1 . Implement Addition and Subtraction methods to demonstrate overloaded methods and constructors.

Here is one possible way to implement addition and subtraction methods to demonstrate overloaded methods and constructors:

// A class that represents a mathematical expression

class Expression {

// Instance variables to store the operands and the operator

private double operand1;

private double operand2;

private char operator;

// A constructor that takes two operands and an operator as parameters

public Expression(double operand1, double operand2, char operator) {

this.operand1 = operand1;

this.operand2 = operand2;

this.operator = operator;

}

// A constructor that takes a single operand as a parameter and assumes the operator is '+'

public Expression(double operand1) {

this(operand1, 0, '+');

}

// A constructor that takes no parameters and sets the default values to zero and '+'

public Expression() {

this(0, 0, '+');

}

// A method that performs the addition operation on the operands and returns the result

public double add() {

return operand1 + operand2;

}

// A method that performs the subtraction operation on the operands and returns the result

public double subtract() {

return operand1 - operand2;

}

// An overloaded method that performs the addition operation on two given parameters and returns the result

public double add(double x, double y) {

return x + y;

}

// An overloaded method that performs the subtraction operation on two given parameters and returns the result

public double subtract(double x, double y) {

return x - y;

}

// A method that evaluates the expression based on the operator and returns the result

public double evaluate() {

switch (operator) {

case '+':

return add();

case '-':

return subtract();

default:

return Double.NaN; // Not a number

}

}

// A method that returns a string representation of the expression

public String toString() {

return operand1 + " " + operator + " " + operand2;

}

}

// A class to test the Expression class

class TestExpression {

public static void main(String[] args) {

// Create some expressions using different constructors

Expression e1 = new Expression(5, 3, '-');

Expression e2 = new Expression(7);

Expression e3 = new Expression();

// Print the expressions and their evaluations

System.out.println(e1 + " = " + e1.evaluate());

System.out.println(e2 + " = " + e2.evaluate());

System.out.println(e3 + " = " + e3.evaluate());

// Use the overloaded methods to perform some calculations

System.out.println("10 + 5 = " + e1.add(10, 5));

System.out.println("10 - 5 = " + e1.subtract(10, 5));

}

}

2.What restrictions are placed on method overloading?

[Method overloading is a technique that allows a class to have more than one method with the same name but different parameters1](https://www.geeksforgeeks.org/method-overloading-in-java/). Some of the restrictions that are placed on method overloading are:

* The methods must have different parameter lists. The parameter list can differ by the number, type, or order of the parameters[1](https://www.geeksforgeeks.org/method-overloading-in-java/)[2](https://www.baeldung.com/java-method-overload-override).
* The methods cannot be overloaded by changing only the return type. This would cause ambiguity for the compiler[3](https://www.java67.com/2012/09/what-is-rules-of-overloading-and-overriding-in-java.html)[4](https://www.freecodecamp.org/news/method-overloading-and-overriding-in-java/).
* The methods cannot be overloaded by changing only the access modifier. This would also cause ambiguity for the compiler[3](https://www.java67.com/2012/09/what-is-rules-of-overloading-and-overriding-in-java.html).
* The methods cannot be overloaded by declaring different exceptions. This would not affect the method signature[3](https://www.java67.com/2012/09/what-is-rules-of-overloading-and-overriding-in-java.html).
* The methods must be in the same class or subclass. Method overloading is not applicable across classes[3](https://www.java67.com/2012/09/what-is-rules-of-overloading-and-overriding-in-java.html).

These restrictions ensure that the compiler can resolve the correct method to call based on the arguments passed at compile time. Method overloading is also known as static polymorphism or compile-time polymorphism

3.How are this() and super() used with constructors?

[The this() and super() keywords are used to call different constructors of the same or parent class respectively1](https://www.geeksforgeeks.org/super-and-this-keywords-in-java/)[2](https://stackoverflow.com/questions/10381244/why-cant-this-and-super-both-be-used-together-in-a-constructor). They can be used with constructors for various purposes, such as:

* [To reuse the common code of different constructors by calling one constructor from another1](https://www.geeksforgeeks.org/super-and-this-keywords-in-java/)[2](https://stackoverflow.com/questions/10381244/why-cant-this-and-super-both-be-used-together-in-a-constructor).
* To initialize the parent class fields or invoke the parent class methods from the subclass constructor[1](https://www.geeksforgeeks.org/super-and-this-keywords-in-java/)[3](https://www.tutorialspoint.com/Can-you-use-both-this-and-super-in-a-constructor-in-Java).
* [To resolve the name conflicts between the parent class and subclass fields or methods1](https://www.geeksforgeeks.org/super-and-this-keywords-in-java/)[4](https://www.javatpoint.com/this-vs-super-in-java).

However, there are some rules that must be followed when using this() and super() with constructors:

* [They must be the first statement in the constructor2](https://stackoverflow.com/questions/10381244/why-cant-this-and-super-both-be-used-together-in-a-constructor)[3](https://www.tutorialspoint.com/Can-you-use-both-this-and-super-in-a-constructor-in-Java).
* They cannot be used together in the same constructor[2](https://stackoverflow.com/questions/10381244/why-cant-this-and-super-both-be-used-together-in-a-constructor)[5](https://stackoverflow.com/questions/4023344/difference-between-this-andsuper-keywords-in-java).
* They cannot be used in a static context[2](https://stackoverflow.com/questions/10381244/why-cant-this-and-super-both-be-used-together-in-a-constructor) .

For example, consider the following code that illustrates the use of this() and super() with constructors:

// A parent class

class Person {

// A field to store the name

private String name;

// A constructor that takes a name as a parameter

public Person(String name) {

this.name = name;

}

// A method that returns the name

public String getName() {

return name;

}

}

// A subclass of Person

class Student extends Person {

// A field to store the roll number

private int rollNo;

// A constructor that takes a name and a roll number as parameters

public Student(String name, int rollNo) {

super(name); // calling the parent class constructor

this.rollNo = rollNo;

}

// A method that returns the roll number

public int getRollNo() {

return rollNo;

}

}

// A class to test the Student class

class TestStudent {

public static void main(String[] args) {

// Create a student object using the constructor

Student s = new Student("Alice", 101);

// Print the student details using the inherited and defined methods

System.out.println("Name: " + s.getName());

System.out.println("Roll No: " + s.getRollNo());

}

}

4.What is method overloading?

[Method overloading is a feature of Java that allows a class to have more than one method with the same name but different parameters1](https://www.javatpoint.com/method-overloading-in-java)[2](https://www.geeksforgeeks.org/method-overloading-in-java/). [It is also known as compile-time polymorphism, static polymorphism, or early binding2](https://www.geeksforgeeks.org/method-overloading-in-java/)[3](https://www.w3schools.in/java/method-overloading).

Method overloading can be useful for performing similar tasks with different inputs or outputs. [For example, we can overload a method named add to perform addition of two or more numbers of different types1](https://www.javatpoint.com/method-overloading-in-java)[2](https://www.geeksforgeeks.org/method-overloading-in-java/).

Some of the rules for method overloading are:

* The methods must have different parameter lists. [The parameter list can differ by the number, type, or order of the parameters1](https://www.javatpoint.com/method-overloading-in-java)[2](https://www.geeksforgeeks.org/method-overloading-in-java/).
* The methods cannot be overloaded by changing only the return type. This would cause ambiguity for the compiler[4](https://www.mygreatlearning.com/blog/method-overloading-in-java/).
* The methods must be in the same class or subclass. Method overloading is not applicable across classes.

Method overloading is resolved at compile time based on the arguments passed to the method call. [The compiler chooses the most specific method that matches the arguments](https://www.geeksforgeeks.org/method-overloading-in-java/)

**5.Write a java program for subsidy calculation for citizen given by government during floods. The attributes for citizen name, address, adhar no, subsidy amount. There are three types normal citizen, senior citizen, physically challenged citizen. Senior citizen has further attribute age. subsidy is calculated based on Age range 65 to 75 10000 Rs, 75 and above 15000Rs.For , physically challenged citizen further  attribute added  level of physically challenged values are 1,2 or 3. . subsidy is calculated based on level .For Level 1 25000 Rs  Level 2 40000 Rs  Level 3 50000 Rs .For normal citizen flat 5000 Rs .Implement runtime polymorphism by using dynamic dispatch method calculateSubsidy() being abstract method in Citizen class.And Use respective Constructors.**

/ A class to represent a citizen

abstract class Citizen {

// Attributes for citizen name, address, adhar no, subsidy amount

String name;

String address;

String adharNo;

int subsidyAmount;

// A constructor to initialize the attributes

public Citizen(String name, String address, String adharNo) {

this.name = name;

this.address = address;

this.adharNo = adharNo;

this.subsidyAmount = 0;

}

// An abstract method to calculate the subsidy amount

public abstract void calculateSubsidy();

}

// A class to represent a senior citizen

class SeniorCitizen extends Citizen {

// An additional attribute for age

int age;

// A constructor to initialize the attributes

public SeniorCitizen(String name, String address, String adharNo, int age) {

super(name, address, adharNo); // Calling the superclass constructor

this.age = age;

}

// A method to calculate the subsidy amount based on age range

public void calculateSubsidy() {

if (age >= 65 && age < 75) {

subsidyAmount = 10000; // Rs. 10000 for age range 65 to 75

} else if (age >= 75) {

subsidyAmount = 15000; // Rs. 15000 for age range 75 and above

}

}

}

// A class to represent a physically challenged citizen

class PhysicallyChallengedCitizen extends Citizen {

// An additional attribute for level of physically challenged

int level;

// A constructor to initialize the attributes

public PhysicallyChallengedCitizen(String name, String address, String adharNo, int level) {

super(name, address, adharNo); // Calling the superclass constructor

this.level = level;

}

// A method to calculate the subsidy amount based on level

public void calculateSubsidy() {

if (level == 1) {

subsidyAmount = 25000; // Rs. 25000 for level 1

} else if (level == 2) {

subsidyAmount = 40000; // Rs. 40000 for level 2

} else if (level == 3) {

subsidyAmount = 50000; // Rs.50000 for level 3

}

}

}

// A class to represent a normal citizen

class NormalCitizen extends Citizen {

// A constructor to initialize the attributes

public NormalCitizen(String name, String address, String adharNo) {

super(name, address, adharNo); // Calling the superclass constructor

}

// A method to calculate the subsidy amount as flat Rs.5000

public void calculateSubsidy() {

subsidyAmount = 5000;

}

}

// A class to test the program

class Test {

public static void main(String[] args) {

// Creating an array of citizen objects of different types

Citizen[] citizens = new Citizen[4];

citizens[0] = new SeniorCitizen("Rajesh", "Delhi", "1234567890",67);

citizens[1] = new PhysicallyChallengedCitizen("Sunita", "Mumbai", "2345678901",2);

citizens[2] = new NormalCitizen("Ravi", "Chennai", "3456789012");

citizens[3] = new SeniorCitizen("Kamala", "Kolkata", "4567890123",76);

// Looping through the array and calling the calculateSubsidy method for each object using dynamic dispatch

for (int i =0; i < citizens.length; i++) {

citizens[i].calculateSubsidy();

}

// Displaying the details of each citizen along with their subsidy amount

System.out.println("Name\tAddress\tAdhar No\tSubsidy Amount");

for (int i =0; i < citizens.length; i++) {

System.out.println(citizens[i].name + "\t" + citizens[i].address + "\t" + citizens[i].adharNo + "\t" + citizens[i].subsidyAmount);

}

}

}