1. ARITHMETIC EXCEPTION:

```
import java.lang.Math;

public class ArithmeticExceptionHandling {

    public static void main(String[] args) {
        // a) Divided by Zero
        try {
            int a = 10;
            int b = 0;
            int result = a / b;
            System.out.println("Result of division: " + result);
        } catch (ArithmeticException e) {
            System.out.println("Exception: Division by zero is not allowed.");
        }

        // b) Logarithm of negative or zero
        try {
            double num = -1;
            double logResult = Math.log(num);
            System.out.println("Logarithm result: " + logResult);
        } catch (ArithmeticException e) {
            System.out.println("Exception: Logarithm of negative number or zero is not
allowed.");
        } catch (Exception e) {
            System.out.println("Exception: Invalid logarithmic operation.");
        }

        // c) Tan 90 Degree
        try {
            double angle = 90.0;
            double radians = Math.toRadians(angle);
            double tanResult = Math.tan(radians);
            if (Double.isInfinite(tanResult)) {
                throw new ArithmeticException("Tangent of 90 degrees is undefined.");
            }
            System.out.println("Tangent of 90 degrees: " + tanResult);
        } catch (ArithmeticException e) {
            System.out.println("Exception: " + e.getMessage());
        }

        // d) Zero power Zero
        try {
            double base = 0;
            double exponent = 0;
            double powResult = Math.pow(base, exponent);
            if (Double.isNaN(powResult)) {
                throw new ArithmeticException("Zero raised to the power of zero is undefined.");
            }
            System.out.println("Zero power zero result: " + powResult);
        } catch (ArithmeticException e) {
            System.out.println("Exception: " + e.getMessage());
        }
    }
}
```

2. BOUND EXCEPTION

```
public class ArrayIndexOutOfBoundsHandling {

    public static void main(String[] args) {
        // a) Accessing an array element outside its bound
        try {
            int[] arr = {1, 2, 3, 4, 5};
            System.out.println("Accessing element at index 5: " + arr[5]); // Invalid index
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Exception: Accessing an element outside the array bounds.");
        }

        // b) Iterating beyond the array length
        try {
            int[] arr = {10, 20, 30, 40};
            for (int i = 0; i <= arr.length; i++) { // Invalid loop condition (should be i < arr.length)
                System.out.println("Array element: " + arr[i]);
            }
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Exception: Iterated beyond the array length.");
        }

        // c) Nested array and incorrect index
        try {
            int[][] nestedArray = {
                {1, 2, 3},
                {4, 5},
                {6, 7, 8, 9}
            };
            System.out.println("Accessing nested element [2][4]: " + nestedArray[2][4]); // Invalid
index
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Exception: Incorrect index in nested array.");
        }

        // d) Passing incorrect array to a method
        try {
            int[] arr = null;
            printArray(arr); // Passing a null array
        } catch (NullPointerException e) {
            System.out.println("Exception: Passed a null array to the method.");
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Exception: Array index out of bounds in the method.");
        }
    }

    // Method to print the array elements
    public static void printArray(int[] arr) {
        if (arr == null) {
            throw new NullPointerException("Array is null");
        }
        for (int i = 0; i < arr.length; i++) {
            System.out.println("Array element at index " + i + ": " + arr[i]);
        }
    }
}
```

3. NUMBER FORMAT EXCEPTION:

```
import java.text.DecimalFormat;
import java.text.NumberFormat;
import java.text.ParseException;
import java.util.Locale;
import java.util.Scanner;

public class NumberFormatExceptionHandling {

    public static void main(String[] args) {
        // a) Parsing a Non-Numeric String
        try {
            String nonNumeric = "ABC123";
            int num = Integer.parseInt(nonNumeric); // This will throw NumberFormatException
            System.out.println("Parsed number: " + num);
        } catch (NumberFormatException e) {
            System.out.println("Exception: Unable to parse non-numeric string.");
        }

        // b) Reading User Input Without Validation
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter a number: ");
        String userInput = scanner.nextLine();
        try {
            double userNumber = Double.parseDouble(userInput); // Can fail if input is
non-numeric
            System.out.println("User entered: " + userNumber);
        } catch (NumberFormatException e) {
            System.out.println("Exception: Invalid input. Please enter a valid numeric value.");
        }

        // c) Formatting Issues in “DecimalFormat”
        try {
            DecimalFormat df = new DecimalFormat("#.##");
            String invalidDecimal = "12.34.56"; // Invalid format
            Number parsedNumber = df.parse(invalidDecimal); // This will throw a
ParseException
            System.out.println("Parsed number: " + parsedNumber);
        } catch (ParseException e) {
            System.out.println("Exception: Invalid number format in decimal string.");
        }

        // d) Incorrectly using localized decimal separators
        try {
            String localizedDecimal = "1,234.56"; // US format
            NumberFormat nf = NumberFormat.getInstance(Locale.GERMANY); // Expecting
German format, which uses "," as a decimal point
            Number number = nf.parse(localizedDecimal); // Parsing using incorrect locale
            System.out.println("Parsed number (localized): " + number);
        } catch (ParseException e) {
            System.out.println("Exception: Incorrectly using localized decimal separators.");
        }
    }
}
```

4. ILLEGAL ARGUMENT EXCEPTION:

```
public class IllegalArgumentExceptionHandling {

   public static void main(String[] args) {
      // a) Negative Argument in the method requiring non-negative values
      try {
         calculateSquareRoot(-5); // Passing a negative value
      } catch (IllegalArgumentException e) {
         System.out.println("Exception: " + e.getMessage());
      }

      // b) Invalid Enum Constant Passed to a Method
      try {
         String invalidDay = "HOLIDAY";
         printDay(Day.valueOf(invalidDay)); // Passing an invalid enum constant
      } catch (IllegalArgumentException e) {
         System.out.println("Exception: Invalid enum constant passed.");
      }

      // c) Setting an Invalid Range for a Method Parameter
      try {
         setPercentage(150); // Passing a value out of the valid range (0-100)
      } catch (IllegalArgumentException e) {
         System.out.println("Exception: " + e.getMessage());
      }

      // d) Empty or Null String Argument in a Method Requiring Non-Empty String
      try {
         greetUser(""); // Passing an empty string
      } catch (IllegalArgumentException e) {
         System.out.println("Exception: " + e.getMessage());
      }
   }

   // a) Method requiring non-negative values
   public static double calculateSquareRoot(int number) {
      if (number < 0) {
         throw new IllegalArgumentException("Negative value: Square root of negative
numbers is not defined.");
      }
      return Math.sqrt(number);
   }

   // Enum for days of the week
   enum Day {
      MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY
```

```
    }

    // b) Method accepting Enum constants
    public static void printDay(Day day) {
        System.out.println("Today is: " + day);
    }

    // c) Method with a valid range check for percentage (0-100)
    public static void setPercentage(int percentage) {
        if (percentage < 0 || percentage > 100) {
            throw new IllegalArgumentException("Invalid percentage: " + percentage + ". It
should be between 0 and 100.");
        }
        System.out.println("Valid percentage: " + percentage);
    }

    // d) Method requiring non-empty string
    public static void greetUser(String name) {
        if (name == null || name.isEmpty()) {
            throw new IllegalArgumentException("Name cannot be null or empty.");
        }
        System.out.println("Hello, " + name + "!");
    }
}
```

5. CUSTOM EXCEPTION - NEGATIVE VALUES

```
import java.util.Scanner;

// Main class to handle the exception
public class CustomExceptionDemo {

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

        System.out.print("Enter a positive number: ");
        int number = scanner.nextInt();

        try {
            checkPositiveNumber(number);
            System.out.println("You entered: " + number);
        } catch (NegativeValueException e) {
            System.out.println("Exception: " + e.getMessage());
        }
    }

    // Method to check if the number is negative
    public static void checkPositiveNumber(int number) throws NegativeValueException {
        if (number < 0) {
            throw new NegativeValueException("Negative values are not allowed: " + number);
        }
    }
}
```