

# Subject Name (Gujarati)

4343203 -- Summer 2025

Semester 1 Study Material

*Detailed Solutions and Explanations*

## પ્રશ્ન 1(અ) [3 ગુણ]

Java માં identifier ને નામ આપવાના નિયમોની યાદી માન્ય અને અમાન્ય ઉદાહરણો સાથે બનાવો.

### જવાબ

#### Java Identifier ના નિયમો:

| નિયમ         | વર્ણન   | માન્ય ઉદાહરણ          | અમાન્ય ઉદાહરણ  |
|--------------|---|-----------------------|----------------|
| શરૂઆતી       | letter, underscore અથવા                           | name, _value, \$cost  | 2name, #id     |
| અક્ષર        | dollar sign થી શરૂ                                |                       |                |
| પણીના અક્ષરો | letters, digits,<br>underscore, dollar હોઈ<br>શકે | student123, user_name | my-var, class@ |
| Keyword      | Java reserved words                               | myClass, userName     | class, int     |
| પ્રતિબંધ     | વાપરી શકાતા નથી                                   |                       |                |
| Case         | Identifier case-sensitive                         | Name ≠ name           | -              |
| Sensitivity  | છે  |                       |                |
| લંબાઈ        | કોઈ લંબાઈની મર્યાદા નથી                           | verylongvariablename  | -              |

### મેમરી ટ્રીક

“Letters First, Keywords Never, Case Counts”

## પ્રશ્ન 1(બ) [4 ગુણ]

Java ના operator ની યાદી બનાવો. Arithmetic અને Logical operator ને સમજાવો.

### જવાબ

#### Java Operator ના પ્રકારો:

| Operator નો પ્રકાર | ઉદાહરણો                     |
|--------------------|-----------------------------|
| Arithmetic         | +, -, *, /, %               |
| Relational         | ==, !=, <, >, <=, >=        |
| Logical            | &&,   , !                   |
| Assignment         | =, +=, -=, *=, /=           |
| Unary              | ++, --, +, -, !             |
| Bitwise            | &,  , ^, ~, <<, >>          |
| Ternary            | condition ? value1 : value2 |

### Arithmetic Operators:

- Addition (+): બે operand નો સરવાળો કરે છે
- Subtraction (-): પહેલામંથી બીજો બાદ કરે છે
- \*\*Multiplication (\*): બે operand નો ગુણાકાર કરે છે
- Division (/): પહેલાને બીજા વડે ભાગ આપે છે
- Modulus (%): ભાગાકારનો શેષ આપે છે

### Logical Operators:

- AND (&&): બંને શરતો સાચી હોય તો true આપે છે
- OR (||): ઓછામાં ઓછી એક શરત સાચી હોય તો true આપે છે
- NOT (!): logical state ને ઉલટાવે છે

### મેમરી ટ્રીક

``Add Subtract Multiply Divide Remainder, And Or Not''

## પ્રશ્ન 1(ક) [7 ગુણ]

3 અંકડાની સંખ્યાને વિપરીત કરવા માટે Java પ્રોગ્રામ લખો. ઉદાહરણ તરીકે, સંખ્યા 653 છે તો તેની વિપરીત સંખ્યા 356 છે.

### જવાબ

```
import java.util.Scanner;

public class ReverseNumber {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("3           : ");
        int num = sc.nextInt();

        int reverse = 0;
        int temp = num;

        while (temp != 0) {
            reverse = reverse * 10 + temp % 10;
            temp = temp / 10;
        }

        System.out.println("      : " + num);
        System.out.println("      : " + reverse);
    }
}
```

### Algorithm:

- છેલ્લો અંકડો કાઢો: Modulus operator (%) નો ઉપયોગ કરો
- વિપરીત સંખ્યા બનાવો: 10 થી ગુણો અને અંકડો ઉમેરો
- છેલ્લો અંકડો દૂર કરો: Integer division (/) નો ઉપયોગ કરો
- પુનરાવર્તન કરો: મૂળ સંખ્યા 0 થાય ત્યાં સુધી

### મેમરી ટ્રીક

``Extract, Build, Remove, Repeat''

## પ્રશ્ન 1(ક) OR [7 ગુણ]

\*\*3\*3 મેટ્રિક્સનો સરવાળો કરવા માટેનો Java પ્રોગ્રામ લખો.\*\*

## જવાબ

```
import java.util.Scanner;

public class MatrixAddition {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int[][] matrix1 = new int[3][3];
        int[][] matrix2 = new int[3][3];
        int[][] result = new int[3][3];

        //
        System.out.println("          :");
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                matrix1[i][j] = sc.nextInt();
            }
        }

        //
        System.out.println("          :");
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                matrix2[i][j] = sc.nextInt();
            }
        }

        //
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                result[i][j] = matrix1[i][j] + matrix2[i][j];
            }
        }

        //
        System.out.println("          :");
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                System.out.print(result[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

### Matrix Addition ના પગલાં:

- **Array બનાવો:** ત્રણ 3x3 integer array
- **મેટ્રિક્સ દાખલ કરો:** બંને મેટ્રિક્સ માટે values વાંચો
- **સંબંધિત elements ઉમેરો:**  $result[i][j] = matrix1[i][j] + matrix2[i][j]$
- **પરિણામ દર્શાવો:** સરવાળાનું મેટ્રિક્સ print કરો

## મેમરી ટ્રીક

“Create, Input, Add, Display”

## પ્રશ્ન 2(અ) [3 ગુણ]

પેરામીટ્રાઇડ કન્સ્ટ્રક્ટર નો ઉપયોગ દર્શાવતો Java પ્રોગ્રામ લખો.

## જવાબ

```

class Student \{
    private String name;
    private int rollNo;

    // Parameterized Constructor
    public Student(String name, int rollNo) \{
        this.name = name;
        this.rollNo = rollNo;
    \}

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

public class ParameterizedConstructor \{
    public static void main(String[] args) \{
        Student s1 = new Student(" ", 101);
        s1.display();
    \}
\}

```

### Parameterized Constructor ની વિશેષતાઓ:

- પ્રોમીટર લે છે: Object બનાવતી વખતે values સ્વીકારે છે
- Instance variables initialize કરે છે: Object ની state સેટ કરે છે
- Class જેતું જ નામ: Constructor નું નામ class સાથે મેળ ખાય છે
- Return type નથી: Constructor નો return type હોતો નથી

## મેમરી ટ્રીક

``Parameters Initialize Same-name No-return''

## પ્રશ્ન 2(બ) [4 ગુણા]

ઉદાહરણ સાથે નીચેના શબ્દોની બેસીક સીન્ટેક્સ આપો: (1) કલાસ બનાવવા માટે, (2) ઓફજેક્ટ બનાવવા માટે, (3) મેથ્ડને વ્યાખ્યાયિત કરવા માટે, (4) ચલ(વરીએલલ)ને ડિકલર કરવા માટે.

## જવાબ

### Java Basic Syntax:

| ઘટક                  | Syntax                                      | ઉદાહરણ                  |
|----------------------|---|-------------------------|
| Class બનાવવું        | class ClassName { }                         | class Car { }           |
| Object બનાવવું       | ClassName objectName =<br>new ClassName();  | Car myCar = new Car();  |
| Method વ્યાખ્યા      | returnType<br>methodName(parameters) {<br>} | public void start() { } |
| Variable declaration | dataType variableName;                      | int age;                |

## સંપૂર્ણ ઉદાહરણ:

```
class Car \{
    int speed; // Class
    // Variable Declaration

    public void accelerate() \{ // Method
        speed += 10;
    \}
\}

public class Main \{
    public static void main(String[] args) \{
        Car myCar = new Car(); // Object
    \}
\}
```

## મેમરી ટ્રીક

``Class Object Method Variable - COMV''

## પ્રશ્ન 2(ક) [7 ગુણ]

Java માં એક પ્રોગ્રામ લખો જેમાં Student નામનો Class છે. જેમાં enrollmentNo અને name નામ ના બે ઇન્સ્ટન્ટ વેરીએબલ હોય. Student Class ના 3 ઓફ્જેક્ટ main મેથડમાં બનાવો અને Student's ના નામ દર્શાવો.

### જવાબ

```
class Student \{
    String enrollmentNo;
    String name;

    // Student data initialize      constructor
    public Student(String enrollmentNo, String name) \{
        this.enrollmentNo = enrollmentNo;
        this.name = name;
    \}

    // Student      display      method
    public void displayName() \{
        System.out.println("      : " + name);
    \}
\}

public class StudentDemo \{
    public static void main(String[] args) \{
        // Student class 3 objects
        Student s1 = new Student("CS001", " ");
        Student s2 = new Student("CS002", " ");
        Student s3 = new Student("CS003", " ");

        //
        s1.displayName();
        s2.displayName();
        s3.displayName();
    \}
\}
```

### પ્રોગ્રામની રૂચના:

- Class વ્યાખ્યા: Instance variables સાથે Student class

- **Constructor:** enrollmentNo અને name initialize કરવા માટે
- **Method:** displayName() વિદ્યાર્થીનું નામ બતાવવા માટે
- **Object બનાવું:** main method માં ત્રણ Student objects
- **Method calling:** displayName() વાપરીને નામ દર્શાવવા

### મેમરી ટ્રીક

“Define Initialize Display Create Call”

---

## પ્રશ્ન 2(અ OR) [3 ગુણ]

ડિફોલ્ટ કન્સ્ટ્રક્ટર નો ઉપયોગ દર્શાવતો Java પ્રોગ્રામ લખો.

### જવાબ

```
class Rectangle {
    int length;
    int width;

    // Default Constructor
    public Rectangle() {
        length = 5;
        width = 3;
        System.out.println("Default constructor      ");
    }

    public void displayArea() {
        System.out.println("      : " + (length * width));
    }
}

public class DefaultConstructor {
    public static void main(String[] args) {
        Rectangle r1 = new Rectangle();
        r1.displayArea();
    }
}
```

### Default Constructor ની વિશેષતાઓ:

- કોઈ પેરામેટર નથી: કોઈ arguments લેતો નથી
- **Default values:** Instance variables માટે default values સેટ કરે છે
- આપોઆપ કોલ: Object બનાવતી વખતે કોલ થાય છે
- **Class જેવું જ નામ:** Constructor નું નામ class સાથે મેળ ખાય છે

### મેમરી ટ્રીક

“No-parameters Default Automatic Same-name”

---

## પ્રશ્ન 2(બ OR) [4 ગુણ]

પ્રોસ્સિજર-ઓરિએન્ટેડ પ્રોગ્રામિંગ અને ઓફ્જેક્ટ-ઓરિએન્ટેડ પ્રોગ્રામિંગ વચ્ચે ચાર તફાવત આપો.

## જવાબ

### POP vs OOP તુલના:

| પાસું        | Procedure Oriented Programming | Object-Oriented Programming       |
|--------------|--------------------------------|-----------------------------------|
| અભિગમ        | Top-down approach              | Bottom-up approach                |
| ધ્યાન        | Functions અને procedures પર    | Objects અને classes પર            |
| Data સુરક્ષા | Data hiding નથી, global access | Data encapsulation અને hiding     |
| સમસ્યા ઉકેલ  | Functions માં વહેંચતું         | Objects માં વહેંચતું              |
| Code         | મર્યાદિત પુનઃઉપયોગ             | Inheritance દ્વારા ઉચ્ચ પુનઃઉપયોગ |
| પુનઃઉપયોગ    |                                |                                   |
| જાળવણી       | જાળવતું મુશ્કેલ                | જાળવતું અને સુધારતું સરળ          |

### મુખ્ય તફાવતો:

- રચના: POP functions વાપરે છે, OOP classes વાપરે છે
- સુરક્ષા: OOP વધુ સારી data protection આપે છે
- પુનઃઉપયોગ: OOP inheritance અને polymorphism ને support કરે છે
- જાળવણી: OOP code જાળવવા માટે સરળ છે

## મેપરી ટ્રીક

"Structure Security Reusability Maintenance"

## પ્રશ્ન 2(ક OR) [7 ગુણ]

Java માં એક પ્રોગ્રામ લખો જેમાં Shape નામની Class હોય જે 2 ઓવરલોડ મેથસ area (ફ્લોટ radius) અને area (ફ્લોટ length, ફ્લોટ width) ધરાવતો હોય. ઓવરલોડ પદ્ધતિઓનો ઉપયોગ કરીને વર્તુળ અને લંબચોરસનો area (વિસ્તાર) દર્શાવો.

## જવાબ

```

class Shape {
    //           method
    public void area(float radius) {
        float circleArea = 3.14f * radius * radius;
        System.out.println("      : " + circleArea);
    }

    //           overloaded method
    public void area(float length, float width) {
        float rectangleArea = length * width;
        System.out.println("      : " + rectangleArea);
    }
}

public class MethodOverloading {
    public static void main(String[] args) {
        Shape shape = new Shape();

        // radius 5
        shape.area(5.0f);

        // length 4   width 6
        shape.area(4.0f, 6.0f);
    }
}

```

### Method Overloading ની વિભાવનાઓ:

- સમાન method નામ: બંને methods નું નામ "area"
- અલગ પેરામીટર: એક radius લે છે, બીજો length અને width લે છે

- **Compile-time polymorphism:** Method compile time पર पसंद थाय છે
- પેરામીટર તફાવત: પેરામીટરની સંખ્યા અથવા પ્રકાર અલગ

### મેમરી ટ્રીક

“Same-name Different-parameters Compile-time Parameter-differentiation”

### પ્રશ્ન 3(અ) [3 ગુણ]

સિંગલ ઇનહેરેટન્સ દર્શાવવા માટે Java પ્રોગ્રામ લખો.

#### જવાબ

```
// Parent class
class Animal {
    public void eat() {
        System.out.println("      ");
    }

    public void sleep() {
        System.out.println("      ");
    }
}

// Animal      Child class
class Dog extends Animal {
    public void bark() {
        System.out.println("      ");
    }
}

public class SingleInheritance {
    public static void main(String[] args) {
        Dog dog = new Dog();

        // Animal class      inherited methods
        dog.eat();
        dog.sleep();

        // Dog class      method
        dog.bark();
    }
}
```

**Single Inheritance** ની વિશેષતાઓ:

- એક parent: Child class એક parent class માંથી inherit કરે છે
- **extends keyword:** Inheritance સંબંધ સ્થાપિત કરવા માટે વપરાય છે
- **Method inheritance:** Child class parent ના methods inherit કરે છે
- **IS-A સંબંધ:** Dog IS-A Animal

### મેમરી ટ્રીક

“One-parent Extends Method IS-A”

### પ્રશ્ન 3(બ) [4 ગુણ]

Java માં એબ્સ્ટ્રેક્ટ કલાસને ઉદાહરણ સાથે વ્યાખ્યાપિત કરો.

#### જવાબ

**Abstract Class વ્યાખ્યા:** Abstract class એ એવો class છે જેનો instantiate કરી શકતો નથી અને તેમાં abstract methods (implementation વિનાના methods) હોઈ શકે છે.

```
// Abstract class
abstract class Vehicle {
    String brand;

    // method
    public void displayBrand() {
        System.out.println(" : " + brand);
    }

    // Abstract methods (implementation )
    public abstract void start();
    public abstract void stop();
}

// Abstract class extend concrete class
class Car extends Vehicle {
    public Car(String brand) {
        this.brand = brand;
    }

    // Abstract methods implement
    public void start() {
        System.out.println(" ");
    }

    public void stop() {
        System.out.println(" ");
    }
}

public class AbstractDemo {
    public static void main(String[] args) {
        Car car = new Car(" ");
        car.displayBrand();
        car.start();
        car.stop();
    }
}
```

**Abstract Class ની વિશેષતાઓ:**

- **Instantiate કરી શકતો નથી:** સીધા objects બનાવી શકતા નથી
- **Abstract methods:** Body વિનાના methods
- **Concrete methods:** Implementation સાથેના સામાન્ય methods
- **Extend કરું જરૂરી:** Child classes ને abstract methods implement કરવા પડે છે

#### મેમરી ટ્રીક

“Cannot-instantiate Abstract-methods Concrete-methods Must-extend”

### પ્રશ્ન 3(ક) [7 ગુણ]

ઇન્ટરફેસનો ઉપયોગ કરીને મદ્દીપલ ઇનહેરીટન્સને ઇમ્પ્રૈટ કરવા માટે Java માં પ્રોગ્રામ લખો.

## જવાબ

```
// interface
interface Flyable {
    void fly();
}

// interface
interface Swimmable {
    void swim();
}

// interfaces implement class
class Duck implements Flyable, Swimmable {
    private String name;

    public Duck(String name) {
        this.name = name;
    }

    // Flyable interface    fly method implement
    public void fly() {
        System.out.println(name + "      ");
    }

    // Swimmable interface    swim method implement
    public void swim() {
        System.out.println(name + "      ");
    }

    public void walk() {
        System.out.println(name + "      ");
    }
}

public class MultipleInheritance {
    public static void main(String[] args) {
        Duck duck = new Duck("      ");

        // Interfaces    methods
        duck.fly();
        duck.swim();

        //    method
        duck.walk();
    }
}
```

### Interfaces દ્વારા Multiple Inheritance:

- અનેક interfaces: Class અનેક interfaces implement કરી શકે છે
- implements keyword: Interfaces implement કરવા માટે વપરાય છે
- બધા methods implement કરવા જરૂરી: Interface ના બધા methods implement કરવા પડે છે
- Diamond problem હલ કરે છે: Multiple inheritance ની અસ્પષ્ટતા ટાળે છે

## મેમરી ટ્રીક

“Multiple-interfaces Implements Must-implement Solves-diamond”

### પ્રશ્ન 3(અ OR) [3 ગુણ]

મલ્ટીલેવલ ઇનહેરિટેન્સ દર્શાવવા માટે Java પ્રોગ્રામ લખો.

#### જવાબ

```
// Grandparent class
class Animal {
    public void breathe() {
        System.out.println("           ");
    }
}

// Animal      inherit      Parent class
class Mammal extends Animal {
    public void giveBirth() {
        System.out.println("           ");
    }
}

// Mammal      inherit      Child class
class Dog extends Mammal {
    public void bark() {
        System.out.println("           ");
    }
}

public class MultilevelInheritance {
    public static void main(String[] args) {
        Dog dog = new Dog();

        // Animal class (grandparent)      method
        dog.breathe();

        // Mammal class (parent)      method
        dog.giveBirth();

        // Dog class      method
        dog.bark();
    }
}
```

#### Multilevel Inheritance ની વિશેષતાઓ:

- Inheritance ની સંકળ:** Child → Parent → Grandparent
- અનેક સ્તરો:** બે કરતાં વધુ સ્તરો inheritance ના
- Transitive inheritance:** Properties સ્તરો દ્વારા પસાર થાય છે
- extends keyword:** દરેક સ્તર extends વાપરે છે

#### મેમરી ટ્રીક

“Chain Multiple Transitive Extends”

### પ્રશ્ન 3(બ OR) [4 ગુણ]

પેકેજ વ્યાખ્યાચિત કરો અને ઉદાહરણ સાથે પેકેજ બનાવવા માટે સીન્ટેક્સ લખો.

#### જવાબ

**Package વ્યાખ્યા:** Package એ namespace છે જે સંબંધિત classes અને interfaces ને organize કરે છે. તે access protection અને namespace management પ્રદાન કરે છે.

### Package Syntax:

```
package packageName;
```

#### ઉદાહરણ:

File: mypackage/Calculator.java

```
package mypackage;
```

```
public class Calculator {\n    public int add(int a, int b) {\n        return a + b;\n    }\n\n    public int subtract(int a, int b) {\n        return a {-} b;\n    }\n}
```

File: TestCalculator.java

```
import mypackage.Calculator;\n\npublic class TestCalculator {\n    public static void main(String[] args) {\n        Calculator calc = new Calculator();\n\n        System.out.println("      : " + calc.add(10, 5));\n        System.out.println("      : " + calc.subtract(10, 5));\n    }\n}
```

#### Package ના ફાયદા:

- **Namespace management:** નામકરણની ગુંચવણો ટાળે છે
- **Access control:** Class ની visibility control કરે છે
- **Code organization:** સંબંધિત classes ને group કરે છે
- **પુનઃઉપયોગ:** Packaged classes નો સરળ પુનઃઉપયોગ

### મેમરી ટ્રીક

“Namespace Access Organization Reusability”

## પ્રશ્ન 3(ક OR) [7 ગુણ]

મેથડ ઓવરરાઇડિંગ દર્શાવવા માટે Java પ્રોગ્રામ લખો.

### જવાબ

```
// Parent class\nclass Animal {\n    public void makeSound() {\n        System.out.println("      ");\n    }\n\n    public void move() {\n        System.out.println("      ");\n    }\n}\n\n// Parent methods    override    Child class
```

```

class Dog extends Animal \{
    // Method overriding
    @Override
    public void makeSound() \{
        System.out.println("        : ! !");
    \}

    @Override
    public void move() \{
        System.out.println("                ");
    \}
\}

class Cat extends Animal \{
    // Method overriding
    @Override
    public void makeSound() \{
        System.out.println("        : ! !");
    \}

    @Override
    public void move() \{
        System.out.println("                ");
    \}
\}

public class MethodOverriding \{
    public static void main(String[] args) \{
        Animal animal;

        // Dog object
        animal = new Dog();
        animal.makeSound(); // Dog makeSound() call
        animal.move(); // Dog move() call

        System.out.println();

        // Cat object
        animal = new Cat();
        animal.makeSound(); // Cat makeSound() call
        animal.move(); // Cat move() call
    \}
\}

```

#### Method Overriding ની વિશેષતાઓ:

- સમાન method signature: સમાન નામ, પેરામીટર અને return type
- Runtime polymorphism: Method runtime પર નક્કી થાય છે
- @Override annotation: વૈકલ્પિક પણ સુચવેલું
- IS-A સંબંધ: Child class parent ના method ને override કરે છે

#### મેમરી ટ્રીક

``Same-signature Runtime Override IS-A''

#### પ્રશ્ન 4(અ) [3 ગુણ]

Java માં વિવિધ પ્રકારની એરરની ચાદી બનાવો અને સમજાવો.

## જવાબ

Java Error ના પ્રકારો:

| Error નો પ્રકાર            | વર્ણન                                  | ઉદાહરણ                            |
|----------------------------|--|-----------------------------------|
| <b>Compile-time Errors</b> | Compilation દરમિયાન શોધાય છે           | Syntax errors, missing semicolons |
| <b>Runtime Errors</b>      | Program execution દરમિયાન થાય છે       | Division by zero, null pointer    |
| <b>Logical Errors</b>      | Program ચાલે છે પણ ખોટું output આપે છે | ખોટા algorithm logic              |

વિસ્તૃત સમજૂતી:

- **Compile-time:** Compiler દ્વારા અટકાવાય છે, run કરતાં પહેલાં fix કરવું પડશે
- **Runtime:** Program execution દરમિયાન crash થાય છે, exceptions દ્વારા handle થાય છે
- **Logical:** શોધવા સૌથી મુશ્કેલ, program કામ કરે છે પણ પરિણામ ખોટા આવે છે

સામાન્ય ઉદાહરણો:

- **Syntax Error:** Semicolon ગુમ, ખોટા brackets
- **RuntimeException:** ArrayIndexOutOfBoundsException, NullPointerException
- **Logic Error:** ખોટા formula, ખોટી condition

## મેમરી ટ્રીક

“Compile Runtime Logic - CRL”

## પ્રશ્ન 4(બ) [4 ગુણ]

રેપર કલાસ શું છે? કોઈપણ બે રેપર કલાસનો ઉપયોગ સમજાવો.

## જવાબ

**Wrapper Class વ્યાખ્યા:** Wrapper classes primitive data types ના object representation પ્રદાન કરે છે. તેઓ primitives ને objects માં convert કરે છે.

**Wrapper Class ટેબલ:**

| Primitive | Wrapper Class |
|-----------|---------------|
| int       | Integer       |
| double    | Double        |
| boolean   | Boolean       |
| char      | Character     |

### ઉદાહરણ - Integer Wrapper:

```
// Primitive to Object (Boxing)
int num = 100;
Integer intObj = Integer.valueOf(num);

// Object to Primitive (Unboxing)
int value = intObj.intValue();

// Utility methods
String str = "123";
int parsed = Integer.parseInt(str);
```

### ઉદાહરણ - Double Wrapper:

```
// Double object
Double doubleObj = Double.valueOf(45.67);

// String double convert
String str = "123.45";
double value = Double.parseDouble(str);

// Special values check
boolean isNaN = Double.isNaN(doubleObj);
```

### Wrapper Class ના ઉપયોગો:

- **Collections:** Collections માં primitives store કરવા માટે
- **Null values:** Primitives વિપરીત null store કરી શકે છે
- **Utility methods:** Parsing, conversion methods
- **Generics:** Generic types સાથે વાપરવા માટે

## મેમરી ટ્રીક

“Collections Null Utility Generics”

### પ્રશ્ન 4(ક) [7 ગુણ]

બોંકિંગ એપ્લિકેશન વિકસાવવા માટે Java માં એક પ્રોગ્રામ લખો જેમાં વપરાશકર્ત્તા 25000 રૂપિયા જમા કરે અને પછી 20000, 4000 રૂપિયા ઉપાડવાનું શરૂ કરે અને ત્યારબાદ જ્યારે વપરાશકર્ત્તા રૂ. 2000 ઉપાડે તો પ્રોગ્રામ “પૂર્ણ ભંડોળ નથી” એક્સોપ્શન(exception) આપે.

### જવાબ

```
// Custom Exception class
class InsufficientFundException extends Exception \{
    public InsufficientFundException(String message) \{
        super(message);
    \}

// Bank Account class
class BankAccount \{
    private double balance;

    public BankAccount(double initialBalance) \{
        this.balance = initialBalance;
    \}

    public void deposit(double amount) \{
        balance += amount;
    \}
}
```

```

        System.out.println("      : ." + amount);
        System.out.println("      : ." + balance);
    \}

    public void withdraw(double amount) throws InsufficientFundException \{
        if (amount > balance) \{
            throw new InsufficientFundException("      ");
        \}
        balance -= amount;
        System.out.println("      : ." + amount);
        System.out.println("      : ." + balance);
    \}

    public double getBalance() \{
        return balance;
    \}
\}

public class BankingApplication \{
    public static void main(String[] args) \{
        BankAccount account = new BankAccount(0);

        try \{
            // . 25000
            account.deposit(25000);
            System.out.println();

            // . 20000
            account.withdraw(20000);
            System.out.println();

            // . 4000
            account.withdraw(4000);
            System.out.println();

            // . 2000      (exception throw )
            account.withdraw(2000);
        \} catch (InsufficientFundException e) \{
            System.out.println("Exception: " + e.getMessage());
            System.out.println("      : ." + account.getBalance());
        \}
    \}
\}

```

### Exception Handling ના ધરકો:

- **Custom exception:** InsufficientFundException extends Exception
- **throw keyword:** Balance અપૂર્ણ હોય ત્યારે exception throw કરે છે
- **try-catch block:** Exception ને handle કરે છે
- **Exception message:** "પૂર્ણ ભંડાળ નથી" દર્શાવે છે

### Banking Operations:

- **Deposit:** Balance માં પૈસા ઉમેરે છે
- **Withdraw:** પૂર્ણ balance હોય તો પૈસા બાદ કરે છે
- **Balance check:** ઉપાડતાં પહેલાં validate કરે છે
- **Exception handling:** Program crash થતું અટકાવે છે

### મેમરી ટ્રીક

"Custom Throw Try-catch Message, Deposit Withdraw Check Handle"

## પ્રશ્ન 4(અ OR) [3 ગુણ]

શ્રેદના સંપૂર્ણ જીવનચક્ષણું વર્ણન કરો.

### જવાબ

#### Thread Lifecycle States:

```
stateDiagram{-v2}
    direction LR
    [*] {-{->} New}
    New {-{->} Runnable : start()}
    Runnable {-{->} Running : CPU allocated}
    Running {-{->} Runnable : yield()/time slice expired}
    Running {-{->} Blocked : wait()/sleep()/I/O}
    Blocked {-{->} Runnable : notify()/timeout/I/O complete}
    Runnable {-{->} Dead : run() method completes}
    Dead {-{->} [*]}
```

#### Thread States:

- **New:** Thread object બન્યો પણ start થયો નથી
- **Runnable:** Thread run થવા માટે તૈયાર, CPU ની રાહ જોઈ રહ્યો
- **Running:** Thread હાલમાં execute થઈ રહ્યો છે
- **Blocked:** Thread resource અથવા condition ની રાહ જોઈ રહ્યો
- **Dead:** Thread execution પૂર્ણ થયું

#### State Transitions:

- **start():** New → Runnable
- **CPU allocation:** Runnable → Running
  
- **wait()/sleep():** Running → Blocked
- **notify()/timeout:** Blocked → Runnable
- **completion:** Running → Dead

### મેમરી ટ્રીક

"New Runnable Running Blocked Dead"

## પ્રશ્ન 4(બ OR) [4 ગુણ]

એક્સેસ સ્પેસીફિકાયરની યાદી બનાવો અને Java માં તેમના હેતુનું વર્ણન કરો.

### જવાબ

#### Java Access Specifiers:

| Access Specifier | Same Class | Same Package | Subclass | Other Package |
|------------------|------------|--------------|----------|---------------|
| private          | □          | □            | □        | □             |
| default          | □          | □            | □        | □             |
| protected        | □          | □            | □        | □             |
| public           | □          | □            | □        | □             |

## Access Specifier ના હેતુઓ:

### Private:

- **Encapsulation:** Implementation details છુપાવે છે
- **Data સુરક્ષા:** Sensitive data ને સુરક્ષિત રાપે છે
- **Class-only access:** ફક્ત same class માં accessible

### Default (Package-private):

- **Package access:** Same package માં accessible
- **Module organization:** સંબંધિત classes ને group કરે છે
- કોઈ keyword જરૂરી નથી: કોઈ specifier ન મૂક્યું હોય ત્યારે default

### Protected:

- **Inheritance support:** Subclasses માં accessible
- **Package + inheritance:** Same package + subclasses
- **Controlled access:** Private કરતાં વધુ access, public કરતાં ઓછું

### Public:

- **સાર્વાનુક્તિક access:** કચાંથી પણ accessible
- **Interface methods:** Public APIs માટે વપરાય છે
- **મહત્તમ visibility:** કોઈ access restrictions નથી

## મેમરી ટ્રીક

“Private Encapsulates, Default Packages, Protected inherits, Public Universal”

## પ્રશ્ન 4(ક OR) [7 ગુણ]

Java પ્રોગ્રામ લખો જે 2 થ્રેડ ચલાવે છે દર થ્રેડ એક .1000 મીલીસેકન્ડ “થ્રેડ --I” ને પ્રિન્ટ કરે છે, બીજો થ્રેડ દર 2000 મીલીસેકન્ડ -- થ્રેડ “II” ને પ્રિન્ટ કરે છે. થ્રેડ Class ને એક્સટેન્ડ કરીને થ્રેડ બનાવો.

### જવાબ

```
// thread class
class Thread1 extends Thread {
    public void run() {
        try {
            for (int i = 1; i <= 10; i++) {

                System.out.println("Thread1 {- Count: " + i);
                Thread.sleep(1000); // 1000 milliseconds sleep
            }
        } catch (InterruptedException e) {
            System.out.println("Thread1 interrupt : " + e.getMessage());
        }
    }
}

// thread class
class Thread2 extends Thread {
    public void run() {
        try {
            for (int i = 1; i <= 5; i++) {

                System.out.println("Thread2 {- Count: " + i);
                Thread.sleep(2000); // 2000 milliseconds sleep
            }
        } catch (InterruptedException e) {
    }
}
```

```

        System.out.println("Thread2 interrupt : " + e.getMessage());
    \}
\}

public class MultiThreadDemo \{
    public static void main(String[] args) \{
        // Thread objects
        Thread1 t1 = new Thread1();
        Thread2 t2 = new Thread2();

        System.out.println(" threads      ...");

        // threads start
        t1.start();
        t2.start();

        try \{
            // threads complete
            t1.join();
            t2.join();
        \} catch (InterruptedException e) \{
            System.out.println("Main thread interrupt : " + e.getMessage());
        \}

        System.out.println(" threads      ");
    \}
\}

```

#### Multithreading વિભાગનાઓ:

- **Thread class extension:** બંને classes Thread ને extend કરે છે
- **run() method:** Thread execution code સમાવે છે
- **sleep() method:** નિર્દિષ્ટ સમય માટે thread ને pause કરે છે
- **start() method:** Thread execution શરૂ કરે છે
- **join() method:** Thread completion ની રાહ જુઓ છે

#### Thread Synchronization:

- **Concurrent execution:** બંને threads એકસાથે run થાય છે
- **Independent timing:** દરેક thread નો પોતાનો sleep interval છે
- **Exception handling:** InterruptedException catch અને handle થાય છે
- **Thread coordination:** join() ensure કરે છે કે main thread રાહ જુએ

#### મેમરી ટ્રીક

“Extend Run Sleep Start Join”

#### પ્રશ્ન 5(અ) [3 ગુણ]

સ્ટ્રીમ કલાસ શું છે? સ્ટ્રીમ કલાસ કેવી રીતે વર્ગીકૃત કરવામાં આવે છે?

#### જવાબ

**Stream Class વ્યાખ્યા:** Java માં Stream classes input અને output operations handle કરવાની રીત પ્રદાન કરે છે. તેઓ source થી destination સુધી data ના flow ને represent કરે છે.

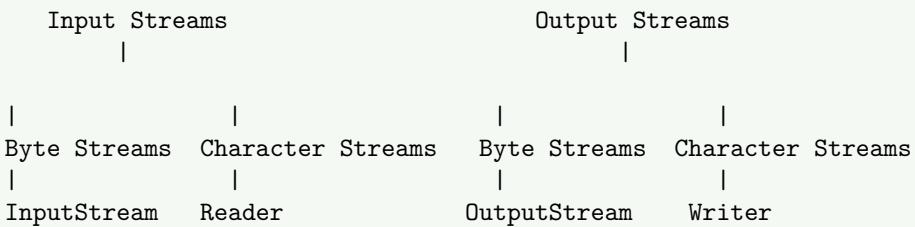
#### Stream Classification:

Stream Classes

|

|

|



### Stream ના પ્રકારો:

| Stream Type  | હેતુ                   | ઉદાહરણ Classes                         |
|--------------|------------------------|--|
| InputStream  | Bytes વાંચવા માટે      | FileInputStream, BufferedInputStream   |
| OutputStream | Bytes લખવા માટે        | FileOutputStream, BufferedOutputStream |
| Reader       | Characters વાંચવા માટે | FileReader, BufferedReader             |
| Writer       | Characters લખવા માટે   | FileWriter, BufferedWriter             |

### Classification ની વિશેષતાઓ:

- દિશા: Input (read) vs Output (write)
- Data type: Byte streams vs Character streams
- કાર્યક્રમતા: Basic vs Buffered streams

### મેમરી ટ્રીક

``Direction Data-type Functionality''

### પ્રશ્ન 5(બ) [4 ગુણ]

ઉદાહરણ સાથે મેથડ ઓવરરાઇટિંગનો હેતુ સમજાવો.

#### જવાબ

**Method Overriding નો હેતુ:** Method overriding subclass ને parent class માં પહેલેથી જ define કરેલા method નું specific implementation પ્રદાન કરવાની મંજૂરી આપે છે.

```

// Parent class
class Shape {
    public void draw() {
        System.out.println("          ");
    }

    public double area() {
        return 0.0;
    }
}

// Parent methods override Child class
class Circle extends Shape {
    private double radius;

    public Circle(double radius) {
        this.radius = radius;
    }

    // draw method overriding
    @Override
    public void draw() {
        System.out.println("      radius " + radius + "      ");
    }
}

```

```

// area method overriding
@Override
public double area() \{
    return 3.14 * radius * radius;
\}

class Rectangle extends Shape \{
    private double length, width;

    public Rectangle(double length, double width) \{
        this.length = length;
        this.width = width;
    \}

    @Override
    public void draw() \{
        System.out.println("      " + length + "x" + width);
    \}

    @Override
    public double area() \{
        return length * width;
    \}
\}

public class OverridingDemo \{
    public static void main(String[] args) \{
        Shape[] shapes = \{
            new Circle(5.0),
            new Rectangle(4.0, 6.0)
        \;;

        for (Shape shape : shapes) \{
            shape.draw();           // Overridden method call
            System.out.println("      : " + shape.area());
            System.out.println();
        \}
    \}
\}

```

#### Method Overriding ના ફાયદા:

- **Runtime polymorphism:** Runtime पર method selection
- **Specific implementation:** Child class specific behavior પ્રદાન કરે છે
- **Code flexibility:** Same interface, અલગ implementations
- **Dynamic method dispatch:** Object type અનુસાર યોગ્ય method call

#### મેમરી ટ્રીક

``Runtime Specific Flexibility Dynamic''

#### પ્રશ્ન 5(ક) [7 ગુણ]

Abc.txt નામની ટેક્સ્ટ ફાઇલ પર વાંચવા અને લખવાની કામગીરી કરવા માટેનો Java પ્રોગ્રામ લખો.



### Try-with-resources વાપરીને વધુ સારો વિકલ્ય:

```
// Try-with-resources
public static void writeToFileImproved(String fileName) \{
    try (FileWriter writer = new FileWriter(fileName)) \{
        writer.write("    method      !{n}");
        writer.write("Try-with-resources           .}{n}");
        System.out.println("Data       .");
    } catch (IOException e) \{
        System.out.println(" : " + e.getMessage());
    }
\}

public static void readFromFileImproved(String fileName) \{
    try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) \{
        String line;
        while ((line = reader.readLine()) != null) \{
            System.out.println(line);
        }
    } catch (IOException e) \{
        System.out.println(" : " + e.getMessage());
    }
\}
```

### File Operation ના ઘટકો:

- **FileWriter**: File માં character data લખવા માટે વપરાય છે
- **FileReader**: File માંથી character data વાંચવા માટે વપરાય છે
- **BufferedReader**: Buffering સાથે વધુ કાર્યક્ષમ વાંચવા માટે
- **Exception handling**: IOException અને FileNotFoundException
- **Resource management**: Memory leaks ટાળવા માટે streams બંધ કરવા

### File Handling ના પગલાં:

- **Stream બનાવો**: FileWriter/FileReader object
- **Operation કરો**: write()/readLine() methods
- **Exceptions handle કરો**: try-catch blocks
- **Resources બંધ કરો**: close() method અથવા try-with-resources

### મેમરી ટ્રીક

“Create Perform Handle Close”

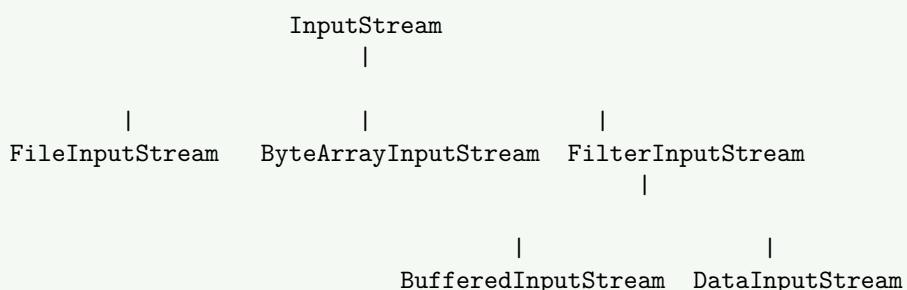
## પ્રશ્ન 5(અ OR) [3 ગુણ]

ઇનપુટ સ્ટ્રીમ સમજાવો.

### જવાબ

**InputStream વ્યાખ્યા:** InputStream એ abstract class છે જે bytes ના input stream ને represent કરે છે. તે input stream of bytes represent કરતા બધા classes નો superclass છે.

### InputStream Hierarchy:



## સામાન્ય InputStream Methods:

| Method       | વર્ણન                       | ઉદાહરણ                         |
|--------------|-----------------------------|--------------------------------|
| read()       | એક byte વાંચે છે            | int b = in.read();             |
| read(byte[]) | Bytes ને array માં વાંચે છે | in.read(buffer);               |
| available()  | ઉપલબ્ધ bytes પરત કરે છે     | int count =<br>in.available(); |
| close()      | Stream બંધ કરે છે           | in.close();                    |
| skip(long)   | નિર્દિષ્ટ bytes skip કરે છે | in.skip(10);                   |

## ઉપયોગનું ઉદાહરણ:

```
try (FileInputStream fis = new FileInputStream("data.txt")) {
    int data;
    while ((data = fis.read()) != {-}1) {
        System.out.print((char) data);
    }
} catch (IOException e) {
    e.printStackTrace();
}
```

## InputStream ની વિશેષતાઓ:

- Abstract class:** સીધા instantiate કરી શકતો નથી
- Byte-oriented:** Byte data handle કરે છે
- Input operations:** વિવિધ sources માંથી data વાંચવા માટે
- Resource management:** ઉપયોગ પછી બંધ કરવો જરૂરી

## મેમરી ટ્રીક

“Abstract Byte Input Resource”

## પ્રશ્ન 5(બ OR) [4 ગુણ]

Java માં પેકેજ વ્યાખ્યાયિત કરો. યોગ્ય સીનેક્સ અને એક ઉદાહરણ સાથે Java માં પેકેજ કેવી રીતે ઇમ્પ્લિમેન્ટ કરી શકાય તે લખો.

## જવાબ

**Package વ્યાખ્યા:** Java માં package એ namespace છે જે સંબંધિત classes અને interfaces ને સાથે organize કરે છે. તે access protection અને namespace management પ્રદાન કરે છે.

### Package Implementation Syntax:

```
// 1. Package declaration ( )
package packageName;

// 2. Import statements
import java.util.*;
import anotherPackage.ClassName;

// 3. Class definition
public class ClassName {
    // class body
}
```

### Package બનાવવાના પગલાં:

| પગલું              | કિયા                           | ઉદાહરણ          |
|--------------------|--------------------------------|-----------------|
| 1. Directory બનાવો | Package name સાથે folder બનાવો | mkdir mypackage |

- 2. Package declaration
- 3. Compile
- 4. Run

Package statement ઉમેરો  
યોગ્ય classpath સાથે compile કરો  
Fully qualified name સાથે run કરો

```
package mypackage;  
javac -d . ClassName.java  
java mypackage.ClassName
```

સંપૂર્ણ ઉદાહરણ:

**File: utilities/MathOperations.java**

```
package utilities;

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

    public static int multiply(int a, int b) {
        return a * b;
    }

    public static double calculateArea(double radius) {
        return 3.14 * radius * radius;
    }
}
```

**File: utilities/StringOperations.java**

```
package utilities;

public class StringOperations {
    public static String reverse(String str) {
        return new StringBuilder(str).reverse().toString();
    }

    public static boolean isPalindrome(String str) {
        String reversed = reverse(str);
        return str.equals(reversed);
    }
}
```

**File: TestPackage.java**

```
import utilities.MathOperations;
import utilities.StringOperations;

public class TestPackage {
    public static void main(String[] args) {
        // MathOperations
        int sum = MathOperations.add(10, 20);
        int product = MathOperations.multiply(5, 4);
        double area = MathOperations.calculateArea(7.0);

        System.out.println(" : " + sum);
        System.out.println(" : " + product);
        System.out.println(" : " + area);

        // StringOperations
        String original = "hello";
        String reversed = StringOperations.reverse(original);
        boolean isPalindrome = StringOperations.isPalindrome("madam");

        System.out.println(" : " + original);
        System.out.println(" : " + reversed);
        System.out.println(" {madam palindrome ? } + isPalindrome");
    }
}
```

**Package ના ફાયદા:**

- **Namespace management:** નામકરણની સમસ્યાઓ ટાળે છે

- **Access control:** Package-private access level
- **Code organization:** સંબંધિત functionality ને group કરે છે
- **પુનઃઉપયોગ:** Import અને વાપરવા માટે સરળ

## મેમરી ટ્રીક

“Namespace Access Organization Reusability”

## પ્રશ્ન 5(ક OR) [7 ગુણ]

List નો ઉપયોગ દર્શાવવા માટે Java મા પ્રોગ્રામ લખો .1) ArrayList બનાવો અને અઠવાડિયાના દિવસો ઉમેરો (સ્ટ્રિંગ સ્વરૂપમા) 2) LinkedList બનાવો અને મહિના ઉમેરો (સ્ટ્રીંગ સ્વરૂપમા) (બંને List ને ડિસ્પલે કરો.

### જવાબ

```
import java.util.*;

public class ListDemo {
    public static void main(String[] args) {
        //          ArrayList demonstrate
        demonstrateArrayList();

        System.out.println("{n}" + "=" .repeat(50) + "{n}");

        //          LinkedList demonstrate
        demonstrateLinkedList();

        System.out.println("{n}" + "=" .repeat(50) + "{n}");

        //      lists      comparison
        compareListOperations();
    }

    // ArrayList demonstrate      method
    public static void demonstrateArrayList() {
        System.out.println("ARRAYLIST DEMONSTRATION");
        System.out.println("=====");

        // ArrayList
        ArrayList<String> weekdays = new ArrayList<()>;

        //
        weekdays.add("    ");
        weekdays.add("    ");
        weekdays.add("    ");
        weekdays.add("    ");
        weekdays.add("    ");
        weekdays.add("    ");
        weekdays.add("    ");

        // ArrayList display
        System.out.println("ArrayList           :");
        for (int i = 0; i < weekdays.size(); i++) {
            System.out.println((i + 1) + ". " + weekdays.get(i));
        }

        // ArrayList specific operations
        System.out.println("{n}ArrayList Operations:");
    }
}
```

```

System.out.println(" : " + weekdays.size());
System.out.println(" : " + weekdays.get(0));
System.out.println(" : " + weekdays.get(weekdays.size() {-} 1));
System.out.println(" { : " + weekdays.contains(" "));

// Enhanced for loop
System.out.println("{n}Enhanced For Loop :");
for (String day : weekdays) {
    System.out.print(day + " ");
}
System.out.println();
}

// LinkedList demonstrate      method
public static void demonstrateLinkedList() {
    System.out.println("LINKEDLIST DEMONSTRATION");
    System.out.println("=====");

    // LinkedList
    LinkedList<String> months = new LinkedList();

    //
    months.add(" ");
    months.add(" ");

    // LinkedList display
    System.out.println("LinkedList :");
    for (int i = 0; i < months.size(); i++) {
        System.out.println((i + 1) + ". " + months.get(i));
    }
}

// LinkedList specific operations
System.out.println("{n}LinkedList Operations:");
System.out.println(" : " + months.size());
System.out.println(" : " + months.getFirst());
System.out.println(" : " + months.getLast());

//
months.addFirst(" ");
months.addLast(" ");

System.out.println("{n} :");
System.out.println(" element: " + months.getFirst());
System.out.println(" element: " + months.getLast());
System.out.println(" : " + months.size());

//   elements
months.removeFirst();
months.removeLast();

// Iterator

```

```

System.out.println("{n}Iterator      :");
Iterator<String> iterator = months.iterator();
while (iterator.hasNext()) \{
    System.out.print(iterator.next() + " ");
\}
System.out.println();
\}

// List operations      comparison      method
public static void compareListOperations() \{
    System.out.println("LIST COMPARISON");
    System.out.println("=====");
    // Sample data      lists
    ArrayList<String> arrayList = new ArrayList<()>;
    LinkedList<String> linkedList = new LinkedList<()>

    // Sample data
    String[] data = \{"A", "B", "C", "D", "E"\};

    for (String item : data) \{
        arrayList.add(item);
        linkedList.add(item);
    \}

    System.out.println("ArrayList: " + arrayList);
    System.out.println("LinkedList: " + linkedList);

    // Performance comparison info
    System.out.println("{n}Performance      : ");
    System.out.println("ArrayList {-      : Random access, Memory efficient"});
    System.out.println("LinkedList {-      : Insertion/Deletion, Dynamic size"});
\}
\}

```

### List Interface ની વિશેષતાઓ:

| વિશેષતા            | ArrayList           | LinkedList               |
|--------------------|---------------------|--------------------------|
| આંતરિક રચના        | Dynamic array       | Doubly linked list       |
| Access Time        | O(1) random access  | O(n) sequential access   |
| Insertion/Deletion | O(n) at middle      | O(1) at ends             |
| Memory             | ઓછી memory overhead | Pointers માટે વધુ memory |

### સામાન્ય List Methods:

- **add():** List માં element ઠોડે છે
- **get():** Index દ્વારા element મેળવે છે
- **size():** Elements ની સંખ્યા પરત કરે છે
- **contains():** Element અસ્તિત્વ ચકાસે છે
- **remove():** Element દૂર કરે છે
- **iterator():** Traversal માટે iterator પરત કરે છે

### List ના ફાયદા:

- **Dynamic size:** આપોઆપ વધે અને ઘટે છે
- **Ordered collection:** Insertion order જાળવે છે
- **Duplicate elements:** Duplicate values ની મંજૂરી આપે છે
- **Index-based access:** Position દ્વારા elements access કરો

### મેમરી ટ્રીક

“Dynamic Ordered Duplicate Index-based”