```java
// Filename: MethodOverloadingExample.java

public class MethodOverloadingExample {

    // Method with two integer parameters
    public int add(int a, int b) {
        return a + b;
    }

    // Method with three double parameters
    public double add(double a, double b, double c) {
        return a + b + c;
    }

    // Method with two strings as parameters
    public String concatenateStrings(String str1, String str2) {
        return str1 + str2;
    }

    public static void main(String[] args) {
        MethodOverloadingExample example = new MethodOverloadingExample();

        // Using the add method with integers
        int resultInt = example.add(5, 10);
        System.out.println("Sum of integers: " + resultInt);

        // Using the add method with doubles
        double resultDouble = example.add(2.5, 3.5, 1.0);
        System.out.println("Sum of doubles: " + resultDouble);

        // Using the concatenateStrings method
        String concatenatedString = example.concatenateStrings("Hello, ",
"World!");
        System.out.println("Concatenated String: " + concatenatedString);
    }
}
// Filename: ConstructorOverloadingExample.java

public class ConstructorOverloadingExample {

    private int intValue;
    private double doubleValue;
    private String stringValue;

    // Constructor with no parameters
    public ConstructorOverloadingExample() {
        System.out.println("Default Constructor");
    }

    // Constructor with one integer parameter
    public ConstructorOverloadingExample(int value) {
        this.intValue = value;
        System.out.println("Constructor with int parameter: " + intValue);
    }

    // Constructor with one double parameter
    public ConstructorOverloadingExample(double value) {
        this.doubleValue = value;
        System.out.println("Constructor with double parameter: " + doubleValue);
    }

    // Constructor with one string parameter
    public ConstructorOverloadingExample(String value) {
        this.stringValue = value;
        System.out.println("Constructor with String parameter: " + stringValue);
    }

    public static void main(String[] args) {
        // Creating objects using different constructors
        ConstructorOverloadingExample defaultConstructor = new
ConstructorOverloadingExample();
        ConstructorOverloadingExample intConstructor = new
ConstructorOverloadingExample(42);
        ConstructorOverloadingExample doubleConstructor = new
ConstructorOverloadingExample(3.14);
        ConstructorOverloadingExample stringConstructor = new
ConstructorOverloadingExample("Hello, Constructor!");

        // Accessing values in the objects
        System.out.println("Values in intConstructor: " + intConstructor.intValue);
        System.out.println("Values in doubleConstructor: " +
doubleConstructor.doubleValue);
        System.out.println("Values in stringConstructor: " +
stringConstructor.stringValue);
    }
}
// File: OOPJCompilerExample.java

// Base class with method overloading
class Calculator {
    public int add(int a, int b) {
        return a + b;
    }

    public double add(double a, double b) {
        return a + b;
    }
}

// Subclass with multilevel inheritance
class AdvancedCalculator extends Calculator {
    public int multiply(int a, int b) {
        return a * b;
    }
}

// Another subclass with hierarchical inheritance
class ScientificCalculator extends Calculator {
    public double power(double base, double exponent) {
        return Math.pow(base, exponent);
    }
}

// Main class to demonstrate the program
public class OOPJCompilerExample {
    public static void main(String[] args) {
        // Method overloading example
        Calculator basicCalculator = new Calculator();
        int resultInt = basicCalculator.add(5, 10);
        double resultDouble = basicCalculator.add(3.5, 7.2);

        System.out.println("Method Overloading:");
        System.out.println("Integer Addition Result: " + resultInt);
        System.out.println("Double Addition Result: " + resultDouble);

        // Multilevel inheritance example
        AdvancedCalculator advancedCalculator = new AdvancedCalculator();
        int resultMultiply = advancedCalculator.multiply(4, 5);

        System.out.println("\nMultilevel Inheritance:");
        System.out.println("Multiplication Result: " + resultMultiply);

        // Hierarchical inheritance example
        ScientificCalculator scientificCalculator = new ScientificCalculator();
        double resultPower = scientificCalculator.power(2, 3);

        System.out.println("\nHierarchical Inheritance:");
        System.out.println("Power Result: " + resultPower);
    }
}
// File: shapes/Circle.java
package shapes;

public class Circle {
    private double radius;

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

    public double calculateArea() {
        return Math.PI * radius * radius;
    }

    // Method Overloading
    public double calculateArea(double scaleFactor) {
        return scaleFactor * Math.PI * radius * radius;
    }
}

// File: shapes/Rectangle.java
package shapes;

public class Rectangle {
    protected double length;
    protected double width;

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

    protected double calculateArea() {
        return length * width;
    }
}
```

```
// File: Main.java
import shapes.Circle;
import shapes.Rectangle;

public class Main {
    public static void main(String[] args) {
        // Creating objects and demonstrating method overloading
        Circle circle = new Circle(5.0);
        System.out.println("Area of Circle: " + circle.calculateArea());
        System.out.println("Area of Circle with scale factor: " +
circle.calculateArea(2.0));

        Rectangle rectangle = new Rectangle(4.0, 6.0);
        System.out.println("Area of Rectangle: " + rectangle.calculateArea());

        // Accessing members with different access modifiers
        System.out.println("\nAccessing members with different access modifiers:");

        // Public members can be accessed from any class
        System.out.println("Public access - Radius of Circle: " + circle.radius);

        // Protected members can be accessed within the same package and subclasses
        System.out.println("Protected access - Length of Rectangle: " +
rectangle.length);
        System.out.println("Protected access - Width of Rectangle: " +
rectangle.width);

        // Private members can only be accessed within the same class
        // Uncommenting the line below will result in a compilation error
        // System.out.println("Private access - Radius of Circle: " +
circle.radius);
    }
}
// Interface1.java
interface Interface1 {
    void method1();
}

// Interface2.java
interface Interface2 {
    void method2();
}

// MultipleInheritance.java
class MultipleInheritance implements Interface1, Interface2 {
    public void method1() {
        System.out.println("Method from Interface1");
    }

    public void method2() {
        System.out.println("Method from Interface2");
    }

    public void additionalMethod() {
        System.out.println("Additional method in the class");
    }

    public static void main(String[] args) {
        MultipleInheritance obj = new MultipleInheritance();
        // Calling methods from interfaces
        obj.method1();
        obj.method2();

        // Calling additional method in the class
        obj.additionalMethod();
    }
}
public class StringHandlingExample {

    public static void main(String[] args) {
        // Initialize a sample string
        String myString = "   Hello, World!   ";

        // 1. charAt(int index)
        char charAtIndex = myString.charAt(7);
        System.out.println("Character at index 7: " + charAtIndex);

        // 2. length()
        int lengthOfString = myString.length();
        System.out.println("Length of the string: " + lengthOfString);

        // 3. substring(int beginIndex)
        String subString = myString.substring(7);
        System.out.println("Substring starting from index 7: " + subString);

        // 4. substring(int beginIndex, int endIndex)
        String subStringRange = myString.substring(7, 12);
        System.out.println("Substring from index 7 to 11: " + subStringRange);

        // 5. concat(String str)
        String concatenatedString = myString.concat(" This is additional
content.");
        System.out.println("Concatenated String: " + concatenatedString);

        // 6. indexOf(String str)
        int indexOfWorld = myString.indexOf("World");
        System.out.println("Index of 'World': " + indexOfWorld);

        // 7. lastIndexOf(String str)
        int lastIndexOfSpace = myString.lastIndexOf(" ");
        System.out.println("Last Index of space: " + lastIndexOfSpace);

        // 8. equals(Object obj)
        boolean isEqualToHello = myString.equals("Hello");
        System.out.println("Is the string equal to 'Hello'? " + isEqualToHello);

        // 9. equalsIgnoreCase(String str)
        boolean isEqualToHelloIgnoreCase = myString.equalsIgnoreCase("hello");
        System.out.println("Is the string equal to 'hello' (case-insensitive)? " +
isEqualToHelloIgnoreCase);

        // 10. startsWith(String prefix)
        boolean startsWithHello = myString.startsWith("Hello");
        System.out.println("Does the string start with 'Hello'? " +
startsWithHello);

        // 11. endsWith(String suffix)
        boolean endsWithExclamation = myString.endsWith("!");
        System.out.println("Does the string end with an exclamation mark? " +
endsWithExclamation);

        // 12. replace(char oldChar, char newChar)
        String replacedString = myString.replace('o', '0');
        System.out.println("String after replacing 'o' with '0': " +
replacedString);

        // 13. replaceAll(String regex, String replacement)
        String regexReplacedString = myString.replaceAll("[aeiouAEIOU]", "*");
        System.out.println("String after replacing vowels with '*': " +
regexReplacedString);

        // 14. toLowerCase()
        String lowerCaseString = myString.toLowerCase();
        System.out.println("String in lowercase: " + lowerCaseString);

        // 15. toUpperCase()
        String upperCaseString = myString.toUpperCase();
        System.out.println("String in uppercase: " + upperCaseString);

        // 16. toCharArray()
        char[] charArray = myString.toCharArray();
        System.out.print("Character array representation: ");
        for (char c : charArray) {
            System.out.print(c + " ");
        }
        System.out.println();

        // 17. isEmpty()
        boolean isEmpty = myString.isEmpty();
        System.out.println("Is the string empty? " + isEmpty);

        // 18. trim()
        String trimmedString = myString.trim();
        System.out.println("String after trimming leading and trailing whitespaces:
'" + trimmedString + "'");

        // 19. valueOf(primitive data type or Object)
        String valueOfString = String.valueOf(123);
        System.out.println("String representation of the integer 123: " +
valueOfString);

        // 20. compareTo(String anotherString)
        int comparisonResult = myString.compareTo("Hello, Universe!");
        System.out.println("Comparison result with 'Hello, Universe!': " +
comparisonResult);

        // 21. compareToIgnoreCase(String anotherString)
        int comparisonResultIgnoreCase = myString.compareToIgnoreCase("hello,
universe!");
        System.out.println("Case-insensitive comparison result with 'hello,
universe!': " + comparisonResultIgnoreCase);

        // 22. contains(CharSequence sequence)
        boolean containsSequence = myString.contains("lo");
        System.out.println("Does the string contain the sequence 'lo'? " +
containsSequence);
```

```java
    }
}
// Custom exception class
class InvalidAgeException extends Exception {
    public InvalidAgeException(String message) {
        super(message);
    }
}

// Main class demonstrating built-in and custom exceptions
public class ExceptionExample {
    // Method to check if the age is valid
    private static void validateAge(int age) throws InvalidAgeException {
        if (age < 0 || age > 120) {
            throw new InvalidAgeException("Invalid age: " + age);
        }
    }

    public static void main(String[] args) {
        try {
            // Example of using a built-in exception (ArithmeticException)
            int result = 10 / 0; // This will throw ArithmeticException

            // Example of using a custom exception
            int age = -5;
            validateAge(age);
        } catch (ArithmeticException e) {
            System.out.println("Caught ArithmeticException: " + e.getMessage());
        } catch (InvalidAgeException e) {
            System.out.println("Caught InvalidAgeException: " + e.getMessage());
        } catch (Exception e) {
            System.out.println("Caught generic exception: " + e.getMessage());
        }
    }
}
```

```java
// Custom exception class
class InvalidAgeException extends Exception {
    public InvalidAgeException(String message) {
        super(message);
    }
}

// Main class demonstrating built-in and custom exceptions
public class ExceptionExample {
    // Method to check if the age is valid
    private static void validateAge(int age) throws InvalidAgeException {
        if (age < 0 || age > 120) {
            throw new InvalidAgeException("Invalid age: " + age);
        }
    }

    public static void main(String[] args) {
        try {
            // Example of using a built-in exception (ArithmeticException)
            int result = 10 / 0; // This will throw ArithmeticException

            // Example of using a custom exception
            int age = -5;
            validateAge(age);
        } catch (ArithmeticException e) {
            System.out.println("Caught ArithmeticException: " + e.getMessage());
        } catch (InvalidAgeException e) {
            System.out.println("Caught InvalidAgeException: " + e.getMessage());
        } catch (Exception e) {
            System.out.println("Caught generic exception: " + e.getMessage());
        }
    }
}
// Custom exception class
class NegativeNumberException extends Exception {
    public NegativeNumberException(String message) {
        super(message);
    }
}

// Main class demonstrating built-in and custom exceptions
public class ExceptionExample2 {
    // Method to process an array and throw ArrayIndexOutOfBoundsException
    private static void processArray(int[] array, int index) {
        if (index < 0 || index >= array.length) {
            throw new ArrayIndexOutOfBoundsException("Invalid array index: " +
index);
        }
        System.out.println("Array element at index " + index + ": " +
array[index]);
    }

    // Method to check if a number is negative and throw NegativeNumberException
    private static void checkNegativeNumber(int number) throws
NegativeNumberException {
        if (number < 0) {
            throw new NegativeNumberException("Negative number not allowed: " +
number);
        }
    }

    public static void main(String[] args) {
        try {
            // Example of using a built-in exception
(ArrayIndexOutOfBoundsException)
            int[] numbers = {1, 2, 3};
            processArray(numbers, 5);

            // Example of using a custom exception
            int negativeNumber = -10;
            checkNegativeNumber(negativeNumber);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Caught ArrayIndexOutOfBoundsException: " +
e.getMessage());
        } catch (NegativeNumberException e) {
            System.out.println("Caught NegativeNumberException: " +
e.getMessage());
        } catch (Exception e) {
            System.out.println("Caught generic exception: " + e.getMessage());
        }
    }
}
// Custom exception class
```

```
class CustomValidationException extends Exception {
    public CustomValidationException(String message) {
        super(message);
    }
}

// Main class demonstrating built-in and custom exceptions
public class ExceptionExample3 {
    // Method to parse a string as an integer and throw NumberFormatException
    private static void parseAndPrint(String str) throws NumberFormatException {
        int number = Integer.parseInt(str);
        System.out.println("Parsed number: " + number);
    }

    // Method to validate a custom condition and throw CustomValidationException
    private static void validateCondition(boolean condition) throws
CustomValidationException {
        if (!condition) {
            throw new CustomValidationException("Custom condition not met");
        }
    }

    public static void main(String[] args) {
        try {
            // Example of using a built-in exception (NumberFormatException)
            String invalidNumber = "abc";
            parseAndPrint(invalidNumber);

            // Example of using a custom exception
            boolean customCondition = false;
            validateCondition(customCondition);
        } catch (NumberFormatException e) {
            System.out.println("Caught NumberFormatException: " + e.getMessage());
        } catch (CustomValidationException e) {
            System.out.println("Caught CustomValidationException: " +
e.getMessage());
        } catch (Exception e) {
            System.out.println("Caught generic exception: " + e.getMessage());
        }
    }
}
class SharedResource {
    private int data;
    private boolean dataProduced;

    public synchronized void produce(int value) {
        while (dataProduced) {
            try {
                wait(); // Wait until the consumer consumes the data
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
        data = value;
        System.out.println("Produced: " + data);
        dataProduced = true;
        notify(); // Notify the consumer that data is ready
    }
```

```java
    public synchronized int consume() {
        while (!dataProduced) {
            try {
                wait(); // Wait until the producer produces data
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
        System.out.println("Consumed: " + data);
        dataProduced = false;
        notify(); // Notify the producer that data has been consumed
        return data;
    }
}

class Producer extends Thread {
    private SharedResource sharedResource;

    public Producer(SharedResource sharedResource) {
        this.sharedResource = sharedResource;
    }

    @Override
    public void run() {
        for (int i = 1; i <= 5; i++) {
            sharedResource.produce(i);
            try {
                Thread.sleep(1000); // Simulate some time to produce data
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
    }
}

class Consumer extends Thread {
    private SharedResource sharedResource;

    public Consumer(SharedResource sharedResource) {
        this.sharedResource = sharedResource;
    }

    @Override
    public void run() {
        for (int i = 1; i <= 5; i++) {
            sharedResource.consume();
            try {
                Thread.sleep(1000); // Simulate some time to consume data
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
    }
}

public class Main {
    public static void main(String[] args) {
        SharedResource sharedResource = new SharedResource();
        Producer producer = new Producer(sharedResource);
        Consumer consumer = new Consumer(sharedResource);

        producer.start();
        consumer.start();
    }
}

//OneAdd.java
import java.util.*;
class OneAdd
{
  public static List<Integer> addOne(List<Integer> A)
  {
      int n= A.size()-1;
      A.set(n,A.get(n)+1);
      for(int i=n; i>0&& A.get(i)==10; --i)
      {
            A.set(i,0);
            A.set(i-1,A.get(i-1)+1);
      }
      if(A.get(0)==10)
      {
            A.set(0,0);
              A.add(0,1);
      }
      return A;
  }
  public static void main(String args[ ])
  {
      Scanner sc=new Scanner(System.in);
      System.out.println("Enter a number:");
      String num=sc.nextLine();
      List <Integer> al= new ArrayList<Integer>();
      for(int i=0;i<num.length();i++)
      {
            //al.add((int)num.charAt(i)-48);
            al.add(Integer.parseInt(String.valueOf(num.charAt(i))));
      }
      System.out.println(addOne(al));
        System.out.print("Result after adding one is: ");
      for(int st:al)
            System.out.print(st);
      System.out.println();
  }
}
//PostfixEval.java
/*Write a program that takes an arithmetic expression in Reverse Polish
Noatation and returns the number that the expression evaluates to.
 Process subexpressions, keeping values in a stack. How should operators be
handled?
 For example "3,4, +, 2, *,1, +"  The ordinary form for this is (3 + 4) * 2 + 1.*/

import java.util.*;
class PostfixEval
{
  public static int evaluate(String pexp)
  {
      Deque<Integer> res = new LinkedList<Integer>();
      String symbols[ ] = pexp.split(",");
      for (String token : symbols)
      {
            if("+-/*".contains(token))
            {
                  int y = res.removeFirst();
                  int x = res.removeFirst();
                  switch (token.charAt(0))
                  {
                        case '+': res.addFirst(x + y); break ;
                        case '-': res.addFirst(x - y); break ;
                        case '*': res.addFirst(x * y); break ;
                        case '/': res.addFirst(x / y); break ;
                  }
            }
            else
            {
                  res.addFirst(Integer.parseInt(token));
            }
      }
      return res.removeFirst();
  }
  public static void main(String args[])
  {
      Scanner sc=new Scanner(System.in);
      System.out.println("Enter a Postfix Expression:");
      String st=sc.nextLine();
      System.out.println("The result of evaluated postfix expression is:"+
evaluate(st));
  }
}

/*  Input a number 'N' and an array as given below.
Input:N = 2   Array =1,2,3,3,4,4
O/p : 2
Find the least number of unique elements after deleting N numbers of elements from
the array.
In the above example, after deleting N=2 elements from the array.
In above 1,2 will be deleted.
So 3,3,4,4 will be remaining so,
2 unique elements are in the array i.e 3 and 4. */


//import java.lang.*;
import java.io.*;
import java.util.*;

class UniqueElements
{
  public static void main(String args[ ])throws IOException
  {
       BufferedReader br= new BufferedReader(new InputStreamReader(System.in));
       int n= Integer.parseInt(br.readLine());
       String s[ ]= br.readLine().split(",");
         int n1=s.length;
       HashSet<Integer> hs= new HashSet<Integer>();
       for(int i=0; i<n1; i++)
             hs.add(Integer.parseInt(s[i]));
       System.out.println(hs.size()-n);
```

```
    }
}
```