Object Oriented Programming With Java (4341602) - Summer 2025 Solution

Milav Dabgar

May 15, 2025

પ્રશ્ન 1(અ) [3 ગુણ]

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

જવાબ

તફાવત:

કોષ્ટક 1. POP vs OOP

પાસાં	POP	OOP
અભિગમ	ટોપ-ડાઉન અભિગમ	બોટમ-અપ અભિગમ
ફોકસ	ફંક્શન્સ અને પ્રોસિજર્સ	ઓબ્જેક્ટ્સ અને ક્લાસેસ
ડેટા સિક્યોરિટી	ઓછી સુરક્ષિત, ગ્લોબલ ડેટા	વધુ સુરક્ષિત, ડેટા encapsulation
સમસ્યા ઉકેલ	ફંક્શન્સમાં વિભાજન	ઓબ્જેક્ટ્સમાં વિભાજન

મુખ્ય મુદ્દા:
- **POP**: ફંક્શન્સ પ્રાથમિક બિલ્ડિંગ બ્લોક્સ છે.
- **OOP**: ઓબ્જેક્ટ્સમાં ડેટા અને મેથડ્સ બંને સામેલ છે.
- **પુનઃઉપયોગ**: OOP કોડની વધુ સારી પુનઃઉપયોગિતા (Reusability) પ્રદાન કરે છે.

મેમરી ટ્રીક

``POP Functions, OOP Objects''

પ્રશ્ન 1(બ) [4 ગુણ]

OOP ના મૂળભૂત ખ્યાલોની નોંધણી કરો અને સમજાવો.

જવાબ

OOP ના મૂળભૂત ખ્યાલો:
- **Encapsulation**: ક્લાસમાં ડેટા અને મેથડ્સને એકસાથે બાંધવું. તે ડેટાને બહારની દખલથી સુરક્ષિત રાખે છે.
- **Inheritance**: હાલની ક્લાસેસમાંથી નવી ક્લાસેસ બનાવવી. તે કોડના પુનઃઉપયોગને પ્રોત્સાહન આપે છે.
- **Polymorphism**: સમાન મેથડ નામ સાથે વિવિધ implementations (દા.ત., Overloading, Overriding).
- **Abstraction**: યુઝરથી જટિલ implementation વિગતો છુપાવવી અને માત્ર જરૂરી કાર્યક્ષમતા દર્શાવવી.

ફાયદા:
- **કોડ પુનઃઉપયોગ**: Inheritance અને polymorphism દ્વારા.
- **ડેટા સિક્યોરિટી**: Encapsulation દ્વારા.
- **સરળ Maintenance**: મોડ્યુલર અભિગમ અપડેટ્સને સરળ બનાવે છે.

મેમરી ટ્રીક

``Every Intelligent Person Abstracts''

પ્રશ્ન 1(ક) [7 ગુણ]

Constructor વ્યાખ્યાયિત કરો. વિવિધ પ્રકારના Constructors ની નોંધણી કરો અને તેમાંથી કોઈપણ 2 ને યોગ્ય ઉદાહરણ સાથે સમજાવો.

જવાબ

Constructor વ્યાખ્યા: Constructor એ એક વિશેષ મેથડ છે જે ઓબ્જેક્ટ બનાવવામાં આવે ત્યારે તેને initialize કરે છે. તેનું નામ ક્લાસ જેવું જ હોય છે અને તેનો કોઈ return type હોતો નથી.

Constructor ના પ્રકારો:

1. Default Constructor
2. Parameterized Constructor
3. Copy Constructor
4. Private Constructor

Listing 1. Constructor ઉદાહરણ

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

   // 1. Default Constructor
   public Student() {
      name = "Unknown";
      age = 0;
   }

   // 2. Parameterized Constructor
   public Student(String n, int a) {
      name = n;
      age = a;
   }
}

class Main {
   public static void main(String[] args) {
      Student s1 = new Student();       // Default કૉલ કરે છે
      Student s2 = new Student("John", 20); // Parameterized કૉલ કરે છે
   }
}
```

મુખ્ય લક્ષણો:

- **આપોઆપ કૉલ**: ઓબ્જેક્ટ બનાવવા દરમિયાન આપોઆપ કૉલ થાય છે.
- **કોઈ Return Type નથી**: Constructor નો કોઈ return type (void પણ નહીં) હોતો નથી.

મેમરી ટ્રીક

``Constructors Create Objects''

પ્રશ્ન 1(ક OR) [7 ગુણ]

String class સમજાવો. String class ની વિવિધ methods ની યાદી બનાવો અને તેમાંથી કોઈપણ 3 ને યોગ્ય ઉદાહરણ

સાથે સમજાવો.

જવાબ

String Class: Java માં String ક્લાસ immutable (બદલી ન શકાય તેવા) character sequences ને રિપ્રેઝેન્ટ કરે છે. એકવાર બનાવ્યા પછી, String ઓબ્જેક્ટ્સ બદલી શકાતા નથી.
String Methods:

કોષ્ટક 2. String Methods

Method	હેતુ
length()	string ની લંબાઈ પરત કરે છે.
charAt(index)	આપેલ index પર રહેલો character પરત કરે છે.
substring(start, end)	start થી end-1 સુધીની substring કાઢે છે.
indexOf(char)	character ની પ્રથમ પોઝિશન શોધે છે.
toUpperCase()	બધા characters ને uppercase માં કન્વર્ટ કરે છે.

Listing 2. String Methods Demo

```
public class StringDemo {
   public static void main(String[] args) {
      String str = "Hello World";

      // 1. length() method
      System.out.println("Length: " + str.length()); // 11

      // 2. charAt() method
      System.out.println("Char at 0: " + str.charAt(0)); // H

      // 3. substring() method
      System.out.println("Substring: " + str.substring(0, 5)); // Hello
   }
}
```

મુખ્ય મુદ્દા:
- **Immutable**: String ઓબ્જેક્ટ્સ બદલી શકાતા નથી.
- **Memory Efficient**: Storage માટે string pool નો ઉપયોગ કરે છે.

મેમરી ટ્રીક

``Strings Store Text''

પ્રશ્ન 2(અ) [3 ગુણ]

Garbage Collection વ્યાખ્યાયિત કરો. JAVA પ્રોગ્રામિંગમાં Garbage Collection નું મહત્વ જણાવો.

જવાબ

Garbage Collection વ્યાખ્યા: આ એક ઓટોમેટિક મેમરી મેનેજમેન્ટ પ્રક્રિયા છે જે એવા ઓબ્જેક્ટ્સ દ્વારા કબજે કરાયેલી મેમરી પાછી મેળવે છે (reclaim કરે છે) જેનો હવે પ્રોગ્રામમાં કોઈ reference (ઉપયોગ) નથી.
મહત્વ:
- **આપોઆપ મેમરી મેનેજમેન્ટ**: ડેવલપરે મેન્યુઅલી મેમરી allocate કે deallocate કરવાની જરૂર રહેતી નથી.
- **Memory Leaks ટાળે છે**: આપોઆપ unused મેમરી મુક્ત કરે છે, જેનાથી leaks નું જોખમ ઘટે છે.
- **Application Performance**: મેમરી ઉપયોગને optimize કરે છે.

ફાયદા:
- **Productivity**: પ્રોગ્રામર મેમરીને બદલે business logic પર ધ્યાન આપી શકે છે.

- **Reliability**: મેમરી issues ને લીધે થતા crashes ઘટાડે છે.

મેમરી ટ્રીક

``Garbage Collector Cleans Memory''

પ્રશ્ન 2(બ) [4 ગુણ]

Object ને Garbage collection માટે eligible બનાવવાની ચાર રીતોની યાદી બનાવો.

જવાબ

GC Eligibility ની રીતો: જ્યારે કોઈ ઓબ્જેક્ટનો કોઈ reference રહેતો નથી ત્યારે તે Garbage Collection માટે eligible બને છે.

કોષ્ટક 3. GC Eligibility Methods

રીત	વર્ણન
Reference ને Null કરવું	સ્પષ્ટપણે object reference ને null સેટ કરવું.
Reference ફરીથી Assign કરવું	Reference variable ને બીજા કોઈ object પર point કરવું.
Anonymous Objects	Reference variable વિના objects બનાવવા.
Island of Isolation	જ્યારે બે કે વધુ ઓબ્જેક્ટ્સ માત્ર એકબીજાને refer કરતા હોય અને બહારથી કોઈ access ન હોય.

ઉદાહરણો:
1. Student s = new Student(); s = null;
2. Student s1 = new Student(); s1 = new Student();
3. new Student();

મેમરી ટ્રીક

``Null References Attract Islands''

પ્રશ્ન 2(ક) [7 ગુણ]

Static block દર્શાવવા માટે JAVA પ્રોગ્રામ લખો જે main પહેલા execute થાય છે. તેનું મહત્વ સમજાવો.

જવાબ

Listing 3. Static Block Demo

```
public class StaticBlockDemo {
    static int count;

    // Static block - Main પહેલા execute થાય છે
    static {
        System.out.println("Static block executed first");
        count = 10;
        System.out.println("Count initialized to: " + count);
    }

    public static void main(String[] args) {
        System.out.println("Main method started");
        System.out.println("Count value: " + count);
    }
```

```
}
```

આઉટપુટ:

```
Static block executed first
Count initialized to: 10
Main method started
Count value: 10
```

મહત્વ અને ઉપયોગ:
- **પ્રારંભિક Initialization**: જ્યારે ક્લાસ મેમરીમાં લોડ થાય છે ત્યારે મેઈન મેથડ પહેલા જ static block આપોઆપ execute થાય છે.
- **Class Loading**: તે class loading વખતે માત્ર એક જ વાર રન થાય છે.
- **ઉપયોગ**: Static variables initialize કરવા અથવા native libraries લોડ કરવા માટે ઉપયોગી છે.

મેમરી ટ્રીક

``Static Blocks Start Before Main''

પ્રશ્ન 2(અ OR) [3 ગુણ]

JAVA માં Minor/Incremental અને Major/Full Garbage collection નું વર્ણન કરો.

જવાબ

Garbage Collection ના પ્રકારો:

કોષ્ટક 4. Minor vs Major GC

પ્રકાર	વર્ણન	આવર્તન (Frequency)
Minor GC	**Young Generation** (Eden space) સાફ કરે છે. ટૂંકા-જીવનવાળા ઓબ્જેક્ટ્સ દૂર કરે છે.	વારંવાર, ઝડપી
Major GC	**Old Generation** (Tenured space) સાફ કરે છે. આખી heap (Full GC) આવરી લે છે.	ઓછું વારંવાર, ધીમું

વિગતો:
- **Minor GC**: Eden space ભરાઈ જાય ત્યારે trigger થાય છે. તે ઝડપી છે અને ઓછું pause આપે છે.
- **Major GC**: Old Generation ભરાઈ જાય ત્યારે trigger થાય છે. તે "Stop the World" pauses આપી શકે છે જે performance ને અસર કરે છે.

મેમરી ટ્રીક

``Minor Frequent, Major Slow''

પ્રશ્ન 2(બ OR) [4 ગુણ]

JAVA માં finalize() method ને તેના ફાયદાઓ સાથે સમજાવો.

જવાબ

finalize() Method: આ Object ક્લાસમાં વ્યાખ્યાયિત એક મેથડ છે. Garbage Collector ઓબ્જેક્ટને destroy કરતા અને મેમરી reclaim કરતા પહેલા આ મેથડને કૉલ કરે છે. તેનો ઉપયોગ cleanup processing માટે થાય છે.
Syntax:

```
protected void finalize() throws Throwable {
```

```
    // Cleanup code
}
```

ફાયદા:
- **Resource Cleanup**: જો પ્રોગ્રામર ફાઇલ સ્ટ્રીમ્સ અથવા ડેટાબેઝ કનેક્શન્સ બંધ કરવાનું ભૂલી જાય, તો આ મેથડ ઉપયોગી બને છે.
- **Safety Net**: તે resources મુક્ત કરવા માટે છેલ્લી તક પૂરી પાડે છે.

મેમરી ટ્રીક

``Finalize Frees Resources''

પ્રશ્ન 2(ક OR) [7 ગુણ]

public static void main(String[] args) ની syntax સમજાવો. Command line argument તરીકે લેવાયેલ input ને છાપવા માટે JAVA પ્રોગ્રામ લખો.

જવાબ

Main Method Syntax સમજૂતી: public static void main(String[] args)
- **public**: Access modifier છે, જે મેથડને ગમે ત્યાંથી accessible બનાવે છે (JVM તેને કૉલ કરી શકે).
- **static**: ઓબ્જેક્ટ બનાવ્યા વિના JVM આ મેથડને કૉલ કરી શકે છે.
- **void**: આ મેથડ કોઈ value return કરતી નથી.
- **main**: આ JVM દ્વારા ઓળખાતું ચોક્કસ મેથડ નામ છે.
- **String[] args**: Command-line arguments સ્ટોર કરતું String array.

Listing 4. Command Line Arguments Demo

```
public class CommandLineDemo {
    public static void main(String[] args) {
        System.out.println("Arguments ની સંખ્યા: " + args.length);

        if(args.length > 0) {
            System.out.println("Command line arguments:");
            for(int i = 0; i < args.length; i++) {
                System.out.println("Arg " + i + ": " + args[i]);
            }
        } else {
            System.out.println("કોઈ arguments મળ્યા નથી");
        }
    }
}
```

Execution અને Output: java CommandLineDemo Hello World 123

```
Arguments ની સંખ્યા: 3
Command line arguments:
Arg 0: Hello
Arg 1: World
Arg 2: 123
```

પ્રશ્ન 3(અ) [3 ગુણ]

વિવિધ Java access modifiers ની યાદી બનાવો અને સમજાવો.

જવાબ

Java Access Modifiers: Access modifiers ક્લાસ, કન્સ્ટ્રક્ટર, વેરિયેબલ, મેથડ અથવા ડેટા મેમ્બરનો scope (visibility) નક્કી કરે છે.

કોષ્ટક 5. Access Modifiers Scope

Modifier	Class	Package	Subclass	World
public	હા	હા	હા	હા
protected	હા	હા	હા	ના
default	હા	હા	ના	ના
private	હા	ના	ના	ના

ઉપયોગ:
- **public**: બધે accessible છે.
- **protected**: Inheritance માટે વપરાય છે; package માં અને બહાર subclasses દ્વારા accessible છે.
- **default**: (કોઈ keyword નહીં) Package-private છે.
- **private**: માત્ર defining ક્લાસમાં જ restricted છે.

મેમરી ટ્રીક

``Public Protected Default Private''

પ્રશ્ન 3(બ) [4 ગુણ]

JAVA માં interface નું વર્ણન કરો. Executable ઉદાહરણ સાથે interface નો inheritance દર્શાવો.

જવાબ

Java માં Interface: Interface એ Java માં એક reference type છે. તે ક્લાસ જેવું જ છે પરંતુ તે કરાર (contract) રજૂ કરે છે. તેમાં માત્ર constants, method signatures, default methods, static methods, અને nested types હોઈ શકે છે. Interfaces ને instantiate કરી શકાતા નથી—તે માત્ર ક્લાસ દ્વારા implement અથવા અન્ય interfaces દ્વારા extend કરી શકાય છે.

Listing 5. Interface Inheritance

```
// Parent interface
interface Animal {
    void sound();
}

// Child interface Animal થી inherit કરે છે
interface Mammal extends Animal {
    void walk();
}

// Child interface ને implement કરતી class
class Dog implements Mammal {
    public void sound() {
        System.out.println("Dog barks");
    }

    public void walk() {
        System.out.println("Dog walks on four legs");
    }
}

class Main {
    public static void main(String[] args) {
        Dog d = new Dog();
```

```
        d.sound();
        d.walk();
    }
}
```

મેમરી ટ્રીક

``Interfaces Inherit Contracts''

પ્રશ્ન 3(ક) [7 ગુણ]

super keyword વ્યાખ્યાયિત કરો અને executable JAVA પ્રોગ્રામ સાથે super keyword નો ઉપયોગ દર્શાવો

જવાબ

super Keyword: Java માં super keyword એ એક reference variable છે જેનો ઉપયોગ immediate parent class ના object ને refer કરવા માટે થાય છે. તેનો ઉપયોગ current class ના scope ને bypass કરીને parent class ના members ને access કરવા માટે થાય છે.

ઉપયોગ:

1. **Variable Access**: Immediate parent class instance variable ને refer કરવા (`super.variable`).
2. **Method Call**: Immediate parent class method ને invoke કરવા (`super.method()`).
3. **Constructor Call**: Immediate parent class constructor ને invoke કરવા (`super()`).

Listing 6. super Keyword નો ઉપયોગ

```
class Animal {
    String name = "Animal";

    Animal(String type) {
        System.out.println("Animal constructor: " + type);
    }

    void sound() {
        System.out.println("Animal makes sound");
    }
}

class Dog extends Animal {
    String name = "Dog";

    Dog() {
        // 1. Calling parent constructor
        super("Mammal");
        System.out.println("Dog constructor");
    }

    void sound() {
        // 2. Calling parent method
        super.sound();
        System.out.println("Dog barks");
    }

    void display() {
        // 3. Accessing parent variable
        System.out.println("Parent name: " + super.name);
        System.out.println("Child name: " + this.name);
    }
}
```

```
class Main {
    public static void main(String[] args) {
        Dog d = new Dog();
        d.sound();
        d.display();
    }
}
```

મેમરી ટ્રીક

``Super Calls Parent''

પ્રશ્ન 3(અ OR) [3 ગુણ]

JAVA માં package ને વ્યવહારુ ઉદાહરણ સાથે સમજાવો.

જવાબ

Package: Java માં package એ ક્લાસીસ, સબ-પેકેજીસ અને ઈન્ટરફેસના જૂથને સમાવિષ્ટ (encapsulate) કરવાની એક પદ્ધતિ છે.
ફાયદા:

- **Organization**: બહેતર કોડ મેનેજમેન્ટ માટે વર્ગો (દા.ત., `model`, `view`, `controller`) ને વર્ગીકૃત કરે છે.
- **Access Control**: Access protection પ્રદાન કરે છે (default/protected access).
- **Namespace Management**: નામકરણ સંઘર્ષો (naming conflicts) અટકાવે છે.

Listing 7. Package બનાવવું અને વાપરવું

```
// File: com/company/model/Student.java
package com.company.model;

public class Student {
    private String name;
    public void setName(String name) { this.name = name; }
}

// File: Main.java
import com.company.model.Student;

public class Main {
    public static void main(String[] args) {
        Student s = new Student();
        s.setName("John");
    }
}
```

મેમરી ટ્રીક

``Packages Organize Classes''

પ્રશ્ન 3(બ OR) [4 ગુણ]

વ્યવહારુ ઉદાહરણ સાથે abstract અને final keywords સમજાવો.

જવાબ

કોષ્ટક 6. Abstract vs Final

Keyword	હેતુ	ઉપયોગ સ્તર
abstract	અધૂરી implementation (template) વ્યાખ્યાયિત કરે છે. Instantiate કરી શકાતું નથી.	Class, Method
final	અચલ (constant) અથવા ન બદલી શકાય તેવી entity વ્યાખ્યાયિત કરે છે. ફેરફાર/એક્સ્ટેંશન અટકાવે છે.	Class, Method, Variable

Listing 8. Abstract અને Final Demo

```
// Abstract class
abstract class Shape {
    final double PI = 3.14;  // final variable (Constant)

    abstract void draw();    // abstract method (No body)

    final void display() {   // final method (Override ન કરી શકાય)
        System.out.println("Displaying shape");
    }
}

// Final class (Extend ન કરી શકાય)
final class Circle extends Shape {
    void draw() {
        System.out.println("Drawing circle");
    }
}
```

મેમરી ટ્રીક

``Abstract Allows, Final Forbids''

પ્રશ્ન 3(ક OR) [7 ગુણ]

Java Programming language context માં Dynamic Method Dispatch વ્યાખ્યાયિત કરો. Dynamic Method Dispatch દર્શાવતો executable પ્રોગ્રામ બનાવો.

જવાબ

Dynamic Method Dispatch (Runtime Polymorphism): આ એક પદ્ધતિ છે જેના દ્વારા overridden method નો કૉલ compile-time ને બદલે runtime એ resolve થાય છે. આ parent class reference variable નો ઉપયોગ કરીને child class object ને refer કરીને implement કરવામાં આવે છે.

તે કેવી રીતે કામ કરે છે:

1. Variable દ્વારા refer થતા actual object type ને retrieve કરે છે.
2. Virtual Method Table (vtable) માં method શોધે છે.
3. Actual object ને અનુરૂપ method implementation invoke કરે છે.

Listing 9. Dynamic Dispatch ઉદાહરણ

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

```
}

// Derived classes
class Dog extends Animal {
    void sound() {
        System.out.println("Dog barks");
    }
}

class Cat extends Animal {
    void sound() {
        System.out.println("Cat meows");
    }
}

class DynamicDispatchDemo {
    public static void main(String[] args) {
        Animal ref;  // Reference variable

        // Runtime method resolution

        ref = new Dog(); // Dog object ને refer કરે છે
        ref.sound();     // Dog ની sound() કૉલ થાય છે

        ref = new Cat(); // Cat object ને refer કરે છે
        ref.sound();     // Cat ની sound() કૉલ થાય છે

        ref = new Animal();
        ref.sound();     // Animal ની sound() કૉલ થાય છે
    }
}
```

આઉટપુટ:

Dog barks
Cat meows
Animal makes sound

મેમરી ટ્રીક

``Dynamic Dispatch Decides Runtime''

પ્રશ્ન 4(અ) [3 ગુણ]

Exception Handling માં throw અને finally keywords સમજાવો.

જવાબ

throw:
- Method અથવા code block માંથી સ્પષ્ટપણે exception throw કરવા માટે વપરાય છે.
- મુખ્યત્વે custom exceptions throw કરવા માટે વપરાય છે.
- Syntax: throw new Exception("Message");

finally:
- try અથવા catch પછીનો બ્લોક છે.
- Exception આવ્યું હોય કે ન આવ્યું હોય, તે **હંમેશા execute થાય છે**.
- સામાન્ય રીતે cleanup code (files બંધ કરવા, resources release કરવા) માટે વપરાય છે.

Listing 10. Throw અને Finally

```java
try {
   if(age < 0)
      throw new IllegalArgumentException("Invalid age");
} catch(Exception e) {
   System.out.println(e);
} finally {
   System.out.println("આ હંમેશા પ્રિન્ટ થશે.");
}
```

મેમરી ટ્રીક

``Throw Creates, Finally Cleans''

પ્રશ્ન 4(બ) [4 ગુણ]

JAVA માં try...catch block દર્શાવતો પ્રોગ્રામ લખો

જવાબ

Listing 11. Try-Catch Demo

```java
public class TryCatchDemo {
   public static void main(String[] args) {
      try {
         int[] arr = {1, 2, 3};
         // આ લાઇન ArrayIndexOutOfBoundsException આપશે
         System.out.println("Array element: " + arr[5]);

         // આ લાઇન execute નહીં થાય
         int result = 10 / 0;

      } catch(ArrayIndexOutOfBoundsException e) {
         System.out.println("Array index error: " + e.getMessage());

      } catch(ArithmeticException e) {
         System.out.println("Math error: " + e.getMessage());

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

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

આઉટપુટ:

Array index error: Index 5 out of bounds for length 3
Program continues...

મેમરી ટ્રીક

``Try Code, Catch Errors''

પ્રશ્ન 4(ક) [7 ગુણ]

ArrayIndexOutOfBoundsException Exception વ્યાખ્યાયિત કરો. તેને પ્રદર્શિત કરતો એક કાર્યક્ષમ JAVA પ્રોગ્રામ લખો. Input(ઓ) નો પણ ઉલ્લેખ કરો જે આ Exception ને વધારશે.

જવાબ

વ્યાખ્યા: ArrayIndexOutOfBoundsException એ એક runtime exception છે જે ત્યારે thrown થાય છે જ્યારે કોડ અયોગ્ય index સાથે array element ને access કરવાનો પ્રયાસ કરે છે. Index નકારાત્મક હોય અથવા array ના કદ કરતા મોટો કે બરાબર હોય ત્યારે આવું થાય છે.

Listing 12. ArrayIndexOutOfBounds Demo

```
public class ArrayExceptionDemo {
   public static void main(String[] args) {
      int[] numbers = {10, 20, 30, 40, 50}; // Array size: 5

      try {
         // Valid access
         System.out.println("Element at 2: " + numbers[2]);

         // INVALID access - exception raise કરશે
         // Index 10 એ length 5 કરતા મોટો છે
         System.out.println("Element at 10: " + numbers[10]);

      } catch(ArrayIndexOutOfBoundsException e) {
         System.out.println("Exception caught: " + e.getMessage());
         System.out.println("Invalid index accessed!");
      }
   }
}
```

Exception raise કરતા Inputs:

- **નકારાત્મક Index**: numbers[-1]
- **Index >= Length**: numbers[5] (કેમ કે valid indices 0-4 છે)
- **Empty Array**: ખાલી array પર numbers[0]

મેમરી ટ્રીક

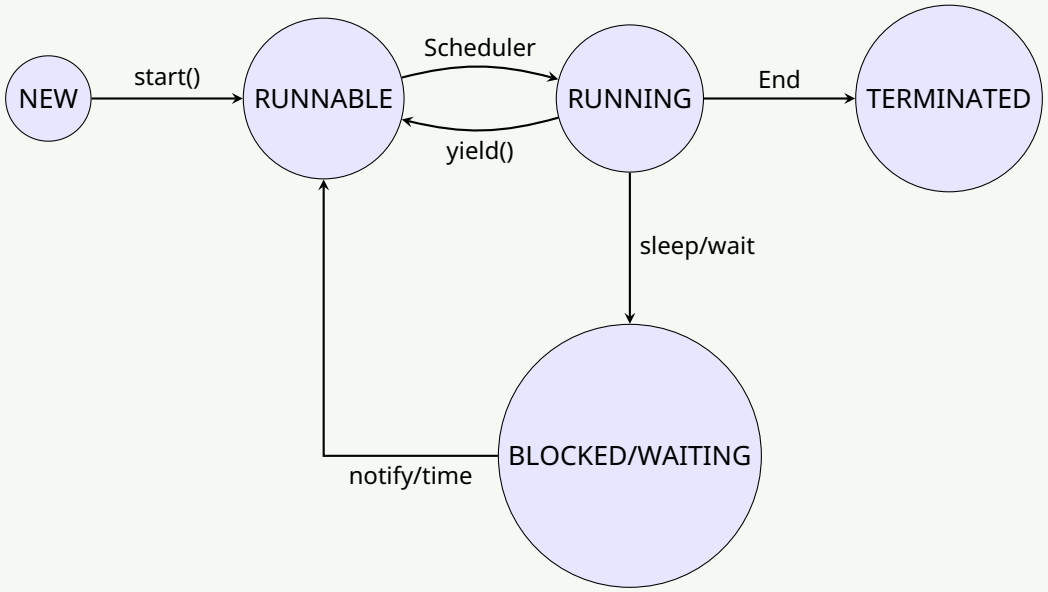
``Array Bounds Break Programs''

પ્રશ્ન 4(અ OR) [3 ગુણ]

JAVA માં Thread ના life cycle ને ઉદાહરણ સાથે દોરો અને સમજાવો.

જવાબ

Thread Life Cycle: Thread તેના જીવનકાળ દરમિયાન વિવિધ અવસ્થાઓમાંથી પસાર થાય છે.



આકૃતિ 1. Thread Life Cycle

અવસ્થાઓ (States):
- **NEW**: Thread બન્યો છે પણ શરૂ થયો નથી.
- **RUNNABLE**: Run કરવા માટે તૈયાર છે, CPU ની રાહ જોઈ રહ્યો છે.
- **RUNNING**: હાલમાં execute થઈ રહ્યો છે.
- **BLOCKED/WAITING**: Resource અથવા અન્ય thread માટે રાહ જોઈ રહ્યો છે.
- **TERMINATED**: Execution પૂર્ણ થયું છે.

મેમરી ટ્રીક

``New Runs, Blocks Wait, Terminates''

પ્રશ્ન 4(બ OR) [4 ગુણ]

JAVA Optional class સમજાવો. Optional class ની OfNullable() પદ્ધતિનું વર્ણન કરો.

જવાબ

Optional Class: Java 8 માં દાખલ કરાયેલ, java.util.Optional એ એક container object છે જે non-null value ધરાવી શકે છે અથવા ન પણ ધરાવી શકે. null નો ઉપયોગ કર્યા વિના value ની ગેરહાજરી રજૂ કરવા માટે તેનો ઉપયોગ થાય છે, જે NullPointerException ટાળવામાં મદદ કરે છે.
ofNullable() Method:
- **વર્ણન**: જો value non-null હોય, તો તે value સાથેનું Optional પરત કરે છે, અન્યથા ખાલી Optional પરત કરે છે.
- **Syntax**: static <T> Optional<T> ofNullable(T value)

Listing 13. Optional.ofNullable Demo

```
import java.util.Optional;

public class OptionalDemo {
    public static void main(String[] args) {
        String name1 = "John";
        String name2 = null;

        // "John" ધરાવતું Optional પરત કરે છે
        Optional<String> opt1 = Optional.ofNullable(name1);
```

```
10
11      // ખાલી Optional પરત કરે છે □□□□( Exception throw થતું નથી)
12      Optional<String> opt2 = Optional.ofNullable(name2);
13
14      System.out.println("opt1 હાજર: " + opt1.isPresent()); // true
15      System.out.println("opt2 હાજર: " + opt2.isPresent()); // false
16
17      // orElse સાથે ઉપયોગ
18      System.out.println(opt2.orElse("Default Name")); // prints Default Name
19    }
20  }
```

મેમરી ટ્રીક

``Optional Offers Null Safety''

પ્રશ્ન 4(ક OR) [7 ગુણ]

નેસ્ટેડ try...catch block દર્શાવતો કાર્યક્ષમ JAVA પ્રોગ્રામ લખો.

જવાબ

Listing 14. Nested Try-Catch

```
1  public class NestedTryCatchDemo {
2    public static void main(String[] args) {
3      try {
4        System.out.println("બાહ્ય try block શરૂ થયો");
5
6        try {
7          System.out.println("આંતરિક try block શરૂ થયો");
8          int[] numbers = {10, 20};
9
10         // Causes ArrayIndexOutOfBoundsException
11         System.out.println(numbers[5]);
12
13       } catch(ArrayIndexOutOfBoundsException e) {
14         System.out.println("આંતરિક catch: Array index error");
15         // Modifying exception flow - throwing runtime exception
16         throw new RuntimeException("આંતરિક block માં error");
17       }
18
19     } catch(RuntimeException e) {
20       System.out.println("બાહ્ય catch: Runtime error - " + e.getMessage());
21
22     } catch(Exception e) {
23       System.out.println("બાહ્ય catch: સામાન્ય error");
24
25     } finally {
26       System.out.println("બાહ્ય finally: Cleanup");
27     }
28   }
29 }
```

મુખ્ય લક્ષણો:

- **સ્તરીય Handling**: વિશિષ્ટ errors આંતરિક બ્લોકમાં, સામાન્ય errors બાહ્ય બ્લોકમાં handle થાય છે.
- **Flow**: જો આંતરિક catch તેને handle કરે, તો re-throw ન થાય ત્યાં સુધી બાહ્ય catch સુધી વાત પહોંચતી નથી.

પ્રશ્ન 5(અ) [3 ગુણ]

JAVA માં executable કોડ સાથે thread synchronization સમજાવો.

જવાબ

Thread Synchronization: Synchronization એ બહુવિધ threads દ્વારા shared resources ના access ને control કરવાની ક્ષમતા છે. તે thread interference અને memory consistency errors અટકાવે છે.

Listing 15. Thread Synchronization

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

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

   public int getCount() { return count; }
}

class SyncDemo extends Thread {
   Counter counter;

   SyncDemo(Counter c) { counter = c; }

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

ફાયદા:

- **Data Consistency**: જ્યારે બહુવિધ threads સમાન ડેટામાં ફેરફાર કરે ત્યારે આવશ્યક છે.
- **Thread Safety**: ખાતરી કરે છે કે એક સમયે માત્ર એક જ thread resource ને access કરે છે.

મેમરી ટ્રીક

``Synchronize Secures Shared Data''

પ્રશ્ન 5(બ) [4 ગુણ]

JAVA માં વિવિધ stream classes ની નોંધણી કરો. Executable ઉદાહરણ સાથે કોઈપણ એકને સમજાવો.

જવાબ

Stream Classes:

કોષ્ટક 7. Common Stream Classes

Class	હેતુ	પ્રકાર
FileInputStream	File માંથી bytes વાંચવા	Byte Stream
FileOutputStream	File માં bytes લખવા	Byte Stream
BufferedReader	Buffered character reading	Character Stream
PrintWriter	Formatted text output	Character Stream

Listing 16. FileInputStream Example

```
import java.io.*;

public class StreamDemo {
   public static void main(String[] args) {
      try {
         // Write data
         FileOutputStream fos = new FileOutputStream("test.txt");
         String data = "Hello World";
         fos.write(data.getBytes());
         fos.close();

         // Read data
         FileInputStream fis = new FileInputStream("test.txt");
         int ch;
         while((ch = fis.read()) != -1) {
            System.out.print((char)ch);
         }
         fis.close();

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

મેમરી ટ્રીક

``Streams Send Data''

પ્રશ્ન 5(ક) [7 ગુણ]

Thread નો ઉપયોગ કરીને આપેલ બે પૂર્ણાંક સંખ્યાઓ વચ્ચે વિષમ સંખ્યાઓ દર્શાવવા માટે Thread class ને વિસ્તારતો JAVA પ્રોગ્રામ લખો.

જવાબ

Listing 17. Odd Number Thread

```
class OddNumberThread extends Thread {
   private int start;
   private int end;

   public OddNumberThread(int start, int end) {
      this.start = start;
      this.end = end;
   }

   @Override
   public void run() {
      System.out.println("Thread started: " + getName());

      for(int i = start; i <= end; i++) {
         if(i % 2 != 0) {  // Check if odd
            System.out.println("Odd number: " + i);
            try {
```

```
                Thread.sleep(500);  // Pause
            } catch(InterruptedException e) {
                System.out.println("Thread interrupted");
            }
        }
    }
    System.out.println("Thread completed: " + getName());
  }
}

public class OddNumberDemo {
  public static void main(String[] args) {
    // Create threads
    OddNumberThread t1 = new OddNumberThread(1, 10);

    t1.setName("OddThread-1");
    t1.start();

    try {
      t1.join(); // Wait for completion
    } catch(InterruptedException e) {
      e.printStackTrace();
    }

    System.out.println("Main thread completed");
  }
}
```

આઉટપુટ:

```
Thread started: OddThread-1
Odd number: 1
Odd number: 3
...
Odd number: 9
Thread completed: OddThread-1
Main thread completed
```

મેમરી ટ્રીક

``Threads Take Turns''

પ્રશ્ન 5(અ OR) [3 ગુણ]

JAVA માં Thread class ની join() અને alive() પદ્ધતિઓ સમજાવો.

જવાબ

Thread Methods:

- **join()**: આ મેથડ એક thread ને બીજા thread ના completion ની રાહ જોવાની મંજૂરી આપે છે. જો t.join() કૉલ કરવામાં આવે, તો વર્તમાન thread જ્યાં સુધી thread t સમાપ્ત ન થાય ત્યાં સુધી execution અટકાવી દે છે.
- **isAlive()**: આ મેથડ ચકાસે છે કે thread હજુ ચાલી રહ્યો છે કે નહીં. જો thread શરૂ થયો હોય અને હજુ સુધી મૃત્યુ પામ્યો ન હોય તો તે true પરત કરે છે, અન્યથા false.

Listing 18. Join અને IsAlive

```
class TestThread extends Thread {
  public void run() {
    try { sleep(500); } catch(InterruptedException e) {}
```

```
    }
}

public class Main {
    public static void main(String[] args) throws InterruptedException {
        TestThread t = new TestThread();
        System.out.println("Before start: " + t.isAlive()); // false

        t.start();
        System.out.println("After start: " + t.isAlive()); // true

        t.join(); // Wait for completion
        System.out.println("After join: " + t.isAlive()); // false
    }
}
```

મેમરી ટ્રીક

``Join Waits, Alive Checks''

પ્રશ્ન 5(બ OR) [4 ગુણ]

JAVA માં user-defined exceptions ને વ્યાખ્યાયિત કરો. User-defined exceptions બતાવવા માટે પ્રોગ્રામ લખો

જવાબ

User-defined Exceptions: Java યુઝર્સને Exception ક્લાસ (checked) અથવા RuntimeException ક્લાસ (unchecked) ને extend કરીને તેમના પોતાના exception classes વ્યાખ્યાયિત કરવાની મંજૂરી આપે છે. તેનો ઉપયોગ application-specific logical errors ને handle કરવા માટે થાય છે.

Listing 19. Custom Exception

```
// 1. Create custom exception class
class AgeValidationException extends Exception {
    public AgeValidationException(String message) {
        super(message);
    }
}

class Person {
    public void setAge(int age) throws AgeValidationException {
        if(age < 0) {
            throw new AgeValidationException("Age cannot be negative: " + age);
        }
        System.out.println("Valid age: " + age);
    }
}

public class UserDefinedExceptionDemo {
    public static void main(String[] args) {
        Person p = new Person();
        try {
            p.setAge(-5); // Invalid
        } catch(AgeValidationException e) {
            System.out.println("Caught: " + e.getMessage());
        }
    }
}
```

મેમરી ટ્રીક

``Custom Exceptions Catch Specific Errors''

પ્રશ્ન 5(ક OR) [7 ગુણ]

ફાઇલ a.txt ની સામગ્રીને b.txt પર કોપી કરવા માટે JAVA પ્રોગ્રામ લખો.

જવાબ

Listing 20. File Copy Program

```
import java.io.*;

public class FileCopyDemo {
    public static void main(String[] args) {
        String source = "a.txt";
        String target = "b.txt";

        // Method: FileInputStream અને FileOutputStream નો ઉપયોગ કરીને
        copyFile(source, target);
    }

    public static void copyFile(String src, String dest) {
        FileInputStream fis = null;
        FileOutputStream fos = null;

        try {
            // Initialize streams
            fis = new FileInputStream(src);
            fos = new FileOutputStream(dest);

            // Read and write byte by byte
            int ch;
            while((ch = fis.read()) != -1) {
                fos.write(ch);
            }

            System.out.println("File copied successfully!");

        } catch(FileNotFoundException e) {
            System.out.println("File not found: " + e.getMessage());
        } catch(IOException e) {
            System.out.println("IO Error: " + e.getMessage());
        } finally {
            // Close streams in finally block
            try {
                if(fis != null) fis.close();
                if(fos != null) fos.close();
            } catch(IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Logic:
1. FileInputStream નો ઉપયોગ કરીને source file ને read mode માં ખોલો.
2. FileOutputStream નો ઉપયોગ કરીને destination file ને write mode માં ખોલો.
3. Source માંથી byte વાંચો અને જ્યાં સુધી file નો અંત (-1) ન આવે ત્યાં સુધી destination પર લખો.
4. બંને streams ને બંધ કરો જેથી system resources મુક્ત થાય.

મેમરી ટ્રીક

``Files Flow From Source To Target''