ઑબ્જેક્ટ ઓરિએન્ટેડ પ્રોગ્રામિંગ વિથ જાવા (4341602) - સમર 2023 સોલ્યુશન

Milav Dabgar

July 15, 2023

પ્રશ્ન 1(a) [3 ગુણ]

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

જવાબ

કોષ્ટક 1. POP vs OOP

પાસું	POP	OOP
ધ્યાન	ફંક્શન્સ/પ્રોસિજર્સ	ઑબ્જેક્ટ્સ અને ક્લાસિઝ
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 ઑબ્જેક્ટ્સની આસપાસ ગોઠવાયેલું જેમાં ડેટા અને મેથડ્સ બંને હોય છે

મેમરી ટ્રીક

``POP Functions, OOP Objects''

પ્રશ્ન 1(b) [4 ગુણ]

યોગ્ય 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.

કોષ્ટક 2. Super કીવર્ડના ઉપયોગો

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

Listing 1. Super કીવર્ડ ઉદાહરણ

```
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 ને કૉલ કરે છે
        System.out.println("Dog eats bones");
    }
    void display() {
        System.out.println(super.name); // "Animal" પ્રિન્ટ કરે છે
    }
}
```

મેમરી ટ્રીક

``Super Calls Parent''

પ્રશ્ન 1(c) [7 ગુણ]

વ્યાખ્યાયિત કરો: મેથડ ઓવરરાઇડિંગ. મેથડ ઓવરરાઇડિંગ માટેના નિયમોની યાદી બનાવો. એક જાવા પ્રોગ્રામ લખો જે મેથડ ઓવરરાઇડિંગને implement કરે છે.

જવાબ

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

કોષ્ટક 3. મેથડ ઓવરરાઇડિંગના 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

Listing 2. મેથડ ઓવરરાઇડિંગ ઉદાહરણ

```
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(); // આઉટપુટ: Drawing a circle
    }
}
```

મેમરી ટ્રીક

``Override Same Signature''

પ્રશ્ન 1(c OR) [7 ગુણ]

વર્ણવો: ઇન્ટરફેસ. મલ્ટિપલ inheritance ને દર્શાવવા માટે interface નો ઉપયોગ કરીને જાવા પ્રોગ્રામ લખો.

જવાબ

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

કોષ્ટક 4. Interface ની વિશેષતાઓ

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

Listing 3. Interface ઉદાહરણ

```
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

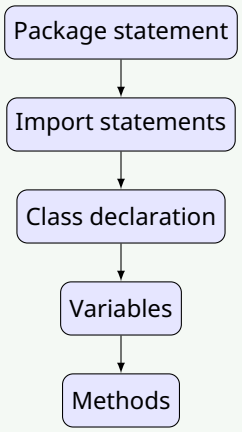
``Interface Multiple Implementation''

પ્રશ્ન 2(a) [3 ગુણ]

ઉદાહરણ સાથે Java પ્રોગ્રામ સ્ટ્રક્ચર સમજાવો.

જવાબ

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



આકૃતિ 1. Java પ્રોગ્રામ સ્ટ્રક્ચર

Listing 4. પ્રોગ્રામ સ્ટ્રક્ચર ઉદાહરણ

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

public class HelloWorld {   // Class
   static int count = 0;   // Variable

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

મેમરી ટ્રીક

``Package Import Class Main''

પ્રશ્ન 2(b) [4 ગુણ]

યોગ્ય ઉદાહરણ સાથે static કીવર્ડ સમજાવો.

જવાબ

Static કીવર્ડ class નું છે instance નું નહીં. memory એક વાર allocate થાય છે.

કોષ્ટક 5. Static ના ઉપયોગો

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

Listing 5. Static કીવર્ડ ઉદાહરણ

```
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(); // કોઈ object ની જરૂર નથી
   }
}
```

મેમરી ટ્રીક

``Static Shared by Class''

પ્રશ્ન 2(c) [7 ગુણ]

વ્યાખ્યાયિત કરો: કન્સ્ટ્રક્ટર. તેના પ્રકારોની યાદી બનાવો. પેરામીટરાઇઝ્ડ અને કોપી કન્સ્ટ્રક્ટરને યોગ્ય ઉદાહરણ સાથે સમજાવો.

જવાબ

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

કોષ્ટક 6. કન્સ્ટ્રક્ટરના પ્રકારો

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

Listing 6. કન્સ્ટ્રક્ટર ઉદાહરણ

```
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 Initializes Objects''

પ્રશ્ન 2(a OR) [3 ગુણ]

જાવામાં પ્રિમિટિવ ડેટા પ્રકારો અને યુઝર ડિફાઇન્ડ ડેટા પ્રકારો સમજાવો.

જવાબ

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

કોષ્ટક 7. 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 માં store થાય, ઝડપી access
- **User Defined**: heap માં store થાય, જટિલ operations

મેમરી ટ્રીક

``Primitive Built-in, User Custom''

પ્રશ્ન 2(b OR) [4 ગુણ]

યોગ્ય ઉદાહરણ સાથે this કીવર્ડ સમજાવો.

જવાબ

This કીવર્ડ વર્તમાન object instance ને refer કરે છે, instance અને local variables વચ્ચે ભેદ પાડવા માટે વાપરાય છે.

કોષ્ટક 8. This કીવર્ડના ઉપયોગો

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

Listing 7. This કીવર્ડ ઉદાહરણ

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

   Student(String name, int age) {
      this.name = name;    // this instance અને parameter વચ્ચે
      this.age = age;      // ભેદ પાડે છે
   }

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

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

મેમરી ટ્રીક

``This Current Object''

પ્રશ્ન 2(c OR) [7 ગુણ]

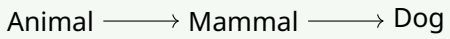
ઇનહેરિટન્સ વ્યાખ્યાયિત કરો. તેના પ્રકારોની યાદી બનાવો. multilevel અને hierarchical ઇનહેરિટન્સને યોગ્ય ઉદાહરણ સાથે સમજાવો.

જવાબ

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

કોષ્ટક 9. Inheritance ના પ્રકારો

પ્રકાર	વર્ણન	સ્ટ્રક્ચર
Single	એક parent, એક child	A → B
Multilevel	inheritance ની chain	A → B → C
Hierarchical	એક parent, અનેક children	A → B, A → C
Multiple	અનેક parents (interfaces)	B,C → A



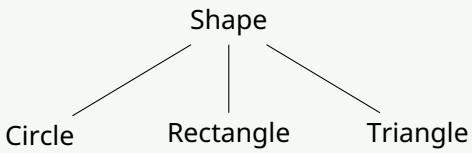
આકૃતિ 2. Multilevel Inheritance

Listing 8. Multilevel Inheritance ઉદાહરણ

```
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"); }
}
```



આકૃતિ 3. Hierarchical Inheritance

Listing 9. Hierarchical Inheritance ઉદાહરણ

```
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"); }
}
```

મેમરી ટ્રીક

``Inheritance Properties Shares''

પ્રશ્ન 3(a) [3 ગુણ]

જાવામાં ટાઇપ કન્વર્ઝન અને કાસ્ટિંગ સમજાવો.

જવાબ

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

કોષ્ટક 10. Type Conversion

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

Listing 10. Conversion vs Casting

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

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

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

મેમરી ટ્રીક

``Implicit Auto, Explicit Manual''

પ્રશ્ન 3(b) [4 ગુણ]

Java માં ઉપયોગમાં લેવાતા વિવિધ visibility controls સમજાવો.

જવાબ

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

કોષ્ટક 11. Access Modifiers

Modifier	Class	Pkg	Sub	Diff Pkg
private	✓	×	×	×
default	✓	✓	×	×
protected	✓	✓	✓	×
public	✓	✓	✓	✓

Listing 11. Access Modifiers ઉદાહરણ

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

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

મેમરી ટ્રીક

``Private Package Protected Public''

પ્રશ્ન 3(c) [7 ગુણ]

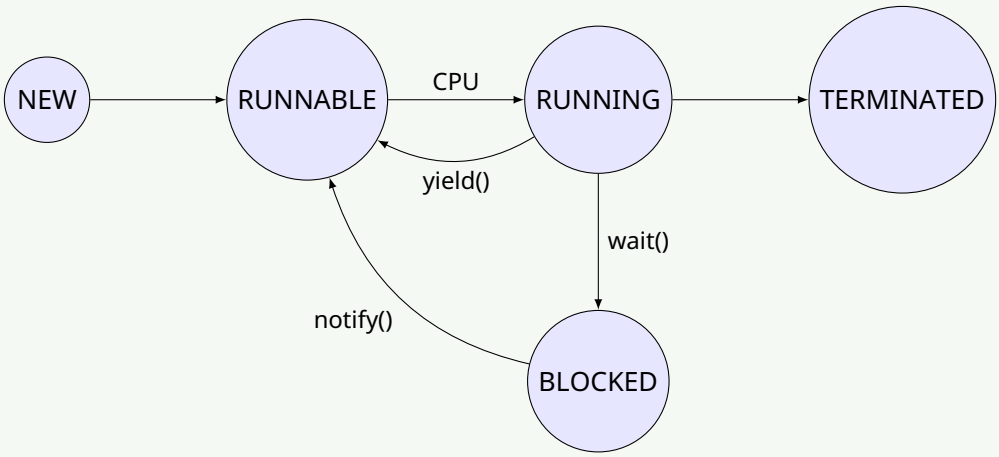
વ્યાખ્યાયિત કરો: થ્રેડ. થ્રેડ બનાવવા માટે ઉપયોગમાં લેવાતી વિવિધ પદ્ધતિઓની સૂચિ બનાવો. થ્રેડની લાઇફ સાઇકલ વિગતવાર સમજાવો.

જવાબ

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

કોષ્ટક 12. Thread બનાવવા પદ્ધતિઓ

પદ્ધતિ	વર્ણન	ઉદાહરણ
Extending	Thread class inherit	class MyThread extends Thread
Boolean	Runnable implement	class MyTask implements Runnable



આકૃતિ 4. Thread Life Cycle

કોષ્ટક 13. 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

Listing 12. Thread Creation Example

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

// પદ્ધતિ 2: Runnable implement કરવું
```

```java
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(a OR) [3 ગુણ]

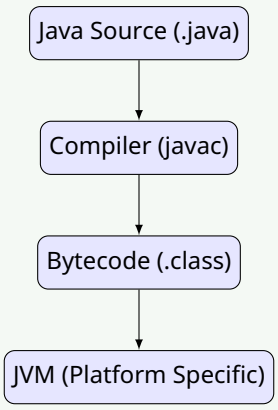
java માં JVM નો હેતુ સમજાવો.

જવાબ

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

કોષ્ટક 14. 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 5. JVM Workflow

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

મેમરી ટ્રીક

``JVM Java Virtual Machine''

પ્રશ્ન 3(b OR) [4 ગુણ]

વ્યાખ્યાયિત કરો: પેકેજ. યોગ્ય ઉદાહરણ સાથે પેકેજ બનાવવા માટેના પગલાંઓ લખો.

જવાબ

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

કોષ્ટક 15. Package ના ફાયદા

ફાયદો	વર્ણન
Namespace	નામની ટકરાર ટાળે
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

Listing 13. Package Example

```
// 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));
    }
}
```

મેમરી ટ્રીક

``Package Class Group''

પ્રશ્ન 3(c OR) [7 ગુણ]

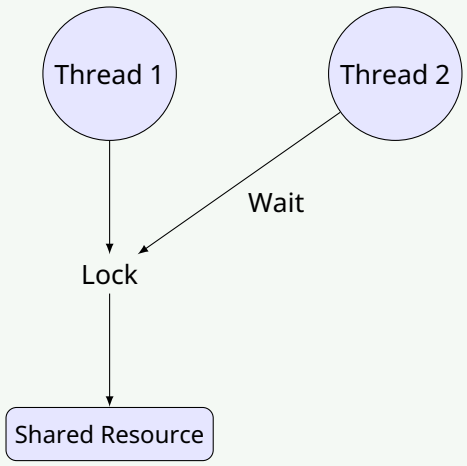
થ્રેડમાં Synchronization ને યોગ્ય ઉદાહરણ સાથે સમજાવો.

જવાબ

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

કોષ્ટક 16. Synchronization ના પ્રકારો

પ્રકાર	વર્ણન	ઉપયોગ
Synchronized method	આખી method lock થાય	synchronized void
Synchronized block	specific block lock થાય	synchronized(obj)
Static synchronization	Class level locking	static synchronized



આકૃતિ 6. Synchronization Mechanism

Listing 14. Synchronization Example

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

   public synchronized void increment() {
      count++;
   }

   public void decrement() {
      synchronized(this) {
         count--;
      }
   }
}
```

મેમરી ટ્રીક

``Synchronization Prevents Race Conditions''

પ્રશ્ન 4(a) [3 ગુણ]

સ્ટ્રિંગ ક્લાસ અને સ્ટ્રિંગબફર ક્લાસ વચ્ચે તફાવત કરો.

જવાબ

કોષ્ટક 17. String vs StringBuffer

પાસું	String	StringBuffer
Mutability	Immutable (બદલાતું નથી)	Mutable (બદલાય છે)
Performance	concatenation માટે ધીમું	concatenation માટે ઝડપું
Memory	દર વખતે નવું object બનાવે	હાલનું object modify કરે
Thread Safety	Thread safe	Thread safe

Listing 15. String vs StringBuffer

```
// String - Immutable
String s1 = "Hello";
s1 = s1 + " World";  // નવું String object બનાવે છે

// StringBuffer - Mutable
StringBuffer sb = new StringBuffer("Hello");
sb.append(" World");  // હાલનું object modify કરે છે
```

મેમરી ટ્રીક

``String Immutable, StringBuffer Mutable''

પ્રશ્ન 4(b) [4 ગુણ]

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

જવાબ

Listing 16. Array Sum and Average

```
class ArraySum {
    public static void main(String[] args) {
        int[] numbers = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};
        int sum = 0;

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

        // સરેરાશ Calculate
        double average = (double) sum / numbers.length;

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

આઉટપુટ:

Sum: 550
Average: 55.0

મેમરી ટ્રીક

``Loop Sum Divide Average''

પ્રશ્ન 4(c) [7 ગુણ]

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

જવાબ

I) Abstract Class: class જેનું instantiation થઈ શકતું નથી, abstract methods હોય છે.

કોષ્ટક 18. Abstract Class ની વિશેષતાઓ

વિશેષતા	વર્ણન
Cannot instantiate	object બનાવી શકાતું નથી
Abstract methods	implementation વિનાની methods
Inheritance	Subclasses implement methods

Listing 17. Abstract Class Example

```
abstract class Shape {
   abstract void draw();
}

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

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

કોષ્ટક 19. Final Class ની વિશેષતાઓ

વિશેષતા	વર્ણન
No inheritance	extend કરી શકાતું નથી
Security	modification અટકાવે છે
ઉદાહરણો	String, Integer, System

Listing 18. Final Class Example

```
final class FinalClass {
   void display() {
      System.out.println("This is final class");
   }
}
// class Sub extends FinalClass { } // ભૂલ
```

મેમરી ટ્રીક

``Abstract Incomplete, Final Complete''

પ્રશ્ન 4(a OR) [3 ગુણ]

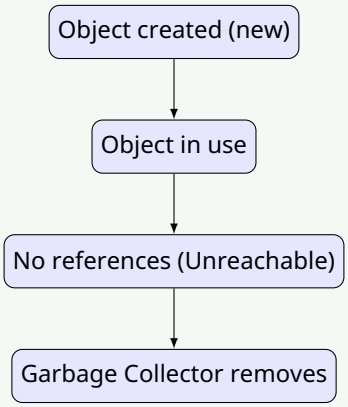
જાવામાં Garbage Collection સમજાવો.

જવાબ

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

કોષ્ટક 20. GC ના ફાયદા

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



આકૃતિ 7. Garbage Collection Process

મેમરી ટ્રીક

``GC Automatic Memory Cleanup''

પ્રશ્ન 4(b OR) [4 ગુણ]

'Divide by Zero' એરર માટે યુઝર ડિફાઇન્ડ એક્સેપ્શન હેન્ડલ કરવા માટે જાવા પ્રોગ્રામ લખો.

જવાબ

Listing 19. User Defined Exception

```
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("શૂન્ય થી ભાગ ન આપી શકાય!");
      }
```

```
        return (double) a / b;
    }
}

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

મેમરી ટ્રીક

``Custom Exception Handle Error''

પ્રશ્ન 4(c OR) [7 ગુણ]

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

જવાબ

Listing 20. Multiple Try-Catch Blocks

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

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

        // Multiple catch
        try {
            int result = 10 / 0;  // Arithmetic
        }
        catch(ArithmeticException e) {
            System.out.println("Math ભૂલ: " + e.getMessage());
        }
        catch(Exception e) {
            System.out.println("અન્ય ભૂલ: " + e.getMessage());
        }
        finally {
```

```
            System.out.println("પ્રોગ્રામ પૂર્ણ થયું");
        }
    }
}
```

મેમરી ટ્રીક

``Multiple Try Multiple Catch''

પ્રશ્ન 5(a) [3 ગુણ]

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

જવાબ

Listing 21. File Write Example

```
import java.io.*;

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

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

            // ફાઇલમાં ડેટા લખો
            writer.write("નમસ્તે દુનિયા!\n");
            writer.write("આ Java ફાઇલ લેખન ડેમો છે.\n");

            writer.close();
            System.out.println("ફાઇલ સફળતાપૂર્વક બનાવવામાં આવી!");

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

મેમરી ટ્રીક

``File Writer Write Close''

પ્રશ્ન 5(b) [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 હંમેશા execute થાય છે.

કોષ્ટક 21. Throw vs Finally

કીવર્ડ	હેતુ	ઉપયોગ
throw	Explicitly exception throw કરે	throw new Ex()
finally	હંમેશા cleanup code execute કરે	finally { }

Listing 22. Throw and Finally

```
class ThrowFinallyDemo {
   public static void checkAge(int age) throws Exception {
      if(age < 18) throw new Exception("અમાન્ય ઉંમર");
   }

   public static void main(String[] args) {
      try {
         checkAge(15);
      } catch(Exception e) {
         System.out.println("ભૂલ: " + e.getMessage());
      } finally {
         System.out.println("હંમેશા execute થાય છે");
      }
   }
}
```

મેમરી ટ્રીક

``Throw Exception, Finally Always''

પ્રશ્ન 5(c) [7 ગુણ]

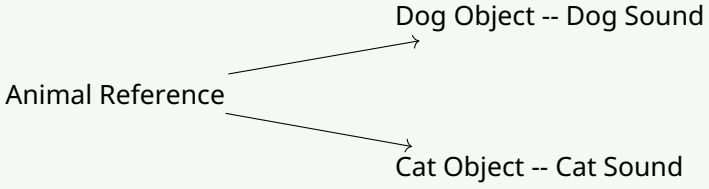
વર્ણવો: પોલીમોર્ફિઝમ. જાવામાં યોગ્ય ઉદાહરણ સાથે રન ટાઇમ પોલીમોર્ફિઝમ સમજાવો.

જવાબ

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

કોષ્ટક 22. Polymorphism ના પ્રકારો

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



આકૃતિ 8. Runtime Polymorphism

Listing 23. Runtime Polymorphism

```
class Animal {
   void makeSound() { System.out.println("પ્રાણી અવાજ કરે છે"); }
}

class Dog extends Animal {
   void makeSound() { System.out.println("કૂતરો ભસે છે"); }
}

class Cat extends Animal {
   void makeSound() { System.out.println("બિલાડી મ્યાં કરે છે"); }
}

class Main {
   public static void main(String[] args) {
      Animal a;

      a = new Dog();
      a.makeSound(); // કૂતરો ભસે છે

      a = new Cat();
      a.makeSound(); // બિલાડી મ્યાં કરે છે
   }
}
```

મેમરી ટ્રીક

``Polymorphism Many Forms Runtime''

પ્રશ્ન 5(a OR) [3 ગુણ]

જાવામાં એક પ્રોગ્રામ લખો જે બાઇટ બાય બાઇટ ફાઇલના કન્ટેન્ટ વાંચે અને તેને બીજી ફાઇલમાં કોપી કરે.

જવાબ

Listing 24. Byte Stream File Copy

```
import java.io.*;

class FileCopyDemo {
   public static void main(String[] args) {
      try {
         FileInputStream input = new FileInputStream("in.txt");
         FileOutputStream output = new FileOutputStream("out.txt");

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

         input.close();
         output.close();
      } catch(IOException e) {
         System.out.println("ભૂલ: " + e.getMessage());
      }
   }
}
```

મેમરી ટ્રીક

``Read Byte Write Byte''

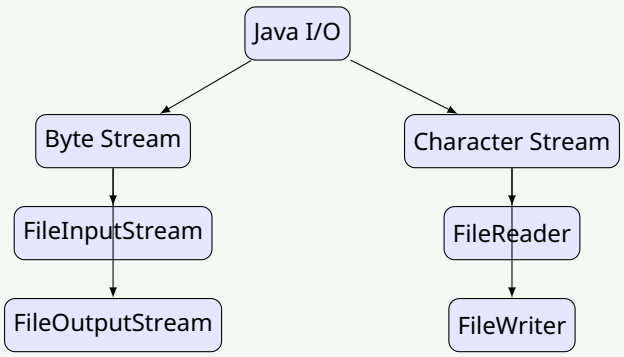
પ્રશ્ન 5(b OR) [4 ગુણ]

Java સાથે ઉપલબ્ધ વિવિધ I/O વર્ગો સમજાવો.

જવાબ

કોષ્ટક 23. Java I/O Classes

Class Type	Class Name	હેતુ
Byte Stream	FileInputStream	ફાઇલથી bytes વાંચે
Byte Stream	FileOutputStream	ફાઇલમાં bytes લખે
Char Stream	FileReader	ફાઇલથી chars વાંચે
Char Stream	FileWriter	ફાઇલમાં chars લખે
Buffered	BufferedReader	કાર્યક્ષમ વાંચન
Buffered	BufferedWriter	કાર્યક્ષમ લખવુ



આકૃતિ 9. I/O Class Hierarchy

મેમરી ટ્રીક

``Byte Character Buffered Streams''

પ્રશ્ન 5(c OR) [7 ગુણ]

જાવા પ્રોગ્રામ લખો જે બે થ્રેડોને એક્ઝિક્યુટ કરે છે. એક થ્રેડ દર 3 સેકન્ડે "Java Programming" દર્શાવે છે, અને બીજો દર 6 સેકન્ડે "Semester - 4th IT" દર્શાવે છે.

જવાબ

Listing 25. Thread Timing Example

```
class JavaThread extends Thread {
   public void run() {
```

```
        try {
            while(true) {
                System.out.println("Java Programming");
                Thread.sleep(3000);
            }
        } catch(InterruptedException e) {}
    }
}

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

class Main {
    public static void main(String[] args) {
        new JavaThread().start();
        new SemThread().start();
    }
}
```

મેમરી ટ્રીક

``Two Threads Different Timing''