1)

```
import java.util.Arrays;

public class ThirdLargest {

  public static int findThirdLargest(int[] arr) {

  if (arr.length < 3) {

    return -1;   // Indicates the array does not have enough elements

  } // Sort the array in ascending order

    Arrays.sort(arr);   // Return the third largest number

    return arr[arr.length - 3];

  }

  public static void main(String[] args)  {

    int[] arr = {10, 4, 3, 50, 23, 90};

    if (arr.length < 3)  {

      System.out.println("Array must have at least three elements.");

    }

    else {

      int thirdLargest = findThirdLargest(arr);

      System.out.println("The third largest number is: " + thirdLargest);

    }

}

}
```

2)

```
import java.util.ArrayList;


public class DuplicateElements {
    public static void main(String[] args) {
        // Example ArrayList
        ArrayList<Integer> list = new ArrayList<>();
        list.add(10);
        list.add(20);
        list.add(30);
        list.add(10);
        list.add(40);
        list.add(20);

        System.out.println("Duplicate elements:");

        // Create a new ArrayList to track already printed duplicates
        ArrayList<Integer> printedDuplicates = new ArrayList<>();

        for (int i = 0; i < list.size(); i++) {
            for (int j = i + 1; j < list.size(); j++) {
                if (list.get(i).equals(list.get(j)) && !printedDuplicates.contains(list.get(i))) {
                    System.out.println(list.get(i));
                    printedDuplicates.add(list.get(i)); // Track the duplicate
                    break;
         // Avoid further checks for this duplicate
                }
            }
        }
    }
}
```

3)

```
import java.util.Arrays;

public class FirstKSmallElements {
    public static void main(String[] args) {
        int[] numbers = {6, 7, 4, 5, 9, 8, 3};
        int k = 2;

        // Stream to sort and get the first k smallest elements
        Arrays.stream(numbers)
              .sorted()       // Sort the array in ascending order
              .limit(k)       // Limit to the first k elements
              .forEach(System.out::println); // Print each element
    }
}
```

4)

```
 class Desktop {
 private String brand, processor;
 private int ramSize;

 public Desktop(String brand, String processor, int ramSize) {
 this.brand = brand;
 this.processor = processor;
 this.ramSize = ramSize;
 }

 public void upgradeRam(int additionalRam) {
 ramSize += additionalRam;
 }

 @Override
 public String toString() {
 return String.format("Desktop{brand='%s', processor='%s', ramSize=%d}", brand, processor,
ramSize);
 }
}

public class ComputerTest {
 public static void main(String[] args) {
 Desktop desktop = new Desktop("Dell", "Intel i7", 8);
 System.out.println(desktop);
 desktop.upgradeRam(4);
 System.out.println("After RAM upgrade: " + desktop);
 }
}
```

5)

```
import java.util.stream.IntStream;

public class PrimeSumCalculator {

public static void main(String[] args) {

                        int start = 10, end = 50; // Compute the sum of primes in the specified range

                        int sumOfPrimes = IntStream.rangeClosed(start, end)
.filter(PrimeSumCalculator::isPrime) .sum();

                        System.out.println("Total sum of prime numbers: " + sumOfPrimes);

            } // Helper method to determine if a number is prime

            private static boolean isPrime(int number) {

            if (number < 2) return false; // Numbers less than 2 are not prime

                        return IntStream.rangeClosed(2, (int) Math.sqrt(number)) .allMatch(divisor ->
number % divisor != 0);

}

}
```

6)

```
abstract class GeometricShape {
    abstract double calculateArea();
    abstract double calculatePerimeter();
}

class Triangle extends GeometricShape {
    private double base, height, sideA, sideB;

    public Triangle(double base, double height, double sideA, double sideB) {
        this.base = base;
        this.height = height;
        this.sideA = sideA;
        this.sideB = sideB;
    }

    @Override
    double calculateArea() {
        return 0.5 * base * height;
    }

    @Override
    double calculatePerimeter() {
        return base + sideA + sideB;
    }
}

class Square extends GeometricShape {
    private double sideLength;

    public Square(double sideLength) {
        this.sideLength = sideLength;
    }

    @Override
    double calculateArea() {
        return Math.pow(sideLength, 2);
    }

    @Override
    double calculatePerimeter() {
        return 4 * sideLength;
    }
}

public class GeometryDemo {
    public static void main(String[] args) {
        GeometricShape triangle = new Triangle(5, 3, 4, 6);
        GeometricShape square = new Square(4);

        System.out.println("Triangle - Area: " + triangle.calculateArea() +
                           ", Perimeter: " + triangle.calculatePerimeter());
        System.out.println("Square - Area: " + square.calculateArea() +
                           ", Perimeter: " + square.calculatePerimeter());
    }
}
```

8)

```
SELECT name, commission

FROM sales

WHERE city = 'Paris'

GROUP BY name, commission

HAVING commission > 0.10

ORDER BY commission DESC;
```

9)

```
CREATE VIEW paris_salespeople AS

SELECT name, commission

FROM sales

WHERE city = 'Paris'
```

10)

```
SELECT *

FROM sales

WHERE city = 'Rome' AND salesman_id > 5005 AND commission < 0.15;
```