SHEET 6 SOLUTIONS

SHEHAB MAHMOUD SALAH | 2100320

GENERAL NOTE: to **copy** code **from** this PDF document, **copy each block of code** separately to **not** lose the code's formatting, alternatively you can find all the **source code** to **PROGRAMMING EXERCISES** on my **GitHub**: https://bit.ly/CSE231Sheets , **happy compiling!**

1. (**NumberMath.java** on GitHub)

```
import java.util.InputMismatchException;
import java.util.Scanner;

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

        while (true) {
            try {
                System.out.print("Enter the first integer: ");
                int num1 = scanner.nextInt();
                System.out.print("Enter the second integer: ");
                int num2 = scanner.nextInt();

                // Display the results of arithmetic operations
                System.out.println("Sum: " + calculator.add(num1, num2));
                System.out.println("Difference: " + calculator.subtract(num1,
num2));
                System.out.println("Product: " + calculator.multiply(num1,
num2));
                System.out.println("Quotient: " + calculator.divide(num1,
num2));
                System.out.println("Division Remainder: " +
calculator.remainder(num1, num2));

                break; // Exit loop after successful operations
            } catch (InputMismatchException e) {
                System.out.println("Invalid input. Please enter valid
integers.");
                scanner.nextLine(); // Clear the buffer
            } catch (ArithmeticException | DivisionByZeroException e) {
                System.out.println("Error: " + e.getMessage());
            }
        }

        scanner.close();
    }

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

    public int subtract(int a, int b) {
        return a - b;
    }

    public int multiply(int a, int b) {
        return a * b;
    }
```

```java
    public String divide(int a, int b) throws DivisionByZeroException {
        if (b == 0) {
            throw new DivisionByZeroException(a);
        }
        return String.valueOf(a / b);
    }

    public int remainder(int a, int b) throws DivisionByZeroException {
        if (b == 0) {
            throw new DivisionByZeroException(a, true);
        }
        return a % b;
    }

    public static class DivisionByZeroException extends Exception {
        public DivisionByZeroException(int a) {
            super((a == 0 ? "Undefined quantity" : (a > 0 ? "+infinity" : "-
infinity")));
        }

        public DivisionByZeroException(int a, boolean remainder) {
            super("Division remainder: 0");
        }
    }
}
```

(**DoubleNumberMath.java** on GitHub)

2.

```java
import java.util.InputMismatchException;
import java.util.Scanner;

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

        while (true) {
            try {
                System.out.print("Enter the first double: ");
                double num1 = scanner.nextDouble();
                System.out.print("Enter the second double: ");
                double num2 = scanner.nextDouble();

                // Display the results of arithmetic operations
                System.out.println("Sum: " + calculator.add(num1, num2));
                System.out.println("Difference: " + calculator.subtract(num1,
num2));
                System.out.println("Product: " + calculator.multiply(num1,
num2));
                System.out.println("Quotient: " + calculator.divide(num1,
num2));
                System.out.println("Division Remainder: " +
calculator.remainder(num1, num2));

                break; // Exit loop after successful operations
            } catch (InputMismatchException e) {
                System.out.println("Invalid input. Please enter valid double
values.");
                scanner.nextLine(); // Clear the buffer
            }
        }
        scanner.close();
    }

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

    public double subtract(double a, double b) {
        return a - b;
    }

    public double multiply(double a, double b) {
        return a * b;
    }

    public double divide(double a, double b) {
        return a / b;
    }

    public double remainder(double a, double b) {
        return a % b;
    }
}
```

3. (BankAccount.java on GitHub)

```
public class BankAccount {
    private String accountNum;
    private String holderName;
    private float balance;
    private boolean isOpen;

    public BankAccount(String accountNum, String holderName, float balance) {
        this.accountNum = accountNum;
        this.holderName = holderName;
        this.balance = balance;
        this.isOpen = true; // Account is open by default
    }

    public void print() {
        System.out.println("Account Number: " + accountNum);
        System.out.println("Account Holder: " + holderName);
        System.out.println("Balance: " + balance);
        System.out.println("Account Status: " + (isOpen ? "Open" :
"Closed"));
    }

    public float getBalance() {
        return balance;
    }

    public void close() {
        if (!isOpen) {
            throw new BankAccountException("Account already closed.");
        }
        if (balance != 0) {
            throw new BankAccountException("Cannot close an account that has
money.");
        }
        isOpen = false;
    }

    public void reOpen() {
        if (isOpen) {
            throw new BankAccountException("Account is already open.");
        }
        isOpen = true;
    }

    public void deposit(float amount) {
        if (amount <= 0) {
            throw new BankAccountException("Deposit amount must be
positive.");
        }
        if (!isOpen) {
            throw new BankAccountException("Cannot deposit to a closed
account.");
        }
        balance += amount;
    }

    public void withdraw(float amount) {
        if (amount <= 0) {
            throw new BankAccountException("Withdrawal amount must be
positive.");
        }
        if (!isOpen) {
            throw new BankAccountException("Cannot withdraw from a closed
account.");
        }
        if (balance < amount) {
            throw new BankAccountException("Insufficient funds.");
        }
        balance -= amount;
    }

    public void transferTo(BankAccount other, float amount) {
        if (amount <= 0) {
            throw new BankAccountException("Transfer amount must be
positive.");
        }
        if (!isOpen || !other.isOpen) {
            throw new BankAccountException("One or both accounts are
closed.");
        }
        if (balance < amount) {
            throw new BankAccountException("Insufficient funds for
transfer.");
        }
        this.withdraw(amount);
        other.deposit(amount);
    }

    public static void printAll(BankAccount[] accounts) {
        for (BankAccount account : accounts) {
            account.print();
        }
    }

    public static int find(BankAccount[] accounts, String accountNum) {
        for (int i = 0; i < accounts.length; i++) {
            if (accounts[i].accountNum.equals(accountNum)) {
                return i;
            }
        }
        return -1;
    }
}

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

public class BankDemo {
    public static void main(String[] args) {
        BankAccount BankAccount1 = new BankAccount("12345", "SomeDude", 500);
        BankAccount BankAccount2 = new BankAccount("67890", "OtherDude",
1000);

        BankAccount1.print(); // Print details of BankAccount1.

        BankAccount1.close(); // This will throw an exception since the
balance is not zero.
        BankAccount1.deposit(200); // This will throw an exception as the
BankAccount is already closed.
        BankAccount1.withdraw(100); // This will also throw an exception due
to BankAccount closure.

        BankAccount1.reOpen(); // This should succeed after closure.
        BankAccount1.transferTo(BankAccount2, 2000); // This will throw an
exception due to insufficient funds.

        BankAccount.printAll(new BankAccount[] {BankAccount1, BankAccount2});
        System.out.println("Index of BankAccount Number '12345': " +
BankAccount.find(new BankAccount[] {BankAccount1, BankAccount2}, "12345"));
    }
}
```

4. Modifying the bank account to be **checked**, instead would mean throwing our user-defined exception in the declared methods, as follows:

(**CheckedBankAccount.java** on GitHub)

```
public class CheckedBankAccount {
    private String accountNum;
    private String holderName;
    private float balance;
    private boolean isOpen;

    public CheckedBankAccount(String accountNum, String holderName, float
balance) {
        this.accountNum = accountNum;
        this.holderName = holderName;
        this.balance = balance;
        this.isOpen = true; // Account is open by default
    }

    public void print() {
        System.out.println("Account Number: " + accountNum);
```

```java
        System.out.println("Account Holder: " + holderName);
        System.out.println("Balance: " + balance);
        System.out.println("Account Status: " + (isOpen ? "Open" :
"Closed"));
    }

    public float getBalance() {
        return balance;
    }

    public void close() throws BankAccountException {
        if (!isOpen) {
            throw new BankAccountException("Account already closed.");
        }
        if (balance != 0) {
            throw new BankAccountException("Cannot close account with non-
zero balance.");
        }
        isOpen = false;
    }

    public void reOpen() throws BankAccountException {
        if (isOpen) {
            throw new BankAccountException("Account is already open.");
        }
        isOpen = true;
    }

    public void deposit(float amount) throws BankAccountException {
        if (amount <= 0) {
            throw new BankAccountException("Deposit amount must be
positive.");
        }
        if (!isOpen) {
            throw new BankAccountException("Cannot deposit to a closed
account.");
        }
        balance += amount;
    }

    public void withdraw(float amount) throws BankAccountException {
        if (amount <= 0) {
            throw new BankAccountException("Withdrawal amount must be
positive.");
        }
        if (!isOpen) {
            throw new BankAccountException("Cannot withdraw from a closed
account.");
        }
        if (balance < amount) {
            throw new BankAccountException("Insufficient funds.");
        }
        balance -= amount;
    }

    public void transferTo(CheckedBankAccount other, float amount) throws
BankAccountException {
        if (amount <= 0) {
            throw new BankAccountException("Transfer amount must be
positive.");
```

```
        }
        if (!isOpen || !other.isOpen) {
            throw new BankAccountException("One or both accounts are 
closed.");
        }
        if (balance < amount) {
            throw new BankAccountException("Insufficient funds for 
transfer.");
        }
        this.withdraw(amount);
        other.deposit(amount);
    }

    public static void printAll(CheckedBankAccount[] accounts) {
        for (CheckedBankAccount account : accounts) {
            account.print();
        }
    }

    public static int find(CheckedBankAccount[] accounts, String accountNum) 
{
        for (int i = 0; i < accounts.length; i++) {
            if (accounts[i].accountNum.equals(accountNum)) {
                return i;
            }
        }
        return -1;
    }
}
```

(**CheckedBankDemo.java** on GitHub)

```
public class CheckedBankDemo {
    public static void main(String[] args) {
        CheckedBankAccount account1 = new CheckedBankAccount("12345", 
"SomeDude", 500);

        try {
            account1.print();
            account1.close(); // This will throw an exception
        } catch (BankAccountException e) {
            System.out.println("Exception: " + e.getMessage());
        }

        try {
            account1.deposit(200); // This should succeed
        } catch (BankAccountException e) {
            System.out.println("Exception: " + e.getMessage());
        }

        try {
            account1.withdraw(100); // This should succeed
        } catch (BankAccountException e) {
            System.out.println("Exception: " + e.getMessage());
        }

        try {
            account1.reOpen(); // Will throw an exception since account is 
not closed
        } catch (BankAccountException e) {
            System.out.println("Exception: " + e.getMessage());
        }

        // Demonstrating transfer with exception handling
        CheckedBankAccount account2 = new CheckedBankAccount("67890",
"OtherDude", 1000);
        try {
            account1.transferTo(account2, 2000); // Will throw an exception
due to insufficient funds
        } catch (BankAccountException e) {
            System.out.println("Exception: " + e.getMessage());
        }
    }
}
```

5. **For each of the given inputs:**

a- 16,8,10 ---->

```
x=2
y=0
Finally.
Done.
```

b- “X”,10,0 ---->

```
Non-integer inputs.
Finally.
Done.
```

c- 16,8,10 ---->

```
Cannot divide by Zero.
Finally.
Done.
```

d- 10,”X”,17 ---->

```
x=4
Non-integer inputs.
Finally.
Done.
```

e- 7,150, 0 ---->

```
x=5
(Unhandled ArrayIndexOutOfBoundsException is thrown)
```

f- 87,18, 0 ---->

```
x=0
Cannot divide by Zero.
Finally.
Done.
```

This concludes Sheet (6) Solutions, this document + source code to all programming exercises available on https://bit.ly/CSE231Sheets **.**