

Java Programming (4343203) Summer-2024 Solutions - Gujarati

પ્રશ્ન 1(a): જાવામાં Garbage collection સમજાવો. (ગુણ: 03)

Garbage Collection એ Java ની ઓટોમેટિક મેમરી મેનેજમેન્ટ સિસ્ટમ છે.

- હેતુ: આપોઆપ વણવપરાયેલી વસ્તુઓ ને દૂર કરે છે અને મેમરી ખાલી કરે છે
- કાર્યપ્રણાલી: JVM કોઈ રેફરન્સ વગરની વસ્તુઓને ઓળખે છે અને દૂર કરે છે
- ફાયદાઓ: મેમરી લીકેજ અને મેન્યુઅલ મેમરી મેનેજમેન્ટની ભૂલોને રોકે છે

આ રીતે યાદ રાખો "GC-APB":

- Garbage collector Automatically Prevents મેમરી લીકેજ Backing up મેમરી

પ્રશ્ન 1(b): JVM ને વિગતવાર સમજાવો. (ગુણ: 04)

JVM (Java Virtual Machine) એ એન્જિન છે જે Java પ્રોગ્રામ ચલાવે છે.

- પ્લેટફોર્મ ઇન્ડિપેન્ડન્સ: Java ની "Write Once, Run Anywhere" ક્ષમતા આપે છે
- ઘટકો:

- Class Loader: મેમરીમાં ક્લાસ ફાઇલો લોડ કરે છે
- Runtime Data Areas: એક્ઝેક્યુશન દરમિયાન પ્રોગ્રામ ડેટા સ્ટોર કરે છે
- Execution Engine: બાઇટકોડને મશીન કોડમાં ઇન્ટરપ્રેટ કરે છે
- Garbage Collector: બિનજરૂરી ઓબ્જેક્ટ્સ દૂર કરે છે

આ રીતે યાદ રાખો "LERG":

- ક્લાસ Loads કરે, બાઇટકોડ Executes કરે, બધે Runs થાય, Garbage collects કરે

પ્રશ્ન 1(c): Fibonacci series પ્રિન્ટ કરવા માટેનો જાવા પ્રોગ્રામ લખો. (ગુણ: 07)

```
import java.util.Scanner;

public class FibonacciSeries {
    public static void main(String[] args) {
        // યુઝર પાસેથી ઇનપુટ લો
        Scanner sc = new Scanner(System.in);
        System.out.print("ટર્મની સંખ્યા દાખલ કરો: ");
        int n = sc.nextInt();

        // પ્રથમ બે ટર્મ્સ ઇનિશિયલાઇઝ કરો
        int first = 0, second = 1;

        System.out.println(n + " ટર્મ માટે ફિબોનાચી શ્રેણી:");

        // શ્રેણી પ્રિન્ટ કરવા માટે લૂપ
        for (int i = 1; i <= n; i++) {
```

```


        System.out.print(first + " ");

        // આગલી ટર્મ ગણો
        int next = first + second;
        first = second;
        second = next;
    }
    sc.close();
}
}

```

મુખ્ય સ્ટેપ્સ:

- ઇનિશિયલાઇઝ: **first=0, second=1** થી શરૂ કરો
- પ્રિન્ટ: વર્તમાન **first** વેલ્યુ
- ગણતરી: **next = first + second**
- અપડેટ: **first = second** અને **second = next**
- રિપીટ: N ટર્મ્સ સુધી

 આ રીતે યાદ રાખો "IPCUR":

- Initialize, Print, Calculate next, Update values, Repeat

પ્રશ્ન 1(c OR): કમાન્ડ લાઇન arguments નો ઉપયોગ કરીને કોઈપણ દસ સંખ્યાઓ માંથી ન્યૂનતમ શોધવા માટે જવા પ્રોગ્રામ લખો. (ગુણ: 07)

```

public class FindMinimum {
    public static void main(String[] args) {
        // ચેક કરો કે પૂરતા arguments છે કે નહીં
        if (args.length < 10) {
            System.out.println("કૃપા કરીને 10 નંબર દાખલ કરો");
            return;
        }

        // પ્રથમ આર્ગ્યુમેન્ટને ઇન્ટીજરમાં કન્વર્ટ કરો અને તેને ન્યૂનતમ માનો
        int min = Integer.parseInt(args[0]);

        // બાકીના નંબરો ચેક કરો
        for (int i = 1; i < 10; i++) {
            int num = Integer.parseInt(args[i]);
            // જો વર્તમાન નંબર નાનો હોય તો min અપડેટ કરો
            if (num < min) {
                min = num;
            }
        }

        System.out.println("ન્યૂનતમ સંખ્યા છે: " + min);
    }
}

```

મુખ્ય સ્ટેપ્સ:

- વાંચો: **command line arguments** માંથી સંખ્યાઓ મેળવો
- ઇનિશિયલાઇઝ: **min** = પહેલી સંખ્યા સેટ કરો
- તુલના કરો: દરેક સંખ્યા **min** કરતાં નાની છે કે નહીં તપાસો
- અપડેટ: જો વર્તમાન સંખ્યા નાની હોય, તો **min** = વર્તમાન સંખ્યા
- આઉટપુટ: સૌથી નાની સંખ્યા પ્રિન્ટ કરો

આ રીતે યાદ રાખો "RICUO":

- Read arguments, Initialize minimum, Compare દરેક સંખ્યા, Update જો નાની હોય, Output રિઝલ્ટ

પ્રશ્ન 2(a): Java OOP ના મૂળભૂત કોન્સેપ્ટની યાદી બનાવો. કોઈપણ એકને વિગતવાર સમજાવો. (ગુણ: 03)

Java માં OOP ના મૂળભૂત કોન્સેપ્ટ:

- **Encapsulation**: ડેટા અને મેથડને સિંગલ યુનિટ (ક્લાસ) માં રાખવાની પ્રક્રિયા
- **Inheritance**: નવા ક્લાસ બનાવવા જે મૌજૂદ ક્લાસની વિશેષતાઓનો ઉપયોગ કરે
- **Polymorphism**: વિવિધ ક્લાસના ઓબ્જેક્ટ્સ એક જ મેથડ નામનો ઉપયોગ કરે
- **Abstraction**: ઇમ્પ્લિમેન્ટેશન ડિટેલ્સ છુપાવીને, માત્ર જરૂરી ફંક્શનલિટી બતાવવી

આ રીતે યાદ રાખો "EIPA":

- Encapsulation, Inheritance, Polymorphism, Abstraction

પ્રશ્ન 2(b): final કી-વર્ડ ઉદાહરણ સાથે સમજાવો. (ગુણ: 04)

final કી-વર્ડ Java એપ્લિકેશનને અપરિવર્તનીય/નોન-એક્સટેન્ડેબલ બનાવે છે.

```
// Final વેરિએબલ (કોન્સ્ટન્ટ)
final double PI = 3.14159;

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

// Final ક્લાસ (એક્સટેન્ડ ન થઈ શકે)
final class SecureClass {
    // ક્લાસ ઇમ્પ્લિમેન્ટેશન
}
```

final નો ઉપયોગ:

- **final વેરિએબલ**: કોન્સ્ટન્ટ્સ બનાવે છે (ફરીથી એસાઇન ન કરી શકાય)
- **final મેથડ**: સબક્લાસમાં મેથડ ઓવરરાઇડિંગ અટકાવે છે
- **final ક્લાસ**: ક્લાસ ઇનહેરિટન્સ રોકે છે (કોઈ સબક્લાસ બનાવી ન શકાય)

આ રીતે યાદ રાખો "VCM":

- Variables કોન્સ્ટન્ટ બને, Classes extend ન થઈ શકે, Methods ઓવરરાઇડ ન થઈ શકે

પ્રશ્ન 2(c): કન્સ્ટ્રક્ટર શું છે? Parameterized કન્સ્ટ્રક્ટર ને ઉદાહરણ સાથે સમજાવો. (ગુણ: 07)

કન્સ્ટ્રક્ટર એ ખાસ મેથડ છે જે ઓબ્જેક્ટ બનતી વખતે ઇનિશિયલાઇઝ કરે છે.

```
public class Student {
    int id;
    String name;

    // Parameterized કન્સ્ટ્રક્ટર
    public Student(int studentId, String studentName) {
        id = studentId;          // id ઇનિશિયલાઇઝ કરો
        name = studentName;      // name ઇનિશિયલાઇઝ કરો
    }

    public void display() {
        System.out.println("ID: " + id + ", Name: " + name);
    }

    public static void main(String[] args) {
        // કન્સ્ટ્રક્ટર વાપરીને ઓબ્જેક્ટ બનાવો
        Student s1 = new Student(101, "રવિ");
        s1.display();
    }
}
```

Parameterized કન્સ્ટ્રક્ટર:

- ઓબ્જેક્ટ બનાવતી વખતે પેરામીટર્સ સ્વીકારે છે
- આપેલા મૂલ્યો સાથે ઓબ્જેક્ટ એટ્રિબ્યુટ્સ ઇનિશિયલાઇઝ કરે છે
- ક્લાસ જેવું જ નામ હોય છે પરંતુ પેરામીટર્સ સાથે
- કોઈ રિટર્ન ટાઇપ નથી, void પણ નહીં
- `new` વાપરીને ઓબ્જેક્ટ બનાવતી વખતે આપોઆપ કોલ થાય છે

આ રીતે યાદ રાખો "PAINS":

- Parameters સ્વીકારે, Attributes ઇનિશિયલાઇઝ થાય, Identical નામ ક્લાસ જેવું, No રિટર્ન ટાઇપ, Same-time ઓબ્જેક્ટ ક્રિએશન સાથે એકિઝક્યુશન

પ્રશ્ન 2(a OR): ઉદાહરણ સાથે જાવા પ્રોગ્રામ સ્ટ્રક્ચર સમજાવો. (ગુણ: 03)

Java પ્રોગ્રામ સ્ટ્રક્ચર:

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


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

// 3. ક્લાસ ડિક્લેરેશન
public class HelloWorld {
```

```
// 4. મેઇન મેથડ
public static void main(String[] args) {
    // 5. પ્રોગ્રામ સ્ટેટમેન્ટ્સ
    System.out.println("Hello world!");
}
}
```

મુખ્ય ઘટકો:

- પેકેજ ડિક્લેરેશન: સંબંધિત ક્લાસને સંગઠિત કરવા
- ઇમ્પોર્ટ સ્ટેટમેન્ટ્સ: અન્ય પેકેજમાંથી ક્લાસ એક્સેસ કરવા
- ક્લાસ ડિક્લેરેશન: ઓબ્જેક્ટ્સ માટે બ્લુપ્રિન્ટ
- મેઇન મેથડ: પ્રોગ્રામનો એન્ટ્રી પોઇન્ટ
- પ્રોગ્રામ સ્ટેટમેન્ટ્સ: કોડ ઇન્સ્ટ્રક્શન્સ

 આ રીતે યાદ રાખો "PICMS":

- Package, Imports, Class, Main method, Statements

પ્રશ્ન 2(b OR): યોગ્ય ઉદાહરણ સાથે static કી-વર્ડ સમજાવો. (ગુણ: 04)

static કી-વર્ડ એવા એલિમેન્ટ્સ બનાવે છે જે ઓબ્જેક્ટને બદલે ક્લાસને બિલ્લોગ કરે છે.

```
public class Counter {
    // Static વેરિએબલ (બધા ઓબ્જેક્ટ્સ દ્વારા શેર કરાય છે)
    static int count = 0;

    // કન્સ્ટ્રક્ટર
    Counter() {
        count++; // કાઉન્ટર વધારો
    }

    // Static મેથડ
    static void displayCount() {
        System.out.println("Count: " + count);
    }

    public static void main(String[] args) {
        // ઓબ્જેક્ટ વગર static મેથડ કોલ કરો
        Counter.displayCount(); // આઉટપુટ: Count: 0

        // ઓબ્જેક્ટ્સ બનાવો
        Counter c1 = new Counter();
        Counter c2 = new Counter();

        // ફરીથી static મેથડ કોલ કરો
        Counter.displayCount(); // આઉટપુટ: Count: 2
    }
}
```

static નો ઉપયોગ:

- static વેરિએબલ: બધા ઓબ્જેક્ટ્સ દ્વારા શેર કરાયેલ સિંગલ કોપી
- static મેથડ: ઓબ્જેક્ટ્સ બનાવ્યા વગર કોલ કરી શકાય
- static બ્લોક: ક્લાસ મેમરીમાં લોડ થાય ત્યારે એક્ઝિક્યુટ થાય છે

📝 આ રીતે યાદ રાખો "COS":

- Class-level એક્સેસ, One copy શેર થયેલ, Same બધા ઓબ્જેક્ટ માટે

પ્રશ્ન 2(c OR): ઇનહેરીટન્સ વ્યાખ્યાયિત કરો. તેના પ્રકારોની યાદી બનાવો. Multilevel અને Hierarchical ઇનહેરીટન્સ ને યોગ્ય ઉદાહરણ સાથે જાવો. (ગુણ: 07)

ઇનહેરીટન્સ એ મેકેનિઝમ છે જેમાં નવો ક્લાસ હાલના ક્લાસની પ્રોપર્ટીઝ મેળવે છે.

ઇનહેરીટન્સ ના પ્રકારો:

- **Single:** એક સબક્લાસ એક સુપરક્લાસને extend કરે
- **Multilevel:** ઇનહેરીટન્સની ચેઇન ($A \rightarrow B \rightarrow C$)
- **Hierarchical:** ઘણા સબક્લાસ એક સુપરક્લાસને extend કરે
- **Multiple:** એક ક્લાસ ઘણા ક્લાસને extend કરે (ઇન્ટરફેસ દ્વારા સપોર્ટેડ)
- **Hybrid:** ઇનહેરીટન્સ પ્રકારોનું કોમ્બિનેશન

Multilevel Inheritance ઉદાહરણ:

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

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

class Puppy extends Dog {
    void weep() { System.out.println("રડે છે..."); }

    public static void main(String args[]) {
        Puppy p = new Puppy();
        p.eat();    // Animal માંથી
        p.bark();   // Dog માંથી
        p.weep();   // Puppy માંથી
    }
}
```

Hierarchical Inheritance ઉદાહરણ:

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

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

```

class Cat extends Animal {
    void meow() { System.out.println("મ્યાઉ કરે છે..."); }

    public static void main(String args[]) {
        Cat c = new Cat();
        c.eat(); // Animal માંથી
        c.meow(); // Cat માંથી

        Dog d = new Dog();
        d.eat(); // Animal માંથી
        d.bark(); // Dog માંથી
    }
}

```

 ઇનહેરીટન્સના પ્રકારો આ રીતે યાદ રાખો "SMHMH":

- Single, Multilevel, Hierarchical, Multiple, Hybrid

પ્રશ્ન 3(a): this કી-વર્ડને યોગ્ય ઉદાહરણ સાથે જાવો. (ગુણ: 03)

this કી-વર્ડ મેથડ અથવા કન્સ્ટ્રક્ટરમાં વર્તમાન ઓબ્જેક્ટને સંદર્ભિત કરે છે.

```

public class Student {
    int rollNo;
    String name;

    // પેરામીટર્સ સાથે કન્સ્ટ્રક્ટર
    Student(int rollNo, String name) {
        this.rollNo = rollNo; // this વર્તમાન ઓબ્જેક્ટને સંદર્ભિત કરે છે
        this.name = name;
    }

    void display() {
        System.out.println(rollNo + " " + name);
    }

    public static void main(String args[]) {
        Student s1 = new Student(111, "કરણ");
        s1.display();
    }
}

```

this નો ઉપયોગ:

- ઇન્સ્ટન્સ વેરિએબલ અને પેરામીટર્સ વચ્ચે ભેદ કરે છે
- વર્તમાન ક્લાસ મેથડ/કન્સ્ટ્રક્ટર્સ કોલ કરે છે
- વર્તમાન ઓબ્જેક્ટ પાછું મોકલે છે

 આ રીતે યાદ રાખો "DIR":

- વેરિએબલ્સ વચ્ચે Differentiates, મેથડ્સ Invokes, વર્તમાન ઓબ્જેક્ટ Returns

પ્રશ્ન 3(b): જાવામાં વિવિધ એક્સેસ કંટ્રોલ સમજાવો. (ગુણ: 04)

એક્સેસ મોડિફાયર્સ ક્લાસ, મેથડ, અને વેરિએબલની દૃશ્યતા નિયંત્રિત કરે છે.

મોડિફાયર	ક્લાસ	પેકેજ	સબક્લાસ	વર્લ્ડ
private	✓	X	X	X
default	✓	✓	X	X
protected	✓	✓	✓	X
public	✓	✓	✓	✓

એક્સેસ લેવલ્સ:

- **private**: માત્ર ક્લાસની અંદર જ એક્સેસિબલ
- **default**: એક જ પેકેજમાં એક્સેસિબલ
- **protected**: પેકેજ અને સબક્લાસમાં એક્સેસિબલ
- **public**: ગમે ત્યાંથી એક્સેસિબલ

આ રીતે યાદ રાખો "PriDefProPub":

- **Private** (ક્લાસ), **Default** (પેકેજ), **Protected** (પેકેજ+સબક્લાસ), **Public** (બધે)

પ્રશ્ન 3(c): ઇન્ટરફેસ શું છે? ઇન્ટરફેસ દ્વારા ઉદાહરણ સાથે multiple inheritance સમજાવો. (ગુણ: 07)

ઇન્ટરફેસ એ કોન્ટ્રાક્ટ છે જેમાં abstract મેથડ્સ અને કોન્સ્ટન્ટ્સ હોય છે જે ક્લાસે ઇમ્પ્લિમેન્ટ કરવા જોઈએ.

```
// ઇન્ટરફેસ ડિફાઇન કરો
interface Printable {
    void print();
}

interface Showable {
    void show();
}

// બંને ઇન્ટરફેસ ઇમ્પ્લિમેન્ટ કરો
class Magazine implements Printable, Showable {
    // બંને ઇન્ટરફેસની બધી મેથડ્સ ઇમ્પ્લિમેન્ટ કરો
    public void print() {
        System.out.println("મેગેઝિન પ્રિન્ટ થાય છે...");
    }

    public void show() {
        System.out.println("મેગેઝિન દેખાય છે...");
    }

    public static void main(String args[]) {
        Magazine m = new Magazine();
        m.print();
    }
}
```



```
m.show();  
}  
}
```

ઈન્ટરફેસ ફીચર્સ:

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

આ રીતે યાદ રાખો "MAPLE":

- **M**ultiple inheritance, **A**bstract methods માત્ર, **P**ublic by default, **L**ike a contract, **E**asy implementation

પ્રશ્ન 3(a OR): super કી-વર્ડ ઉદાહરણ સાથે સમજાવો. (ગુણ: 03)

super કી-વર્ડ પેરન્ટ ક્લાસના ઓબ્જેક્ટ્સ/મેથડ્સને સંદર્ભિત કરે છે.

```
class Animal {  
    String color = "સફેદ";  
  
    void eat() {  
        System.out.println("ખાય છે...");  
    }  
}  
  
class Dog extends Animal {  
    String color = "કાળી";  
  
    void printColor() {  
        System.out.println(color);           // કાળી પ્રિન્ટ કરે છે  
        System.out.println(super.color);     // સફેદ પ્રિન્ટ કરે છે  
    }  
  
    void eat() {  
        super.eat(); // પેરન્ટ ક્લાસ મેથડ કોલ કરે છે  
        System.out.println("રોટલી ખાય છે...");  
    }  
}
```

super નો ઉપયોગ:

- પેરન્ટ ક્લાસ વેરિએબલ્સ એક્સેસ કરે છે
- પેરન્ટ ક્લાસ મેથડ્સ કોલ કરે છે
- પેરન્ટ ક્લાસ કન્સ્ટ્રક્ટર કોલ કરે છે

આ રીતે યાદ રાખો "VMC":

- પેરન્ટ ક્લાસના **V**ariables, **M**ethods, અને **C**onstructors એક્સેસ કરે

પ્રશ્ન 3(b OR): પેકેજ શું છે? પેકેજ બનાવવાના પગલાં લખો અને તેનું ઉદાહરણ આપો. (ગુણ: 04)

પેકેજ એ નેમસ્પેસ છે જે સંબંધિત ક્લાસ અને ઇન્ટરફેસને સંગઠિત કરે છે.

પેકેજ બનાવવાના પગલાં:

1. સોર્સ ફાઇલના ટોચ પર પેકેજ ડિક્લેર કરો
2. javac -d option સાથે કમ્પાઇલ કરો
3. અન્ય ક્લાસમાં ઉપયોગ કરવા માટે પેકેજ ઇમ્પોર્ટ કરો

```
// પગલું 1: પેકેજ ડિક્લેર કરો (Calculator.java તરીકે સેવ કરો)
package mathutils;

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

// પગલું 2: javac -d . Calculator.java સાથે કમ્પાઇલ કરો

// પગલું 3: બીજા ક્લાસમાં પેકેજનો ઉપયોગ કરો
import mathutils.Calculator;

class TestCalculator {
    public static void main(String args[]) {
        Calculator calc = new Calculator();
        System.out.println(calc.add(10, 20));
    }
}
```

પેકેજના ફાયદા:

- સંબંધિત ક્લાસને સંગઠિત કરે છે
- નામ કોન્ફ્લિક્ટ્સ અટકાવે છે
- એક્સેસ કંટ્રોલ પ્રદાન કરે છે

📁 આ રીતે પેકેજ બનાવટ યાદ રાખો "DCI":

- પેકેજ **D**eclare કરો, -d સાથે **C**ompile કરો, ઉપયોગ કરવા **I**mport કરો

પ્રશ્ન 3(c OR): વ્યાખ્યાયિત કરો: શ્રેડ ઓવરરાઇડિંગ. શ્રેડ ઓવરરાઇડિંગ માટેના નિયમોની યાદી બનાવો તથા એક જાવા પ્રોગ્રામ લખો જે શ્રેડ ઓવરરાઇડિંગને ઇમ્પલેમેન્ટ કરે. (ગુણ: 07)

મેથડ ઓવરરાઇડિંગ એ સબક્લાસમાં એક મેથડને ફરીથી ડિફાઇન કરવાની પ્રક્રિયા છે જે પહેલેથી પેરન્ટ ક્લાસમાં ડિફાઇન થયેલ છે.

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

```

}

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

    public static void main(String[] args) {
        Animal myAnimal = new Animal();
        myAnimal.makeSound(); // આઉટપુટ: પ્રાણી અવાજ કરે છે


        Dog myDog = new Dog();
        myDog.makeSound(); // આઉટપુટ: કૂતરો ભસે છે

        // પોલિમોર્ફિઝમ
        Animal animal = new Dog();
        animal.makeSound(); // આઉટપુટ: કૂતરો ભસે છે
    }
}

```

મેથડ ઓવરરાઇડિંગના નિયમો:

- મેથડનું નામ પેરન્ટ ક્લાસની મેથડ જેવું જ સમાન નામ હોવું જોઈએ
- મેથડના પેરન્ટ ક્લાસની મેથડ જેવા જ સમાન પેરામીટર્સ હોવા જોઈએ
- IS-A સંબંધ (ઇન્હેરિટન્સ) હોવો જરૂરી છે
- એક્સેસ મોડિફાયર સમાન અથવા વધુ પરમિસિવ હોવો જોઈએ
- રિટર્ન ટાઇપ સમાન અથવા covariant હોવું જોઈએ
- final અથવા static મેથડ્સ ઓવરરાઇડ ન કરી શકાય

 નિયમો આ રીતે યાદ રાખો "SPIARS":

- Same નામ, Parameters મેચ થવા જોઈએ, Inheritance જરૂરી, Access સમાન/વાઇડર, Return type સમાન/સબક્લાસ, Static/final ઓવરરાઇડ ન થઈ શકે

પ્રશ્ન 4(a): યોગ્ય ઉદાહરણ સાથે abstract class સમજાવો. (ગુણ: 03)

Abstract Class એ પ્રતિબંધિત ક્લાસ છે જેને સીધી રીતે ઇન્સ્ટેન્શિયેટ કરી શકાતી નથી.

```

abstract class Shape {
    // Abstract મેથડ (બોડી વગર)
    abstract void draw();

    // concrete મેથડ
    void resize() {
        System.out.println("આકાર રીસાઈઝ થાય છે");
    }
}

class Circle extends Shape {
    // Abstract મેથડ ઇમ્પ્લિમેન્ટ કરવી
    void draw() {

```

```

        System.out.println("વર્તુળ દોરે છે");
    }

    public static void main(String[] args) {
        // Shape s = new Shape(); // એરર: ઇન્સ્ટેન્શિયેટ ન કરી શકાય
        Circle c = new Circle();
        c.draw();
        c.resize();
    }
}

```

Abstract Class ફીચર્સ:

- સીધી રીતે ઇન્સ્ટેન્શિયેટ ન કરી શકાય
- **abstract** મેથડ્સ (બોડી વગર) ધરાવી શકે છે
- **concrete** મેથડ્સ (બોડી સાથે) ધરાવી શકે છે
- સબક્લાસએ **abstract** મેથડ્સ ઇમ્પ્લિમેન્ટ કરવી જ પડે

આ રીતે યાદ રાખો "NACI":

- No ઇન્સ્ટેન્શિયેશન, Abstract મેથડ્સ મળી શકે, Concrete મેથડ્સ મળી શકે, Implementation જરૂરી

પ્રશ્ન 4(b): થ્રેડ શું છે? થ્રેડ જીવનચક્ર સમજાવો. (ગુણ: 04)

થ્રેડ એ લાઇટવેઇટ પ્રોસેસ છે જે કોન્કરન્ટ એક્ઝિક્યુશન શક્ય બનાવે છે.

થ્રેડ લાઇફ સાયકલ સ્ટેટ્સ:

- **New**: થ્રેડ બનાવેલ છે પણ શરૂ થયેલ નથી
- **Runnable**: થ્રેડ ચલાવવા માટે તૈયાર છે
- **Running**: હાલમાં એક્ઝિક્યુટ થઈ રહી છે
- **Blocked/Waiting**: અસ્થાયી રૂપે નિષ્ક્રિય છે
- **Terminated**: એક્ઝિક્યુશન પૂર્ણ થયું છે

ટ્રાન્ઝિશન્સ:

- **start()**: New → Runnable
- **run()**: Runnable → Running
- **sleep()/wait()**: Running → Waiting
- **notify()**: Waiting → Runnable
- **run() પૂર્ણ થાય**: Running → Terminated

આ રીતે યાદ રાખો "NRWBT":

- New, Runnable, Running, Waiting/Blocked, Terminated

પ્રશ્ન 4(c): જાવામાં એક પ્રોગ્રામ લખો જે Thread Class નો અમલ કરીને બહુવિધ થ્રેડો બનાવે છે. (ગુણ: 07)

```

class MyThread extends Thread {
    private String threadName;

    // કન્સ્ટ્રક્ટર
    public MyThread(String name) {
        this.threadName = name;
    }
}

```

```

    }

    // run મેથડ ઓવરરાઇડ કરો
    @Override
    public void run() {
        try {
            for (int i = 1; i <= 3; i++) {
                System.out.println(threadName + ": Count " + i);
                Thread.sleep(1000); // 1 સેકન્ડ માટે પોઝ
            }
        } catch (InterruptedException e) {
            System.out.println(threadName + " interrupted.");
        }
        System.out.println(threadName + " finished.");
    }
}

public class MultiThreadDemo {
    public static void main(String[] args) {
        // બહુવિધ થ્રેડ્સ બનાવો
        MyThread thread1 = new MyThread("Thread-1");
        MyThread thread2 = new MyThread("Thread-2");

        // થ્રેડ્સ શરૂ કરો
        thread1.start();
        thread2.start();

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

```

બહુવિધ થ્રેડ્સ બનાવવા:

- **Thread ક્લાસ extend કરો:** Thread ક્લાસને extend કરતો સબક્લાસ બનાવો
- **run() મેથડ ઓવરરાઇડ કરો:** થ્રેડ શું કરશે તે ડિફાઇન કરો
- **થ્રેડ ઓબ્જેક્ટ્સ બનાવો:** તમારા થ્રેડ સબક્લાસને ઇન્સ્ટેન્શિયેટ કરો
- **start() મેથડ કોલ કરો:** થ્રેડ એક્ઝિક્યુશન શરૂ કરો

 આ રીતે યાદ રાખો "ECOS":

- **E**xtend Thread, **r**un મેથડ **C**reate કરો, **O**bject બનાવો, **S**tart કરો

પ્રશ્ન 4(a OR): યોગ્ય ઉદાહરણ સાથે final class સમજાવો. (ગુણ: 03)

final ક્લાસ એવો ક્લાસ છે જેને **extend** ન કરી શકાય (કોઈ સબક્લાસ બની શકતા નથી).

```

final class FinalClass {
    void display() {
        System.out.println("આ એક final ક્લાસ છે");
    }
}

// એરર: final ક્લાસને extend ન કરી શકાય
// class ChildClass extends FinalClass {

```

```
// ...
// }

class FinalClassDemo {
    public static void main(String[] args) {
        FinalClass fc = new FinalClass();
        fc.display();
    }
}
```

final ક્લાસના ફાયદા:

- સિક્યોરિટી: સંવેદનશીલ ક્લાસમાં ફેરફાર રોકે છે
- અપરિવર્તનીયતા: ક્લાસ બિહેવિયર બદલાય નહીં તેની ખાતરી કરે છે
- ઓપ્ટિમાઇઝેશન: કમ્પાઇલર final ક્લાસને ઓપ્ટિમાઇઝ કરી શકે છે

 આ રીતે યાદ રાખો "SIO":

- Security વધારે, Immutability ગેરંટી, Optimization શક્ય

પ્રશ્ન 4(b OR): યોગ્ય ઉદાહરણ સાથે thread ની પ્રાથમિકતાઓ સમજાવો. (ગુણ: 04)

થ્રેડ પ્રાયોરિટી થ્રેડના એક્ટિવેશનનું મહત્વ નક્કી કરે છે.

```
class PriorityThread extends Thread {
    PriorityThread(String name) {
        super(name);
    }

    public void run() {
        System.out.println("ચાલતી થ્રેડ: " +
            getName() +
            ", પ્રાયોરિટી: " +
            getPriority());
    }
}

public class ThreadPriorityDemo {
    public static void main(String[] args) {
        // થ્રેડ્સ બનાવો
        PriorityThread t1 = new PriorityThread("ઓછી પ્રાયોરિટી");
        PriorityThread t2 = new PriorityThread("સામાન્ય પ્રાયોરિટી");
        PriorityThread t3 = new PriorityThread("ઉચ્ચ પ્રાયોરિટી");

        // પ્રાયોરિટી સેટ કરો
        t1.setPriority(Thread.MIN_PRIORITY); // 1
        // t2 ડિફોલ્ટ પ્રાયોરિટી વાપરે છે // 5
        t3.setPriority(Thread.MAX_PRIORITY); // 10

        // થ્રેડ્સ શરૂ કરો
        t1.start();
        t2.start();
    }
}
```

```
t3.start();  
}  
}
```

પ્રાયોરિટી વિગતો:

- 1 (MIN_PRIORITY) થી 10 (MAX_PRIORITY) સુધીની રેન્જ
- ડિફોલ્ટ 5 (NORM_PRIORITY) છે
- ઊંચી પ્રાયોરિટી ધરાવતી થ્રેડને એક્ઝિક્યુશનમાં પસંદગી મળે છે
- પ્રાયોરિટી માત્ર શેડ્યુલરને હિંટ છે, ગેરંટી નથી

📄 આ રીતે યાદ રાખો "RPH":

- Range 1-10, ઊંચા મૂલ્ય માટે Preference, શેડ્યુલર માટે Hint

પ્રશ્ન 4(c OR): Exception શું છે? Arithmetic Exception નો ઉપયોગ દર્શાવતો પ્રોગ્રામ લખો. (ગુણ: 07)

Exception એ અસામાન્ય સ્થિતિ છે જે પ્રોગ્રામના સામાન્ય પ્રવાહને અવરોધે છે.

```
public class ArithmeticExceptionDemo {  
    public static void main(String[] args) {  
        try {  
            // Exception થઈ શકે તેવો કોડ  
            int a = 30, b = 0;  
            System.out.println("ડિવાઇડ કરવાનો પ્રયાસ: " + a + "/" + b);  
  
            // આ ArithmeticException થી કરશે  
            int result = a / b;  
  
            // જો exception થાય તો આ એક્ઝિક્યુટ નહીં થાય  
            System.out.println("પરિણામ: " + result);  
  
        } catch (ArithmeticException e) {  
            // Exception હેન્ડલર  
            System.out.println("Exception પકડાયું: " + e.getMessage());  
            System.out.println("શૂન્ય વડે ભાગી શકાતું નથી!");  
  
        } finally {  
            // હંમેશા એક્ઝિક્યુટ થાય  
            System.out.println("Finally બ્લોક એક્ઝિક્યુટ થયો");  
        }  
  
        System.out.println("બાકીનો કોડ ચાલુ રહે છે...");  
    }  
}
```

Exception હેન્ડલિંગ એલિમેન્ટ્સ:

- **try:** એવો કોડ ધરાવે છે જે exception થી કરી શકે
- **catch:** exception હેન્ડલ કરે છે
- **finally:** exception ના આધાર વિના એક્ઝિક્યુટ થાય છે

- **throw**: મેન્યુઅલી exception થ્રો કરે છે
- **throws**: મેથડ કઈ exceptions થ્રો કરી શકે તે જાહેર કરે છે

 આ રીતે યાદ રાખો "TCFTTS":

- Try જોખમી કોડ, Catch સમસ્યાઓ, Finally ક્લીન અપ, Throw જ્યારે જરૂરી, Throws ઘોષણા માટે, Safe એક્ટિવેશન

પ્રશ્ન 5(a): એરેની 10 સંખ્યાઓનો સરવાળો અને સરેરાશ શોધવા માટેનો જાવા પ્રોગ્રામ લખો. (ગુણ: 03)

```
public class ArraySumAverage {
    public static void main(String[] args) {
        // એરે ઇનિશિયલાઇઝ કરો
        int[] numbers = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};

        // સરવાળા અને સરેરાશ માટે વેરિએબલ્સ
        int sum = 0;
        double average;

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

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

        // પરિણામ પ્રિન્ટ કરો
        System.out.println("સરવાળો = " + sum);
        System.out.println("સરેરાશ = " + average);
    }
}
```

સ્ટેપ્સ:

- એરે ઇનિશિયલાઇઝ કરો 10 વેલ્યુ સાથે
- લૂપ વાપરીને બધા એલિમેન્ટ્સનો સરવાળો કરો
- સરેરાશ = સરવાળો / એલિમેન્ટ્સની સંખ્યા
- પરિણામ ડિસ્પ્લે કરો

 આ રીતે યાદ રાખો "ISAD":

- Initialize એરે, Sum એલિમેન્ટ્સ, Average ગણતરી, Display પરિણામ

પ્રશ્ન 5(b): 'DivideByZero' એરર માટે યુઝર ડિફાઈન્ડ Exception હેન્ડલ કરવા માટે જાવા પ્રોગ્રામ લખો. (ગુણ: 04)

```
// કસ્ટમ exception ક્લાસ
class DivideByZeroException extends Exception {
    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 {
            System.out.println(divide(10, 2)); // સારી રીતે કામ કરશે
            System.out.println(divide(20, 0)); // exception શ્રી કરશે
        } catch (DivideByZeroException e) {
            System.out.println("કસ્ટમ Exception: " + e.getMessage());
        }
    }
}

```

કસ્ટમ Exception બનાવવું:

- Exception ક્લાસ **extend** કરો
- મેસેજ સાથે **કસ્ટમીઝેડ** બનાવો
- કન્ડિશન આવે ત્યારે exception **throw** કરો
- તેને હેન્ડલ કરવા માટે **catch** કરો

આ રીતે યાદ રાખો "ETCW":

- **E**xtend Exception, **T**hrowable મેસેજ બનાવો, **C**reate અને **throw**, **W**rite હેન્ડલર

પ્રશ્ન 5(c): ટેક્સ્ટ ફાઇલ બનાવવા માટે જાવા પ્રોગ્રામ લખો અને ટેક્સ્ટ ફાઇલ પર રીડ ઓપરેશન કરો. (ગુણ: 07)

```

import java.io.File;
import java.io.FileWriter;
import java.io.FileReader;
import java.io.BufferedReader;
import java.io.IOException;

public class FileReadWriteDemo {
    public static void main(String[] args) {
        try {
            // ફાઇલ બનાવો
            File file = new File("sample.txt");
            if (file.createNewFile()) {
                System.out.println("ફાઇલ બનાવી: " + file.getName());
            }

            // ફાઇલમાં લખો
            FileWriter writer = new FileWriter(file);
            writer.write("હેલો વર્લ્ડ!\nઆ એક સેમ્પલ ટેક્સ્ટ ફાઇલ છે.\nJava I/O સરળ છે.");
            writer.close();
        }
    }
}

```

```

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

        // ફાઇલમાંથી વાંચો
        System.out.println("\nફાઇલ કન્ટેન્ટ:");
        FileReader reader = new FileReader(file);
        BufferedReader buffReader = new BufferedReader(reader);

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

        // રીડર બંધ કરો
        buffReader.close();

    } catch (IOException e) {
        System.out.println("એરર આવી.");
        e.printStackTrace();
    }
}
}

```

ફાઇલ ઓપરેશન્સ:

- File ક્લાસથી ફાઇલ બનાવો
- FileWriter વાપરીને કન્ટેન્ટ લખો
- FileReader અને BufferedReader વાપરીને કન્ટેન્ટ વાંચો
- વાપર્યા પછી રિસોર્સિસ બંધ કરો
- try-catch સાથે exceptions હેન્ડલ કરો

આ રીતે યાદ રાખો "CWRCH":

- ફાઇલ **C**reate કરો, કન્ટેન્ટ **W**rite કરો, કન્ટેન્ટ **R**ead કરો, રિસોર્સિસ **C**lose કરો, એક્સેપ્શન **H**andle કરો

પ્રશ્ન 5(a OR): Java I/O પ્રક્રિયા સમજાવો. (ગુણ: 03)

Java I/O (Input/Output) ડેટા વાંચવા અને લખવા માટે ક્લાસ પ્રદાન કરે છે.

I/O સ્ટ્રીમ્સ:

- **બાઇટ સ્ટ્રીમ્સ:** બાઇનરી ડેટા હેન્ડલ કરે (FileInputStream, FileOutputStream)
- **કેરેક્ટર સ્ટ્રીમ્સ:** ટેક્સ્ટ ડેટા હેન્ડલ કરે (FileReader, FileWriter)
- **બફર્ડ સ્ટ્રીમ્સ:** પરફોર્મન્સ સુધારે (BufferedReader, BufferedWriter)

પ્રોસેસ ફ્લો:

- સ્ટ્રીમ ખોલો → ડેટા પ્રોસેસ કરો → સ્ટ્રીમ બંધ કરો

આ રીતે યાદ રાખો "BCOP":

- **B**yte અથવા character સ્ટ્રીમ્સ, સોર્સ/ડેસ્ટિનેશનથી **C**onnect, ડેટા પર **O**perate, યોગ્ય રીતે **P**roperly close

પ્રશ્ન 5(b OR): Exception હેન્ડલિંગમાં Throw અને finally કી-વર્ડ ઉદાહરણ સાથે જાવો. (ગુણ: 04)

throw સ્પષ્ટપણે exception થ્રો કરે છે. **finally** કોડને હંમેશા એકિઝક્યુટ કરવાની ખાતરી આપે છે.

```

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

    public static void main(String[] args) {
        try {
            // exception થી કરી શકે તેવો કોડ
            validateAge(15); // આ exception થી કરશે

        } catch (ArithmeticException e) {
            System.out.println("Exception: " + e.getMessage());
        } finally {
            // આ હંમેશા એક્ઝિક્યુટ થશે
            System.out.println("Finally બ્લોક એક્ઝિક્યુટ થયો");
        }

        System.out.println("બાકીનો કોડ...");
    }
}

```

મુખ્ય પોઇન્ટ્સ:

- **throw**: કન્ડિશન આધારિત સ્પષ્ટ exception થી કરે છે
- **finally**: exception પછી પણ હંમેશા એક્ઝિક્યુટ થાય છે
- **throw** નો ઉપયોગ કસ્ટમ વેલિડેશન લોજિક માટે કરો
- **finally** નો ઉપયોગ ક્લીનઅપ ઓપરેશન્સ માટે કરો (ફાઇલ, કનેક્શન બંધ કરવા)

📌 આ રીતે યાદ રાખો "TFC":

- Throw exceptions મેન્યુઅલી, Finally હંમેશા ચાલે, Cleanup રિસોર્સિસ

પ્રશ્ન 5(c OR): ટેક્સ્ટ ફાઇલ ના કન્ટેન્ટ ડિસ્પ્લે કરવા અને ટેક્સ્ટ ફાઇલ પર એપેન્ડ ઓપરેશન કરવા માટે જાવા પ્રોગ્રામ લખો કરો. (ગુણ: 07)

```

import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.BufferedReader;
import java.io.IOException;

public class FileAppendDemo {
    public static void main(String[] args) {
        try {
            // ફાઇલ ન હોય તો બનાવો
            File file = new File("data.txt");
            if (!file.exists()) {

```

```

        // ફાઇલ બનાવો અને શરૂઆતી કન્ટેન્ટ લખો
        FileWriter writer = new FileWriter(file);
        writer.write("મૂળ કન્ટેન્ટ\n");
        writer.close();
        System.out.println("શરૂઆતી કન્ટેન્ટ સાથે ફાઇલ બની");
    }

    // વર્તમાન કન્ટેન્ટ દર્શાવો
    System.out.println("વર્તમાન ફાઇલ કન્ટેન્ટ:");
    FileReader reader = new FileReader(file);
    BufferedReader buffReader = new BufferedReader(reader);

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

    // નવું કન્ટેન્ટ એપેન્ડ કરો (true ફ્લેગ એપેન્ડ મોડ સક્ષમ કરે છે)
    FileWriter appendWriter = new FileWriter(file, true);
    appendWriter.write("આ કન્ટેન્ટ એપેન્ડ કરવામાં આવ્યું છે\n");
    appendWriter.close();
    System.out.println("\nકન્ટેન્ટ સફળતાપૂર્વક એપેન્ડ થયું");

    // અપડેટ કરેલ કન્ટેન્ટ દર્શાવો
    System.out.println("\nઅપડેટ કરેલ ફાઇલ કન્ટેન્ટ:");
    reader = new FileReader(file);
    buffReader = new BufferedReader(reader);

    while ((line = buffReader.readLine()) != null) {
        System.out.println(line);
    }
    buffReader.close();

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

```

એપેન્ડ ઓપરેશન સ્ટેપ્સ:

- પહેલા વર્તમાન કન્ટેન્ટ **ડિસ્પ્લે** કરો
- એપેન્ડ મોડમાં ફાઇલ **ખોલો** (FileWriter સાથે true પેરામીટર)
- છેલ્લે નવું કન્ટેન્ટ **લખો**
- રિસોર્સિસ **બંધ** કરો
- અપડેટ કરેલ કન્ટેન્ટ **ડિસ્પ્લે** કરો

આ રીતે યાદ રાખો "DOWCD":

- **D**isplay ઓરિજિનલ, **O**pen એપેન્ડ મોડમાં, **W**rite નવું કન્ટેન્ટ, **C**lose રાઇટર, **D**isplay અપડેટેડ કન્ટેન્ટ