RICHFIELD

FACULTY OF INFORMATION TECHNOLOGY

PROGRAMMING 731

1ST SEMESTER ASSIGNMENT

Name & Surname: VUYO MTHEMBU **ITS No:** 402202864

Qualification: BScIGT **Semester:** 1 **Module Name:** PROGRAMMING 731

Date Submitted: _____

ASSESSMENT CRITERIA	MARK ALLOCATION	EXAMINER MARKS	MODERATOR MARKS
MARKS FOR CONTENT			
QUESTION ONE	20		
QUESTION TWO	20		
QUESTION THREE	50		
TOTAL	90		
MARKS FOR TECHNICAL ASPECTS			
CODE LAYOUT/STRUCTURE AND COMMENTS	10		
TOTAL MARKS FOR ASSIGNMENT	100		
Examiner's Comments:			
Moderator's Comments:			
Signature of Examiner:		Signature of Moderator:	

TABLE OF CONTENTS

QUESTION 1

```
package testing;
import java.io.*;
import java.util.ArrayList;
import java.util.List;
/**
 *
 * @author Vuyo
 */
public class Testing
{
   /**
    * @param args the command line arguments
    */
   public static void main(String[] args)
   {
      List<Employee> employees = new ArrayList<>();

      //USing the Try Catch for handing Errors
      try
         // Reading the file and increment the employees list
         (BufferedReader br = new BufferedReader(new FileReader("employee.txt")))
      {
         String line = br.readLine();

         while ((line = br.readLine()) != null)
         {
            String[] details = line.split(" ");
            String name = details[0];
            String surname = details[1];
            int yearsWorked = Integer.parseInt(details[2]);
            double salary = Double.parseDouble(details[3]);

            Employee employee = new Employee(name, surname, yearsWorked, salary);
            employees.add(employee);
         }
```

```
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }

        // Updating the salarie for employee
        for (Employee employee : employees)
        {
            employee.updateSalary();
        }

        //Updated details must go back to the file
        try (BufferedWriter bw = new BufferedWriter(new FileWriter("employee.txt")))
        {
            bw.write("Name Surname Years Worked Salary\n");
            for (Employee employee : employees)
            {
                bw.write(employee.toString() + "\n");
            }
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
    }

}

package testing;


public class Employee
{
    //Declaring Variables
    String name;
```

```
String surname;
int yearsWorked;
double salary;

//Constructor
public Employee(String name, String surname, int yearsWorked, double salary)
{
   this.name = name;
   this.surname = surname;
   this.yearsWorked = yearsWorked;
   this.salary = salary;
}

//Method Void to Calculate Salary with if-else statement
void updateSalary()
{
   if (yearsWorked < 5)
   {
      salary += salary * 0.05;
   }
   else if (yearsWorked >= 5 && yearsWorked <= 10)
   {
      salary += salary * 0.15;
   }
   else if (yearsWorked > 10)
   {
      salary += salary * 0.30;
   }
}

 //Method Override
@Override
public String toString()
{
   return name + " " + surname + " " + yearsWorked + " " + salary;
}
```

```
    //Getter Method
    public String getName()
    {
        return name;
    }
    public String getSurname()
    {
        return surname;
    }
    public int getYearsWorked()
    {
        return yearsWorked;
    }
    public double getSalary()
    {
        return salary;
    }

    //Setter Methods
    public void setName(String name)
    {
        this.name = name;
    }
    public void setSurname(String surname)
    {
        this.surname = surname;
    }
    public void setYearsWorked(int YearsWorked)
    {
        this.yearsWorked = yearsWorked;
    }
    public void setSalary(double salary)
    {
        this.salary = salary;
    }
}
```

Question 2

```
package taskdetails;

/**
 *
 * @author Vuyo
 */
public class Taskdetails
{

   /**
    * @param args the command line arguments
    */
   public static void main(String[] args)
   {
      Task task1 = new Task("Code Review", 1, 150);
      Task task2 = new Task("Testing", 2, 200);

      Thread worker1 = new Thread(new TaskWorker(task1, false), "Worker-1");
      Thread worker2 = new Thread(new TaskWorker(task2, true), "Worker-2");

      worker1.setPriority(Thread.MIN_PRIORITY);
      worker2.setPriority(Thread.MAX_PRIORITY);

      worker1.start();
      worker2.start();

      //Using the try catch for handling errors
      try
      {
         worker1.join();
         worker2.join();
      }
      catch (InterruptedException e)
      {
         e.printStackTrace();
```

```
        }

        displayThreadStatus(worker1);
        displayThreadStatus(worker2);
    }

    //Method to displaystatus
    private static void displayThreadStatus(Thread thread)
    {
        System.out.println("Thread Name: " + thread.getTaskName() + ", Status: " +
thread.getState());
    }

}

package taskdetails;

public class Thread
{
    //Declaring Variables
    String taskName;
    int taskID;
    int taskWage;

    //Constructor
    public Thread(String taskName, int taskID, int taskWage)
    {
        this.taskName = taskName;
        this.taskID = taskID;
        this.taskWage = taskWage;
    }

    //Getter Method
    public String getTaskName()
    {
        return taskName;
    }
```

```
    public int getTaskID()
    {
        return taskID;
    }
    public int getTaskWage()
    {
        return taskWage;
    }

    //Setter Method
    public void setTaskName(String taskName)
    {
        this.taskName = taskName;
    }
    public void setTaskID(int taskID)
    {
        this.taskID = taskID;
    }
    public void setTaskWage(int taskWage)
    {
        this.taskWage = taskWage;
    }

    //Method Override
    @Override
    public String toString()
    {
        return "Task Name: " + taskName + ", Task ID: " + taskID + ", Task Wage: " + taskWage;
    }
}

package taskdetails;

public class WorkerThread extends Thread
{
    //Declaring Variables
```

```
Task task;
boolean additionalInfo;

//Constructor
public WorkerThread(Task task, boolean additionalInfo)
{
   this.task = task;
   this.additionalInfo = additionalInfo;
}

//Method Override
@Override
public void run()
{
   //Using the try catch for handling errors
   try
   {
      Thread.sleep(1000);
   }
   catch (InterruptedException e)
   {
      e.printStackTrace();
   }
   if (additionalInfo)
   {
      displayTaskDetailsWithInfo(task);
   }
   else
   {
      displayTaskDetails(task);
   }
   System.out.println("Thread Name: " + Thread.currentThread().getName());
   System.out.println("Thread Priority: " + Thread.currentThread().getPriority());
   System.out.println("Thread Status: " + Thread.currentThread().getState());
}
//Method void to display the details
```

```
    public void displayTaskDetails(Task task)
    {
        System.out.println("Task Name: " + task.getTaskName());
        System.out.println("Task ID: " + task.getTaskID());
        System.out.println("Task Wage: " + task.getTaskWage());
    }

    //Void method for taskDetails
    public void displayTaskDetailsWithAdditionalInfo(Task task)
    {
        System.out.println("Task Name: " + task.getTaskName());
        System.out.println("Task ID: " + task.getTaskID());
        System.out.println("Task Wage: " + task.getTaskWage());
        System.out.println("Additional Information: Task in progress");
    }
```

QUESTION 3

```
package testinggamecard;

/**
 *
 * @author Vuyo
 */
public class TestingGameCard
{

    /**
     * @param args the command line arguments
     */
     public static void main(String[] args)
    {
        //Creating the object of blackjackgame
        BlackjackGame game = new BlackjackGame();
        game.play();
```

```
    }

}

package testinggamecard;

/**
 *
 * @author Vuyo
 */
public class Card
{
    //Declaring Variables
    private String rank;
    private String suit;

    //Constructor
    public Card(String rank, String suit)
    {
        this.rank = rank;
        this.suit = suit;
    }

    //Getter MEthod
    public String getRank()
    {
        return rank;
    }
    public String getSuit()
    {
        return suit;
    }
    public int getValue()
    {
        switch (rank)
        {
```

```
        case "2": case "3": case "4": case "5": case "6": case "7": case "8": case "9": case "10":
            return Integer.parseInt(rank);
        case "J": case "Q": case "K":
            return 10;
        case "A":
            return 11;
        default:
            throw new IllegalArgumentException("Invalid card rank: " + rank);
    }
}
//Setter MEthod
public void setRank(String rank)
{
    this.rank = rank;
}
public void setSuit(String suit)
{
    this.suit = suit;
}
public int setValue(int value)
{
    switch (rank)
    {
        case "2": case "3": case "4": case "5": case "6": case "7": case "8": case "9": case "10":
            return Integer.parseInt(rank);
        case "J": case "Q": case "K":
            return 10;
        case "A":
            return 11;
        default:
            throw new IllegalArgumentException("Invalid card rank: " + rank);
    }
}
//Method Override
@Override
public String toString()
```

```
    {
        return rank + " of " + suit;
    }
}

package testinggamecard;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
/**
 *
 * @author Vuyo
 */
public class Deck
{
    //Declaring Variables
     private List<Card> cards;

    public Deck()
    {
        cards = new ArrayList<>();
        String[] ranks = {"2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K", "A"};
        String[] suits = {"Hearts", "Diamonds", "Clubs", "Spades"};
        for (String rank : ranks)
        {
            for (String suit : suits)
            {
                cards.add(new Card(rank, suit));
            }
        }
        shuffle();
    }

    public void shuffle()
    {
        Collections.shuffle(cards);
```

```
    }

    public Card drawCard()
    {
        if (cards.isEmpty())
        {
            throw new IllegalStateException("Deck is empty");
        }
        return cards.remove(cards.size() - 1);
    }

}
```

```
package testinggamecard;
import java.util.ArrayList;
import java.util.List;
/**
 *
 * @author Vuyo
 */
public class Hand
{
    //Declaring Variables
    private List<Card> cards;

    public Hand()
    {
        cards = new ArrayList<>();
    }

    //Method void addCard
    public void addCard(Card card)
    {
        cards.add(card);
    }
```

```
    //Method calculatePointers
    public int calculatePoints()
    {
        int points = 0;
        int aceCount = 0;

        for (Card card : cards)
        {
            points += card.getValue();
            if (card.getRank().equals("A"))
            {
                aceCount++;
            }
        }

        while (points > 21 && aceCount > 0)
        {
            points -= 10;
            aceCount--;
        }

        return points;
    }

    //Void Method to diplay
    public void displayHand()
    {
        for (Card card : cards)
        {
            System.out.println(card);
        }
    }

}

package testinggamecard;
```

```
import java.util.Scanner;
/**
 *
 * @author Vuyo
 */
public class BlackjackGame
{
    //Declaring Variables
    private Deck deck;
    private Hand playerHand;
    private Hand dealerHand;

    public BlackjackGame()
    {
        deck = new Deck();
        playerHand = new Hand();
        dealerHand = new Hand();
    }

    //Void method
    public void play()
    {
        dealInitialCards();

        System.out.println("Player's hand:");
        playerHand.displayHand();
        System.out.println("Points: " + playerHand.calculatePoints());

        if (playerTurn())
        {
            dealerTurn();
        }

        determineWinner();
    }
```

```
//Void method
private void dealInitialCards()
{
   playerHand.addCard(deck.drawCard());
   playerHand.addCard(deck.drawCard());
   dealerHand.addCard(deck.drawCard());
   dealerHand.addCard(deck.drawCard());
}

//Boolean Method
private boolean playerTurn()
{
   Scanner scanner = new Scanner(System.in);
   while (true)
   {
      System.out.println("Do you want to hit or stand? (hit/stand)");
      String action = scanner.nextLine();

      if (action.equalsIgnoreCase("hit"))
      {
         playerHand.addCard(deck.drawCard());
         System.out.println("Player's hand:");
         playerHand.displayHand();
         System.out.println("Points: " + playerHand.calculatePoints());

         if (playerHand.calculatePoints() > 21)
         {
            System.out.println("You bust!");
            return false;
         }
      }
      else if (action.equalsIgnoreCase("stand"))
      {
         return true;
      }
      else
```

```
        {
            System.out.println("Invalid action. Please choose hit or stand.");
        }
    }
}

//Method void for dealer's Turn
private void dealerTurn()
{
    System.out.println("Dealer's hand:");
    dealerHand.displayHand();
    System.out.println("Points: " + dealerHand.calculatePoints());

    while (dealerHand.calculatePoints() < 17)
    {
        dealerHand.addCard(deck.drawCard());
        System.out.println("Dealer draws a card.");
        System.out.println("Dealer's hand:");
        dealerHand.displayHand();
        System.out.println("Points: " + dealerHand.calculatePoints());
    }
    if (dealerHand.calculatePoints() > 21)
    {
        System.out.println("Dealer busts!");
    }
}

//Void Method for the winner
private void determineWinner()
{
    int playerPoints = playerHand.calculatePoints();
    int dealerPoints = dealerHand.calculatePoints();

    System.out.println("Player's final points: " + playerPoints);
    System.out.println("Dealer's final points: " + dealerPoints);
```

```
        if (playerPoints > 21)
        {
            System.out.println("Dealer wins!");
        }
        else if (dealerPoints > 21 || playerPoints > dealerPoints)
        {
            System.out.println("Player wins!");
        }
        else if (playerPoints == dealerPoints)
        {
            System.out.println("It's a tie!");
        }
        else
        {
            System.out.println("Dealer wins!");
        }
    }
}
```