

Java Programming (4343203) Winter-2024 Solutions - Gujarati

પ્રશ્ન 1(a): Java ના વિવિધ પ્રકારના Primitive data typeની યાદી આપો. (ગુણ: 03)

Java Primitive Data Types:

- **byte**: 8-bit integer (-128 થી 127)
- **short**: 16-bit integer (-32,768 થી 32,767)
- **int**: 32-bit integer (-2^{31} થી $2^{31}-1$)
- **long**: 64-bit integer (-2^{63} થી $2^{63}-1$)
- **float**: 32-bit floating-point નંબર
- **double**: 64-bit floating-point નંબર
- **char**: 16-bit Unicode કેરેક્ટર
- **boolean**: true અથવા false વેલ્યુ

આ રીતે યાદ રાખો "CHILDREN":

- Char, Healthy boolean, Integer, Long, Double, Robust float, Efficient byte, Nifty short

પ્રશ્ન 1(b): યોગ્ય ઉદાહરણ સાથે Java Programનું સ્ટ્રક્ચર સમજાવો. (ગુણ: 04)

Java પ્રોગ્રામનું સ્ટ્રક્ચર:

```
// 1. પેકેજ ડેક્લેરેશન
package myprogram;


// 2. ઇમ્પોર્ટ સ્ટેટમેન્ટ્સ
import java.util.Scanner;

// 3. ક્લાસ ડેક્લેરેશન
public class HelloWorld {
    // 4. મેઇન મેથડ - એન્ટ્રી પોઇન્ટ
    public static void main(String[] args) {
        // 5. વેરીએબલ્સ અને સ્ટેટમેન્ટ્સ
        System.out.println("Hello World!");
    }

    // 6. વધારાની મેથડ્સ
    public void displayMessage() {
        System.out.println("સ્વાગત છે!");
    }
}
```

મુખ્ય ઘટકો:

- પેકેજ ડેક્લેરેશન: સંબંધિત ક્લાસને ગ્રૂપ કરે છે
- ઇમ્પોર્ટ સ્ટેટમેન્ટ્સ: બાહ્ય ક્લાસને એક્સેસ કરે છે
- ક્લાસ ડેક્લેરેશન: ઓબ્જેક્ટ્સનું બ્લૂપ્રિન્ટ
- મેઈન મેથડ: અહીંથી પ્રોગ્રામ એક્ઝિક્યુશન શરૂ થાય છે
- વેરીએબલ્સ/સ્ટેટમેન્ટ્સ: પ્રોગ્રામ લોજિક
- વધારાની મેથડ્સ: રીયુઝેબલ કોડ બ્લોક્સ

 આ રીતે યાદ રાખો "PICMVA":

- **P**ackage, **I**mports, **C**lass, **M**ain method, **V**ariables/statements, **A**dditional methods

પ્રશ્ન 1(c): Java ની arithmetic operatorsની યાદી આપો. કોઈ પણ ત્રણ arithmetic operatorsનો ઉપયોગ કરીને Java Program વિકસાવો અને તેનું output બતાવો. (ગુણ: 07)

Java માં Arithmetic Operators:

- + : સરવાળો (એડિશન)
- - : બાદબાકી (સબ્ટ્રેક્શન)
- * : ગુણાકાર (મલ્ટિપ્લિકેશન)
- / : ભાગાકાર (ડિવિઝન)
- % : મોડ્યુલસ (બાકી)
- ++ : ઇન્ક્રિમેન્ટ
- -- : ડિક્રિમેન્ટ

```
public class ArithmeticDemo {
    public static void main(String[] args) {
        // વેરીએબલ ડેક્લેરેશન અને ઇનિશિયલાઇઝેશન
        int a = 10, b = 3;

        // ત્રણ એરિથમેટિક ઓપરેટર્સનો ઉપયોગ
        int sum = a + b;      // સરવાળો
        int product = a * b;  // ગુણાકાર
        int remainder = a % b; // મોડ્યુલસ

        // પરિણામ દર્શાવો
        System.out.println("વેલ્યુ: a = " + a + ", b = " + b);
        System.out.println("સરવાળો: " + a + " + " + b + " = " + sum);
        System.out.println("ગુણાકાર: " + a + " * " + b + " = " + product);
        System.out.println("બાકી: " + a + " % " + b + " = " + remainder);
    }
}
```

આઉટપુટ:

```
વેલ્યુ: a = 10, b = 3
સરવાળો: 10 + 3 = 13
ગુણાકાર: 10 * 3 = 30
બાકી: 10 % 3 = 1
```

📝 એરિથમેટિક ઓપરેટર્સને આ રીતે યાદ રાખો "ASMDIP":

- Addition, Subtraction, Multiplication, Division, Increment, Percentage (modulus)

પ્રશ્ન 1(c OR): Javaમાં for લૂપ માટેની સિન્ટેક્સ લખો. ૧ થી ૧૦ વચ્ચે આવતા પ્રાઈમ નંબર શોધવા માટેનો java કોડ વિકસાવો. (ગુણ: 07)

Java for લૂપ સિન્ટેક્સ:

```
for (initialization; condition; update) {  
    // code to be executed  
}
```

૧ થી ૧૦ વચ્ચેના પ્રાઈમ નંબર શોધવાનો પ્રોગ્રામ:

```
public class PrimeNumbers {  
    public static void main(String[] args) {  
        System.out.println("૧ અને ૧૦ વચ્ચેના પ્રાઈમ નંબર:");  
  
        // ૧ થી ૧૦ સુધીના નંબરમાં લૂપ ફરો  
        for (int i = 1; i <= 10; i++) {  
            boolean isPrime = true;  
  
            // ચેક કરો કે નંબર પ્રાઈમ છે  
            if (i == 1) {  
                isPrime = false; // ૧ પ્રાઈમ નથી  
            } else {  
                // ફેક્ટર્સ માટે ચેક કરો  
                for (int j = 2; j < i; j++) {  
                    if (i % j == 0) {  
                        isPrime = false;  
                        break;  
                    }  
                }  
            }  
  
            // પ્રાઈમ હોય તો પ્રિન્ટ કરો  
            if (isPrime) {  
                System.out.print(i + " ");  
            }  
        }  
    }  
}
```

આઉટપુટ:

```
૧ અને ૧૦ વચ્ચેના પ્રાઈમ નંબર:  
2 3 5 7
```

મુખ્ય સ્ટેપ્સ:

- ૧ થી ૧૦ સુધીના નંબરમાં લૂપ ફરો
- દરેક નંબર માટે, શરૂઆતમાં તેને પ્રાઈમ માની લો
- ૧ માટે ખાસ કેસ હેન્ડલ કરો (પ્રાઈમ નથી)
- ૨ થી (n-1) સુધીના કોઈ પણ નંબરથી ભાગી શકાય છે કે નહીં તપાસો
- જો ભાગી શકાય તો નંબર પ્રાઈમ નથી
- બધા પ્રાઈમ નંબર પ્રિન્ટ કરો

અલ્ગોરિથમને આ રીતે યાદ રાખો "IACHP":

- Iterate નંબર્સ, Assume પ્રાઈમ, Check ડિવિઝર્સ, Handle સ્પેશિયલ કેસિસ, Print રિઝલ્ટ્સ

પ્રશ્ન 2(a): Procedure-Oriented Programming (POP) અને Object-Oriented Programming (OOP) ના તફાવતોની યાદી આપો. (ગુણ: 03)

POP vs OOP:

Procedure-Oriented Programming	Object-Oriented Programming
ફોકસ પ્રોસીજર/ફંક્શન પર	ફોકસ ઓબ્જેક્ટ્સ પર
પ્રોગ્રામને ફંક્શન્સમાં વિભાજિત કરે છે	પ્રોગ્રામને ઓબ્જેક્ટ્સમાં વિભાજિત કરે છે
ડેટા અને ફંક્શન્સ અલગ છે	ડેટા અને ફંક્શન્સને એકસાથે એનકેપ્સ્યુલેટ કરે છે
ઓછી સુરક્ષિત (ગ્લોબલ ડેટા)	વધુ સુરક્ષિત (ડેટા હાઈડિંગ)
ઉદાહરણો: C, FORTRAN	ઉદાહરણો: Java, C++, Python

આ રીતે યાદ રાખો "FDSEM":

- Focus (પ્રોસીજર vs ઓબ્જેક્ટ્સ), Division (ફંક્શન્સ vs ઓબ્જેક્ટ્સ), Structure (અલગ vs સંયુક્ત), Encapsulation (નથી vs છે), Models (પ્રોસીજરલ vs ઓબ્જેક્ટ)

પ્રશ્ન 2(b): યોગ્ય ઉદાહરણ સાથે static કીવર્ડ સમજાવો. (ગુણ: 04)

static કીવર્ડ એવા એલિમેન્ટ્સ બનાવે છે જે ઇન્સ્ટન્સ ને બદલે ક્લાસને બિલોગ કરે છે.

```
public class Counter {  
    // Static વેરિએબલ (બધા ઓબ્જેક્ટ્સ દ્વારા શેર થયેલ)  
    static int count = 0;  
  
    // કન્ટ્રીક્ટર  
    Counter() {  
        count++; // શેડ કાઉન્ટર વધારે છે  
    }  
  
    // Static મેથડ  
    static void displayCount() {  
        System.out.println("કુલ બનાવેલા ઓબ્જેક્ટ્સ: " + count);  
    }  
}
```

```

public static void main(String[] args) {
    // ઓબ્જેક્ટ્સ બનાવો
    Counter c1 = new Counter();
    Counter c2 = new Counter();

    // static મેથડ કોલ કરો (ઓબ્જેક્ટ વગર)
    Counter.displayCount(); // આઉટપુટ: કુલ બનાવેલા ઓબ્જેક્ટ્સ: 2
}
}

```

static નો ઉપયોગ:

- **static વેરિએબલ:** બધા ઓબ્જેક્ટ્સ દ્વારા શેર થયેલ સિંગલ કોપી
- **static મેથડ:** ઓબ્જેક્ટ બનાવ્યા વગર કોલ કરી શકાય
- **static બ્લોક:** જ્યારે ક્લાસ મેમરીમાં લોડ થાય ત્યારે એક્ઝિક્યુટ થાય

 **static ઉપયોગ આ રીતે યાદ રાખો "SCM":**

- Shared ઓબ્જેક્ટ્સમાં, Class-level એક્સેસ, Method કોલિંગ ઓબ્જેક્ટ વગર

પ્રશ્ન 2(c): Constructorની વ્યાખ્યા આપો. Constructorના વિવિધ પ્રકારોની યાદી આપો. Parameterized constructor સમજાવવા માટેનો java code વિકસાવો. (ગુણ: 07)

Constructor એ સ્પેશિયલ મેથડ છે જે ઓબ્જેક્ટ બનાવતી વખતે ઇનિશિયલાઇઝ કરે છે.

Constructor ના પ્રકારો:

- **Default Constructor:** પેરામીટર્સ વગર
- **Parameterized Constructor:** પેરામીટર્સ સાથે
- **Copy Constructor:** બીજા ઓબ્જેક્ટ પરથી ઓબ્જેક્ટ બનાવે

```

public class Student {
    // ઇન્સ્ટન્સ વેરિએબલ્સ
    int id;
    String name;

    // Default constructor
    Student() {
        id = 0;
        name = "અજ્ઞાત";
        System.out.println("Default constructor કોલ થયો");
    }

    // Parameterized constructor
    Student(int studentId, String studentName) {
        id = studentId;
        name = studentName;
        System.out.println("Parameterized constructor કોલ થયો");
    }

    // Display મેથડ
    void display() {

```

```

        System.out.println("ID: " + id + ", નામ: " + name);
    }

    public static void main(String[] args) {
        // Constructors વાપરીને ઓબ્જેક્ટ્સ બનાવવા
        Student s1 = new Student(); // Default constructor
        Student s2 = new Student(101, "અવેક્સ"); // Parameterized constructor

        // ઓબ્જેક્ટ ડેટા દર્શાવો
        s1.display(); // આઉટપુટ: ID: 0, નામ: અજ્ઞાત
        s2.display(); // આઉટપુટ: ID: 101, નામ: અવેક્સ
    }
}

```

Parameterized Constructor ની લાક્ષણિકતાઓ:

- ઓબ્જેક્ટ બનાવતી વખતે પેરામીટર્સ સ્વીકારે છે
- આપેલા મૂલ્યો સાથે ઓબ્જેક્ટ ઇનિશિયલાઇઝ કરે છે
- ક્લાસ જેવું જ નામ ધરાવે છે
- રિટર્ન ટાઇપ નથી
- ઓબ્જેક્ટ ક્રિએટ થાય ત્યારે ઓટોમેટિક કોલ થાય છે

constructor પ્રકારો આ રીતે યાદ રાખો "DPC":

- Default (પેરામીટર વગર), Parameterized (પેરામીટર સાથે), Copy (ઓબ્જેક્ટ્સ ક્લોન)

પ્રશ્ન 2(a OR): java માં મૂળભૂત OOP conceptsની યાદી આપો અને કોઈ પણ એક સમજાવો. (ગુણ: 03)

Java માં મૂળભૂત OOP Concepts:

- **Encapsulation:** ડેટા અને મેથડ્સને સિંગલ યુનિટ (ક્લાસ) માં રાખવાની પ્રક્રિયા
- **Inheritance:** એવા નવા ક્લાસ બનાવવા જે હાલના ક્લાસની પ્રોપર્ટીઝનો ઉપયોગ કરે
- **Polymorphism:** ઓબ્જેક્ટ્સ વિવિધ રીતે એક જ મેથડના નામનો પ્રતિસાદ આપે
- **Abstraction:** ઇમ્પ્લિમેન્ટેશન ડિટેલ્સ છુપાવીને, માત્ર જરૂરી ફંક્શનલિટી બતાવવી

Encapsulation ઉદાહરણ:

```

class Person {
    // પ્રાઇવેટ વેરિએબલ્સ (છુપાયેલા)
    private String name;
    private int age;

    // પબ્લિક મેથડ્સ (એક્સેસિબલ)
    public void setName(String name) { this.name = name; }
    public String getName() { return name; }
}

```

આ રીતે યાદ રાખો "EIPA":

- Encapsulation, Inheritance, Polymorphism, Abstraction

પ્રશ્ન 2(b OR): યોગ્ય ઉદાહરણ સાથે final કીવર્ડ સમજાવો. (ગુણ: 04)

final કીવર્ડ એલિમેન્ટ્સને અપરિવર્તનીય અથવા નોન-એક્સટેન્ડેબલ બનાવે છે.

```
// Final વેરિએબલ (કોન્સ્ટન્ટ)
public class FinalDemo {
    final double PI = 3.14159; // બદલી શકાતું નથી

    // Final મેથડ (ઓવરરાઇડ ન થઈ શકે)
    final void display() {
        System.out.println("આ મેથડ ઓવરરાઇડ થઈ શકતી નથી");
    }
}

// Final ક્લાસ (એક્સટેન્ડ ન થઈ શકે)
final class SecureClass {
    void show() {
        System.out.println("આ ક્લાસ એક્સટેન્ડ થઈ શકતો નથી");
    }
}

// આ એરર થશે: ફાઇનલ ક્લાસ એક્સટેન્ડ ન થઈ શકે
// class ChildClass extends SecureClass {}
```

final નો ઉપયોગ:

- final વેરિએબલ: કોન્સ્ટન્ટ્સ બનાવે (બદલી શકાતા નથી)
- final મેથડ: સબક્લાસમાં મેથડ ઓવરરાઇડિંગ અટકાવે છે
- final ક્લાસ: ક્લાસ ઇનહેરિટન્સ રોકે છે (કોઈ સબક્લાસ બનાવી ન શકાય)

આ રીતે યાદ રાખો "CMV":

- Constant વેરિએબલ્સ, Method ઓવરરાઇડ ન થઈ શકે, Veto ઇનહેરિટન્સ

પ્રશ્ન 2(c OR): java access modifier માટેનો scope લખો. public modifier સમજાવવા માટેનો java code વિકસાવો. (ગુણ: 07)

Java માં Access Modifiers Scope:

મોડિફાયર	ક્લાસ	પેકેજ	સબક્લાસ	વર્લ્ડ
private	✓	X	X	X
default (મોડિફાયર વગર)	✓	✓	X	X
protected	✓	✓	✓	X
public	✓	✓	✓	✓

```
// File: PublicDemo.java
package demo;

// Public ક્લાસ - ગમે ત્યાંથી એક્સેસિબલ
```

```

public class PublicDemo {
    // Public વેરિએબલ
    public String message = "Hello world";

    // Public મેથડ
    public void display() {
        System.out.println(message);
    }
}

// File: MainApp.java
package application;

// બીજા પેકેજમાંથી ઇમ્પોર્ટ
import demo.PublicDemo;

public class MainApp {
    public static void main(String[] args) {
        // અલગ પેકેજના ક્લાસનો ઓબ્જેક્ટ બનાવવો
        PublicDemo obj = new PublicDemo();

        // પબ્લિક મેમ્બર વેરિએબલ એક્સેસ કરવો
        System.out.println("મેસેજ: " + obj.message);

        // પબ્લિક મેથડ કોલ કરવી
        obj.display();
    }
}

```

Public Modifier ની લાક્ષણિકતાઓ:

- ગમે તે ક્લાસથી એક્સેસિબલ
- ગમે તે પેકેજથી એક્સેસિબલ
- એક્સેસ પ્રતિબંધો નથી
- એવા ક્લાસ, મેથડ અને વેરિએબલ્સ માટે વપરાય છે જે વ્યાપકપણે એક્સેસિબલ હોવા જોઈએ
- પબ્લિક ઇન્ટરફેસ વાળા ક્લાસ પબ્લિક મેથડ્સ વાપરે છે

📌 એક્સેસ સ્કોપ આ રીતે યાદ રાખો "CDPS":

- Class-only (private), Default પેકેજ, Protected ઇન્ટરિટન્સ, System-wide (public)

પ્રશ્ન 3(a): વિવિધ પ્રકારના inheritance ની યાદી આપો અને કોઈ પણ એક ઉદાહરણ સાથે સમજાવો. (ગુણ: 03)

Inheritance ના પ્રકારો:

- **Single Inheritance:** એક સબક્લાસ એક સુપરક્લાસને એક્સટેન્ડ કરે
- **Multilevel Inheritance:** ઇન્ટરિટન્સની ચેઇન (A→B→C)
- **Hierarchical Inheritance:** ઘણા સબક્લાસ એક સુપરક્લાસને એક્સટેન્ડ કરે
- **Multiple Inheritance:** એક ક્લાસ ઘણા ક્લાસને એક્સટેન્ડ કરે (ઇન્ટરફેસ દ્વારા)
- **Hybrid Inheritance:** ઇન્ટરિટન્સ પ્રકારોનું કોમ્બિનેશન

Single Inheritance ઉદાહરણ:


```
// પેરેન્ટ ક્લાસ
class Animal {
    void eat() {
        System.out.println("પ્રાણી ખાય છે");
    }
}

// ચાઇલ્ડ ક્લાસ
class Dog extends Animal {
    void bark() {
        System.out.println("કૂતરો ભસે છે");
    }

    public static void main(String[] args) {
        Dog d = new Dog();
        d.eat(); // ઇન્હેરિટેડ મેથડ
        d.bark(); // પોતાની મેથડ
    }
}
```

📄 ઇન્હેરિટેન્સ પ્રકારો આ રીતે યાદ રાખો "SMMHH":

- Single, Multilevel, Multiple, Hierarchical, Hybrid

પ્રશ્ન 3(b): કોઈ પણ બે String buffer class methods યોગ્ય ઉદાહરણ સાથે સમજાવો. (ગુણ: 04)

StringBuffer Class Methods:

```
public class StringBufferDemo {
    public static void main(String[] args) {
        StringBuffer sb = new StringBuffer("Hello");

        // 1. append() - અંતે કન્ટેન્ટ ઉમેરે છે
        sb.append(" world");
        System.out.println("append પછી: " + sb); // Hello world

        // 2. insert() - સ્પેસિફિક પોઝિશન પર કન્ટેન્ટ દાખલ કરે છે
        sb.insert(5, " Beautiful");
        System.out.println("insert પછી: " + sb); // Hello Beautiful world

        // અન્ય સામાન્ય મેથડ્સ:
        // 3. replace() - ટેક્સ્ટ બદલે છે
        // 4. delete() - કેરેક્ટર્સ દૂર કરે છે
        // 5. reverse() - કન્ટેન્ટને ઉલટાવે છે
    }
}
```

StringBuffer ની મુખ્ય મેથડ્સ:

- **append(String s):** બફરના અંતે ટેક્સ્ટ ઉમેરે છે
- **insert(int offset, String s):** સ્પેસિફિક પોઝિશનમાં ટેક્સ્ટ દાખલ કરે છે
- **replace(int start, int end, String s):** સ્ટાર્ટથી એન્ડ સુધીનો ટેક્સ્ટ બદલે છે

- **delete(int start, int end):** સ્ટાર્ટથી એન્ડ સુધીના કેરેક્ટર્સ દૂર કરે છે
- **reverse():** બધા કેરેક્ટર્સને ઉલટાવે છે

✏ આ રીતે યાદ રાખો "AIRDOR":

- Append ટેક્સ્ટ, Insert પોઝિશન પર, Replace કન્ટેન્ટ, Delete ભાગો, Reverse બધું

પ્રશ્ન 3(c): Interfaceની વ્યાખ્યા આપો. Interfaceની મદદ થી multiple inheritance નો java program લખો. (ગુણ: 07)

Interface એ abstract methodsનો કલેક્શન છે જેને ક્લાસિસ ઇમ્પ્લિમેન્ટ કરી શકે છે.

```
// પહેલો ઇન્ટરફેસ
interface Printable {
    void print(); // Abstract મેથડ
}

// બીજો ઇન્ટરફેસ
interface Showable {
    void show(); // Abstract મેથડ
}

// બંને ઇન્ટરફેસ ઇમ્પ્લિમેન્ટ કરતો ક્લાસ
class Document implements Printable, Showable {
    // બંને ઇન્ટરફેસની બધી મેથડ્સ ઇમ્પ્લિમેન્ટ કરો
    public void print() {
        System.out.println("ડોક્યુમેન્ટ પ્રિન્ટિંગ");
    }

    public void show() {
        System.out.println("ડોક્યુમેન્ટ શોઇંગ");
    }

    public static void main(String[] args) {
        Document doc = new Document();

        // ઇમ્પ્લિમેન્ટેડ મેથડ્સ કોલ કરવી
        doc.print();
        doc.show();

        // ઇન્ટરફેસ રેફરન્સ વાપરવું
        Printable p = new Document();
        p.print();

        Showable s = new Document();
        s.show();
    }
}
```

Interface ની લાક્ષણિકતાઓ:

- **abstract** મેથડ્સ ધરાવે છે (ઇમ્પ્લિમેન્ટેશન વગર)
- Java માં **multiple inheritance** શક્ય બનાવે છે
- મેથડ્સ અંતર્નિહિત રીતે **public** અને **abstract** હોય છે
- વેરિએબલ્સ અંતર્નિહિત રીતે **public, static, final** હોય છે
- ક્લાસિસ ઇન્ટરફેસને **implement** કરે છે (extend નહીં)

📝 ઇન્ટરફેસ ફીચર્સ આ રીતે યાદ રાખો "PAVIA":

- **P**ublic મેથડ્સ, **A**bstract માત્ર, **V**ariables કોન્સ્ટન્ટ્સ છે, **I**mplementation જરૂરી, **A**llows multiple inheritance

પ્રશ્ન 3(a OR): Abstract class અને Interface નો તફાવત આપો. (ગુણ: 03)

Abstract Class vs Interface:

Abstract Class	Interface
abstract કીવર્ડ વાપરે છે	interface કીવર્ડ વાપરે છે
બંને abstract અને concrete મેથડ્સ ધરાવી શકે	માત્ર abstract મેથડ્સ ધરાવે છે (Java 8 પહેલા)
કન્સ્ટ્રક્ટર્સ ધરાવી શકે	કન્સ્ટ્રક્ટર્સ ન ધરાવી શકે
ગમે તે પ્રકારના વેરિએબલ્સ ધરાવી શકે	વેરિએબલ્સ public static final જ હોય
માત્ર સિંગલ ઇન્હેરિટન્સ સપોર્ટ કરે	મલ્ટિપલ ઇન્હેરિટન્સ શક્ય બનાવે
એક્સેસ મોડિફાયર્સ ધરાવી શકે	મેથડ્સ અંતર્નિહિત રીતે public હોય

📝 આ રીતે યાદ રાખો "KCVIAM":

- **K**eyword અલગ, **C**oncrete મેથડ્સ મંજૂર/અમંજૂર, **V**ariable પ્રતિબંધો, **I**nheritance પ્રકાર, **A**ccess મોડિફાયર્સ, **M**ethod ઇમ્પ્લિમેન્ટેશન

પ્રશ્ન 3(b OR): કોઈ પણ બે String class methods યોગ્ય ઉદાહરણ સાથે સમજાવો. (ગુણ: 04)

String Class Methods:

```
public class StringMethodsDemo {
    public static void main(String[] args) {
        String str = "Hello Java Programming";

        // 1. length() - સ્ટ્રિંગની લંબાઈ પાછી આપે છે
        int len = str.length();
        System.out.println("લંબાઈ: " + len); // 23

        // 2. substring() - સ્ટ્રિંગનો ભાગ કાઢે છે
        String sub = str.substring(6, 10);
        System.out.println("સબસ્ટ્રિંગ: " + sub); // Java


        // અન્ય સામાન્ય મેથડ્સ:
```

```
// 3. indexOf() - ટેક્સ્ટનું પોઝિશન શોધે છે
// 4. equals() - સ્ટ્રિંગ્સને સરખાવે છે
// 5. replace() - ટેક્સ્ટ બદલે છે

}
```

String ની મુખ્ય મેથડ્સ:

- **length():** કેરેક્ટર્સની સંખ્યા પાછી આપે છે
- **substring(int start, int end):** સ્ટ્રિંગનો ભાગ પાછો આપે છે
- **indexOf(String str):** પ્રથમ ઓકરન્સનું પોઝિશન પાછું આપે છે
- **equals(Object obj):** સ્ટ્રિંગ કન્ટેન્ટ સરખાવે છે
- **replace(char old, char new):** કેરેક્ટર્સ બદલે છે

 આ રીતે યાદ રાખો "LASER":

- Length ગણતરી, Acquire સબસ્ટ્રિંગ, Search indexOf સાથે, Equals કમ્પેરિઝન, Replace ટેક્સ્ટ

પ્રશ્ન 3(c OR): Package સમજાવો અને package create કરવા માટેના સ્ટેપ્સની યાદી બનાવો. (ગુણ: 07)

Package એ **namespace** છે જે સંબંધિત ક્લાસ અને ઇન્ટરફેસને સંગઠિત કરે છે.

Package બનાવવાના સ્ટેપ્સ:

1. સોર્સ ફાઇલના ટોચે **package ડિક્લેર** કરો
2. -d ઓપ્શન સાથે **કમ્પાઇલ** કરો
3. અન્ય ક્લાસમાં ઉપયોગ કરવા માટે **ઇમ્પોર્ટ** કરો

```
// સ્ટેપ 1: પેકેજ ડિક્લેર કરો (Calculator.java)
package mathutil;

public class Calculator {
    public int add(int a, int b) {
        return a + b;
    }

    public int subtract(int a, int b) {
        return a - b;
    }
}

// સ્ટેપ 2: javac -d . Calculator.java સાથે કોમ્પાઇલ કરો

// સ્ટેપ 3: બીજી ફાઇલમાં પેકેજનો ઉપયોગ કરો (TestCalculator.java)
import mathutil.Calculator;

public class TestCalculator {
    public static void main(String[] args) {
        // પેકેજમાંથી ઓબ્જેક્ટ બનાવવો
        Calculator calc = new Calculator();

        // મેથડ્સનો ઉપયોગ
    }
}
```

```
        System.out.println("5 + 3 = " + calc.add(5, 3));
        System.out.println("5 - 3 = " + calc.subtract(5, 3));
    }
}
```

Package ના ફાયદા:

- ફંક્શનાલિટી દ્વારા ક્લાસિસને ઓર્ગનાઇઝ કરે છે
- નેમિંગ કોન્ફ્લિક્ટ્સ અટકાવે છે
- એક્સેસ કંટ્રોલ પ્રદાન કરે છે
- ડેટા એનકેપ્સ્યુલેશન શક્ય બનાવે છે
- ક્લાસિસને શોધવાનું સરળ બનાવે છે

📝 પેકેજ ક્રિએશન સ્ટેપ્સ આ રીતે યાદ રાખો "DCIE":

- **D**eclare પેકેજ નામ, **C**ompile -d સાથે, **I**mport અન્ય ક્લાસમાં, **E**xecute તમારો પ્રોગ્રામ

પ્રશ્ન 4(a): java માં errorના પ્રકારોની યાદી આપો. (ગુણ: 03)

Java માં Errors ના પ્રકારો:

- **Syntax Errors:** કોડમાં ગ્રામર મિસ્ટેક્સ જે કોમ્પાઇલેશન દરમિયાન શોધાય છે
- **Runtime Errors:** પ્રોગ્રામ એક્ઝિક્યુશન દરમિયાન થતી સમસ્યાઓ
- **Logical Errors:** કોડ યાવે છે પરંતુ ખોટા પરિણામો આપે છે
- **Compilation Errors:** કમ્પાઇલર દ્વારા મળતી ભૂલો (દા.ત. સેમિકોલન ખૂટે છે)
- **LinkageErrors:** રનટાઇમ પર ક્લાસિસને લિંક કરવામાં સમસ્યાઓ

ઉદાહરણો:

```
// Syntax Error
if (x > 5) // કર્વી બ્રેસિસ ખૂટે છે

// Runtime Error
int[] arr = new int[5];
arr[10] = 50; // ArrayIndexOutOfBoundsException

// Logical Error
int sum = a - b; // સરવાળો હોવો જોઈએ પણ બાદબાકી વાપરી
```

📝 આ રીતે યાદ રાખો "SRCLL":

- **S**yntax, **R**untime, **C**ompilation, **L**ogical, **L**inkage errors

પ્રશ્ન 4(b): try catch block યોગ્ય ઉદાહરણ સાથે સમજાવો. (ગુણ: 04)

try-catch બ્લોક્સ Java પ્રોગ્રામ્સમાં exceptions હેન્ડલ કરે છે.

```
public class TryCatchDemo {
    public static void main(String[] args) {
        try {
            // exception થઈ શકે તેવો કોડ
            int result = 10 / 0; // ArithmeticException
            System.out.println("પરિણામ: " + result); // એક્ઝિક્યુટ નહીં થાય
        }
    }
}
```

```

catch (ArithmeticException e) {
    // Exception હેન્ડલર
    System.out.println("એરર: " + e.getMessage());

    // ઓપ્શનલ: સ્ટેક ટ્રેસ પ્રિન્ટ કરો
    // e.printStackTrace();
}
finally {
    // હંમેશા એક્ઝિક્યુટ થાય છે
    System.out.println("આ હંમેશા એક્ઝિક્યુટ થાય છે");
}

System.out.println("પ્રોગ્રામ ચાલુ રહે છે...");
}
}

```

try-catch ઘટકો:

- **try બ્લોક:** એવો કોડ ધરાવે છે જે **exceptions** શ્રી કરી શકે
- **catch બ્લોક:** ચોક્કસ પ્રકારના **exceptions** હેન્ડલ કરે છે
- **finally બ્લોક:** એવો કોડ જે **હંમેશા એક્ઝિક્યુટ થાય છે** (ઓપ્શનલ)
- વિવિધ પ્રકારના exceptions હેન્ડલ કરવા **મલ્ટિપલ catch બ્લોક્સ** હોઈ શકે
- try-catch વગર, exceptions **પ્રોગ્રામ ટર્મિનેશન** કરાવે છે

આ રીતે યાદ રાખો "TECH":

- **T**ry જોખમી કોડ, **E**xception પકડાય છે, **C**atch સમસ્યાઓ હેન્ડલ કરે, **H**ealing પછી ચાલુ રહે

પ્રશ્ન 4(c): method overloading અને overriding વચ્ચેના ચાર તફાવત આપો. method overriding સમજાવવા માટેનો java program લખો. (ગુણ: 07)

Method Overloading vs Method Overriding:

Method Overloading	Method Overriding
એક જ ક્લાસમાં	પેરેન્ટ-ચાઈલ્ડ ક્લાસિસમાં
અલગ પેરામીટર્સ	સમાન પેરામીટર્સ
કોમ્પાઇલ-ટાઇમ પોલિમોર્ફિઝમ	રનટાઇમ પોલિમોર્ફિઝમ
રિટર્ન ટાઇપ અલગ હોઈ શકે	રિટર્ન ટાઇપ સમાન અથવા covariant હોવું જોઈએ
Static મેથડ્સ ઓવરલોડ થઈ શકે	Static મેથડ્સ ઓવરરાઇડ ન થઈ શકે
એક્સેસ મોડિફાયર્સ બદલી શકાય	એક્સેસિબિલિટી પ્રતિબંધિત ન કરી શકાય

```

// Method overriding ઉદાહરણ
class Animal {
    // પેરેન્ટ ક્લાસ મેથડ
    void makeSound() {
        System.out.println("પ્રાણી અવાજ કરે છે");
    }
}

```

```

    }
}

class Dog extends Animal {
    // યાદ્દ ક્લાસમાં ઓવરરાઇડેડ મેથડ
    @Override
    void makeSound() {
        System.out.println("કૂતરો ભસે છે");
    }
}

class Cat extends Animal {
    // અન્ય ઓવરરાઇડેડ મેથડ
    @Override
    void makeSound() {
        System.out.println("બિલાડી મ્યાઉં કરે છે");
    }

    public static void main(String[] args) {
        // ઓબ્જેક્ટ્સ બનાવો
        Animal a = new Animal();
        Animal d = new Dog();
        Animal c = new Cat();

        // મેથડ બિહેવિયર એક્ચ્યુઅલ ઓબ્જેક્ટ પર આધારિત
        a.makeSound(); // પ્રાણી અવાજ કરે છે
        d.makeSound(); // કૂતરો ભસે છે
        c.makeSound(); // બિલાડી મ્યાઉં કરે છે
    }
}

```

Method Overriding ના નિયમો:

- મેથડનું સમાન નામ હોવું જોઈએ
- મેથડના સમાન પેરામીટર્સ હોવા જોઈએ
- IS-A સંબંધ (ઇન્ટેરિટન્સ) હોવો જોઈએ
- રિટર્ન ટાઇપ સમાન અથવા covariant હોઈ શકે
- એક્સેસ લેવલ ઓછું ન કરી શકાય

 ઓવરરાઇડિંગ નિયમો આ રીતે યાદ રાખો "NISRA":

- **N**ame મેથ થવું જોઈએ, **I**nheritance જરૂરી, **S**ignature મેથ થવું જોઈએ, **R**eturn type સમાન/covariant, **A**ccess સમાન/વાઇડર

પ્રશ્ન 4(a OR): કોઈ પણ ચાર inbuilt exceptions ની યાદી આપો. (ગુણ: 03)

Java માં Inbuilt Exceptions:

- **ArithmeticException:** ગણિતીય ભૂલો (દા.ત. શૂન્ય વડે ભાગાકાર)
- **NullPointerException:** null ઓબ્જેક્ટ રેફરન્સ એક્સેસ કરવું
- **ArrayIndexOutOfBoundsException:** ઇન્ડેક્સ એરે ઇન્ડેક્સ
- **NumberFormatException:** કન્વર્ઝનમાં ઇન્ડેક્સ નંબર ફોર્મેટ

- **ClassCastException:** ઇનકમ્પેટિબલ ક્લાસિસ વચ્ચે ઇનવેલિડ કાસ્ટિંગ
- **IllegalArgumentException:** મેથડને ઇમ્પ્રોપર આર્ગ્યુમેન્ટ મળે
- **IOException:** ઇનપુટ/આઉટપુટ ઓપરેશન ફેલ્યોર
- **FileNotFoundException:** ફાઇલ એક્સેસ સમસ્યાઓ

ઉદાહરણો:

```
// ArithmeticException
int x = 10 / 0;

// NullPointerException
String str = null;
int length = str.length();

// ArrayIndexOutOfBoundsException
int[] arr = new int[5];
int value = arr[10];

// NumberFormatException
int num = Integer.parseInt("abc");
```

 સામાન્ય exceptions આ રીતે યાદ રાખો "ANNIE":

- Arithmetic, NullPointerException, NumberFormat, IndexOutOfBoundsException, Exception

પ્રશ્ન 4(b OR): યોગ્ય ઉદાહરણ સાથે "throw" કીવર્ડ સમજાવો. (ગુણ: 04)

throw કીવર્ડ સ્પષ્ટપણે exception શ્રો કરે છે.

```
public class ThrowDemo {
    // throw નો ઉપયોગ કરતી મેથડ
    static void validateAge(int age) {
        if (age < 18) {
            // સ્પષ્ટપણે exception શ્રો કરો
            throw new ArithmeticException("મતદાન માટે યોગ્ય નથી");
        } else {
            System.out.println("મતદાન આપવા આપનું સ્વાગત છે!");
        }
    }

    public static void main(String[] args) {
        try {
            // exception શ્રો કરી શકે તેવી મેથડ કોલ કરો
            validateAge(15); // આ exception શ્રો કરશે

            // જો exception થાય તો આ એક્ઝિક્યુટ નહીં થાય
            System.out.println("વેલિડેશન પછી");
        } catch (ArithmeticException e) {
            // શ્રો થયેલ exception હેન્ડલ કરો
            System.out.println("Exception પકડાયું: " + e.getMessage());
        }
    }
}
```



```

    }

    System.out.println("પ્રોગ્રામ ચાલુ રહે છે...");
}
}

```

throw નો ઉપયોગ:

- exception મેન્યુઅલી થ્રો કરે છે
- કસ્ટમ વેલિડેશન માટે વપરાય છે
- સ્ટાન્ડર્ડ અથવા કસ્ટમ exceptions થ્રો કરી શકે
- exception હેન્ડલિંગમાં કંટ્રોલ ફ્લો માટે જરૂરી
- try-catch અથવા throws સાથે હેન્ડલ કરવું જરૂરી

📄 આ રીતે યાદ રાખો "MCCTH":

- Manually exception બનાવો, Custom વેલિડેશન્સ, Control flow બદલવું, Throw મેથડમાં, Handling જરૂરી

પ્રશ્ન 4(c OR): 'this' કીવર્ડ 'Super' કીવર્ડ સાથે સરખાવો. યોગ્ય ઉદાહરણ સાથે super કીવર્ડ સમજાવો. (ગુણ: 07)

'this' vs 'super' કીવર્ડ્સ:

'this' કીવર્ડ	'super' કીવર્ડ
વર્તમાન ક્લાસ ઓબ્જેક્ટનો રેફરન્સ	પેરન્ટ ક્લાસ ઓબ્જેક્ટનો રેફરન્સ
વર્તમાન ક્લાસ મેમ્બર્સ એક્સેસ કરે	પેરન્ટ ક્લાસ મેમ્બર્સ એક્સેસ કરે
સમાન ક્લાસમાં કન્સ્ટ્રક્ટર યેઇનિંગ માટે	પેરન્ટ કન્સ્ટ્રક્ટર કોલ કરવા માટે
વેરિએબલ શેડોઇંગ રિઝોલ્વ કરે	ઓવરરાઇડેન મેથડ્સ/વેરિએબલ્સ એક્સેસ કરે
વર્તમાન ક્લાસ કોન્ટેક્સ્ટમાં વપરાય	ઇન્હેરિટન્સ કોન્ટેક્સ્ટમાં વપરાય

super કીવર્ડ ઉદાહરણ:

```

class Animal {
    String color = "સફેદ";

    void eat() {
        System.out.println("પ્રાણી ખાય છે");
    }
}

class Dog extends Animal {
    String color = "કાળો";

    void printColor() {
        // વર્તમાન ક્લાસ વેરિએબલ એક્સેસ કરો
        System.out.println("ફૂતરાનો રંગ: " + color);

        // પેરન્ટ ક્લાસ વેરિએબલ એક્સેસ કરો
    }
}

```

```

        System.out.println("પ્રાણીનો રંગ: " + super.color);
    }

    void doActions() {
        // વર્તમાન ક્લાસ મેથડ કોલ કરો
        eat();

        // પેરન્ટ ક્લાસ મેથડ કોલ કરો
        super.eat();
    }

    public static void main(String[] args) {
        Dog d = new Dog();
        d.printColor(); // બંને રંગ પ્રિન્ટ કરે
        d.doActions(); // બંને eat મેથડ્સ કોલ કરે
    }
}

```

super કીવર્ડના ઉપયોગ:

- પેરન્ટ ક્લાસ વેરિએબલ્સ ઍક્સેસ કરવા
- પેરન્ટ ક્લાસ મેથડ્સ કોલ કરવા
- પેરન્ટ ક્લાસ કન્સ્ટ્રક્ટર ઇન્વોક કરવા
- ઓવરરાઇડેન મેમ્બર્સ વચ્ચે તફાવત દર્શાવવા
- મેથડ ઓવરરાઇડિંગમાં આવશ્યક

 **super ઉપયોગ આ રીતે યાદ રાખો "VMCDO":**

- પેરન્ટના **V**ariables, પેરન્ટની **M**ethods, **C**onstructor કોલિંગ, **D**istinguish ઓવરરાઇડેન મેમ્બર્સ, **O**verride ઇમ્પ્લિમેન્ટેશન

પ્રશ્ન 5(a): વિવિધ Stream Classes ની યાદી આપો. (ગુણ: 03)

Java Stream Classes:

Byte Stream Classes (બાઇનરી ડેટા):

- **FileInputStream:** ફાઇલમાંથી બાઇટ્સ વાંચે
- **FileOutputStream:** ફાઇલમાં બાઇટ્સ લખે
- **BufferedInputStream:** એફિશિયન્સી માટે બફર્ડ ઇનપુટ
- **BufferedOutputStream:** એફિશિયન્સી માટે બફર્ડ આઉટપુટ
- **DataInputStream:** પ્રિમિટિવ ડેટા ટાઇપ્સ વાંચે
- **DataOutputStream:** પ્રિમિટિવ ડેટા ટાઇપ્સ લખે

Character Stream Classes (ટેક્સ્ટ ડેટા):

- **FileReader:** ફાઇલમાંથી કેરેક્ટર્સ વાંચે
- **FileWriter:** ફાઇલમાં કેરેક્ટર્સ લખે
- **BufferedReader:** readLine() સાથે બફર્ડ રીડિંગ
- **BufferedWriter:** newLine() સાથે બફર્ડ રાઇટિંગ
- **PrintWriter:** વધારેલી રાઇટિંગ કેપેબિલિટીઝ

 **આ રીતે યાદ રાખો "FBI-CRP":**

- **F**ile સ્ટ્રીમ્સ, **B**uffered સ્ટ્રીમ્સ, **I**ntput/Output સ્ટ્રીમ્સ, **C**haracter સ્ટ્રીમ્સ, **R**eaders, **P**rinters

પ્રશ્ન 5(b): 'Divide by Zero' એરર માટે યુઝર ડીફાઇન્ડ Exception હેન્ડલ કરવા માટે જાવા પ્રોગ્રામ લખો. (ગુણ: 04)

```
// કસ્ટમ exception
class DivideByZeroException extends Exception {
    // કન્સ્ટ્રક્ટર
    public DivideByZeroException(String message) {
        super(message);
    }
}

public class CustomExceptionDemo {
    // કસ્ટમ exception શ્રી કરી શકે તેવી મેથડ
    static double divide(int a, int b) throws DivideByZeroException {
        if (b == 0) {
            throw new DivideByZeroException("શૂન્ય વડે ભાગી શકાતું નથી!");
        }
        return (double) a / b;
    }

    public static void main(String[] args) {
        try {
            // કેટલાક ભાગાકાર પ્રયાસ કરો
            System.out.println("10/2 = " + divide(10, 2)); // સારી રીતે કામ કરશે
            System.out.println("20/0 = " + divide(20, 0)); // exception શ્રી કરશે
        } catch (DivideByZeroException e) {
            System.out.println("કસ્ટમ Exception: " + e.getMessage());
        }
        System.out.println("પ્રોગ્રામ ચાલુ રહે છે...");
    }
}
```

કસ્ટમ Exception સ્ટેપ્સ:

- Exception એક્સટેન્ડ કરતો ક્લાસ બનાવો
- પેરન્ટને મેસેજ પાસ કરતો કન્સ્ટ્રક્ટર ડિફાઇન કરો
- exception ઊભું કરવા throw વાપરો
- try-catch અથવા throws સાથે હેન્ડલ કરો

આ રીતે યાદ રાખો "CETH":

- Create કસ્ટમ ક્લાસ, Extend Exception, Throw જ્યારે જરૂરી, Handle યોગ્ય રીતે

પ્રશ્ન 5(c): જાવામાં એક પ્રોગ્રામ લખો જે બાઈટ બાય બાઈટ ફાઈલના કન્ટેન્ટ વાંચે અને તેને બીજી ફાઈલ માં કોપી કરે. (ગુણ: 07)

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class FileCopyByteByByte {
    public static void main(String[] args) {
```

```

// સોર્સ અને ડેસ્ટિનેશન ફાઇલ્સ ડિફાઇન કરો
String sourceFile = "source.txt";
String destFile = "destination.txt";

// સ્ટ્રીમ્સ ડિસ્કવેર કરો
FileInputStream fis = null;
FileOutputStream fos = null;

try {
    // સ્ટ્રીમ્સ ઇનિશિયલાઇઝ કરો
    fis = new FileInputStream(sourceFile);
    fos = new FileOutputStream(destFile);

    System.out.println("ફાઇલ કોપી થઈ રહી છે...");

    int byteData;
    // ફાઇલને બાઇટ બાઇટ વાંચો (-1 એટલે ફાઇલનો અંત)
    while ((byteData = fis.read()) != -1) {
        // ડેસ્ટિનેશન ફાઇલમાં બાઇટ લખો
        fos.write(byteData);
    }

    System.out.println("ફાઇલ સફળતાપૂર્વક કોપી થઈ ગઈ!");
} catch (IOException e) {
    System.out.println("એરર: " + e.getMessage());
    e.printStackTrace();
} finally {
    // finally બ્લોકમાં સ્ટ્રીમ્સ બંધ કરો
    try {
        if (fis != null) fis.close();
        if (fos != null) fos.close();
        System.out.println("સ્ટ્રીમ્સ બંધ થઈ ગઈ.");
    } catch (IOException e) {
        System.out.println("સ્ટ્રીમ્સ બંધ કરવામાં એરર: " + e.getMessage());
    }
}
}

```

મુખ્ય સ્ટેપ્સ:

- સોર્સ ફાઇલ માટે **FileInputStream** બનાવો
- ડેસ્ટિનેશન ફાઇલ માટે **FileOutputStream** બનાવો
- સોર્સ ફાઇલને read() સાથે બાઇટ બાઇટ વાંચો
- દરેક બાઇટને write() સાથે ડેસ્ટિનેશનમાં લખો
- પૂર્ણ થયા પછી બંને સ્ટ્રીમ્સ બંધ કરો
- સંભવિત IOException હેન્ડલ કરો

આ રીતે યાદ રાખો "CREW-CH":

- **C**reate સ્ટ્રીમ્સ, **R**ead બાઇટ્સ, **E**xamine EOF માટે, **W**rite બાઇટ્સ, **C**lose સ્ટ્રીમ્સ, **H**andle exceptions

પ્રશ્ન 5(a OR): javaના વિવિધ file operationsની યાદી આપો. (ગુણ: 03)

Java માં File Operations:

- **File Creation:** નવી ફાઇલ બનાવવી
- **Reading:** ફાઇલમાંથી ડેટા વાંચવો
- **Writing:** ફાઇલમાં ડેટા લખવો
- **Appending:** હાલની ફાઇલમાં ડેટા ઉમેરવો
- **Deleting:** ફાઇલ દૂર કરવી
- **Renaming:** ફાઇલના નામ બદલવા
- **Copying:** ફાઇલ કોપી કરવી
- **Checking Existence:** ફાઇલ અસ્તિત્વમાં છે કે નહીં તપાસવું
- **Getting File Info:** સાઇઝ, પાથ, પરમિશન્સ વગેરે
- **Directory Operations:** ડિરેક્ટરી બનાવવી, વિસ્ટિંગ, નેવિગેશન

ઉદાહરણ કોડ:

```
File file = new File("test.txt");
boolean exists = file.exists();    // અસ્તિત્વ ચેક કરવું
boolean created = file.createNewFile(); // નવી ફાઇલ બનાવવી
long size = file.length();         // ફાઇલ સાઇઝ મેળવવી
boolean deleted = file.delete();   // ફાઇલ ડિલીટ કરવી
```

આ રીતે યાદ રાખો "CRWADCEG":

- Create, Read, Write, Append, Delete, Copy, Existence check, Get info

પ્રશ્ન 5(b OR): એક્સેપ્શન હેન્ડલિંગ માં finally block સમજાવતો જાવા પ્રોગ્રામ લખો. (ગુણ: 04)

```
import java.io.FileInputStream;
import java.io.IOException;

public class FinallyBlockDemo {
    public static void main(String[] args) {
        FileInputStream fis = null;

        try {
            // exception થી કરી શકે તેવો કોડ
            fis = new FileInputStream("nonexistent.txt");
            int data = fis.read();
            System.out.println("વંચાયેલ ડેટા: " + data);
        } catch (IOException e) {
            // Exception હેન્ડલ
            System.out.println("Exception પકડાયું: " + e.getMessage());
        } finally {
            // આ હંમેશા એક્ઝિક્યુટ થાય - ક્લીનઅપ માટે આદર્શ
            System.out.println("Finally બ્લોક એક્ઝિક્યુટ થયો");
        }
    }
}
```

```

        // exception હોય કે ન હોય રિસોર્સિસ બંધ કરો
        try {
            if (fis != null) fis.close();
            System.out.println("સ્ટ્રીમ બંધ થઈ");
        } catch (IOException e) {
            System.out.println("સ્ટ્રીમ બંધ કરવામાં એરર");
        }
    }

    System.out.println("પ્રોગ્રામ ચાલુ રહે છે...");
}
}

```

finally બ્લોકનો હેતુ:

- exception થાય કે ના થાય **હંમેશા એકિઝક્યુટ** થાય
- **ક્લીનઅપ ઓપરેશન્સ** (ફાઇલ્સ/કનેક્શન્સ બંધ કરવા) માટે વપરાય
- try/catch માં **return સ્ટેટમેન્ટ** હોય તો પણ એકિઝક્યુટ થાય
- **System.exit()** કોલ થાય તો જ એકિઝક્યુટ નથી થતું
- **રિસોર્સ મેનેજમેન્ટ** માટે અગત્યનું

 આ રીતે યાદ રાખો "ACER":

- Always એકિઝક્યુટ થાય, Cleanup ઓપરેશન્સ, Ensures રિસોર્સ રિલીઝ, Runs try-catch પછી

પ્રશ્ન 5(c OR): ફાઇલ ક્રિએટ કરવા અને તેમાં લખવા માટેનો જાવા પ્રોગ્રામ લખો. (ગુણ: 07)

```

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;

public class FilewriteDemo {
    public static void main(String[] args) {
        // રિસોર્સિસ ડિક્લેર કરો
        Scanner scanner = null;
        FileWriter writer = null;

        try {
            // ફાઇલ ઓબ્જેક્ટ બનાવો
            File file = new File("sample.txt");

            // જો ફાઇલ ન હોય તો નવી બનાવો
            if (file.createNewFile()) {
                System.out.println("ફાઇલ બનાવી: " + file.getName());
            } else {
                System.out.println("ફાઇલ પહેલેથી અસ્તિત્વમાં છે.");
            }

            // ફાઇલ રાઇટર ઇનિશિયલાઇઝ કરો
            writer = new FileWriter(file);

```

```

// યુઝર ઇનપુટ મેળવો
scanner = new Scanner(System.in);
System.out.println("ફાઇલમાં લખવા માટે ટેક્સ્ટ દાખલ કરો ('exit' લખો બહાર નીકળવા માટે):");

String line;
while (true) {
    line = scanner.nextLine();
    if (line.equalsIgnoreCase("exit")) break;

    // ફાઇલમાં લખો
    writer.write(line + "\n");
}

System.out.println("કન્ટેન્ટ ફાઇલમાં સફળતાપૂર્વક લખાયું!");

} catch (IOException e) {
    System.out.println("એરર આવી: " + e.getMessage());
    e.printStackTrace();
} finally {
    // રિસોર્સિસ બંધ કરો
    try {
        if (writer != null) writer.close();
        if (scanner != null) scanner.close();
    } catch (IOException e) {
        System.out.println("રિસોર્સિસ બંધ કરવામાં એરર: " + e.getMessage());
    }
}
}
}

```

ફાઇલ રાઇટ ઓપરેશન સ્ટેપ્સ:

- File ઓબ્જેક્ટ બનાવો
- ફાઇલ અસ્તિત્વમાં છે કે નવી બનાવવી તે ચેક કરો
- FileWriter ઇનિશિયલાઇઝ કરો
- લખવા માટે કન્ટેન્ટ મેળવો (યુઝર/સોર્સથી)
- ફાઇલમાં કન્ટેન્ટ લખો
- રિસોર્સિસ (રાઇટર, સ્કેનર) બંધ કરો

આ રીતે યાદ રાખો "CCIGWC":

- **C**reate ફાઇલ ઓબ્જેક્ટ, **C**heck અસ્તિત્વ, **I**nitialize રાઇટર, **G**et કન્ટેન્ટ, **W**rite ફાઇલમાં, **C**lose રિસોર્સિસ