Object Oriented Programming With Java (4341602) - Summer 2024 Solution (Gujarati)

Milav Dabgar

June 13, 2024

પ્રશ્ન 1(a) [3 ગુણ]

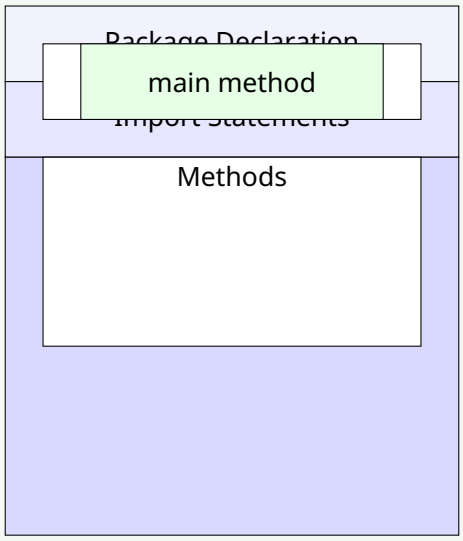
Explain the basic structure of Java program.

જવાબ

Basic Structure: Java પ્રોગ્રામમાં classes, methods અને statements ચોક્કસ શ્રેણીમાં ગોઠવાયેલા હોય છે.

કોષ્ટક 1. મૂળભૂત માળખું ઘટકો

ઘટક	વર્ણન
Package Declaration	વૈકલ્પિક, package સભ્યપદ દર્શાવે છે
Import Statements	જરૂરી classes/packages આયાત કરે છે
Class Declaration	મુખ્ય class વ્યાખ્યા, objects માટે blueprint
Main Method	પ્રવેશ બિંદુ public static void main(String[] args)



આકૃતિ 1. Java પ્રોગ્રામનું માળખું

- **Package**: સંબંધિત classes ને જૂથબદ્ધ કરે છે (જેમ કે package com.example;)
- **Import**: બાહ્ય classes ને access કરે છે (જેમ કે import java.util.*;)
- **Class**: બધો કોડ અહીં હોય છે
- **Main method**: પ્રોગ્રામ execution અહીંથી શરૂ થાય છે

મેમરી ટ્રીક

``PICM - Package, Import, Class, Main''

પ્રશ્ન 1(b) [4 ગુણ]

List out different features of java. Explain any two.

જવાબ

Java ના લક્ષણો:

1. **Platform Independent**: એક વાર લખો, બધે ચલાવો (WORA)
2. **Object Oriented**: બધું object છે
3. **Simple**: સરળ syntax, pointers નથી
4. **Secure**: built-in સુરક્ષા લક્ષણો, bytecode verification
5. **Robust**: મજબૂત memory management, exceotopm handling
6. **Multithreaded**: concurrent execution આધાર

1. Platform Independence: Java કોડ bytecode માં compile થાય છે, જે platform-neutral છે. આ bytecode ને JVM (Java Virtual Machine) interpret કરે છે, જે દરેક OS (Windows, Linux, Mac) માટે અલગ હોય છે.

2. Object Oriented: Java વાસ્તવિક દુનિયાની વસ્તુઓને objects અને classes દ્વારા model કરે છે. તે મુખ્ય OOP સિદ્ધાંતોને આધાર આપે છે:

- **Encapsulation**: ડેટા અને methods ને બાંધવું
- **Inheritance**: કોડ પુનઃઉપયોગ
- **Polymorphism**: એક જ interface, અલગ સ્વરૂપ

મેમરી ટ્રીક

``POSRMM - Platform, Object, Simple, Robust, Multithreaded, Memory''

પ્રશ્ન 1(c) [7 ગુણ]

Write a program in java to find out sum of the digits of entered number.

જવાબ

Listing 1. અંકોનો સરવાળો

```
public class DigitSum {
   public static void main(String[] args) {
      if (args.length == 0) {
         System.out.println("Please provide a number.");
         return;
      }

      int number = Integer.parseInt(args[0]);
      int sum = 0;
      int temp = Math.abs(number);

      while (temp > 0) {
         sum += temp % 10;
         temp /= 10;
      }

      System.out.println("Sum of digits: " + sum);
   }
```

```
}
```

કોષ્ટક 2. Algorithm Trace (ઇનપુટ: 123)

પગલું	ક્રિયા	પરિણામ
1	છેલ્લો અંક કાઢો	$123\%10 = 3$
2	સરવાળામાં ઉમેરો	$sum = 0 + 3 = 3$
3	છેલ્લો અંક હટાવો	$123/10 = 12$
4	અંક કાઢો	$12\%10 = 2$
5	સરવાળામાં ઉમેરો	$sum = 3 + 2 = 5$
6	અંક હટાવો	$12/10 = 1$
7	અંક કાઢો	$1\%10 = 1$
8	સરવાળામાં ઉમેરો	$sum = 5 + 1 = 6$

મેમરી ટ્રીક

``EARD - Extract, Add, Remove, Done''

પ્રશ્ન 1(c OR) [7 ગુણ]

Write a program in java to find out maximum from any ten numbers using command line argument.

જવાબ

Listing 2. મહત્તમ શોધો

```
public class FindMaximum {
   public static void main(String[] args) {
      if (args.length < 10) {
         System.out.println("Please enter 10 numbers");
         return;
      }

      int max = Integer.parseInt(args[0]);

      for (int i = 1; i < 10; i++) {
         int current = Integer.parseInt(args[i]);
         if (current > max) {
            max = current;
         }
      }

      System.out.println("Maximum number: " + max);
   }
}
```

કોષ્ટક 3. Logic Steps

પગલું	ક્રિયા
Validation	10 arguments છે કે કેમ તે તપાસો
Initialization	પ્રથમ સંખ્યાને max ધારો
Comparison	બાકીની સંખ્યાઓ સાથે સરખાવો
Update	જો current > max, તો max = current
Output	max છાપો

મેમરી ટ્રીક

``VCIU - Validate, Compare, Initialize, Update''

પ્રશ્ન 2(a) [3 ગુણ]

List out different concept of oop. Explain anyone in detail.

જવાબ

OOP ની વિભાવનાઓ:
- Encapsulation
- Inheritance
- Polymorphism
- Abstraction

Encapsulation: Encapsulation એ ડેટા અને methods ને એક unit (class) માં જોડવાની પદ્ધતિ છે. તે ડેટાને બહારની દખલગીરીથી સુરક્ષિત કરે છે.
- **Data Hiding**: ડેટાને private જાહેર કરીને સીધો access રોકવામાં આવે છે.
- **Accessors/Mutators**: ડેટા access કરવા માટે public getter અને setter methods આપવામાં આવે છે.
- **ફાયદા**: સુરક્ષા, જાળવણી અને લવચીકતા વધારે છે.

મેમરી ટ્રીક

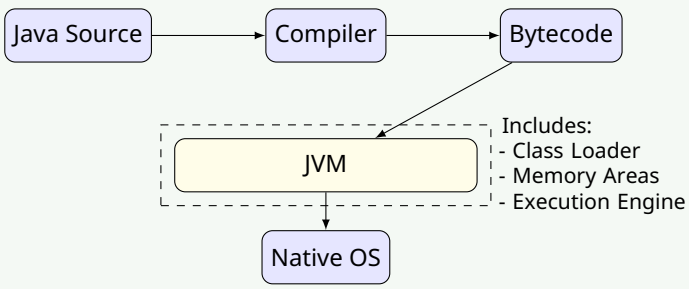
``EIPA - Encapsulation, Inheritance, Polymorphism, Abstraction''

પ્રશ્ન 2(b) [4 ગુણ]

Explain JVM in detail.

જવાબ

JVM (Java Virtual Machine) એ engine છે જે Java bytecode execute કરે છે.



આકૃતિ 2. JVM આર્કિટેક્ચર

કોષ્ટક 4. JVM ઘટકો

ઘટક	કાર્ય
Class Loader	.class files ને memory માં લોડ કરે છે
Memory Areas	મેમરી ફાળવે છે (Heap, Stack, Method Area)
Execution Engine	Bytecode interpret કરે છે અથવા JIT compiler વાપરે છે
JIT Compiler	ઝડપ માટે કોડને native machine code માં બદલે છે

મુખ્ય લક્ષણો:
- **Platform Independence**: JVM Java ને portable બનાવે છે.
- **Memory Management**: મેમરી ફાળવણી અને garbage collection સંભાળે છે.

મેમરી ટ્રીક

``CEMJ - Class loader, Execution, Memory, JIT''

પ્રશ્ન 2(c) [7 ગુણ]

Explain constructor overloading with example.

જવાબ

Constructor Overloading: એક જ class માં અલગ parameters યાદી સાથે એકથી વધુ constructors હોવાની પદ્ધતિ.

Listing 3. Constructor Overloading

```
public class Student {
    private String name;
    private int age;
    private String course;

    // 1. Default constructor
    public Student() {
        this.name = "Unknown";
        this.age = 0;
        this.course = "Not Assigned";
    }

    // 2. Constructor with one parameter
    public Student(String name) {
        this.name = name;
        this.age = 0;
        this.course = "Not Assigned";
    }

    // 3. Constructor with two parameters
    public Student(String name, int age) {
        this.name = name;
        this.age = age;
        this.course = "Not Assigned";
    }

    // 4. Constructor with all parameters
    public Student(String name, int age, String course) {
        this.name = name;
        this.age = age;
        this.course = course;
    }
```

```
}
```

કોષ્ટક 5. Constructor ના પ્રકારો

Constructor	Parameters	શૈલી
Default	કોઈ નહિ	મૂળભૂત કિંમતો સાથે initialization
Single Param	નામ	માત્ર નામ સાથે નોંધણી
Two Param	નામ, ઉંમર	મૂળભૂત વિગતો સાથે
Full Param	બધા fields	સંપૂર્ણ initialization

નિયમો:
- Class નામ જેવું જ નામ હોવું જોઈએ.
- Parameters યાદી અલગ હોવી જોઈએ (સંખ્યા અથવા પ્રકાર).
- Compile-time પર નક્કી થાય છે (Compile-time Polymorphism).

મેમરી ટ્રીક

``SNDF - Same Name, Different Parameters, Flexible''

પ્રશ્ન 2(a OR) [3 ગુણ]

What is wrapper class? Explain with example.

જવાબ

Wrapper Class: એક class જે primitive data type ને object માં લપેટે છે.

કોષ્ટક 6. Primitive vs Wrapper

Primitive	Wrapper Class
byte	Byte
int	Integer
char	Character
double	Double
boolean	Boolean

Listing 4. Boxing and Unboxing

```
// Boxing: Primitive to Object
int num = 10;
Integer obj = Integer.valueOf(num); // Manual
Integer autoBox = 10;            // Autoboxing

// Unboxing: Object to Primitive
Integer wrapper = new Integer(20);
int value = wrapper.intValue();    // Manual
int autoUnbox = wrapper;          // Auto-unboxing
```

ઉપયોગ: Collection Frameworks (ArrayList, Vector) માં જરૂરી છે જ્યાં માત્ર objects જ સંગ્રહિત થાય છે.

મેમરી ટ્રીક

``BUC - Boxing, Unboxing, Collections''

પ્રશ્ન 2(b OR) [4 ગુણ]

Explain static keyword with example.

જવાબ

Static Keyword: દર્શાવે છે કે સભ્ય class નો છે, instance નો નહિ.

કોષ્ટક 7. Static ઉપયોગો

સભ્ય	વર્તન
Variable	બધા instances વચ્ચે shared memory (class variable)
Method	Object બનાવ્યા વિના કૉલ કરી શકાય
Block	Class લોડ થાય ત્યારે એક વાર execute થાય

Listing 5. Static Example

```
public class Counter {
   static int count = 0;  // Shared variable
   int id;              // Instance variable

   public Counter() {
      count++;          // દરેક નવા object માટે વધારો
      this.id = count;
   }

   public static void showCount() {
      // માત્ર static ડેટા access કરી શકે
      System.out.println("Total objects: " + count);
   }
}
```

- **Memory Efficiency**: Variable માત્ર એક જ વાર બને છે.
- **Utility**: Utility methods માટે વપરાય છે (જેમ કે Math.sqrt()).

મેમરી ટ્રીક

``SCMA - Shared, Class-level, Memory, Access''

પ્રશ્ન 2(c OR) [7 ગુણ]

What is constructor? Explain copy constructor with example.

જવાબ

Constructor: Object ને initialize કરવા માટે વપરાતો ખાસ કોડ બ્લોક.
Copy Constructor: જે object ને એ જ class ના બીજા object નો ઉપયોગ કરીને initialize કરે છે. તે object ની નકલ (clone) બનાવે છે.

Listing 6. Copy Constructor

```java
public class Book {
    private String title;
    private String author;

    // Parameterized constructor
    public Book(String title, String author) {
        this.title = title;
        this.author = author;
    }

    // Copy constructor
    public Book(Book other) {
        this.title = other.title;
        this.author = other.author;
    }
}

// Usage
class Main {
    public static void main(String[] args) {
        Book b1 = new Book("Java Guide", "James");
        Book b2 = new Book(b1); // b1 ની નકલ બનાવે છે
    }
}
```

કોષ્ટક 8. Constructor ગુણધર્મો

ગુણધર્મ	વર્ણન
નામ	Class નામ જેવું જ હોવું જોઈએ
Return Type	કોઈ નહિ (void પણ નહિ)
Invocation	new સમયે આપોઆપ

મેમરી ટ્રીક

``SNAC - Same Name, Automatic Call''

પ્રશ્ન 3(a) [3 ગુણ]

Explain any four-string function in java with example.

જવાબ

String Functions:

કોષ્ટક 9. સામાન્ય String Functions

Function	હેતુ	ઉદાહરણ
length()	અક્ષરોની સંખ્યા આપે છે	"Hi".length() → 2
charAt(i)	i index પરનો અક્ષર આપે છે	"Hi".charAt(0) → 'H'
substring(i)	i થી સ્ટ્રિંગનો ભાગ આપે છે	"Code".substring(2) → "de"
toUpperCase()	અક્ષરોને કેપિટલમાં ફેરવે છે	"java".toUpperCase() → "JAVA"

Listing 7. String Example

```java
String str = "Java Programming";
```

```
int len = str.length();          // 16
char ch = str.charAt(0);         // 'J'
String sub = str.substring(5);   // "Programming"
String upper = str.toUpperCase(); // "JAVA PROGRAMMING"
```

મેમરી ટ્રીક

``LCST - Length, Character, Substring, Transform''

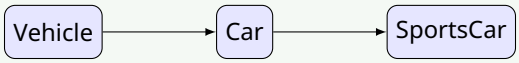
પ્રશ્ન 3(b) [4 ગુણ]

List out different types of inheritance. Explain multilevel inheritance.

જવાબ

Inheritance ના પ્રકારો:
1. Single Inheritance
2. Multilevel Inheritance
3. Hierarchical Inheritance
4. Multiple Inheritance (Interfaces દ્વારા)
5. Hybrid Inheritance

Multilevel Inheritance: એક પદ્ધતિ જ્યાં class ડેરિવેડ class માંથી inherit થાય છે, વારસાની સાંકળ રચે છે.



આકૃતિ 3. Multilevel Inheritance

Listing 8. Multilevel Inheritance

```
class Vehicle { void start() {} }
class Car extends Vehicle { void drive() {} }
class SportsCar extends Car { void race() {} }
```

આ ઉદાહરણમાં, SportsCar Car અને Vehicle બંનેના લક્ષણો વારસામાં મેળવે છે.

મેમરી ટ્રીક

``SMHM - Single, Multilevel, Hierarchical, Multiple''

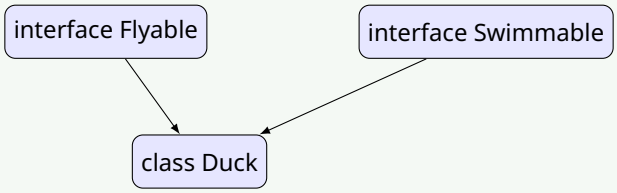
પ્રશ્ન 3(c) [7 ગુણ]

What is interface? Explain multiple inheritance with example.

જવાબ

Interface: Java માં એક અમૂર્ત સંદર્ભ પ્રકાર છે જે class જેવો જ છે પરંતુ તેમાં માત્ર constants, method signatures (ખાલી methods), default methods અને static methods હોય છે. તે સંપૂર્ણ અમૂર્તતા અને multiple inheritance પ્રાપ્ત કરવા માટે વપરાય છે.

Multiple Inheritance: Ambiguity (Diamond Problem) ને ટાળવા માટે Java classes સાથે multiple inheritance ડાયરેક્ટ સપોર્ટ કરતું નથી, પરંતુ તે interfaces દ્વારા સપોર્ટ કરે છે. એક class એકથી વધુ interfaces implement કરી શકે છે.



આકૃતિ 4. Multiple Inheritance (Interfaces)

Listing 9. Multiple Inheritance Example

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

// એક class અનેક interfaces implement કરે છે
class Duck implements Flyable, Swimmable {
   public void fly() {
      System.out.println("Duck is flying");
   }
   public void swim() {
      System.out.println("Duck is swimming");
   }
}
```

કોષ્ટક 10. Interface vs Class

Interface	Class
માત્ર અમૂર્ત (abstract) methods હોઈ શકે	કોંક્રિટ (concrete) methods હોય છે
Variables public static final હોય છે	કોઈ પણ પ્રકારના variable હોઈ શકે
Multiple inheritance સપોર્ટ કરે છે	Single inheritance સપોર્ટ કરે છે

મેમરી ટ્રીક

``CMDS - Contract, Multiple, Diamond-solution''

પ્રશ્ન 3(a OR) [3 ગુણ]

Explain this keyword with example.

જવાબ

'this' Keyword: Java માં એક સંદર્ભ ચલ જે વર્તમાન object નો સંદર્ભ આપે છે.

કોષ્ટક 11. 'this' ના ઉપયોગો

કેસ	હેતુ
Instance Variable	Parameter થી field ને અલગ પાડવા (this.x = x)
Method Call	વર્તમાન class method ને બોલાવવા (this.method())
Constructor Call	Constructors ને સાંકળવા (this())
Return Object	વર્તમાન instance પાછું આપવા (return this)

Listing 10. Using 'this'

```
public class Person {
   String name;

   public Person(String name) {
      this.name = name; // Ambiguity ઉકેલે છે
   }

   public Person getInstance() {
      return this; // વર્તમાન object રિટર્ન કરે છે
   }
}
```

મેમરી ટ્રીક

``CRPM - Current, Resolve, Parameter, Method''

પ્રશ્ન 3(b OR) [4 ગુણ]

Explain method overriding with example.

જવાબ

Method Overriding: જ્યારે પેટા-વર્ગ (subclass) પિતૃ-વર્ગ (parent class) માં પહેલેથી જ વ્યાખ્યાયિત પદ્ધતિ માટે ચોક્કસ અમલીકરણ પ્રદાન કરે છે ત્યારે તે થાય છે. તે Runtime Polymorphism માટે વપરાય છે.

Listing 11. Method Overriding

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

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

class Main {
   public static void main(String[] args) {
      Animal a = new Dog(); // Upcasting
      a.makeSound(); // Output: Dog barks
   }
}
```

કોષ્ટક 12. Overriding ના નિયમો

નિયમ	વર્ણન
Signature	Method નામ અને args સમાન હોવા જોઈએ
Inheritance	IS-A સંબંધ હોવો જોઈએ
Access	Access level વધુ પ્રતિબંધિત હોઈ શકતું નથી
Binding	Runtime પર નક્કી થાય છે (Dynamic Binding)

મેમરી ટ્રીક

``SSRD - Same Signature, Runtime Decision''

પ્રશ્ન 3(c OR) [7 ગુણ]

What is package? Write steps to create a package and give example of it.

જવાબ

Package: એક namespace જે સંબંધિત classes અને interfaces ને સંગઠિત કરે છે. તે મદદ કરે છે:

- નામકરણ તકરાર (naming conflicts) અટકાવવા.
- ઍક્સેસ નિયંત્રિત કરવા (protected/default access).
- Classes શોધવા/ઉપયોગ કરવાનું સરળ બનાવવા.

Package બનાવવા અને વાપરવાના પગલાં:

1. **Directory**: પેકેજ નામ સાથે મેળ ખાતું ફોલ્ડર માળખું બનાવો (દા.ત., com/utils).
2. **Declaration**: ફાઇલની ટોચ પર package com.utils; ઉમેરો.
3. **Compile**: ફોલ્ડર્સ આપમેળે જનરેટ કરવા માટે -d . સાથે કમ્પાઇલ કરો.
4. **Import**: બીજી ફાઇલમાં import com.utils.*; વાપરો.

Listing 12. Package બનાવવું

```
// File: src/com/company/utils/MathUtils.java
package com.company.utils;

public class MathUtils {
    public static int add(int a, int b) {
        return a + b;
    }
}
```

Listing 13. Package વાપરવું

```
// File: src/Calculator.java
import com.company.utils.MathUtils;

public class Calculator {
    public static void main(String[] args) {
        int result = MathUtils.add(5, 10);
        System.out.println("Result: " + result);
    }
}
```

કોષ્ટક 13. Compiling અને Running

ક્રિયા	આદેશ
Compile Package	javac -d . MathUtils.java
Compile Main	javac Calculator.java
Run	java Calculator

મેમરી ટ્રીક

``ONAM - Organization, Namespace, Access, Maintenance''

પ્રશ્ન 4(a) [3 ગુણ]

Explain thread priorities with suitable example.

જવાબ

Thread Priority: Java threads માં 1 થી 10 સુધીની પ્રાધાન્યતા (priority) કિંમતો હોય છે જે Thread Scheduler ને કયું થ્રેડ એક્ઝિક્યુટ કરવું તે નક્કી કરવામાં મદદ કરે છે.

કોષ્ટક 14. Priority Constants

સ્તર	Constant	કિંમત
Min	Thread.MIN_PRIORITY	1
Norm	Thread.NORM_PRIORITY	5 (Default)
Max	Thread.MAX_PRIORITY	10

Listing 14. Thread Priority

```
class MyThread extends Thread {
   public void run() {
      System.out.println("Running: " + getName());
   }
}

public class PriorityDemo {
   public static void main(String[] args) {
      MyThread t1 = new MyThread();
      MyThread t2 = new MyThread();

      t1.setPriority(Thread.MAX_PRIORITY); // 10
      t2.setPriority(Thread.MIN_PRIORITY); // 1

      t1.start();
      t2.start();
   }
}
```

મેમરી ટ્રીક

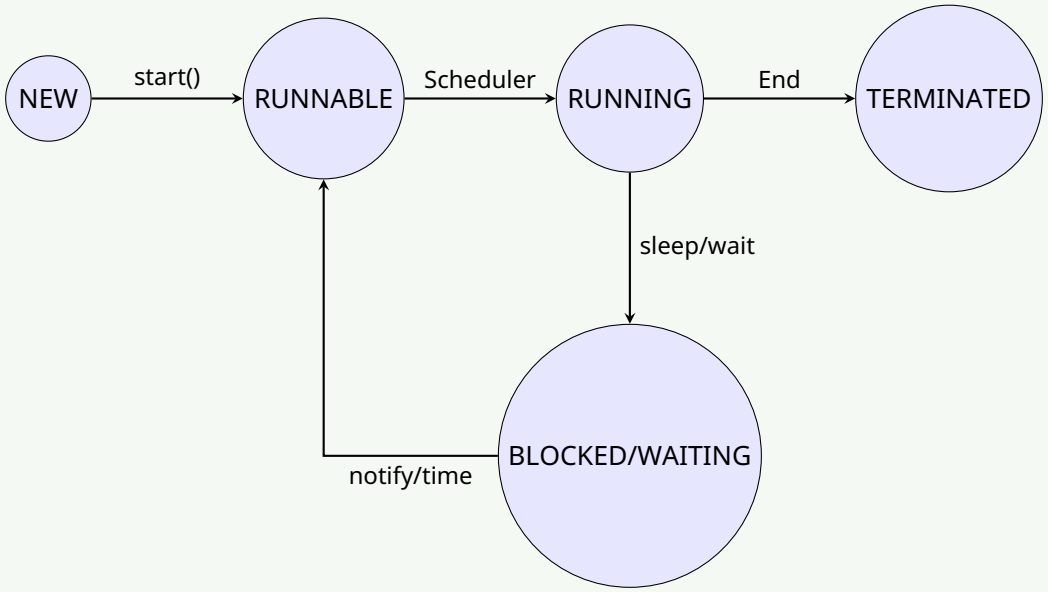
``HNG - Higher priority, Not Guaranteed''

પ્રશ્ન 4(b) [4 ગુણ]

What is Thread? Explain Thread life cycle.

જવાબ

Thread: એક હળવા વજનની subprocess જે અન્ય threads સાથે સમાંતર (concurrently) ચાલે છે.
Thread Life Cycle: થ્રેડ તેના જીવનકાળ દરમિયાન વિવિધ અવસ્થાઓમાંથી પસાર થાય છે.



આકૃતિ 5. Thread Life Cycle

કોષ્ટક 15. Thread અવસ્થાઓ

અવસ્થા	વર્ણન
NEW	Instance બન્યું છે, start() કૉલ નથી થયું
RUNNABLE	ચાલવા માટે તૈયાર, CPU ની રાહ જુએ છે
RUNNING	હાલમાં એક્ઝિક્યુટ થઈ રહ્યું છે
BLOCKED	સંસાધન માટે રાહ જુએ છે
TERMINATED	એક્ઝિક્યુશન પૂર્ણ થયું

મેમરી ટ્રીક

``NRBT - New, Runnable, Blocked, Terminated''

પ્રશ્ન 4(c) [7 ગુણ]

Write a program in java that create the multiple threads by implementing the Thread class.

જવાબ

Listing 15. Multiple Threads

```
class NumberPrinter extends Thread {
   String name;

   NumberPrinter(String name) {
      this.name = name;
   }

   public void run() {
      for(int i=1; i<=3; i++) {
         System.out.println(name + ": " + i);
         try {
            Thread.sleep(500); // Pause
```

```
            } catch(Exception e) {}
        }
    }
}

public class MultiThreadDemo {
    public static void main(String[] args) {
        NumberPrinter t1 = new NumberPrinter("Thread-1");
        NumberPrinter t2 = new NumberPrinter("Thread-2");
        NumberPrinter t3 = new NumberPrinter("Thread-3");

        t1.start();
        t2.start();
        t3.start();

        System.out.println("Main finished");
    }
}
```

પગલાં:
1. Thread class ને extend કરો.
2. Logic સાથે run() method override કરો.
3. Class ના instances બનાવો.
4. એક્ઝિક્યુશન શરૂ કરવા માટે start() કૉલ કરો.

મેમરી ટ્રીક

``EOCS - Extend, Override, Create, Start''

પ્રશ્ન 4(a OR) [3 ગુણ]

Explain basic concept of Exception Handling.

જવાબ

Exception Handling: Runtime errors સંભાળવા માટેની પદ્ધતિ જેથી એપ્લિકેશનનો સામાન્ય પ્રવાહ જળવાઈ રહે.

કોષ્ટક 16. મુખ્ય શબ્દો

Keyword	કાર્ય
try	શંકાસ્પદ કોડનો બ્લોક
catch	Exception સંભાળતો બ્લોક
finally	પરિણામ ગમે તે હોય, એક્ઝિક્યુટ થાય
throw	સ્પષ્ટ રીતે exception ફેંકવા માટે
throws	મેથડ કયા exceptions ફેંકી શકે છે તે જાહેર કરે છે

Listing 16. Exception Syntax

```
try {
    // Risky code
} catch (Exception e) {
    // Handling code
} finally {
    // Cleanup
}
```

મેમરી ટ્રીક

``TRCF - Try, Runtime error, Catch, Finally''

પ્રશ્ન 4(b OR) [4 ગુણ]

Explain multiple catch with suitable example.

જવાબ

Multiple Catch Blocks: એક try બ્લોક પછી અનેક catch બ્લોક હોઈ શકે છે જેથી વિવિધ પ્રકારના exceptions અલગ રીતે સંભાળી શકાય.

Listing 17. Multiple Catch

```
public class MultiCatch {
   public static void main(String[] args) {
      try {
         int a[] = new int[5];
         a[10] = 30 / 0; // Risky code
      } catch (ArithmeticException e) {
         System.out.println("Math Error: " + e);
      } catch (ArrayIndexOutOfBoundsException e) {
         System.out.println("Array Error: " + e);
      } catch (Exception e) {
         // Generic catch છેલ્લે હોવો જોઈએ
         System.out.println("General Error: " + e);
      }
   }
}
```

નિયમો:
- એક સમયે માત્ર એક જ exception થાય છે અને માત્ર એક જ catch બ્લોક એક્ઝિક્યુટ થાય છે.
- ચોક્કસ exceptions સામાન્ય exceptions (Exception parent class) પહેલાં પકડવા જોઈએ.

મેમરી ટ્રીક

``SOOF - Specific first, One executes, Order matters, Finally''

પ્રશ્ન 4(c OR) [7 ગુણ]

What is Exception? Write a program that show the use of Arithmetic Exception.

જવાબ

Exception: પ્રોગ્રામના એક્ઝિક્યુશન દરમિયાન (runtime) થતી અનિચ્છનીય અથવા અણધારી ઘટના જે સૂચનાઓના સામાન્ય પ્રવાહને વિક્ષેપિત કરે છે.
ArithmeticException: જ્યારે અસાધારણ અંકગણિત સ્થિતિ થાય છે, જેમ કે શૂન્ય વડે ભાગાકાર (division by zero), ત્યારે ફેંકવામાં આવતું runtime exception.

Listing 18. Arithmetic Exception Demo

```
public class DivisionDemo {
   public static void main(String[] args) {
      try {
         int numerator = 100;
         int denominator = 0; // Division by zero
```

```
6
7           // This line throws ArithmeticException
8           int result = numerator / denominator;
9
10          System.out.println("Result: " + result);
11      } catch (ArithmeticException e) {
12          System.out.println("Error detected: Division by zero is not allowed.");
13          System.out.println("Exception: " + e.getMessage());
14      } finally {
15          System.out.println("Cleanup actions...");
16      }
17    }
18  }
```

Try-catch બ્લોક વગર, પ્રોગ્રામ ભાગાકારના સમયે જ ક્રેશ થઈ જશે.

મેમરી ટ્રીક

``DZMI - Division by Zero, Mathematical Invalid''

પ્રશ્ન 5(a) [3 ગુણ]

Explain ArrayIndexOutOfBound Exception in Java with example.

જવાબ

ArrayIndexOutOfBoundsException: જ્યારે એરે (array) ને ગેરકાયદેસર index સાથે access કરવામાં આવે ત્યારે ફેંકવામાં આવતું runtime exception. Index નેગેટિવ હોઈ શકે અથવા એરેના કદ કરતાં મોટો કે બરાબર હોઈ શકે.

કોષ્ટક 17. કારણો

કારણ	વર્ણન	ઉદાહરણ
Negative Index	Index < 0	a[-1]
Size Exceeded	Index $\geq$ Length	a[length]
Empty Array	ખાલી એરેનો index 0	a[0]

Listing 19. Array Index Exception

```
int[] numbers = {10, 20, 30}; // Size 3, indices 0-2
try {
  System.out.println(numbers[5]); // Index 5 અમાન્ય છે
} catch (ArrayIndexOutOfBoundsException e) {
  System.out.println("Invalid Index: " + e.getMessage());
}
```

મેમરી ટ્રીક

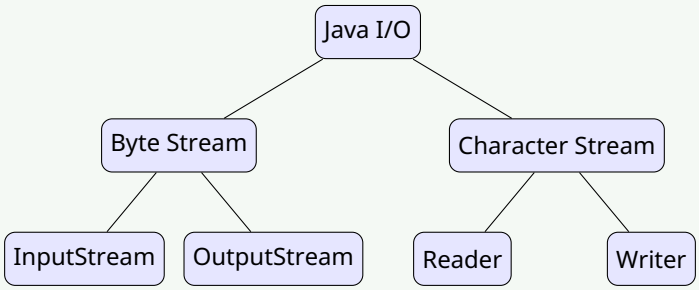
``NIE - Negative, Index-exceed, Empty''

પ્રશ્ન 5(b) [4 ગુણ]

Explain basics of stream classes.

જવાબ

Stream Classes: Stream એ ડેટાનો ક્રમ છે. Java I/O streams પર આધારિત છે.



આકૃતિ 6. Stream Hierarchy

કોષ્ટક 18. Stream શ્રેણીઓ

Stream Type	Data Type	Classes
Byte Stream	Binary Data (Images, etc)	InputStream, OutputStream
Char Stream	Text Data (Strings)	Reader, Writer

Subclasses:
- **File**: FileInputStream, FileWriter, વગેરે.
- **Buffered**: BufferedReader, BufferedOutputStream (કાર્યક્ષમતા માટે).

મેમરી ટ્રીક

``BCIF - Byte, Character, Input/Output, File''

પ્રશ્ન 5(c) [7 ગુણ]

Write a java program to create a text file and perform write operation on the text file.

જવાબ

Listing 20. Write to File

```java
import java.io.FileWriter;
import java.io.IOException;

public class FileWriteDemo {
    public static void main(String[] args) {
        String data = "Welcome to Summer 2024 Solution.";

        // Try-with-resources આપમેળે writer બંધ કરે છે
        try (FileWriter writer = new FileWriter("output.txt")) {

            writer.write(data);

            System.out.println("Successfully wrote to the file.");

        } catch (IOException e) {
            System.out.println("An error occurred.");
            e.printStackTrace();
        }
    }
```

```
}
```

સમજૂતી:
1. **Import**: java.io.FileWriter અને IOException.
2. **FileWriter**: File writer object બનાવે છે.
3. **write()**: સ્ટ્રિંગ ડેટા ફાઇલમાં લખે છે.
4. **close()**: આપમેળે થાય છે, ડેટા સેવ કરે છે અને સંસાધનો મુક્ત કરે છે.

કોષ્ટક 19. વપરાયેલ Methods

Method	વર્ણન
FileWriter(String)	નવી ફાઇલ બનાવે છે (overwrite કરે છે)
write(String)	ટેક્સ્ટ લખે છે
close()	Stream બંધ કરે છે

મેમરી ટ્રીક

``CWCH - Create, Write, Close, Handle''

પ્રશ્ન 5(a OR) [3 ગુણ]

Explain Divide by Zero Exception in Java with example.

જવાબ

Divide by Zero: પૂર્ણાંક (integer) ને શૂન્ય વડે ભાગવાનો પ્રયાસ કરવો એ Java માં ગેરકાયદેસર ક્રિયા છે.

કોષ્ટક 20. વર્તન

Case	પરિણામ	Exception
Integer / 0	અમાન્ય	ArithmeticException
Float / 0.0	Infinity	None
Modulo % 0	અમાન્ય	ArithmeticException

Listing 21. Divide By Zero

```
try {
   int a = 10 / 0; // Throws Exception
} catch (ArithmeticException e) {
   System.out.println("Cannot divide integer by zero");
}

double b = 10.0 / 0.0;
System.out.println(b); // Prints "Infinity"
```

મેમરી ટ્રીક

``IFM - Integer exception, Float infinity, Modulo error''

પ્રશ્ન 5(b OR) [4 ગુણ]

Explain try and catch block with example.

જવાબ

Try-Catch: Exception handling માટેની મુખ્ય પદ્ધતિ.
- **try**: કોડ જે exception ઉત્પન્ન કરી શકે છે તેને ઘેરે છે.
- **catch**: Exception ને કેવી રીતે સંભાળવું તે નક્કી કરે છે.

Listing 22. Try-Catch Example

```
public class Example {
   public static void main(String args[]) {
      try {
         String s = null;
         System.out.println(s.length()); // NullPointer
      } catch(NullPointerException e) {
         System.out.println("Caught Null Pointer Exception");
      }

      System.out.println("Rest of the code...");
   }
}
```

કોષ્ટક 21. પ્રવાહ

પરિસ્થિતિ	Execution પથ
No Exception	try ચાલે છે -> catch છોડવામાં આવે છે
Exception	try અટકે છે -> catch ચાલે છે -> આગળ વધે છે

મેમરી ટ્રીક

``TCF - Try risky, Catch exception, Finally cleanup''

પ્રશ્ન 5(c OR) [7 ગુણ]

Write a java program to display the content of a text file and perform append operation on the text file.

જવાબ

Listing 23. Read and Append

```
import java.io.*;

public class FileReadAppend {
   public static void main(String[] args) {
      String filename = "log.txt";

      // 1. Append કરવા
      try (FileWriter fw = new FileWriter(filename, true);
           BufferedWriter bw = new BufferedWriter(fw)) {

         bw.write("New Log Entry\n");
         System.out.println("Appended to file.");

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

```

```
        // 2. Read ફરીથી
        System.out.println("--- Reading File ---");
        try (FileReader fr = new FileReader(filename);
             BufferedReader br = new BufferedReader(fr)) {

            String line;
            while ((line = br.readLine()) != null) {
                System.out.println(line);
            }

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

મુખ્ય ખ્યાલો:
- **Append Mode**: new FileWriter(file, true) ફાઇલને append mode માં ખોલે છે.
- **BufferedReader**: લાઇન બાય લાઇન વાંચન માટે કાર્યક્ષમ છે.

મેમરી ટ્રીક

``CDADS - Create, Display, Append, Display, Statistics''