Exception Handling in Java

Part 1: Built-in Exception Handlers

Introduction to Built-in Exception Handlers:

Java provides several built-in exception handlers that can be used to handle common exceptions. These include `ArithmeticException`, `ArrayIndexOutOfBoundsException`, `ClassCastException`, `IllegalArgumentException`, `IndexOutOfBoundsException`, `NegativeArraySizeException`, `NullPointerException`, `NumberFormatException`, and `StringIndexOutOfBoundsException`.

Requirements:

1. Create a program that demonstrates the use of built-in exception handlers to handle the following exceptions:
   - `ArithmeticException`: Handle division by zero.
   - `ArrayIndexOutOfBoundsException`: Handle access to an array index that is out of bounds.
   - `ClassCastException`: Handle casting an object to an incompatible class.
   - `IllegalArgumentException`: Handle passing an invalid argument to a method.
2. Use `try-catch` blocks to catch and handle each exception.
3. Provide informative error messages to the user when an exception occurs.

Example Code:

```
// BuiltInExceptions.java
public class BuiltInExceptions {
    public static void main(String[] args) {
        // ArithmeticException
        try {
            int x = 10 / 0;
        } catch (ArithmeticException e) {
            System.out.println("ArithmeticException: " + e.getMessage());
        }

        // ArrayIndexOutOfBoundsException
        int[] array = new int[5];
        try {
            System.out.println(array[10]);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("ArrayIndexOutOfBoundsException: " + e.getMessage())
        }

        // ClassCastException
        Object obj = "Hello";
        try {
            int num = (int) obj;
```

```
        } catch (ClassCastException e) {
            System.out.println("ClassCastException: " + e.getMessage());
        }

        // IllegalArgumentException
        try {
            methodWithInvalidArgument(-1);
        } catch (IllegalArgumentException e) {
            System.out.println("IllegalArgumentException: " + e.getMessage());
        }
    }

    public static void methodWithInvalidArgument(int arg) {
        if (arg < 0) {
            throw new IllegalArgumentException("Argument cannot be negative");
        }
    }
}
```

Part 2: Custom Exception Handling

Objective:

In this lab, you will explore exception handling in Java. You will learn to use `try-catch` blocks, throw exceptions, and create custom exceptions.

Requirements:

1. Create a program that simulates a bank account system. Implement the following:
   - A class `BankAccount` with attributes `accountNumber`, `accountHolderName`, and `balance`.
   - Methods `deposit(double amount)` and `withdraw(double amount)`.
2. Handle exceptions in the `withdraw()` method to prevent overdrafts. Use a `try-catch` block for handling exceptions.
3. Create a custom exception class `InsufficientFundsException` and throw this exception when a withdrawal amount exceeds the balance.
4. Add input validation in `deposit()` to throw an `IllegalArgumentException` if the deposit amount is negative.
5. Write a `main` method to test your program, demonstrating exception handling and custom exception usage.

Exercises:

1. Add a `transfer(double amount, BankAccount recipient)` method to transfer money between accounts. Handle exceptions appropriately.
2. Enhance the program to log all transactions, including failed ones, to a file.

3. Create an option for users to retry a failed transaction using a loop and exception handling.

Hint:

```
// BankAccount.java
public class BankAccount {
    private String accountNumber;
    private String accountHolderName;
    private double balance;

    public BankAccount(String accountNumber, String accountHolderName,
                       double balance) {
        this.accountNumber = accountNumber;
        this.accountHolderName = accountHolderName;
        this.balance = balance;
    }

    public void deposit(double amount) {
        // TODO: Validate input and throw
        // IllegalArgumentException for negative amounts
    }

    public void withdraw(double amount)
                       throws InsufficientFundsException {
        // TODO: Implement withdrawal with exception handling
        // for insufficient balance
    }

    public void displayDetails() {
        // TODO: Print account details
    }
}

// Custom Exception
public class InsufficientFundsException extends Exception {
    public InsufficientFundsException(String message) {
        super(message);
    }
}

// TestBankAccount.java
public class TestBankAccount {
    public static void main(String[] args) {
        // TODO: Test deposit, withdrawal, and transfer
        // functionality with exception handling
    }
}
```