CSC262 Programming in Java

**Final**

|  |  |
| --- | --- |
| First Name | Ryan |
| Last Name | Johnson |
| ID# | 04144579 |

Table of Contents

[How to submit your Work 1](#_Toc16683456)

[P1 – 25 points 2](#_Toc16683457)

[P2 – 30 Points 3](#_Toc16683458)

[P3 – 30 points 4](#_Toc16683459)

[P4 – 15 points 5](#_Toc16683460)

# How to submit your Work

After filling all the parts in this file, please follow the following steps.

1. Add your name and ID to the first page.
2. Save the file in the original format (Docx or Doc)

(please do not convert to other file formats e.g. PDF, ZIP, RAR, …).

1. Rename the file as

*Ryan - Johnson-* CSC262 *–* ***Final*** *-* 04144579*.docx*

**Example:**

John – Smith - CSC262 *–* ***Midterm*** *-* 234566435.docx

1. Upload the file and submit it (only using Blackboard)

# P1 – 25 points

Solve the following problem **using arrays**:

**Past A:** Coupon collector is a classic statistic problem with many practical applications. The problem is to pick objects from a set of objects repeatedly and determine how many picks are needed for all the objects to be picked at least once. A variation of the problem is to pick cards from a shuffled deck of 52 cards repeatedly and find out how many picks are needed before you see one of each suit. Assume a picked card is placed back in the deck before picking another. Write a program to simulate the number of picks needed to get **total of four cards from each different suit** and display the four cards picked (it is possible that a card may be picked twice). Here is a sample run of the program:

**Queen of Spades**

**5 of Clubs**

**Queen of Hearts**

**4 of Diamonds**

**Number of picks: 12**

**Sample run explanation:** *As you see in the above run, 12 picks are made to get the four cards from different suits. The other 8 picks (12-4=8) were from the same previously picked suits, so they are not printed. So we continue picking a card until we see at least one card from each of the for suits.*

**Note:** The card pick is with replacement, meaning that when you pick a card from the deck of 52 card, you put it back in the deck. There is chance to see the previously selected card again.

**Part B:** Put part A in a for loop and repeat it **10,000** times and report the average number of total picks we should have to see 4 cards from different suits.

|  |
| --- |
| Your code for this problem |
| import java.util.Random;  import java.util.ArrayList;  import java.util.Arrays;  import java.util.HashSet;  import java.util.Set;  public class JavaFinalP1 {  public static void main(String[] args) {    System.*out*.println("Welcome to the random 52 card draw program!! I present to you 52 cards! So here you go!");  Random rand = new Random();  int cardnumb = rand.nextInt(13);  int cardsuit = rand.nextInt(4);  int a=0, b=0, c=0, d=0;  int suitArray[] = {a, b, c, d};  int makeSuitCount=0;  int diffSuitCount=0;  int tries =1;  int count= 0;  ArrayList<Integer> ListTries  = new ArrayList<Integer>();    System.*out*.println("");  while(count != 10000)  {  while(diffSuitCount==0)  {  while(makeSuitCount < 4)  {  switch (cardnumb)  {  case 0: System.*out*.print("You have drawn an Ace "); break;  case 1: System.*out*.print("You have drawn a 2 "); break;  case 2: System.*out*.print("You have drawn a 3 "); break;  case 3: System.*out*.print("You have drawn a 4 "); break;  case 4: System.*out*.print("You have drawn a 5 "); break;  case 5: System.*out*.print("You have drawn a 6 "); break;  case 6: System.*out*.print("You have drawn a 7 "); break;  case 7: System.*out*.print("You have drawn an 8 "); break;  case 8: System.*out*.print("You have drawn a 9 "); break;  case 9: System.*out*.print("You have drawn a 10 "); break;  case 10: System.*out*.print("You have drawn a Jack "); break;  case 11: System.*out*.print("You have drawn a Queen "); break;  case 12: System.*out*.print("You have drawn a King "); break;  }    switch (cardsuit)  {  case 0: System.*out*.print("of Hearts"); break;  case 1: System.*out*.print("of Clubs"); break;  case 2: System.*out*.print("of Diamonds"); break;  case 3: System.*out*.print("of Spades"); break;  }    if(makeSuitCount == 0)  a=cardsuit;  else if(makeSuitCount == 1)  b=cardsuit;  else if(makeSuitCount == 2)  c=cardsuit;  else if(makeSuitCount == 3)  d=cardsuit;  cardnumb = rand.nextInt(13);  cardsuit = rand.nextInt(4);  System.*out*.println("");  makeSuitCount = makeSuitCount+1;  //System.out.println(a + " "+b+" "+c+" "+d);    }  makeSuitCount =0;  System.*out*.println("");  if(a != b && a != c && a != d && b != c && b != d && c != d)  {  diffSuitCount = diffSuitCount+1;    }  else  {  tries = tries+1;  }  }    System.*out*.println(tries + " Tries required to get all different suites");  ListTries.add(tries);    tries = 1;  System.*out*.println("Added Tries: " + ListTries.get(count));  double sum = 0;    for(int i = 0; i < ListTries.size(); i++)  sum += ListTries.get(i);  count=count+1;  System.*out*.println("\nGames: "+ count + "\nAverage picks per success: "+sum/ListTries.size() + "\nTotal Tries: " + sum );    makeSuitCount=0;  diffSuitCount=0;  }    }  } |

Put the result in the following box.

|  |
| --- |
| The result of the query |
|  |

# P2 – 30 Points

**(The Person, Student, Employee, Faculty, and Staff classes)**

Design a class named Person and its two derived classes named **Student** and **Employee**. Make **Faculty** and **Staff** derived classes of **Employee**. A person has a name, address, phone number, and e-mail address. A student has a class status (freshman, sophomore, junior, or senior). An employee has an office, salary, and **datehired**. Define a class named **MyDate** that contains the fields **year**, **month**, and **day**. A faculty member has office hours and a rank. A staff member has a title. Define an **abstract** **toString** function in the **Person** class and override it in each class to display the class name and the person’s name.

Implement the classes. Write a test program that creates a **Person**, **Student**, **Employee**, **Faculty**, and **Staff**, and invokes their **toString()** functions.

|  |
| --- |
| Your code for this problem |
| Person.java  public class Person {  String name;  String address;  String phone;  String email;    Person(String name, String address, String phone, String email)  {  this.name = name;  this.address = address;  this.phone = phone;  this.email = email;  }  public String getName()  {  return name;  }  public void setName(String name)  {  this.name = name;  }  public String getAddress()  {  return address;  }  public void setAddress(String address)  {  this.address = address;  }  public String getPhone()  {  return phone;  }  public void setPhone(String phone)  {  this.phone = phone;  }  public String getEmail()  {  return email;  }  public void setEmail(String email)  {  this.email = email;  }    @Override  public String toString() {  return String.*format*("%s %s %s %s",  getName(), getAddress(), getPhone(), getEmail());  }  public static void main(String[] args) {  Person p1 = new Person ("Joe Shmo", "123 A st.", "222-222-2222", "js@yahoo.com");  Student s1 = new Student("Ryan Johnson", "2755 Westfield Ave", "111-111-1111", "rj@yahoo.com", "Freshman");  Employee e1 = new Employee("Jane Amazing", "3 B st.", "333-333-3333", "jaj@yahoo.com", "Phisical Education", 65480, "01/20/2010");  Faculty f1 = new Faculty("Crazy Funguy", "4 C st.", "444-444-4444", "cf@yahoo.com", "Phisical Education", 52100, "03/12/2012", "9am-5pm", "Teacher");  Staff sf1 = new Staff("Newguy Tempy", "5 D st.", "555-555-5555", "nt@yahoo.com", "Science", 20000, "05/28/2020", "Intern");  MyDate d1 = new MyDate("05", "11", "1979");    System.*out*.println(p1);  System.*out*.println("\n"+s1);  System.*out*.println("\n"+e1);  System.*out*.println("\n"+f1);  System.*out*.println("\n"+sf1);  System.*out*.println("\nTest MyDate");  d1.myDatePrint();  }  }  ///////////////////////////  Employee.java  public class Employee extends Person {  String office;  double salary;  String dateHired;    Employee(String name, String address, String phone, String email, String office, double salary, String dateHired) {  super(name, address, phone, email);  this.office=office;  this.salary=salary;  this.dateHired=dateHired;    }  public String toString()  {  return "Office: " + office + "\nName: " + name + "\nAddress: " + address + "\nPhone: "  + phone + "\nEmail: " + email + "\nSalary: $" + salary + "0" + "\nDate Hired: " + dateHired;  }    }  ////////////////////////  Faculty.java  public class Faculty extends Employee {  String officeHours;  String rank;    Faculty(String name, String address, String phone, String email, String office, double salary, String dateHired, String officeHours, String rank) {  super(name, address, phone, email, office, salary, dateHired);  this.officeHours=officeHours;  this.rank=rank;    }  public String toString()  {  return "Office: " + office+ "\nRank: " + rank + "\nName: " + name + "\nAddress: " + address + "\nPhone: "  + phone + "\nEmail: " + email + "\nSalary: $" + salary + "0" + "\nDate Hired: " + dateHired+ "\nOffice Hours: " + officeHours;  }    }  /////////////////////  Staff.java  public class Staff extends Employee {  String title;    Staff(String name, String address, String phone, String email, String office, double salary, String dateHired, String title) {  super(name, address, phone, email, office, salary, dateHired);  this.title=title;      }  public String toString()  {  return "Office: " + office+ "\nTitle: " + title + "\nName: " + name + "\nAddress: " + address + "\nPhone: "  + phone + "\nEmail: " + email + "\nSalary: $" + salary + "0" + "\nDate Hired: " + dateHired;  }    }  //////////////////////////  Student.java  public class Student extends Person {    String status;    Student(String name, String address, String phone, String email, String status) {  super(name, address, phone, email);  this.status= status;    }  public String toString()  {  return "Status: " + status + "\nName: " + name + "\nAddress: " + address + "\nPhone: "  + phone + "\nEmail: " + email;  }    }  ////////////////////////  MyDate.java  public class MyDate {  String year;  String month;  String day;  MyDate()  {  year="00";  month="00";  day="00";  }    MyDate(String year, String month, String day) {    this.year=year;  this.month=month;  this.day=day;    }  public void myDatePrint()  {  System.*out*.println(month + "/" + day + "/" + year);  }    } |

Put the result in the following box.

|  |
| --- |
| The result of the query |
| Joe Shmo 123 A st. 222-222-2222 js@yahoo.com  Status: Freshman  Name: Ryan Johnson  Address: 2755 Westfield Ave  Phone: 111-111-1111  Email: rj@yahoo.com  Office: Phisical Education  Name: Jane Amazing  Address: 3 B st.  Phone: 333-333-3333  Email: jaj@yahoo.com  Salary: $65480.00  Date Hired: 01/20/2010  Office: Phisical Education  Rank: Teacher  Name: Crazy Funguy  Address: 4 C st.  Phone: 444-444-4444  Email: cf@yahoo.com  Salary: $52100.00  Date Hired: 03/12/2012  Office Hours: 9am-5pm  Office: Science  Title: Intern  Name: Newguy Tempy  Address: 5 D st.  Phone: 555-555-5555  Email: nt@yahoo.com  Salary: $20000.00  Date Hired: 05/28/2020  Test MyDate  11/1979/05 |

# P3 – 30 points

A salesman wants to go to five different cities and sell some products. The locations of the cities are listed in the following table.

|  |  |  |
| --- | --- | --- |
| **City #** | **X\_location** | **Y\_location** |
| City 1 | 1 | 1 |
| City 2 | 1 | 3 |
| City 3 | 4 | 1 |
| City 4 | 5 | 3 |
| City 5 | 3 | 5 |

The distance between two cities is defined as the Euclidean distance. That is:

Distance = **sqrt(** (x1 – x2)^2 + (y1 – y2)^2 **)**

For example, the distance between cities 1 and 2 will be:

Distance = **sqrt(** (1 – 1)^2 + (1 – 3)^2 **) = sqrt(** 4 **) = 2**

**The purpose:** The salesman starts his journey from city 1. He then has 4 remaining options for the next city (city 2, city 3, city 4, city 5). If he chooses city 3 as the next destination, then he will have 3 remaining options (city 2, city 4, city 5). He wants to travel all the cities and then come back to the start location (City 1). Not all the paths will be a good choice. We want to help the salesman by finding the shortest path (best order of cities to visit (visit all of them once) starting from city 1).

**Steps:**

**Step 1 [7 points]:** Create a class **City** with **x** and **y** as the class variables. The constructor with argument will get **x** and **y** and will initialize the city. Add a member function **getDistanceFrom()** to the class that gets a **city** as the input and finds the distance between the two cities.

**city1.getDistanceFrom(city2)** will be distance between city 1 and 2.

**Step 2 [6 points]:** Create5 city objects and initialize their location (x and y) using the above table. Then put all of the five cities in a vector.

**Step 3 [7 points]:** Create a two dimensional array or vector **DistanceVec** of size 5 \* 5 and initialize it such that **DistanceVec[i,j]** is the distance between **city\_i** and **city\_j**. Print the **distanceVec** and show the distance among all cities.

**Step 4 [15 points]:** If the salesman starts from the city 1, search all the possible paths and find the optimal path (order of cities to visit) that leads to the minimum total travel distance. Display the found optimal path (order of cities to travel) in your sample run.

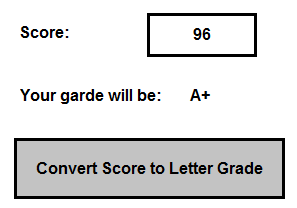
|  |
| --- |
| Your code for this problem |
| import java.lang.Math;  import java.util.Vector;  public class City {  int x;  int y;  double dis;  public City(int x, int y)  {  this.x = x;  this.y = y;  }  public int getX()  {  return x;  }  public int getY()  {  return y;  }  public double getDistanceFrom(City cx)  {  double a = Math.*pow*((cx.getX() - x), 2);  double b = Math.*pow*((cx.getY() - y), 2);  dis = Math.*sqrt*((a+b));  return dis;  }    public static void main(String[] args) {    City City1 = new City(1, 2);  City City2 = new City(1, 3);  City City3 = new City(4, 1);  City City4 = new City(5, 3);  City City5 = new City(3, 5);  Vector<City> cityVector = new Vector<City>();  cityVector.add(City1);  cityVector.add(City2);  cityVector.add(City3);  cityVector.add(City4);  cityVector.add(City5);    double [][] DistanceVec = new double [5][5];  DistanceVec[0][0] = City1.getDistanceFrom(City1);  DistanceVec[0][1] = City1.getDistanceFrom(City2);  DistanceVec[0][2] = City1.getDistanceFrom(City3);  DistanceVec[0][3] = City1.getDistanceFrom(City4);  DistanceVec[0][4] = City1.getDistanceFrom(City5);  DistanceVec[1][0] = City2.getDistanceFrom(City1);  DistanceVec[1][1] = City2.getDistanceFrom(City2);  DistanceVec[1][2] = City2.getDistanceFrom(City3);  DistanceVec[1][3] = City2.getDistanceFrom(City4);  DistanceVec[1][4] = City2.getDistanceFrom(City5);  DistanceVec[2][0] = City3.getDistanceFrom(City1);  DistanceVec[2][1] = City3.getDistanceFrom(City2);  DistanceVec[2][2] = City3.getDistanceFrom(City3);  DistanceVec[2][3] = City3.getDistanceFrom(City4);  DistanceVec[2][4] = City3.getDistanceFrom(City5);  DistanceVec[3][0] = City4.getDistanceFrom(City1);  DistanceVec[3][1] = City4.getDistanceFrom(City2);  DistanceVec[3][2] = City4.getDistanceFrom(City3);  DistanceVec[3][3] = City4.getDistanceFrom(City4);  DistanceVec[3][4] = City4.getDistanceFrom(City5);  DistanceVec[4][0] = City5.getDistanceFrom(City1);  DistanceVec[4][1] = City5.getDistanceFrom(City2);  DistanceVec[4][2] = City5.getDistanceFrom(City3);  DistanceVec[4][3] = City5.getDistanceFrom(City4);  DistanceVec[4][4] = City5.getDistanceFrom(City5);      for(int i=0; i < 5; i++)  {  System.*out*.println("");  for(int j=0; j<5; j++)  {  System.*out*.println(DistanceVec[i][j]);  }  }    System.*out*.println("The shortest rout is to go the the cities in the following order:");    String c1="City 1";  String c2="temp";  String c3="temp";  String c4="temp";  String c5="temp";  String c6= "City 1";  ///////////////    if(DistanceVec[0][1] < DistanceVec[0][2] && DistanceVec[0][1] < DistanceVec[0][3] && DistanceVec[0][1] < DistanceVec[0][4])  {  c2="City 2";  }  if(DistanceVec[0][2] < DistanceVec[0][1] && DistanceVec[0][2] < DistanceVec[0][3] && DistanceVec[0][2] < DistanceVec[0][4])  {  c2="City 3";  }  if (DistanceVec[0][3] < DistanceVec[0][1] && DistanceVec[0][3] < DistanceVec[0][2] && DistanceVec[0][3] < DistanceVec[0][4])  {  c2="City 4";  }  if(DistanceVec[0][4] < DistanceVec[0][1] && DistanceVec[0][4] < DistanceVec[0][3] && DistanceVec[0][4] < DistanceVec[0][2])  {  c2="City 5";  }  ////////////////  if(DistanceVec[1][2] < DistanceVec[1][3] && DistanceVec[1][2] < DistanceVec[1][4]);  {  c3="City 3";  }  if(DistanceVec[1][3] < DistanceVec[1][2] && DistanceVec[1][3] < DistanceVec[1][4]);  {  c3="City 4";  }  if(DistanceVec[1][4] < DistanceVec[1][2] && DistanceVec[1][4] < DistanceVec[1][3]);  {  c3="City 5";  }  //////////  if(DistanceVec[2][2] < DistanceVec[2][3]);  {  c4="City 3";  c5="City 4";  }  if(DistanceVec[2][3] < DistanceVec[2][2]);  {  c4="City 4";  c5="City 3";  }        System.*out*.println(c1+" "+c2+" "+c3+" "+c4+" "+c5+" "+c6);    }  } |

Run the code and insert the result in the following box.

|  |
| --- |
| The result |
| 0.0  1.0  3.1622776601683795  4.123105625617661  3.605551275463989  1.0  0.0  3.605551275463989  4.0  2.8284271247461903  3.1622776601683795  3.605551275463989  0.0  2.23606797749979  4.123105625617661  4.123105625617661  4.0  2.23606797749979  0.0  2.8284271247461903  3.605551275463989  2.8284271247461903  4.123105625617661  2.8284271247461903  0.0  The shortest rout is to go the the cities in the following order:  City 1 City 2 City 5 City 4 City 3 City 1 |

# P4 – 15 points

Create a simple GUI that gets the exam score and shows the corresponding letter grade by clicking on the **“Convert Score to Letter Grade”** button.



Here is the table if you need

|  |  |
| --- | --- |
| **Letter Grade** | **Percentage** |
| A | 96% - 100% |
| A- | 90% - 95% |
| B+ | 87% - 89% |
| B | 84% - 86% |
| B- | 80% - 83% |
| C+ | 77% - 79% |
| C | 74% - 76% |
| C- | 70% - 73% |
| D+ | 67% - 58% |
| D | 64% - 66% |
| D- | 60% - 63% |
| F | 00% - 59% |

|  |
| --- |
| Your C++ code for this problem |
| package application;  import javafx.application.Application;  import javafx.event.ActionEvent;  import javafx.event.EventHandler;  import javafx.geometry.Insets;  import javafx.geometry.Pos;  import javafx.stage.Stage;  import javafx.scene.Scene;  import javafx.scene.control.Button;  import javafx.scene.control.TextField;  import javafx.scene.input.KeyCode;  import javafx.scene.input.KeyEvent;  import javafx.scene.layout.HBox;  import javafx.scene.layout.VBox;  import javafx.scene.text.Font;  import javafx.scene.text.FontPosture;  import javafx.scene.text.Text;  public class Main extends Application {  private static double *score* = 0;  public static void main(String[] args) {  *launch*(args);  }  @Override  public void start(Stage window) throws Exception  {  window.setTitle("Score Grade Calculator");    HBox btns = new HBox();  btns.setPadding(new Insets(10, 10, 10, 10));  btns.setSpacing(10);  Button convertbtn = new Button("Convert Score to Letter Grade");  btns.getChildren().addAll(convertbtn);  btns.setAlignment(Pos.*CENTER*);      VBox root = new VBox();  root.setPadding(new Insets(10, 10, 10, 10));  root.setSpacing(10);    Text t1 = new Text (0, 0, "Simple Calculations");  t1.setFont(Font.*font* ("Arial", 18));    Text t2 = new Text (0, 0, "Score");  t1.setFont(Font.*font* ("Arial", 18));    TextField field1 = new TextField();  field1.setFont(Font.*font*(24));  TextField field3 = new TextField();  field3.setFont(Font.*font*(24));    root.getChildren().addAll(t1, field1, t2, field3, btns);  root.setAlignment(Pos.*CENTER*);    field1.setOnKeyPressed(new EventHandler<KeyEvent>()  {  @Override  public void handle(KeyEvent e)  {  if(e.getCode() == KeyCode.*ENTER*)  {  field1.getAccessibleText();  *score* = Integer.*parseInt*(field1.getText());  }  }  });    field3.setOnKeyPressed(new EventHandler<KeyEvent>()  {  @Override  public void handle(KeyEvent e)  {  if(e.getCode() == KeyCode.*ENTER*)//user has to hit enter on each  {    }  }  });  convertbtn.setOnAction(new EventHandler<ActionEvent>()  {  @Override  public void handle(ActionEvent event)  {  if(*score* > 95)  field3.setText("A");  if(*score* > 89 && *score* < 96)  field3.setText("A");  if(*score* > 86 && *score* < 90)  field3.setText("B+");  if(*score* > 83 && *score* < 87)  field3.setText("B");  if(*score* > 79 && *score* < 84)  field3.setText("B-");  if(*score* > 76 && *score* < 80)  field3.setText("C+");  if(*score* > 73 && *score* < 77)  field3.setText("C");  if(*score* > 69 && *score* < 74)  field3.setText("C-");  if(*score* > 66 && *score* < 69)  field3.setText("D+");  if(*score* > 63 && *score* < 67)  field3.setText("D");  if(*score* > 59 && *score* < 64)  field3.setText("D-");  if(*score* < 60)  field3.setText("F");  }    }  );  Scene scene = new Scene(root, 300, 300); //creates a background Width x Height    window.setScene(scene);  window.show();    }  } |

Run the code for two cases with **two different cases** and Insert the result (**screen shot of the GUI with the result**) in the following box.

|  |
| --- |
| The result of the query |
|  |