

# જાવા પ્રોગ્રામિંગ (4343203) – સમર 2024 સોલ્યુશન

Milav Dabgar

June 15, 2024

## પ્રશ્ન 1(અ) [3 ગુણ]

જાવામાં Garbage collection સમજાવો.

### જવાબ

જાવામાં Garbage collection વણવપરાયેલા ઓફ્જેક્ટ્સને દૂર કરીને આપમેળે મેમરી ખાલી કરે છે.

#### કોષ્ટક 1. Garbage Collection પ્રક્રિયા

તબક્કો	વર્ણન
Mark	JVM મેમરીમાં બધા live ઓફ્જેક્ટ્સને ઓળખે છે
Sweep	વણવપરાયેલા ઓફ્જેક્ટ્સ દૂર કરવામાં આવે છે
Compact	બાકીના ઓફ્જેક્ટ્સને જગ્યા ખાલી કરવા માટે પુનર્ગઠિત કરવામાં આવે છે

- આપમેળે: મેન્યુઅલ મેમરી મેનેજમેન્ટની જરૂર નથી
- બેકગ્રાઉન્ડ: અલગ ઓછી પ્રાથમિકતાવાળા શ્રેઢમાં ચાલે છે

### મેમરી ટ્રીક

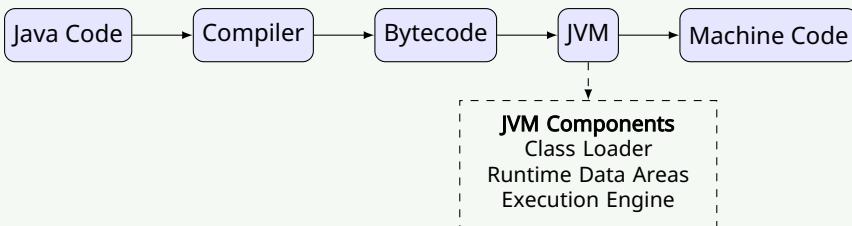
"MSC: Mark-Sweep-Compact આપમેળે મેમરી ખાલી કરે છે"

## પ્રશ્ન 1(બ) [4 ગુણ]

JVM ને વિગતવાર સમજાવો.

### જવાબ

JVM (Java Virtual Machine) એક વર્ચ્યુઅલ મશીન છે જે bytecode ને મશીન કોડમાં રૂપાંતરિત કરીને જાવાની પ્લેટફોર્મ સ્વતંત્રતા આપે છે.



#### આકૃતિ 1. JVM આર્કિટેક્ચર

- પ્લેટફોર્મ સ્વતંત્રતા: એકવાર લખો, બધો ચલાવો
- સુરક્ષા: Bytecode વેરિફિકેશન ખતરનાક કામગીરીને રોકે છે

- ઓફિટિમાઇઝેશન: Just-in-time કમ્પાઇલેશન કામગીરી સુધારે છે

### મેમરી ટ્રીક

“CLASS: Class Loader સુરક્ષિત સિસ્ટમ સંચાલન કરે છે”

## પ્રશ્ન 1(ક) [7 ગુણ]

Fibonacci series પ્રિન્ટ કરવા માટેનો જાવા પ્રોગ્રામ લખો.

### જવાબ

Fibonacci series એવી શ્રેણી બનાવે છે જેમાં દરેક સંખ્યા તેના અગાઉની બે સંખ્યાઓનો સરવાળો હોય.

**Listing 1. Fibonacci Series Program**

```

1 import java.util.Scanner;
2
3 public class FibonacciSeries {
4     public static void main(String[] args) {
5         Scanner input = new Scanner(System.in);
6
7         System.out.print("Enter number of terms: ");
8         int n = input.nextInt();
9
10        int first = 0, second = 1;
11
12        System.out.print("Fibonacci Series: ");
13
14        for (int i = 1; i <= n; i++) {
15            System.out.print(first + " ");
16
17            int next = first + second;
18            first = second;
19            second = next;
20        }
21
22        input.close();
23    }
24 }
```

- પ્રારંભિક: 0 અને 1 થી શરૂઆત કરો
- લૂપ: શ્રેણી બનાવવા માટે N વખત પુનરાવર્તન કરો
- ગણતરી: દરેક સંખ્યા પાછળી બે સંખ્યાનો સરવાળો છે

### મેમરી ટ્રીક

“FSN: પ્રથમ + બીજી = આગળની સંખ્યા શ્રેણીમાં”

## પ્રશ્ન 1(ક) OR [7 ગુણ]

કમાન્ડ લાઇન arguments નો ઉપયોગ કરીને કોઈપણ દસ સંખ્યાઓ માંથી ન્યૂનતમ શોધવા માટે જાવા પ્રોગ્રામ લખો.

## જવાબ

કમાન્ડ લાઇન આર્થ્યુમેન્ટ્સ જાવા પ્રોગ્રામ ચલાવતી વખતે સીધા ઇનપુટ આપવાની સુવિધા આપે છે.

**Listing 2. Find Minimum using Command Line Args**

```

1 public class FindMinimum {
2     public static void main(String[] args) {
3         if (args.length < 10) {
4             System.out.println("Please provide 10 numbers");
5             return;
6         }
7
8         int min = Integer.parseInt(args[0]);
9
10        for (int i = 1; i < 10; i++) {
11            int current = Integer.parseInt(args[i]);
12            if (current < min) {
13                min = current;
14            }
15        }
16
17        System.out.println("Minimum number is: " + min);
18    }
19 }
```

- આર્થ્યુમેન્ટ્સ પારસિંગ: સિંગા આર્થ્યુમેન્ટ્સને ઇન્ટીજરમાં રૂપાંતરિત કરો
- પ્રારંભિક: પ્રથમ સંપ્રાને ન્યૂનતમ તરીકે સેટ કરો
- તુલના: દરેક સંપ્રાને વર્તમાન ન્યૂનતમ સાથે ચકાસો

## મેમરી ટ્રીક

“ICU: શરૂઆત, ચકાસણી, અપડેટ ન્યૂનતમ”

## પ્રશ્ન 2(અ) [3 ગુણ]

Java OOP ના મૂળભૂત ઘ્યાલોની યાદી બનાવો. કોઈપણ એક વિગતવાર સમજાવો.

## જવાબ

જાવા ઓપ્ઝેક્ટ-ઓરિએન્ટેડ પ્રોગ્રામિંગ વાસ્તવિક દુનિયાની વસ્તુઓને મોડેલિંગ કરવા માટે મૂળભૂત સિદ્ધાંતો પર આધારિત છે.

**કોષ્ટક 2. જાવામાં OOP ઘ્યાલો**

ઘ્યાલ	વર્ણન
Encapsulation	ડેટા અને મેથડને એક એકમ તરીકે જોડવું
Inheritance	હાલના class માંથી નવા class બનાવવા
Polymorphism	એક ઇન્ટરફેસ, વિવિધ અમલીકરણો
Abstraction	અમલીકરણની વિગતો છુપાવવી, કાર્યક્ષમતા બતાવવી

- Encapsulation: એક્સેસ કંટ્રોલ દ્વારા ડેટાનું રક્ષણ કરે છે
- ડેટા છુપાવવો: ખાનગી વેરિયેબલ્સ મેથડ્સ દ્વારા એક્સેસ થાય છે

## મેમરી ટ્રીક

“PEAI: પ્રોગ્રામિંગ Encapsulates Abstracts Inherits”

## પ્રશ્ન 2(બ) [4 ગુણ]

final કી-વર્ડ ઉદાહરણ સાથે સમજાવો.

### જવાબ

જાવામાં final કી-વર્ડ ફેરફાર, વારસો અને ઓવરરાઇઝિંગને મર્યાદિત કરવા માટે વપરાય છે.

કોષ્ટક 3. final કી-વર્ડના ઉપયોગો

ઉપયોગ	અસર	ઉદાહરણ
final variable	બદલી શકતું નથી	final int MAX = 100;
final method	ઓવરરાઇઝ કરી શકતી નથી	final void display() {}
final class	વારસામાં લઈ શકતો નથી	final class Math {}

Listing 3. Final Keyword Demo

```

1 public class FinalDemo {
2     final int MAX_VALUE = 100; // નિયતાંક
3
4     final void display() {
5         System.out.println("This method cannot be overridden");
6     }
7 }
8
9 final class MathOperations {
10    // આ class નો વારસો મળી શકતો નથી
11 }
```

### મેમરી ટ્રીક

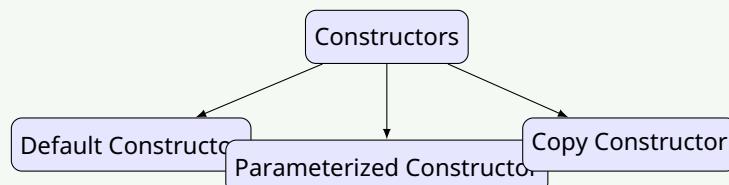
“VCM: Variables Constants Methods બદલી શકતા નથી”

## પ્રશ્ન 2(ક) [7 ગુણ]

કન્સ્ટ્રક્ટર શું છે? Parameterized કન્સ્ટ્રક્ટર ને ઉદાહરણ સાથે સમજાવો.

### જવાબ

કન્સ્ટ્રક્ટર એ ઓફ્જેક્ટ બનાવતી વખતે તેને શરૂઆતી મૂલ્યો આપવા માટેની વિશેષ મેથડ છે.



આકૃતિ 2. કન્સ્ટ્રક્ટરના પ્રકારો

Listing 4. Parameterized Constructor

```

1 public class Student {
2     String name;
3     int age;
```

```

4 // Parameterized constructor
5 Student(String n, int a) {
6     name = n;
7     age = a;
8 }
9
10 void display() {
11     System.out.println("Name: " + name + ", Age: " + age);
12 }
13
14 public static void main(String[] args) {
15     // Object creation using parameterized constructor
16     Student s1 = new Student("John", 20);
17     s1.display();
18 }
19
20 }
```

- પેરામીટર્સ: ઓફજેક્ટ બનાવતી વખતે કિમતો સ્વીકારે છે
- પ્રારંભિક: પાસ કરેલા મૂલ્યો સાથે ઓફજેક્ટ પ્રોપર્ટી સેટ કરે છે
- ઓવરલોડિંગ: અલગ અલગ પેરામીટર્સ સાથે ઘણા કન્સ્ટક્ટર્સ

### મેમરી ટ્રીક

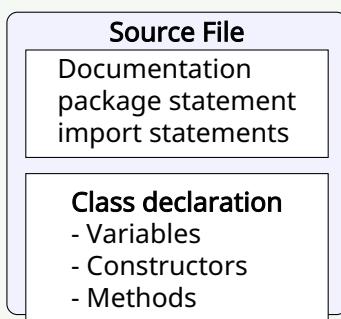
“SPO: વિદ્યાર્થી પેરામીટર્સ ઓફજેક્ટ પ્રોપર્ટી શરૂ કરે છે”

## પ્રશ્ન 2(અ OR) [3 ગુણ]

ઉદાહરણ સાથે જાવા પ્રોગ્રામ સ્ટ્રક્ચર સમજાવો.

### જવાબ

જાવા પ્રોગ્રામ સ્ટ્રક્ચર તાર્કિક રીતે ગોર્ધવાયેલા તત્વોના વિશિષ્ટ ક્રમને અનુસરે છે.



આકૃતિ 3. જાવા પ્રોગ્રામ સ્ટ્રક્ચર

- **Package:** સંબંધિત કલાસને જૂથમાં રાખે છે
- **Import:** બાહ્ય કલાસને સમાવે છે
- **Class:** વેરિયેબલ્સ અને મેથડ્સ ધરાવે છે

### મેમરી ટ્રીક

“PIC: દરેક પ્રોગ્રામમાં Package Imports Class”

## પ્રશ્ન 2(બ) OR) [4 ગુણ]

યોગ્ય ઉદાહરણ સાથે static કી-વર્ડ સમજાવો.

### જવાબ

Static કી-વર્ડ કલાસ-લેવલ વેરિયેબલ્સ અને મેથ્ડ્સ બનાવે છે જે બધા ઓફજેક્ટ્સ વચ્ચે શેર થાય છે.

**કોષ્ટક 4. Static vs Non-Static**

ફીચર	Static	Non-Static
મેમરી	એક કોપી	ઘણી કોપીઓ
એક્સેસ	ઓફજેક્ટ વગર	ઓફજેક્ટ દ્વારા
રેફરન્સ	કલાસ નામ	ઓફજેક્ટ નામ
લોડ થવાનો સમય	કલાસ લોડિંગ	ઓફજેક્ટ બનાવટ

**Listing 5. Static Keyword Demo**

```

1 public class Counter {
2     static int count = 0; // બધા ઓફજેક્ટ્સ માટે શેર
3     int instanceCount = 0; // દરેક ઓફજેક્ટ માટે અલગ
4
5     Counter() {
6         count++;
7         instanceCount++;
8     }
9
10    public static void main(String[] args) {
11        Counter c1 = new Counter();
12        Counter c2 = new Counter();
13
14        System.out.println("Static count: " + Counter.count);
15        System.out.println("c1's instance count: " + c1.instanceCount);
16        System.out.println("c2's instance count: " + c2.instanceCount);
17    }
18 }
```

### મેમરી ટ્રીક

“SCM: Static બધા ઓફજેક્ટ માટે એકવાર મેમરી બનાવે છે”

## પ્રશ્ન 2(ક) OR) [7 ગુણ]

ઇનહેરીટન્સ વ્યાખ્યાચિત કરો. તેના પ્રકારોની ચાદી બનાવો. Multilevel અને Hierarchical ઇનહેરીટન્સ ને યોગ્ય ઉદાહરણ સાથે સમજાવો.

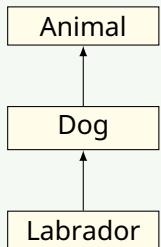
### જવાબ

ઇનહેરીટન્સ એ OOP સિદ્ધાંત છે જેમાં નવો કલાસ હાલના કલાસની સંપત્તિઓ અને વર્તનને મેળવે છે.

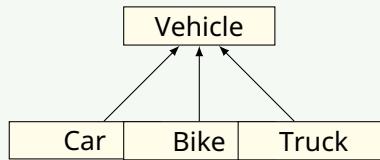
**કોષ્ટક 5. જાવામાં ઇનહેરીટન્સના પ્રકારો**

પ્રકાર	વર્ણન
Single	એક સબકલાસ એક સુપરકલાસથી વિસ્તરે છે
Multilevel	ઇનહેરેટન્સની સંકળ ( $A \rightarrow B \rightarrow C$ )
Hierarchical	ઘણા સબકલાસ એક સુપરકલાસથી વિસ્તરે છે
Multiple	એક કલાસ ઘણા કલાસથી વિસ્તરે છે (ઇન્ટરફેસ દ્વારા)

Multilevel



Hierarchical



આકૃતિ 4. Multilevel vs Hierarchical ઇનહેરેટન્સ

Listing 6. Inheritance Example

```

1 // Multilevel inheritance
2 class Animal {
3     void eat() { System.out.println("eating"); }
4 }
5
6 class Dog extends Animal {
7     void bark() { System.out.println("barking"); }
8 }
9
10 class Labrador extends Dog {
11     void color() { System.out.println("golden"); }
12 }
13
14 // Hierarchical inheritance
15 class Vehicle {
16     void move() { System.out.println("moving"); }
17 }
18
19 class Car extends Vehicle {
20     void wheels() { System.out.println("4 wheels"); }
21 }
22
23 class Bike extends Vehicle {
24     void wheels() { System.out.println("2 wheels"); }
25 }
  
```

### મેમરી ટ્રીક

“SMHM: Single Multilevel Hierarchical ઇનહેરેટન્સના પ્રકારો છે”

### પ્રશ્ન 3(અ) [3 ગુણ]

this કીવર્ડને યોગ્ય ઉદાહરણ સાથે સમજાવો.

## જવાબ

જાવામાં this કી-વર્ડ વર્તમાન ઓફ્જેક્ટનો સંદર્ભ આપે છે, જે ઇન્સ્ટન્સ વેરિયેબલ્સ અને પેરામીટર્સ વર્ચે તફાવત પાડે છે.

કોષ્ટક 6. 'this' કી-વર્ડના ઉપયોગો

ઉપયોગ	હેતુ
this.variable	ઇન્સ્ટન્સ વેરિયેબલ્સ એક્સેસ કરવા
this()	વર્તમાન કલાસના કન્સ્ટ્રક્ટરને કોલ કરવા
return this	વર્તમાન ઓફ્જેક્ટ પાછો આપવા

Listing 7. This Keyword

```

1 public class Student {
2     String name;
3
4     Student(String name) {
5         this.name = name; // ઇન્સ્ટન્સ વેરિયેબલનો સંદર્ભ આપે છે
6     }
7
8     void display() {
9         System.out.println("Name: " + this.name);
10    }
11 }
```

## મેમરી ટ્રીક

“VAR: Variables Access Resolution this નો ઉપયોગ કરીને”

## પ્રશ્ન 3(બ) [4 ગુણ]

જાવામાં વિવિધ એક્સેસ કંટ્રોલ સમજાવો.

## જવાબ

જાવામાં એક્સેસ કંટ્રોલ કલાસેસ, મેથ્ડ્સ અને વેરિયેબલ્સની દૃશ્યતા અને પહોંચને નિયંત્રિત કરે છે.

કોષ્ટક 7. જાવા એક્સેસ મોડિફાયર્સ

મોડિફાયર	કલાસ	પેકેજ	સબકલાસ	જગત
private	✓	✗	✗	✗
default	✓	✓	✗	✗
protected	✓	✓	✓	✗
public	✓	✓	✓	✓

- Private:** માત્ર તે જ કલાસની અંદર
- Default:** તે જ પેકેજની અંદર
- Protected:** પેકેજ અને સબકલાસમાં
- Public:** બધે જ પહોંચ

## મેમરી ટ્રીક

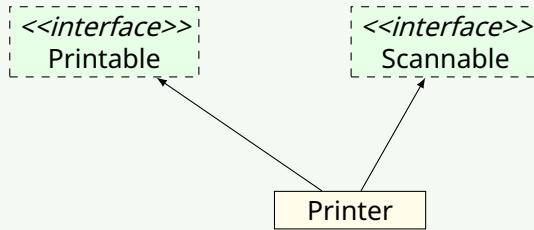
“PDPP: Private Default Protected Public સંકુચિતથી વિશાળ”

## પ્રશ્ન 3(ક) [7 ગુણ]

ઈન્ટરફેસ શું છે? ઈન્ટરફેસ દ્વારા ઉદાહરણ સાથે multiple inheritance સમજાવો.

### જવાબ

ઈન્ટરફેસ એક એવો કરાર છે જે કલાસો શું કરવું જોઈએ તે નિર્દિષ્ટ કરે છે, જેમાં abstract મેથડ્સ, નિયતાંકો અને (Java 8થી) default મેથડ્સ હોય છે.



આકૃતિ 5. ઈન્ટરફેસથી Multiple Inheritance

**Listing 8. Interface Example**

```

1  interface Printable {
2      void print();
3  }
4
5  interface Scannable {
6      void scan();
7  }
8
9 // Multiple inheritance using interfaces
10 class Printer implements Printable, Scannable {
11     public void print() {
12         System.out.println("Printing...");
13     }
14
15     public void scan() {
16         System.out.println("Scanning...");
17     }
18
19     public static void main(String[] args) {
20         Printer p = new Printer();
21         p.print();
22         p.scan();
23     }
24 }
  
```

- કરાર: અમલીકરણ વગર વર્તન વ્યાખ્યાયિત કરે છે
- Implements: કલાસ કરાર પૂર્ણ કરે છે
- Multiple: ધ્રાણ ઈન્ટરફેસ લાગુ કરી શકાય છે

### મેમરી ટ્રીક

“CIM: કરાર Implements Multiple ઈન્ટરફેસ”

## પ્રશ્ન 3(અ) OR) [3 ગુણ]

super કી-વર્ડ ઉદાહરણ સાથે સમજાવો.

**જવાબ**

super કી-વર્ડ પેરન્ટ કલાસનો સંદર્ભ આપે છે, જે પેરન્ટ મેથ્ડુસ, કન્સ્ટ્રક્ટર્સ અને વેરિયેબલ્સ એક્સેસ કરવા વપરાય છે.

કોષ્ટક 8. super કી-વર્ડના ઉપયોગો

ઉપયોગ	હેતુ
super.variable	પેરન્ટ વેરિયેબલ એક્સેસ કરવા
super.method()	પેરન્ટ મેથ્ડ કોલ કરવા
super()	પેરન્ટ કન્સ્ટ્રક્ટર કોલ કરવા

Listing 9. Super Keyword

```

1 class Vehicle {
2     String color = "white";
3
4     void display() {
5         System.out.println("Vehicle class");
6     }
7 }
8
9 class Car extends Vehicle {
10    String color = "black";
11
12    void display() {
13        super.display(); // પેરન્ટ મેથ્ડ કોલ કરે છે
14        System.out.println("Car color: " + color);
15        System.out.println("Vehicle color: " + super.color);
16    }
17 }
```

**મેમરી ટ્રીક**

“VMC: Variables Methods Constructors super દ્વારા એક્સેસ થાય છે”

**પ્રશ્ન 3(બ) OR [4 ગુણ]**

પેકેજ શું છે? પેકેજ બનાવવાના પગલાં લખો અને તેનું ઉદાહરણ આપો.

**જવાબ**

પેકેજ એ જાવામાં સંબંધિત કલાસ અને ઈન્ટરફેસને સંગઠિત કરતું નેમસ્પેસ છે, જે નામકરણ સંઘર્ષને રોકે છે.

કોષ્ટક 9. પેકેજ બનાવવાના પગલાં

પગલું	કિયા
1	ફાઈલની ટોચે પેકેજ નામ જાહેર કરો
2	પેકેજ નામને અનુરૂપ ડિરેક્ટરી સ્ટ્રક્ચર બનાવો
3	જાવા ફાઈલને ડિરેક્ટરીમાં સેવ કરો
4	-d વિકલ્પ સાથે કમ્પાઇલ કરો
5	ઉપયોગ કરવા માટે પેકેજ ઇંપોર્ટ કરો

Listing 10. Package Example

```

1 // પગલું 1: પેકેજ જાહેર કરો (Calculator.java તરીકે સેવ કરો)
2 package mathematics;
3
4 public class Calculator {
5     public int add(int a, int b) {
6         return a + b;
7     }
8 }
9
10 // બીજું ફાઈલમાં (UseCalculator.java)
11 import mathematics.Calculator;
12
13 class UseCalculator {
14     public static void main(String[] args) {
15         Calculator calc = new Calculator();
16         System.out.println(calc.add(10, 20));
17     }
18 }
```

### મેરી ટ્રીક

“DISCO: Declare Import Save Compile Organize”

## પ્રશ્ન 3(ક) OR) [7 ગુણ]

વ્યાખ્યાયિત કરો: થ્રેડ ઓવરરાઇટિંગ. થ્રેડ ઓવરરાઇટિંગ માટેના નિયમોની યાદી બનાવો. થ્રેડ ઓવરરાઇટિંગને ઇમ્પ્લેમેન્ટ કરતો જાવા પ્રોગ્રામ લખો.

### જવાબ

(નોંધ: પ્રશ્નમાં “થ્રેડ ઓવરરાઇટિંગ” પૂછયું છે, પણ જાવામાં “મેથડ ઓવરરાઇટિંગ” હોય છે. જવાબ મેથડ ઓવરરાઇટિંગ સંદર્ભે છે.)  
મેથડ ઓવરરાઇટિંગ ત્યારે થાય છે જ્યારે સબકલાસ તેના પેરન્ટ કલાસમાં વ્યાખ્યાયિત મેથડ માટે ચોક્કસ અમલીકરણ આપે છે.

કોષ્ટક 10. મેથડ ઓવરરાઇટિંગના નિયમો

નિયમ	વર્ણન
એક જ નામ	મેથડનું નામ સરખું હોવું જોઈએ
એક જ પેરામીટર્સ	પેરામીટર સંખ્યા અને પ્રકાર મેળ ખાવા જોઈએ
એક જ રિર્ટન પ્રકાર	રિર્ટન પ્રકાર સરખો અથવા સબટાઇપ હોવો જોઈએ
એક્સેસ મોડિફાયર	વધુ પ્રતિબંધિત ન હોઈ શકે
એક્સોપન્સ	વધુ વિસ્તૃત checked exception ફેંકી ન શકે

Listing 11. Method Overriding

```

1 class Animal {
2     void makeSound() {
3         System.out.println("Animal makes a sound");
4     }
5 }
6
7 class Dog extends Animal {
8     // મેથડ ઓવરરાઇટિંગ
9     @Override
10    void makeSound() {
11        System.out.println("Dog barks");
12    }
13 }
```

```

12 }
13 }
14
15 class Cat extends Animal {
16     // મેથડ ઓવરરાઇટિંગ
17     @Override
18     void makeSound() {
19         System.out.println("Cat meows");
20     }
21 }
22
23 public class MethodOverridingDemo {
24     public static void main(String[] args) {
25         Animal animal = new Animal();
26         Animal dog = new Dog();
27         Animal cat = new Cat();
28
29         animal.makeSound(); // Output: Animal makes a sound
30         dog.makeSound(); // Output: Dog barks
31         cat.makeSound(); // Output: Cat meows
32     }
33 }

```

- નટાઇમ પોલિમોફિઝનમ: નટાઇમ પર મેથડનું રિજોલ્યુશન થાય છે
- @Override: એનોટેશન ખાતરી કરે છે કે મેથડ ઓવરરાઇટ થઈ રહી છે
- ઇનહેરેટન્સ: IS-A સંબંધની જરૂર છે

### મેમરી ટ્રીક

“SPARE: એક જ પેરામીટર્સ, એક્સેસ, રિટર્ન, એક્સોપ્શન્સ”

## પ્રશ્ન 4(અ) [3 ગુણ]

યોગ્ય ઉદાહરણ સાથે abstract class સમજાવો.

### જવાબ

Abstract class ને ઇન્સ્ટન્સ બનાવી ન શકાય અને તેમાં abstract મેથડ્સ હોઈ શકે જે સબક્લાસમાં અમલીકરણ કરવી જરૂરી છે.

કોષ્ટક 11. Abstract Class vs Interface

ફીચર	Abstract Class	Interface
ઇન્સ્ટન્સ	બનાવી ન શકાય	બનાવી ન શકાય
મેથડ્સ	કોન્ક્રીટ અને abstract	Abstract (+ Java 8થી default)
વેરિયેબલ્સ	કોઈપણ પ્રકાર	માત્ર નિયતાંકો
કન્સ્ટ્રક્ટર	ધરાવે છે	ધરાવતું નથી

Listing 12. Abstract Class

```

1 abstract class Shape {
2     // Abstract મેથડ - અમલીકરણ નથી
3     abstract double area();
4
5     // કોન્ક્રીટ મેથડ
6     void display() {
7         System.out.println("This is a shape");

```

```

8 }
9 }
10
11 class Circle extends Shape {
12     double radius;
13
14     Circle(double r) {
15         radius = r;
16     }
17
18     // Abstract મેથડનું અમલીકરણ
19     double area() {
20         return 3.14 * radius * radius;
21     }
22 }
```

### મેમરી ટ્રીક

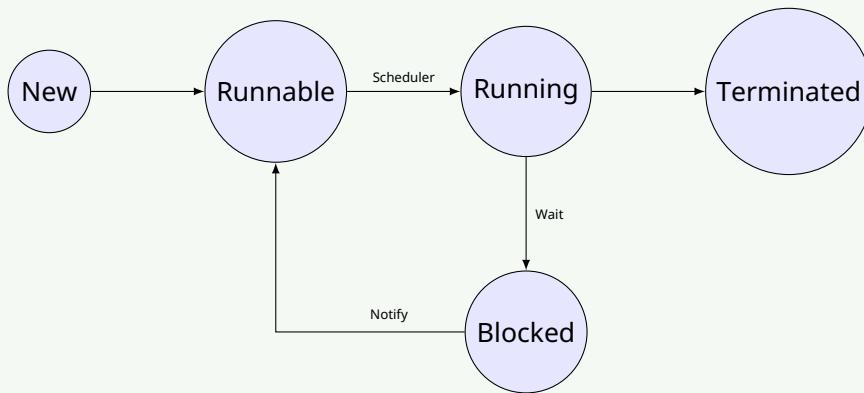
“PAI: આંશિક Abstract અમલીકરણ મુખ્ય છે”

## પ્રશ્ન 4(બ) [4 ગુણ]

થ્રેડ શું છે? થ્રેડ જીવનચક સમજાવો.

### જવાબ

થ્રેડ એ લાઇટવેઈટ સબપ્રોસેસ છે, જે પ્રોસેસિંગની સૌથી નાની એકમ છે જે એક સાથે ચાલતી પ્રક્રિયાઓની મંજૂરી આપે છે.



આકૃતિ 6. થ્રેડ જીવનચક

- **New:** થ્રેડ બનેલ છે પરં શરૂ થયેલ નથી
- **Runnable:** CPU સમય મળે ત્યારે ચાલવા તૈયાર
- **Running:** હલમાં ચાલી રહ્યું છે
- **Blocked/Waiting:** અસ્થાયી રૂપે નિર્દ્ધિય
- **Terminated:** કાર્ય પૂર્ણ થયેલ છે

### મેમરી ટ્રીક

“NRRBT: New Runnable Running Blocked Terminated”

## પ્રશ્ન 4(ક) [7 ગુણ]

જાવામાં એક પ્રોગ્રામ લખો જે Thread Class નો અમલ કરીને બહુવિધ થ્રેડો બનાવે છે.

### જવાબ

Thread class ને અમલ કરીને થ્રેડ બનાવવાથી ઘણા કાર્યો એક સાથે ચલાવી શકાય છે.

**Listing 13. Multithreading Example**

```

1  class MyThread extends Thread {
2      private String threadName;
3
4      MyThread(String name) {
5          this.threadName = name;
6      }
7
8      @Override
9      public void run() {
10         try {
11             for (int i = 1; i <= 5; i++) {
12                 System.out.println(threadName + ":" + i);
13                 Thread.sleep(500);
14             }
15         } catch (InterruptedException e) {
16             System.out.println(threadName + " interrupted");
17         }
18         System.out.println(threadName + " completed");
19     }
20 }
21
22 public class MultiThreadDemo {
23     public static void main(String[] args) {
24         MyThread thread1 = new MyThread("Thread-1");
25         MyThread thread2 = new MyThread("Thread-2");
26         MyThread thread3 = new MyThread("Thread-3");
27
28         thread1.start();
29         thread2.start();
30         thread3.start();
31     }
32 }
```

- **Thread વિસ્તારો:** Thread class વિસ્તારી થ્રેડ બનાવો
- **run()** ઓવરરાઇટ: run મેથડમાં કાર્ય વ્યાખ્યાયિત કરો
- **start():** થ્રેડ ચલાવવાનું શરૂ કરો

### મેમરી ટ્રીક

“ERS: Extend Run Start થ્રેડ બનાવવા માટે”

## પ્રશ્ન 4(અ OR) [3 ગુણ]

ચોંચ ઉદાહરણ સાથે final class સમજાવો.

## જવાબ

Final class નો વારસો મળી શકતો નથી, જેથી તેના ડિઝાઇનમાં ફેરફાર અને વિસ્તરણ અટકાવે છે.

**કોષ્ટક 12.** Final Class લક્ષણો

ફીચર	વર્ણન
ઇનહેરેન્ટસ	સબકલાસ બનાવી શકતો નથી
મેથડ્સ	અંતર્નિહિત final છે
સુરક્ષા	ડિઝાઇન ફેરફારને રોકે છે
ઉદાહરણ	String, Math કલાસ

**Listing 14.** Final Class

```

1 final class Security {
2     void secureMethod() {
3         System.out.println("Secure implementation");
4     }
5 }
6
7 // Error: Cannot extend final class
8 // class HackAttempt extends Security { }
```

- સુરક્ષા: સંવેદનશીલ અમલીકરણનું રક્ષણ કરે છે
- અપરિવર્તનશીલતા: અપરિવર્તનશીલ કલાસ બનાવવામાં મદદ કરે છે
- ઓપ્ટિમાઈઝેશન: JVM final કલાસને ઓપ્ટિમાઈઝ કરી શકે છે

## મેમરી ટ્રીક

“SIO: સુરક્ષા અપરિવર્તનશીલતા ઓપ્ટિમાઈઝેશન”

## પ્રશ્ન 4(બ) OR [4 ગુણ]

યોગ્ય ઉદાહરણ સાથે thread ની પ્રાથમિકતાઓ સમજાવો.

## જવાબ

શેડ પ્રાથમિકતાઓ નક્કી કરે છે કે શેડ્સને અમલીકરણ માટે કયા કમમાં શેડ્યુલ કરવા, 1 (ન્યૂનતમ) થી 10 (ઉચ્ચતમ).

**કોષ્ટક 13.** શેડ પ્રાયોરિટી નિયતાંકો

નિયતાંક	મૂલ્ય	વર્ણન
MIN_PRIORITY	1	-ન્યૂનતમ પ્રાથમિકતા
NORM_PRIORITY	5	ડિફોલ્ટ પ્રાથમિકતા
MAX_PRIORITY	10	ઉચ્ચતમ પ્રાથમિકતા

**Listing 15.** Thread Priority

```

1 class PriorityThread extends Thread {
2     PriorityThread(String name) {
3         super(name);
4     }
5
6     public void run() {
7         System.out.println("Running: " + getName() +
```

```

8         " with priority: " + getPriority());
9     }
10    }
11
12   public class ThreadPriorityDemo {
13       public static void main(String[] args) {
14           PriorityThread low = new PriorityThread("Low Priority");
15           PriorityThread norm = new PriorityThread("Normal Priority");
16           PriorityThread high = new PriorityThread("High Priority");
17
18           low.setPriority(Thread.MIN_PRIORITY);
19           high.setPriority(Thread.MAX_PRIORITY);
20
21           low.start();
22           norm.start();
23           high.start();
24       }
25   }

```

**મેમરી ટ્રીક**

“HNL: ઉચ્ચ સામાન્ય નિમ્ન પ્રાથમિકતાઓ થ્રેડ્સમાં”

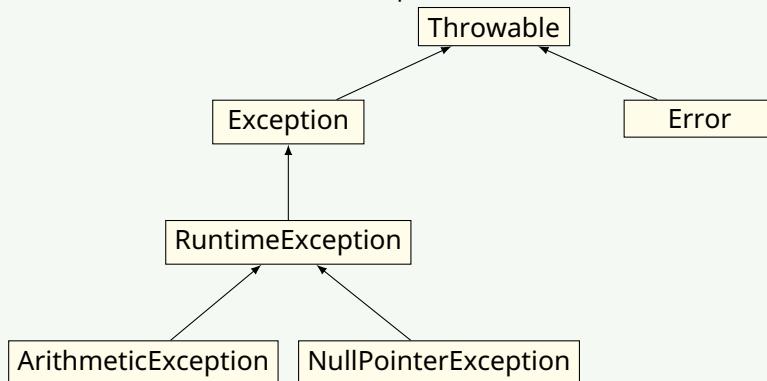
**પ્રશ્ન 4(ક) OR [7 ગુણ]**

Exception શું છે? Arithmetic Exception નો ઉપયોગ દર્શાવતો પ્રોગ્રામ લખો.

**જવાબ**

Exception એ અસામાન્ય સ્થિતિ છે જે પ્રોગ્રામના સામાન્ય પ્રવાહને વિકષેપિત કરે છે.

**કોષ્ટક 14. Exception હાથરાકી**



**Listing 16. Arithmetic Exception Demo**

```

1  public class ArithmaticExceptionDemo {
2      public static void main(String[] args) {
3          try {
4              // આ ArithmaticException ઉત્પન્ત કરશે
5              int result = 100 / 0;
6              System.out.println("Result: " + result);
7          }
8          catch (ArithmaticException e) {

```

```

9     System.out.println("ArithmaticException caught: " + e.getMessage());
10    System.out.println("Cannot divide by zero");
11 }
12 finally {
13     System.out.println("This block always executes");
14 }
15
16 System.out.println("Program continues after exception handling");
17 }
18 }
```

- **Try Block:** એવો કોડ ધરાવે છે જે exception ફેકી શકે છે
- **Catch Block:** ચોક્કસ exception હેન્ડલ કરે છે
- **Finally Block:** exception ફેકાય કે ન ફેકાય, હંમેશા ચાલે છે

#### મેમરી ટ્રીક

“TCF: Try Catch Finally exceptions હેન્ડલ કરે છે”

## પ્રશ્ન 5(અ) [3 ગુણ]

એરેની 10 સંખ્યાઓનો સરવાળો અને સરેરાશ શોધવા માટેનો જાવા પ્રોગ્રામ લખો.

#### જવાબ

એરે એક જ પ્રકારની ઘણી કિંમતો સંગ્રહ છે, જે તત્ત્વોની કમિક પ્રક્રિયા કરવાની મંજૂરી આપે છે.

**Listing 17. Array Sum Average**

```

1 public class ArraySumAverage {
2     public static void main(String[] args) {
3         int[] numbers = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};
4
5         int sum = 0;
6
7         // સરવાળો ગણો
8         for (int i = 0; i < numbers.length; i++) {
9             sum += numbers[i];
10        }
11
12        // સરેરાશ ગણો
13        double average = (double) sum / numbers.length;
14
15        System.out.println("Sum = " + sum);
16        System.out.println("Average = " + average);
17    }
18 }
```

- ધોખાણા: નિશ્ચિત-કદના સંગ્રહ બનાવે છે
- પુનરાવર્તન: તત્ત્વોનો કમિક એક્સોસ્સ
- ગણતરી: પરિણામો માટે મૂલ્યો પર પ્રક્રિયા કરો

#### મેમરી ટ્રીક

“DIC: Declare Iterate Calculate એરે પ્રોસેસિંગ માટે”

## પ્રશ્ન 5(બ) [4 ગુણ]

'DivideByZero' એરર માટે યુઝર ડિફાર્ન્ડ Exception હૈન્ડલ કરવા માટે જાવા પ્રોગ્રામ લખો.

### જવાબ

યુઝર-ડિફાર્ન્ડ exception ચોક્કસ એપ્લિકેશન જરૂરિયાતો માટે કર્યાની exception પ્રકારો બનાવવાની મંજૂરી આપે છે.

**Listing 18. Custom Exception**

```

1 // કર્યાની exception કલાસ
2 class DivideByZeroException extends Exception {
3     public DivideByZeroException(String message) {
4         super(message);
5     }
6 }
7
8 public class CustomExceptionDemo {
9     // કર્યાની exception ફેક્ટરી મેથ્ડ
10    static double divide(int numerator, int denominator) throws DivideByZeroException {
11        if (denominator == 0) {
12            throw new DivideByZeroException("Cannot divide by zero!");
13        }
14        return (double) numerator / denominator;
15    }
16
17    public static void main(String[] args) {
18        try {
19            System.out.println(divide(10, 2));
20            System.out.println(divide(20, 0));
21        } catch (DivideByZeroException e) {
22            System.out.println("Custom exception caught: " + e.getMessage());
23        }
24    }
25 }
```

### મેમરી ટ્રીક

“CTE: Create Throw Exception જ્યારે જરૂર હોય”

## પ્રશ્ન 5(ક) [7 ગુણ]

ટેક્સ્ટ ફાઇલ બનાવવા માટે જાવા પ્રોગ્રામ લખો અને ટેક્સ્ટ ફાઇલ પર રીડ ઓપરેશન કરો.

### જવાબ

જાવા I/O કલાસ ફાઇલો સાથે કામ કરવા માટે સગવડ આપે છે, જે સર્જન, લેખન અને વાંચન ઓપરેશન્સની મંજૂરી આપે છે.

**Listing 19. File Read Write**

```

1 import java.io.FileWriter;
2 import java.io.FileReader;
3 import java.io.IOException;
4 import java.io.BufferedReader;
5
6 public class FileOperationsDemo {
7     public static void main(String[] args) {
8         try {
```

```

9 // ફાઇલ બનાવો અને લખો
10 FileWriter writer = new FileWriter("sample.txt");
11 writer.write("Hello World!\n");
12 writer.write("Welcome to Java File Handling.\n");
13 writer.write("This is the third line.");
14 writer.close();
15 System.out.println("Successfully wrote to the file.");
16
17 // ફાઇલમાંથી વાંચો
18 FileReader reader = new FileReader("sample.txt");
19 BufferedReader buffReader = new BufferedReader(reader);
20
21 String line;
22 System.out.println("\nFile contents:");
23 while ((line = buffReader.readLine()) != null) {
24     System.out.println(line);
25 }
26
27 reader.close();
28
29 } catch (IOException e) {
30     System.out.println("An error occurred: " + e.getMessage());
31 }
32 }
33 }
```

- **FileWriter:** ફાઇલો બનાવે અને લખે છે
- **FileReader:** ફાઇલમાંથી અક્ષર ડેટા વાંચે છે
- **BufferedReader:** લાઇન દ્વારા ટેક્સ્ટ કાર્યક્રમતાથી વાંચે છે

### મેમરી ટ્રીક

“WRC: Write Read Close ફાઇલ ઓપરેશન્સ માટે”

## પ્રશ્ન 5(અ) OR [3 ગુણ]

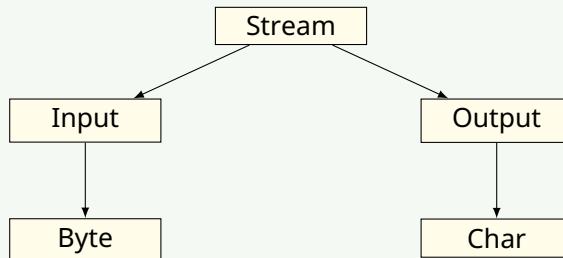
Java I/O પ્રક્રિયા સમજાવો.

### જવાબ

Java I/O પ્રક્રિયામાં સ્ટ્રીમનો ઉપયોગ કરીને વિવિધ સોલોથી ડેટા ટ્રાન્સફર કરવાનો સમાવેશ થાય છે.

કોષ્ટક 15. Java I/O સ્ટ્રીમ પ્રકારો

વર્ગીકરણ	પ્રકારો
દિશા	ઇનપુટ, આઉટપુટ
ડેટા પ્રકાર	બાઇટ સ્ટ્રીમ્સ, કેરેક્ટર સ્ટ્રીમ્સ
કાર્યક્રમતા	બેઝિક, બફર્ડ, ડેટા, ઓફ્જેક્ટ



## આફ્ટિ 7. Java I/O હાયરાર્કો

- સ્ટ્રીમ: સોત અને લક્ષ્ય વચ્ચે વહેતા ડેટાની શ્રેણી
- બફરિંગ: ડિસ્ક એક્સેસ ઘટાડીને કાર્યક્ષમતા સુધારે છે

## મેમરી ટ્રીક

“SBI: સ્ટ્રીમ બફર ઇનપુટ/આઉટપુટ”

## પ્રશ્ન 5(બ) OR [4 ગુણ]

Exception હેન્ડલિંગમાં throw અને finally ઉદાહરણ સાથે સમજાવો.

## જવાબ

Exception હેન્ડલિંગ મેકેનિઝમ્સ ભૂલો દરમિયાન પ્રોગ્રામ ફલોને નિયંત્રિત કરે છે, સુંદર અમલીકરણ સુનિશ્ચિત કરે છે.

## કાષ્ટક 16. throw vs finally

ક્રીયર	throw	finally
હેતુ	સ્પષ્ટપણે exception ફેંકે છે	કોડ અમલીકરણ સુનિશ્ચિત કરે છે
સ્થાન	મેથડની અંદર	try-catch બ્લોક્સ પછી
અમલીકરણ	શરત પૂરી થાય ત્યારે	return હોય તો પણ હંમેશા
ઉપયોગ	કંટ્રોલ ફલો	રિસોર્સ કલીનઅપ

## Listing 20. Throw and Finally

```

1 public class ThrowFinallyDemo {
2     public static void validateAge(int age) {
3         try {
4             if (age < 18) {
5                 throw new ArithmeticException("Not eligible to vote");
6             } else {
7                 System.out.println("Welcome to vote");
8             }
9         } catch (ArithmeticException e) {
10             System.out.println("Exception caught: " + e.getMessage());
11         } finally {
12             System.out.println("Validation process completed");
13         }
14     }
15
16     public static void main(String[] args) {
17         validateAge(15);
18         System.out.println("-----");
19         validateAge(20);
20     }
21 }
```

## મેમરી ટ્રીક

“TERA: Throw Exception, Regardless Always, finally હંમેશા ચાલે છે”

## પ્રશ્ન 5(ક) OR) [7 ગુણ]

ટેક્સ્ટ ફાઇલ ના કન્ટેનર ડિસ્પ્લે કરવા અને ટેક્સ્ટ ફાઇલ પર એપેન્ડ ઓપરેશન કરવા માટે જાવા પ્રોગ્રામ લખો.

### જવાબ

જાવામાં ફાઇલ ઓપરેશન્સ ફાઇલ કન્ટેનરને હેરફેર કરવાની મંજૂરી આપે છે, નવા ડેરા ઉમેરવા સહિત.

**Listing 21. File Append**

```

1 import java.io.*;
2
3 public class FileAppendDemo {
4     public static void main(String[] args) {
5         try {
6             // પ્રારંભિક ફાઇલ બનાવો
7             FileWriter writer = new FileWriter("example.txt");
8             writer.write("Original content line 1\n");
9             writer.write("Original content line 2\n");
10            writer.close();
11
12            // ફાઇલ કન્ટેનર દર્શાવો
13            System.out.println("Original file content:");
14            readFile("example.txt");
15
16            // ફાઇલમાં ઉમેરો
17            FileWriter appendWriter = new FileWriter("example.txt", true);
18            appendWriter.write("Appended content line 1\n");
19            appendWriter.write("Appended content line 2\n");
20            appendWriter.close();
21
22            // અપડેટ થયેલ કન્ટેનર દર્શાવો
23            System.out.println("\nFile content after append:");
24            readFile("example.txt");
25
26        } catch (IOException e) {
27            System.out.println("An error occurred: " + e.getMessage());
28        }
29    }
30
31    // ફાઇલ કન્ટેનર વાંચવા અને દર્શાવવા માટેની મેથડ
32    public static void readFile(String fileName) {
33        try {
34            BufferedReader reader = new BufferedReader(new FileReader(fileName));
35            String line;
36            while ((line = reader.readLine()) != null) {
37                System.out.println(line);
38            }
39            reader.close();
40        } catch (IOException e) {
41            System.out.println("Error reading file: " + e.getMessage());
42        }
43    }
44}

```

- **FileWriter(file, true):** બીજો પેરામીટર એપેન્ડ મોડ સક્ષમ કરે છે
- **BufferedReader:** લાઇન દ્વારા ટેક્સ્ટને કાર્યક્રમતાથી વાંચે છે
- **ફ્રીથી વાપરી શકાય તેવી મેથડ:** વાચન કાર્યક્રમતાને સંકલિત કરે છે

## મેમરી ટ્રીક

“CAD: Create Append Display ફાઇલ ઓપરેશન્સ”