuffer - Mutable
6 StringBuffer sb = new StringBuffer("Hello");
7 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]

એરેની 10 સંખ્યાઓનો સરવાળો અને સરેરાશ મેળવવા માટે જવા પ્રોગ્રામ લખો.

કોડ ફોક:

```

1 class ArraySum {
2     public static void main(String[] args) {
3         // 10 numbers array initialize
4         int[] numbers = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};
5
6         int sum = 0;
7
8         // calculate
9         for(int i = 0; i < numbers.length; i++) {
10             sum += numbers[i];
11         }
12
13         // calculate
14         double average = (double) sum / numbers.length;
15
16         // display
17         System.out.println("Array elements: ");
18         for(int num : numbers) {
19             System.out.print(num + " ");
20         }
21
22         System.out.println("\nSum: " + sum);
23         System.out.println("Average: " + average);
24     }
25 }

```

આઉટપુટ:

```
1 Array elements: 10 20 30 40 50 60 70 80 90 100  
2 Sum: 550  
3 Average: 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) પોઝિય ઉદાહરણ સાથે abstract class સમજાવો. II) પોઝિય 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 ફોલ્ક - Abstract Class:

```
1 abstract class Shape {
2     String color;
3
4     // Abstract method
5     abstract void draw();
6
7     // Concrete method
8     void setColor(String c) {
9         color = c;
10    }
11 }
12
13 class Circle extends Shape {
14     void draw() {
15         System.out.println("Drawing Circle");
16     }
17 }
18
19 class Main {
20     public static void main(String[] args) {
21         // Shape s = new Shape(); //  : Instantiate
22         Circle c = new Circle();
23         c.draw();
24     }
25 }
```

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

ટેબલ: Final Class ની વિશેષતાઓ

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

કોડ ફોલ્ક - Final Class:

```
1 final class FinalClass {
2     void display() {
3         System.out.println("This is final class");
4     }
5 }
6
7 // class SubClass extends FinalClass { } //  : Extend
8
9 class Main {
10     public static void main(String[] args) {
11         FinalClass obj = new FinalClass();
12         obj.display();
13     }
14 }
```

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
Automatic	manual memory management નથી
Memory leak prevention	unreferenced objects ને દૂર કરે
Performance	memory usage optimize કરે
Safety	memory errors અટકાવે

આફિસિયલ:

```

1 +-----+
2 | Object created   |
3 | (new keyword)   |
4 +-----+
5 |           |
6 |           v
7 +-----+
8 | Object in use   |
9 | (has references) |
10 +-----+
11 |           |
12 |           v
13 +-----+
14 | No references   |
15 | (eligible for GC) |
16 +-----+
17 |           |
18 |           v
19 +-----+
20 | Garbage Collector|
21 | removes object   |
22 +-----+

```

- કાર્ય 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

મેમરી ટ્રીક

“GC Automatic Memory Cleanup”

પ્રશ્ન 4(બં OR) [4 ગુણ]

‘Divide by Zero’ એરર માટે ચુઝર ડિફાઇન્ડ એક્સેપ્શન હેન્ડલ કરવા માટે જાવા પ્રોગ્રામ લખો.

જવાબ

કોડ વિલોક્ષણ:

```

1 //      exception class
2 class DivideByZeroException extends Exception {
3     public DivideByZeroException(String message) {
4         super(message);
5     }
6 }
7
8 class Calculator {
9     public static double divide(int a, int b) throws DivideByZeroException {

```

```

0     if(b == 0) {
1         throw new DivideByZeroException("           !");
2     }
3     return (double) a / b;
4 }
5
6
7 class Main {
8     public static void main(String[] args) {
9         try {
10             int num1 = 10;
11             int num2 = 0;
12
13             double result = Calculator.divide(num1, num2);
14             System.out.println("    : " + result);
15
16         } catch(DivideByZeroException e) {
17             System.out.println(" : " + e.getMessage());
18         }
19     }
20 }
```

આઉટપુટ:

```

1 :
2 !
```

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 વ્યૂક:

```

1 class MultipleExceptionDemo {
2     public static void main(String[] args) {
3         //      try block
4         try {
5             int[] arr = {1, 2, 3};
6             System.out.println("Array element: " + arr[5]); // ArrayIndexOutOfBoundsException
7         }
8         catch(ArrayIndexOutOfBoundsException e) {
9             System.out.println("Array index : " + e.getMessage());
10        }
11        catch(Exception e) {
12            System.out.println("      exception: " + e.getMessage());
13        }
14
15        //      try block
16        try {
17            String str = null;
18            System.out.println("String length: " + str.length()); // NullPointerException
19        }
```

```

20     catch(NullPointerException e) {
21         System.out.println("Null pointer : " + e.getMessage());
22     }
23
24     // try block multiple catch
25     try {
26         int a = 10;
27         int b = 0;
28         int result = a / b; // ArithmeticException
29
30         String s = "abc";
31         int num = Integer.parseInt(s); // NumberFormatException
32     }
33     catch(ArithmaticException e) {
34         System.out.println("Arithmatic : " + e.getMessage());
35     }
36     catch(NumberFormatException e) {
37         System.out.println("Number format : " + e.getMessage());
38     }
39     catch(Exception e) {
40         System.out.println(" : " + e.getMessage());
41     }
42     finally {
43         System.out.println(" ");
44     }
45 }

```

આઉટપુટ:

```

1 Array index : Index 5 out of bounds for length 3
2 Null pointer : null
3 Arithmatic : / by zero
4

```

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) [3 ગુણ]

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

જવાબ

કોડ ડાલોક:

```

1 import java.io.*;
2
3 class FileWriteDemo {
4     public static void main(String[] args) {
5         try {
6             //
7             File file = new File("demo.txt");
8
9             // FileWriter object
10            FileWriter writer = new FileWriter(file);

```

```

1
2     // writer.write("      !\n");
3     writer.write(" Java      .\n");
4     writer.write("      .");
5
6
7     // writer
8     writer.close();
9
10    System.out.println("      !");
11
12 } catch(IOException e) {
13     System.out.println(" : " + e.getMessage());
14 }
15
16 }

```

પગલાં:

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

મેમરી ટ્રીક

``File Writer Write Close''

પ્રશ્ન 5(બુ) [4 ગુણ]

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 હુંમેશા execute થાય છે.

ટેબલ: Throw vs Finally

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

કોડ લોક:

```
1 class ThrowFinallyDemo {  
2     public static void checkAge(int age) throws Exception {  
3         if(age < 18) {  
4             throw new Exception("    18");  
5         }  
6         System.out.println("    : " + age);  
7     }  
8  
9     public static void main(String[] args) {  
10        try {  
11            checkAge(15); // Exception throw  
12        }  
13        catch(Exception e) {  
14            System.out.println(" : " + e.getMessage());  
15        }  
16        finally {  
17            System.out.println("Finally block execute ");  
18        }  
19    }  
20}
```

આઉટપુટ:

```
1 : 18  
2 Finally block execute  
3
```

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

મેમરી ટ્રીક

“Throw Exception, Finally હંમેશા”

પ્રશ્ન 5(ક) [7 ગુણ]

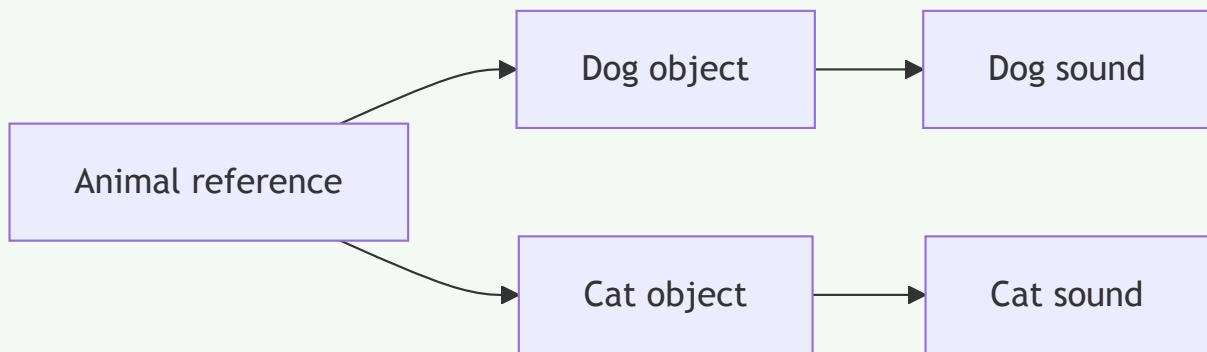
વણવો: પોલીમોર્ફિઝમ. જાવામાં ચોગ્ય ઉદાહરણ સાથે રન ટાઇમ પોલીમોર્ફિઝમ સમજાવો.

જવાબ

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

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

Run-time Polymorphism: Method call runtime पर actual object type આधારે resolve થાય.
આકૃતિ:



કોડ વ્યૂહ:

```

1 class Animal {
2     void makeSound() {
3         System.out.println("Default animal sound");
4     }
5 }
6
7 class Dog extends Animal {
8     @Override
9     void makeSound() {
10        System.out.println("Dog sound");
11    }
12 }
13
14 class Cat extends Animal {
15     @Override
16     void makeSound() {
17        System.out.println("Cat sound");
18    }
19 }
20
21 class Main {
22     public static void main(String[] args) {
23         Animal animal1 = new Dog(); // Upcasting
24         Animal animal2 = new Cat(); // Upcasting
25
26         animal1.makeSound(); // Prints "Dog sound"
27         animal2.makeSound(); // Prints "Cat sound"
28
29         // Animals array
30         Animal[] animals = {new Dog(), new Cat(), new Dog()};
31         for(Animal a : animals) {
32             a.makeSound(); // Dynamic method dispatch
33         }
34     }
35 }
  
```

આઉટપુટ:

```

1
2
3
4
5
6
  
```

વિશેષતાઓ:

- **Dynamic Method Dispatch:** JVM runtime पર 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:

```

1 import java.io.*;
2
3 class FileCopyDemo {
4     public static void main(String[] args) {
5         try {
6             // source file           input stream
7             FileInputStream input = new FileInputStream("source.txt");
8
8             // destination file       output stream
9             FileOutputStream output = new FileOutputStream("destination.txt");
10
11             int byteData;
12
13             //
14             while((byteData = input.read()) != -1) {
15                 output.write(byteData);
16             }
17
18             // streams
19             input.close();
20             output.close();
21
22             System.out.println("          !");
23
24         } catch(IOException e) {
25             System.out.println(" : " + e.getMessage());
26         }
27     }
28 }
```

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
Byte Stream	FileInputStream	ફાઇલમાંથી bytes વાંચો
Byte Stream	FileOutputStream	ફાઇલમાં bytes લખો
Character Stream	FileReader	ફાઇલમાંથી characters વાંચો
Character Stream	FileWriter	ફાઇલમાં characters લખો
Buffered	BufferedReader	કાર્યક્ષમ character reading
Buffered	BufferedWriter	કાર્યક્ષમ character writing

આફ્ટિ: I/O Class Hierarchy

```

1 +-----+
2 |   InputStream   |
3 +-----+
4   |
5     +-- FileInputStream
6     +-- BufferedInputStream
7
8 +-----+
9 |   OutputStream  |
10 +-----+
11   |
12     +-- FileOutputStream
13     +-- BufferedOutputStream
14
15 +-----+
16 |   Reader        |
17 +-----+
18   |
19     +-- FileReader
20     +-- BufferedReader
21
22 +-----+
23 |   Writer         |
24 +-----+
25   |
26     +-- FileWriter
27     +-- BufferedWriter

```

કોડ ઉદાહરણ:

```

1 // Byte streams
2 FileInputStream fis = new FileInputStream("file.txt");
3 FileOutputStream fos = new FileOutputStream("output.txt");
4
5 // Character streams
6 FileReader fr = new FileReader("file.txt");
7 FileWriter fw = new FileWriter("output.txt");
8
9 // Buffered streams
10 BufferedReader br = new BufferedReader(new FileReader("file.txt"));
11 BufferedWriter bw = new BufferedWriter(new FileWriter("output.txt"));

```

મેમરી ટ્રીક

``Byte Character Buffered Streams''

પ્રશ્ન 5(ક) OR) [7 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 વ્યક્ત:

```
1  class JavaThread extends Thread {
2      public void run() {
3          try {
4              while(true) {
5                  System.out.println("Java Programming");
6                  Thread.sleep(3000); // 3
7              }
8          } catch(InterruptedException e) {
9              System.out.println("JavaThread");
10         }
11     }
12 }
13
14 class SemesterThread extends Thread {
15     public void run() {
16         try {
17             while(true) {
18                 System.out.println("Semester - 4th IT");
19                 Thread.sleep(6000); // 6
20             }
21         } catch(InterruptedException e) {
22             System.out.println("SemesterThread");
23         }
24     }
25 }
26
27 class Main {
28     public static void main(String[] args) {
29         // thread objects
30         JavaThread javaThread = new JavaThread();
31         SemesterThread semesterThread = new SemesterThread();
32
33         // threads
34         javaThread.start();
35         semesterThread.start();
36
37         // threads 20
38         try {
39             Thread.sleep(20000);
40             javaThread.interrupt();
41             semesterThread.interrupt();
42         } catch(InterruptedException e) {
43             System.out.println("Main thread");
44         }
45     }
46 }
```

નમૂના આઉટપુટ:

```
1 Java Programming
2 Semester - 4th IT
3 Java Programming
4 Java Programming
5 Semester - 4th IT
6 Java Programming
7 Java Programming
8 Semester - 4th IT
9 ...
```

વિશેષતાઓ:

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

Execution Pattern:

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

મેમરી ટ્રીક

“બે Threads અલગ Timing”