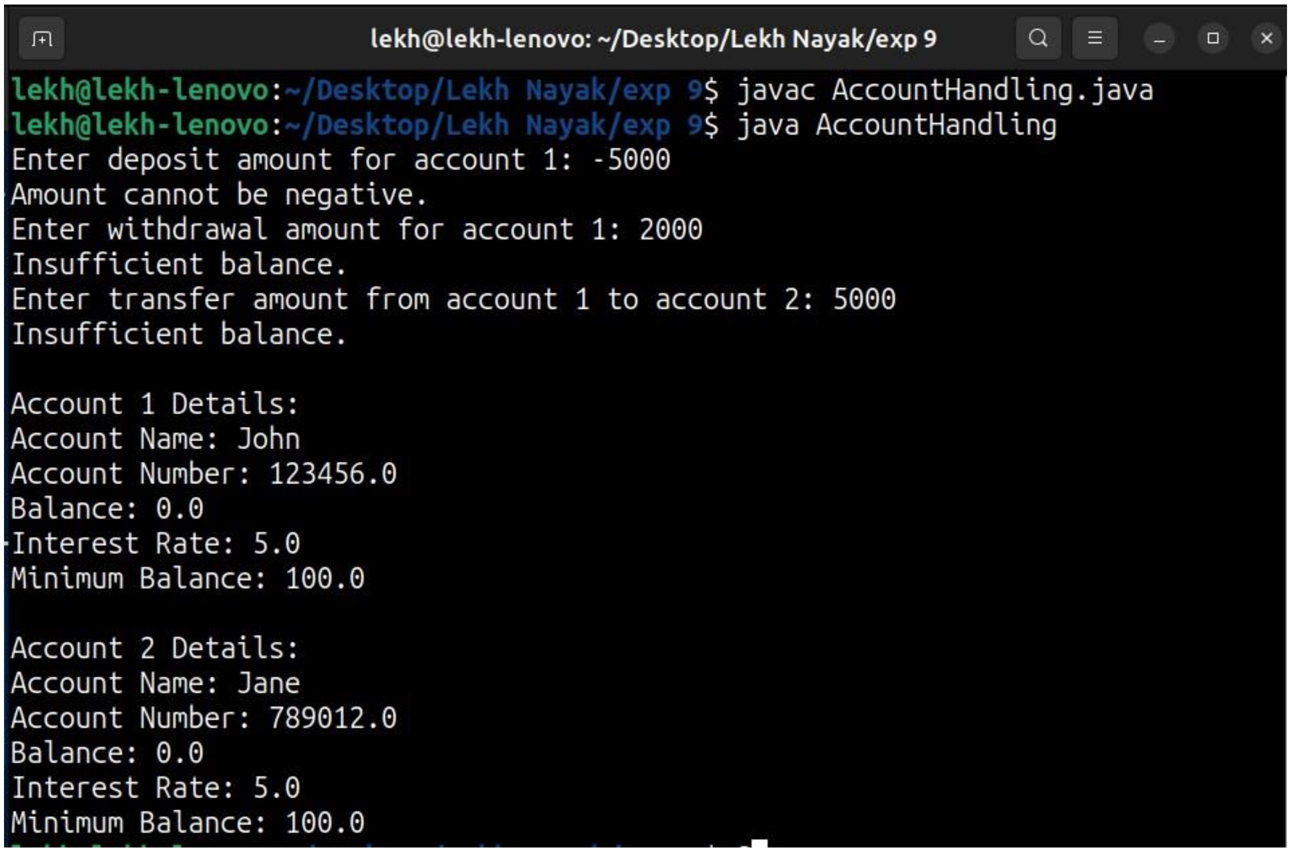
Name	Lekh Sanatan Nayak
UID no.	2023800068
Experiment No.	9

AIM:	Program to demonstrate exception handling
Program 1	
PROBLEM STATEMENT :	1. **There is an abstract class Account** **Attribute:-** **● Name** **● Balance** **● Acc_No** **Method:-** **● Deposit - abstract method** **● withdraw - abstract method** **● display - abstract method** **Saving Account inherits the Account class and provides the implementation for the methods accordingly** **Saving Account class Attribute:-** **● interestRate** **● minBalance** **Method** **● addInterest: handle Arithmetic Exception** **● transfer():** **Note:** **● Balance cannot be less than 0.** **● In a Saving account if minBalance is set then for that the balance cannot go less than that amount. If it goes, an error must be shown.** **● let the user deposit to or withdraw from the account. For each transaction, a message is displayed to indicate the status of the transaction: successful or failed. In case of failure, the failure reason is reported.**

	● The possible Exceptions are negative-amount-exception (in both deposit and withdraw transaction) and insufficient-amount-exception (in withdraw transaction). **For the above scenario write an interactive program in Java. Also, show output for different use cases.**
PROGRAM:	*(code below)*

```
import java.util.*;

// Abstract class representing a generic account
abstract class Account {
    String Name; // Name of the account holder
    double Balance; // Current balance of the account
    double AccNo; // Account number

    // Abstract methods for deposit, withdrawal, and display
    abstract void Deposit(double amount) throws
NegativeAmountException;
    abstract void Withdraw(double amount) throws
NegativeAmountException, InsufficientAmountException;
    abstract void Display();
}

// Subclass representing a savings account, inheriting from Account
class SavingAccount extends Account {
    double interestRate; // Interest rate for the account
    double minBalance; // Minimum balance required

    // Constructor for SavingAccount
    SavingAccount(String Name, double AccNo, double interestRate, double
minBalance) {
        this.Name = Name;
        this.AccNo = AccNo;
        this.interestRate = interestRate;
        this.minBalance = minBalance;
        this.Balance = 0; // Initialize balance to zero
    }

    // Method to deposit money into the account
    void Deposit(double amount) throws NegativeAmountException {
        if (amount < 0) {
            throw new NegativeAmountException(); // Throw exception if amount is negative
        }
        Balance += amount; // Add amount to balance
        System.out.println("Deposit successful. Current balance: " + Balance);
    }

    // Method to withdraw money from the account
    void Withdraw(double amount) throws NegativeAmountException, InsufficientAmountException {
        if (amount < 0) {
            throw new NegativeAmountException(); // Throw exception if amount is negative
        }
        if (Balance - amount < minBalance) {
            throw new InsufficientAmountException(); // Throw exception if balance is insufficient
        }
        Balance -= amount; // Deduct amount from balance
        System.out.println("Withdrawal successful. Current balance: " + Balance);
    }

    // Method to display account details
    void Display() {
        System.out.println("Account Name: " + Name);
        System.out.println("Account Number: " + AccNo);
        System.out.println("Balance: " + Balance);
        System.out.println("Interest Rate: " + interestRate);
        System.out.println("Minimum Balance: " + minBalance);
    }

    // Method to add interest to the account balance
    void addInterest() {
        double interest = Balance * interestRate / 100;
        Balance += interest;
        System.out.println("Interest added. Current balance: " + Balance);
    }

    // Method to transfer money from this account to another account
    void transfer(double amount, SavingAccount targetAccount) throws
NegativeAmountException, InsufficientAmountException {
        Withdraw(amount); // Withdraw amount from this account
        targetAccount.Deposit(amount); // Deposit amount into target account
        System.out.println("Transfer successful.");
    }
}

// Custom exception class for negative amounts
class NegativeAmountException extends Exception {
    NegativeAmountException() {
        System.out.println("Amount cannot be negative."); // Display error
message
    }
}

// Custom exception class for insufficient balance
class InsufficientAmountException extends Exception {
    InsufficientAmountException() {
        System.out.println("Insufficient balance."); // Display error message
    }
}

// Main class for account handling
public class AccountHandling {
    public static void main(String[] args) {
        SavingAccount account1 = new SavingAccount("John", 123456, 5,
100); // Create account 1
        SavingAccount account2 = new SavingAccount("Jane", 789012, 5,
100); // Create account 2

        Scanner sc = new Scanner(System.in); // Create Scanner object for
user input

        // Deposit into account 1
        try {
            System.out.print("Enter deposit amount for account 1: ");
            double depositAmount = sc.nextDouble();
            account1.Deposit(depositAmount); // Call Deposit method
        } catch (NegativeAmountException e) {
            // Handle negative amount exception
        }

        // Withdraw from account 1
        try {
            System.out.print("Enter withdrawal amount for account 1: ");
            double withdrawalAmount = sc.nextDouble();
            account1.Withdraw(withdrawalAmount); // Call Withdraw method
        } catch (NegativeAmountException | InsufficientAmountException e)
{
            // Handle negative amount or insufficient balance exception
        }

        // Transfer from account 1 to account 2
        try {
            System.out.print("Enter transfer amount from account 1 to account
2: ");
            double transferAmount = sc.nextDouble();
            account1.transfer(transferAmount, account2); // Call transfer
method
        } catch (NegativeAmountException | InsufficientAmountException e)
{
            // Handle negative amount or insufficient balance exception
        }

        // Display account details
        System.out.println("\nAccount 1 Details:");
        account1.Display(); // Call Display method for account 1
        System.out.println("\nAccount 2 Details:");
        account2.Display(); // Call Display method for account 2

        sc.close(); // Close Scanner object
    }
}
```

RESULT:



```
lekh@lekh-lenovo:~/Desktop/Lekh Nayak/exp 9$ javac AccountHandling.java
lekh@lekh-lenovo:~/Desktop/Lekh Nayak/exp 9$ java AccountHandling
Enter deposit amount for account 1: -5000
Amount cannot be negative.
Enter withdrawal amount for account 1: 2000
Insufficient balance.
Enter transfer amount from account 1 to account 2: 5000
Insufficient balance.

Account 1 Details:
Account Name: John
Account Number: 123456.0
Balance: 0.0
Interest Rate: 5.0
Minimum Balance: 100.0

Account 2 Details:
Account Name: Jane
Account Number: 789012.0
Balance: 0.0
Interest Rate: 5.0
Minimum Balance: 100.0
```

Program 2

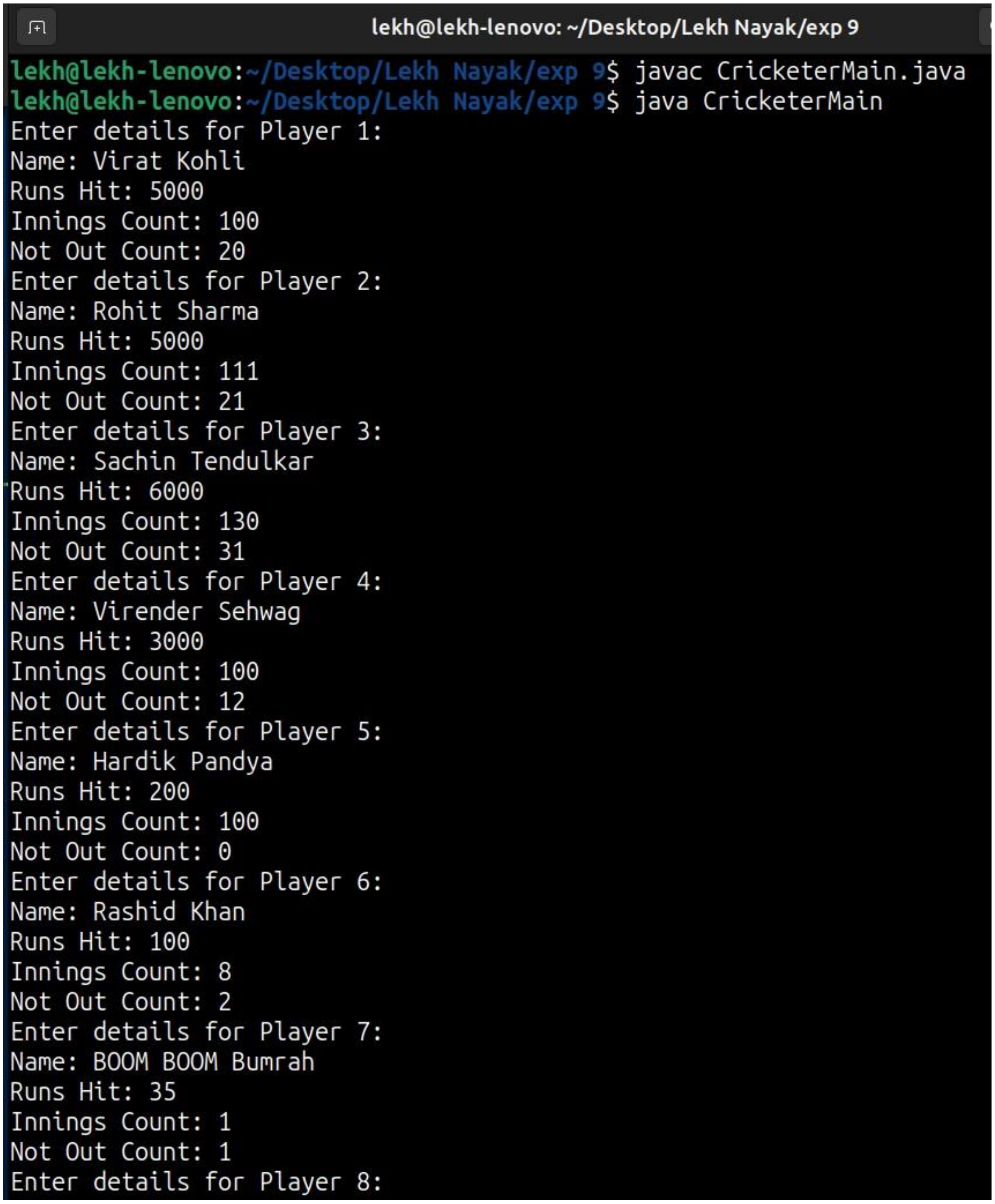
PROBLEM STATEMENT :	1. **Define a class Cricketer which has:-** **Attributes:-** **● player_name** **● runs_hit** **● innings_count** **● not_out_count** **● batting_avg** **Methods:-get_avg** **Make a cricket team with 11 cricketers. For each cricketer, find his batting average. Handle all different errors while calculating this. Also, make a method which will find he list of cricketers in ascending order of their batting average and also display the cricketer stats in this order. If the average of the batting average of the entire team is less than 20 runs then throw a user-defined exception.**

	Note- handle errors like ArrayIndexOutOfBoundsException, ArithmeticException,ArrayStoreException, NumberFormatException, etc
PROGRAM:	

```java
import java.util.*;

// Class to represent a cricketer
class Cricketer {
    String player_name;
    int runs_hit;
    int innings_count;
    int not_out_count;
    float batting_avg;

    // Constructor to initialize cricketer details
    Cricketer(String player_name, int runs_hit, int innings_count, int not_out_count) {
        this.player_name = player_name;
        this.runs_hit = runs_hit;
        this.innings_count = innings_count;
        this.not_out_count = not_out_count;
        this.batting_avg = 0.0f; // Default batting average
    }

    // Method to calculate batting average
    float get_avg() throws ArithmeticException {
        int out_count = innings_count - not_out_count;
        if (out_count == 0) {
            throw new ArithmeticException("Cannot divide by zero.");
        }
        batting_avg = (float) runs_hit / out_count;
        return batting_avg;
    }

    // Override toString method to display cricketer details
    public String toString() {
        return "Player: " + player_name + ", Batting Average: " + batting_avg;
    }
}

// Comparator class to compare cricketers based on batting average
```

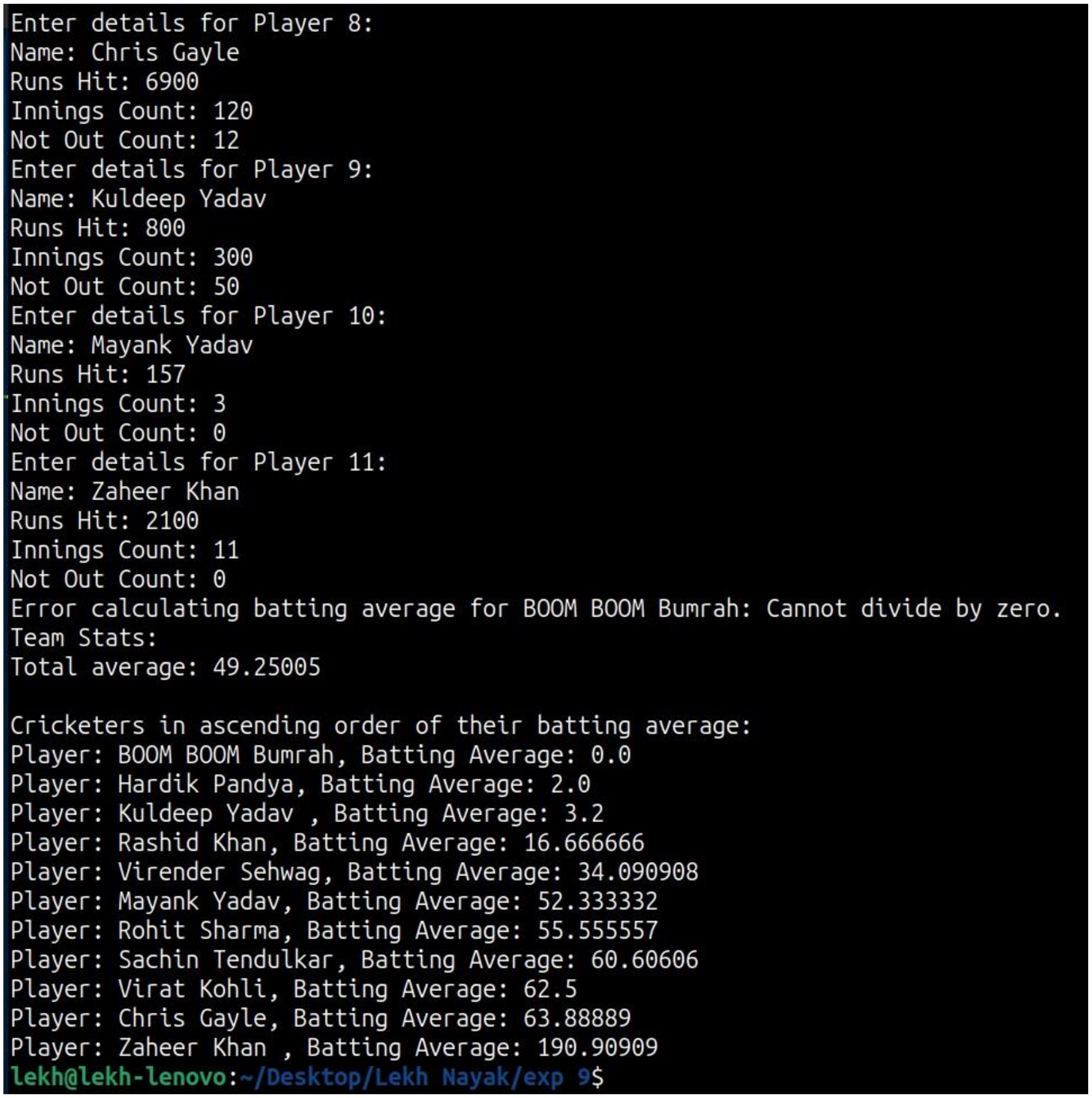
```
class BattingAverageComparator implements Comparator<Cricketer> {
    public int compare (Cricketer c1, Cricketer c2) {
        return Float.compare(c1.batting_avg, c2.batting_avg);
    }
}

// Main class
public class CricketerMain {
    public static void main(String args[]) {
        ArrayList<Cricketer> team = new ArrayList<>(); // List to store
cricketers
        Scanner sc = new Scanner(System.in); // Scanner for user input

        // Loop to input details of 11 players
        for (int i = 1; i <= 11 ; i++) {
            System.out.println("Enter details for Player " + i + ":");
            System.out.print("Name: ");
            String name = sc.nextLine();
            System.out.print("Runs Hit: ");
            int runs = sc.nextInt();
            System.out.print("Innings Count: ");
            int innings = sc.nextInt();
            System.out.print("Not Out Count: ");
            int notOut = sc.nextInt();

            team.add(new Cricketer(name, runs, innings, notOut)); // Add
cricketer to the team
            sc.nextLine(); // Consume newline character
        }

        float totalAvg = 0.0f; // Total average of the team

        // Calculate total average of the team
        for (Cricketer player : team) {
            try {
                totalAvg += player.get_avg(); // Calculate batting average for
each player and add to total
            }
            catch(ArithmeticException e) {
                System.out.println("Error calculating batting average for " +
player.player_name + ": " + e.getMessage());
            }
        }

        // Calculate the average of the entire team
        totalAvg /= team.size();

        // Display the team stats
        System.out.println("Team Stats:");
        System.out.println("Total average: " + totalAvg);

        // Check if the average of the batting average of the entire team is less
than 20 runs
        if (totalAvg < 20) {
            throw new RuntimeException("Average batting average of the team
is less than 20 runs.");
        }

        // Sort cricketers based on batting average
        Collections.sort(team, new BattingAverageComparator());

        // Display cricketers in ascending order of their batting average
        System.out.println("\nCricketers in ascending order of their batting
average:");
        for (Cricketer player : team) {
            System.out.println(player);
        }

        sc.close(); // Close scanner
    }
}
```

RESULT:



lekh@lekh-lenovo: ~/Desktop/Lekh Nayak/exp 9

```
lekh@lekh-lenovo:~/Desktop/Lekh Nayak/exp 9$ javac CricketerMain.java
lekh@lekh-lenovo:~/Desktop/Lekh Nayak/exp 9$ java CricketerMain
Enter details for Player 1:
Name: Virat Kohli
Runs Hit: 5000
Innings Count: 100
Not Out Count: 20
Enter details for Player 2:
Name: Rohit Sharma
Runs Hit: 5000
Innings Count: 111
Not Out Count: 21
Enter details for Player 3:
Name: Sachin Tendulkar
Runs Hit: 6000
Innings Count: 130
Not Out Count: 31
Enter details for Player 4:
Name: Virender Sehwag
Runs Hit: 3000
Innings Count: 100
Not Out Count: 12
Enter details for Player 5:
Name: Hardik Pandya
Runs Hit: 200
Innings Count: 100
Not Out Count: 0
Enter details for Player 6:
Name: Rashid Khan
Runs Hit: 100
Innings Count: 8
Not Out Count: 2
Enter details for Player 7:
Name: BOOM BOOM Bumrah
Runs Hit: 35
Innings Count: 1
Not Out Count: 1
Enter details for Player 8:
```



```
Enter details for Player 8:
Name: Chris Gayle
Runs Hit: 6900
Innings Count: 120
Not Out Count: 12
Enter details for Player 9:
Name: Kuldeep Yadav
Runs Hit: 800
Innings Count: 300
Not Out Count: 50
Enter details for Player 10:
Name: Mayank Yadav
Runs Hit: 157
Innings Count: 3
Not Out Count: 0
Enter details for Player 11:
Name: Zaheer Khan
Runs Hit: 2100
Innings Count: 11
Not Out Count: 0
Error calculating batting average for BOOM BOOM Bumrah: Cannot divide by zero.
Team Stats:
Total average: 49.25005

Cricketers in ascending order of their batting average:
Player: BOOM BOOM Bumrah, Batting Average: 0.0
Player: Hardik Pandya, Batting Average: 2.0
Player: Kuldeep Yadav , Batting Average: 3.2
Player: Rashid Khan, Batting Average: 16.666666
Player: Virender Sehwag, Batting Average: 34.090908
Player: Mayank Yadav, Batting Average: 52.333332
Player: Rohit Sharma, Batting Average: 55.555557
Player: Sachin Tendulkar, Batting Average: 60.60606
Player: Virat Kohli, Batting Average: 62.5
Player: Chris Gayle, Batting Average: 63.88889
Player: Zaheer Khan , Batting Average: 190.90909
lekh@lekh-lenovo:~/Desktop/Lekh Nayak/exp 9$
```

CONCLUSION:	In this experiment, I learned how to use Exception Handling in Java and how to implement them in codes used for solving real-life problems