Name:	**Debjit Ghosal**
UID:	**20233000065**
Experiment No.	**Exp9B**

AIM:	**Exception Handling**
Program 1	
PROBLEM STATEMENT :	**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 the 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:	**/* Write a java program for :** **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 the 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** ***/** **/*** **FLow:** **first make a class for exceptions** **create class for cricketer** **override class**

```
make main class
*/
import java.util.Scanner;
import java.util.Arrays;

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

class Cricketer {
   String playerName;
   int runsHit;
   int inningsCount;
   int notOutCount;
   double battingAvg;

   public Cricketer(String playerName, int runsHit, int inningsCount,
int notOutCount) {
      this.playerName = playerName;
      this.runsHit = runsHit;
      this.inningsCount = inningsCount;
      this.notOutCount = notOutCount;
   }

   public double getAvg() {
      if (inningsCount == 0) {
         return 0.0;
      }
      return (double) runsHit / (inningsCount - notOutCount);
   }

   public String getPlayerName() {
      return playerName;
   }

   public double getBattingAvg() {
      return battingAvg;
   }
```

```
    @Override
    public String toString() {
        return "Player Name: " + playerName +
                ", Runs Hit: " + runsHit +
                ", Innings Count: " + inningsCount +
                ", Not Out Count: " + notOutCount +
                ", Batting Average: " + battingAvg;
    }
}

public class CricketerFinal {
    public static void main(String[] args) {
        Cricketer[] team = new Cricketer[11];

        team[0] = new Cricketer("Player1", 500, 20, 2);
        team[1] = new Cricketer("Player2", 600, 25, 3);
        team[2] = new Cricketer("Player3", 300, 35, 6);
        team[3] = new Cricketer("Player4", 600, 25, 8);
        team[4] = new Cricketer("Player5", 600, 45, 6);
        team[5] = new Cricketer("Player6", 600, 15, 8);
        team[6] = new Cricketer("Player7", 600, 75, 4);
        team[7] = new Cricketer("Player8", 600, 65, 6);
        team[8] = new Cricketer("Player9", 600, 55, 3);
        team[9] = new Cricketer("Player10", 600, 5, 1);
        team[10] = new Cricketer("Player11", 600, 45, 6);


        try {
            double teamAvg = 0.0;
            int playerCount = 0;
            for (Cricketer player : team) {
                if (player != null) {
                    player.battingAvg = player.getAvg();
                    teamAvg += player.getBattingAvg();
                    playerCount++;
                }
            }
            if (playerCount != 0) {
                teamAvg /= playerCount;
```

```
            }
            if (teamAvg < 20) {
                throw new LowTeamBattingAverageException("Team's
average batting average is below 20 runs.");
            }
            Arrays.sort(team, (a, b) -> Double.compare(a.getBattingAvg(),
b.getBattingAvg()));
            System.out.println("Cricketers sorted by batting average:");
            for (Cricketer player : team) {
                if (player != null) {
                    System.out.println(player);
                }
            }
        } catch (LowTeamBattingAverageException e) {
            System.out.println("Error: " + e.getMessage());
        } catch (ArithmeticException | NullPointerException |
ArrayIndexOutOfBoundsException | ArrayStoreException |
NumberFormatException e) {
            System.out.println("Error: " + e.getClass().getSimpleName() +
" occurred while calculating batting average.");
        }
    }
}
```

RESULT:

```
psipl@psipl-OptiPlex-3000:~/Desktop/2023300065$ javac CricketerFinal.java
psipl@psipl-OptiPlex-3000:~/Desktop/2023300065$ java CricketerFinal
Error: NullPointerException occurred while calculating batting average.
psipl@psipl-OptiPlex-3000:~/Desktop/2023300065$
```

```
psipl@psipl-OptiPlex-3000:~/Desktop/2023300065$ javac CricketerFinal.java
psipl@psipl-OptiPlex-3000:~/Desktop/2023300065$ java CricketerFinal
Cricketers sorted by batting average:
Player Name: Player7, Runs Hit: 600, Innings Count: 75, Not Out Count: 4, Batting Average: 8.450704225352112
Player Name: Player8, Runs Hit: 600, Innings Count: 65, Not Out Count: 6, Batting Average: 10.169491525423728
Player Name: Player3, Runs Hit: 300, Innings Count: 35, Not Out Count: 6, Batting Average: 10.344827586206897
Player Name: Player9, Runs Hit: 600, Innings Count: 55, Not Out Count: 3, Batting Average: 11.538461538461538
Player Name: Player5, Runs Hit: 600, Innings Count: 45, Not Out Count: 6, Batting Average: 15.384615384615385
Player Name: Player11, Runs Hit: 600, Innings Count: 45, Not Out Count: 6, Batting Average: 15.384615384615385
Player Name: Player2, Runs Hit: 600, Innings Count: 25, Not Out Count: 3, Batting Average: 27.272727272727273
Player Name: Player1, Runs Hit: 500, Innings Count: 20, Not Out Count: 2, Batting Average: 27.77777777777778
Player Name: Player4, Runs Hit: 600, Innings Count: 25, Not Out Count: 8, Batting Average: 35.294117647058826
Player Name: Player6, Runs Hit: 600, Innings Count: 15, Not Out Count: 8, Batting Average: 85.71428571428571
Player Name: Player10, Runs Hit: 600, Innings Count: 5, Not Out Count: 1, Batting Average: 150.0
psipl@psipl-OptiPlex-3000:~/Desktop/2023300065$
```

Program 2	
PROBLEM STATEMENT :	**Write a program to implement stack operations:** a.Push b. Pop Write a user defined exception to check whether the stack is full or empty.

PROGRAM:	**/*
	Write a java program to implement stack operations:
	
	a.Push b. Pop
	
	Write a user defined exception to check whether the stack is full or empty.
	*/
	
	/*
	Flow:
	User-defined exception for stack full condition
	User-defined exception for stack empty condition
	Stack implementation using array
	then use constructor
	the use the push and pop operations
	Main class to demonstrate stack operations
	*/
	
	import java.util.Scanner;
	class StackFullException extends Exception {
	public StackFullException(String message) {
	super(message);
	}
	}
	
	class StackEmptyException extends Exception {
	public StackEmptyException(String message) {
	super(message);
	}
	}**

```
class Stack {
   private int maxSize;
   private int[] stackArray;
   private int top;

   public Stack(int size) {
      maxSize = size;
      stackArray = new int[maxSize];
      top = -1;
   }

   public void push(int value) throws StackFullException {
      if (top == maxSize - 1) {
         throw new StackFullException("Stack is full");
      }
      stackArray[++top] = value;
   }
   public int pop() throws StackEmptyException {
      if (top == -1) {
         throw new StackEmptyException("Stack is empty");
      }
      return stackArray[top--];
   }
}

public class Push_Pop {
   public static void main(String[] args) {
      Stack stack = new Stack(5);

      try {
         stack.push(10);
         stack.push(20);
         stack.push(30);
         stack.push(40);
         stack.push(50);
         stack.push(60);

         System.out.println("Popped element: " + stack.pop());
         System.out.println("Popped element: " + stack.pop());
```

```
            System.out.println("Popped element: " + stack.pop());
            System.out.println("Popped element: " + stack.pop());
            System.out.println("Popped element: " + stack.pop());
        } catch (StackFullException e) {
            System.out.println("Error: " + e.getMessage());
        } catch (StackEmptyException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

RESULT:

```
psipl@psipl-OptiPlex-3000:~/Desktop/2023300065$ javac Push_Pop.java
psipl@psipl-OptiPlex-3000:~/Desktop/2023300065$ java Push_Pop
Error: Stack is full
psipl@psipl-OptiPlex-3000:~/Desktop/2023300065$
```

```
psipl@psipl-OptiPlex-3000:~/Desktop/2023300065$ javac Push_Pop.java
psipl@psipl-OptiPlex-3000:~/Desktop/2023300065$ java Push_Pop
Popped element: 50
Popped element: 40
Popped element: 30
Popped element: 20
Popped element: 10
psipl@psipl-OptiPlex-3000:~/Desktop/2023300065$ javac Push_Pop.java
psipl@psipl-OptiPlex-3000:~/Desktop/2023300065$ java Push_Pop
Error: Stack is full
psipl@psipl-OptiPlex-3000:~/Desktop/2023300065$
```

CONCLUSION:	I have learnt about exception handling and a more about the push and pop operations.