**1.a.**

/\*\*

\* Prompts the user for a number of dance moves in the range 1 to 3 inclusive.

\* If the number that the user entered is within the range of 1 to 3 inclusive, it will return the number.

\* If the number is not within this range or if the user entered something other than an number the method will set the number of dance moves

\* to 0, and ask the user to enter a numer of dance moves.

\*/

public int promptForNumOfMoves()

{

int moves = 0;

while (! isValidNumOfMoves(moves));

{

try

{

moves = Integer.parseInt(OUDialog.request

("Please enter the number of dance moves to be performed - between 1 and 3 (inclusive)"));

}

catch (NumberFormatException e)

{

moves = 0;

}

}

return moves;

}

**1.b.**

/\*\*

\* This method sets the number of Frogs Dancing to 0, as well as setting the three frogs used as arguments for the FrogDanceCoordination

\* to the colour of green.

\* It then prompt the user if they would like each of the frogs to dance one by one.

\* If you user clicks yes it sets their colour to Red, Position to 1 and increases the number of Frogs dancing by one.

\*/

public void selectDancingFrogs()

{

numOfFrogsDancing = 0;

frog1.setColour(OUColour.GREEN);

frog2.setColour(OUColour.GREEN);

frog3.setColour(OUColour.GREEN);

boolean frog1awnser = OUDialog.confirm("Would you like frog1 to dance");

if ( frog1awnser == true)

{

frog1.setColour(OUColour.RED) ;

frog1.setPosition(1);

numOfFrogsDancing ++;

}

else

{

frog1.setColour(OUColour.GREEN); // This ensures that if the user clicked no frog1 will not dance

}

boolean frog2awnser = OUDialog.confirm("Would you like frog2 to dance");

if ( frog2awnser == true)

{

frog2.setColour(OUColour.RED) ;

frog2.setPosition(1);

numOfFrogsDancing ++;

}

else

{

frog2.setColour(OUColour.GREEN); // This ensures that if the user clicked no frog1 will not dance

}

boolean frog3awnser = OUDialog.confirm("Would you like frog3 to dance");

if ( frog3awnser == true)

{

frog3.setColour(OUColour.RED) ;

frog3.setPosition(1);

numOfFrogsDancing ++;

}

else

{

frog3.setColour(OUColour.GREEN); // This ensures that if the user clicked no frog1 will not dance

}

}

**1.c.**

/\*\*

\* This method asks the user to type the number of dance moves that they want to be performed.

\* It will keep asking the user to input a number until the user enters a valid number.

\* This then calls the method prompForDanceMove.

\* If the number of dancemoves is greater than 1 the user is requested to enter the name of danceMove2.

\* If the number of danceMoves is greater than 2 the user is requested to enter the name of danceMove3.

\*/

public void createDance()

{

int input = 0;

while (! isValidNumOfMoves(input))

try

{

input = Integer.parseInt(OUDialog.request

("Please enter the number of dance moves to be performed"));

}

catch (NumberFormatException e)

{

input = 0;

}

numOfDanceMoves = input;

if (numOfDanceMoves > 1)

{

danceMove1 = promptForDanceMove();

}

if (numOfDanceMoves > 1)

{

danceMove2 = promptForDanceMove();

}

if (numOfDanceMoves > 2)

{

danceMove3 = promptForDanceMove();

}

}

**1.d.i**

/\*\*

\* This methods calls the method selectDancingFrogs and then createDance.

\*/

public void setUpDance()

{

selectDancingFrogs();

createDance();

}

**1.d.ii**

/\*\*

\* This method checks to see if the number of dancing frogs is 1 or more, and if it is it tells all of the frogs to perform danceMove 1.

\* The method then checks to see if the number of dancing frogs is 1 or more and if it is, it tells all of the dancing frogs to perform danceMove2.

\* Then the method checks to see if the number of dancing frogs is 2 or more and if it is, it tell all of the dancing frogs to perfrom danceMove3.

\*/

public void performDance()

{

if (numOfFrogsDancing >= 1)

{

{

if (danceMove1.equals("flip") )

flipDancingFrogs();

}

{

if (danceMove1.equals("slide"))

slideDancingFrogs();

}

if (numOfFrogsDancing > 1)

{

{

if (danceMove2.equals("flip") )

flipDancingFrogs();

}

{

if (danceMove2.equals("slide"))

slideDancingFrogs();

}

}

if (numOfFrogsDancing >2)

{

{

if (danceMove3.equals("flip") )

flipDancingFrogs();

}

{

if (danceMove3.equals("slide"))

slideDancingFrogs();

}

}

}

}

**2.a.i**

public class Payroll

{

private Set <Employee> employeeSet;

/\*\*

\* Constructor for objects of class Payroll

\*/

public Payroll()

{

employeeSet = new HashSet<>();

}

**2.a.ii**

/\*\*

\* 2.A.ii

\* This method creates a new instance of Employee.

\* The string argument is used as the employeeNumber for the new instance.

\* It takes a String as an argument, and does not return anything.

\*/

public void enrollEmployee(String aString)

{

Employee aEmp = new Employee(aString);

employeeSet.add(aEmp);

}

**2.a.iii**

/\*\*

\* 2.A.iii

\* This method prints out each employee's record on is own individual line.

\* It uses the format First Name, Second Name, Employee Name, "Sales", Sales figure.

\* It does not take an argument, and does not return anything.

\*/

public void printEmployees()

{

for (Employee eachElement: employeeSet)

{

System.out.println(eachElement.getFirstName()+" " + eachElement.getSecondName() + " " + eachElement.getEmployeeNumber() + " Sales " + eachElement.getSales() + " ");

}

}

**2.b.**

/\*\*

\* 2.B

\* This method iterated though the set of employees against the employees in the Map.

\* If the employee appears in both, it will update their sales figures to be the total of the two figures, from the set of employees and the map.

\* It takes a map whose keys are strings and values are Integers as an argument, and does not return anythng.

\*/

public void updateSalesFigures(Map <String, Integer> m) // (Sales.getSales());

{

for (Employee eachElement: employeeSet)

{

if (m.containsKey(eachElement.getEmployeeNumber()))

{

eachElement.setSales(eachElement.getSales()+ m.get(eachElement.getEmployeeNumber()));

}

}

}

**2.c.i**

/\*\*

\* 2.C. i

\* This method return true if either the employees first name or second name has a string assigned to it.

\* It returns false if neither of the names is assigned a string.

\* It takes no arguments and returns a boolean value.

\*/

public boolean hasName()

{

return (this.getFirstName() != null) || (this.getSecondName() != null);

}

**2.c.ii**

/\*\*

\* 2.C.ii

\* This method iterates through the set of employees and determines if both their First and Second names are assigned to the value null.

\* If they are, it prompts the user to enter a new first and second name of the employee.

\* The string the user entered is then split at the space and the first half is assigned to the employees first name.

\* The second half is assigned to the employees second name.

\* If the user clicked cancel as opposed to entering a string, the loop continues to the next relevant value.

\* It does not take an argument, and does not return anything.

\*/

public void nameEmployees()

{

for (Employee eachElement: employeeSet)

{

boolean test = (eachElement.getFirstName()== null && eachElement.getSecondName()== null);

if (test == true)

{

String newName = OUDialog.request("Enter first and last name of employee with number: " + eachElement.getEmployeeNumber());

if (newName != null)

{

String[] parts = newName.split(" ");

String part1 = parts[0];

String part2 = parts[1];

eachElement.setFirstName(part1);

eachElement.setSecondName(part2);

}

else

{

continue ;

}

}

}

}

**3.a.**

A library is keeping a record of movies using a map, there the keys of the map are Strings representing the title of the movie. Each key represents a set of Strings representing the actors who star in the movie.

**3.b.**

/\*\*

\* This method populates the map, with a series of movie titles representing the lead actors in the movie.

\* @param void

\* @return void

\*/

public void populateLibary()

{

Set<String> actorSet = new HashSet<>();

actorSet.add("Gwenth Paltro");

actorSet.add("Robert Downie Jr");

listOfMovies.put("Iron Man", actorSet);

actorSet = new HashSet<>();

actorSet.add("Russel Crowe");

actorSet.add("Max von Sydow");

listOfMovies.put("Robin Hood", actorSet);

actorSet = new HashSet<>();

actorSet.add("Colin Firth");

actorSet.add("Mark Strong");

listOfMovies.put("Kingsman", actorSet);

actorSet = new HashSet<>();

actorSet.add("Robert Downie Jr");

actorSet.add("Jude Law");

listOfMovies.put("Sherlock Holmes", actorSet);

actorSet = new HashSet<>();

actorSet.add("Brendan Fraser");

actorSet.add("Rachel Weisz");

listOfMovies.put("The Mummy", actorSet);

actorSet = new HashSet<>();

actorSet.add("Rowan Atkinson");

actorSet.add("Ben Miller");

listOfMovies.put("Johnnie English", actorSet);

actorSet = new HashSet<>();

actorSet.add("Hugh Jackman");

actorSet.add("Tao Okamoto");

listOfMovies.put("The Wolverine", actorSet);

}

**3.c.**

/\*\*

\* 3.C.i

\* This method iterates over a map's keys, and prints out a representation of the map's contence.

\* @param void

\* @return void

\*/

public void printMap()

{

for (String aMovie : listOfMovies.keySet())

{

System.out.println(aMovie + " stars " + listOfMovies.get(aMovie));

}

}

/\*\*

\* 3.C. ii

\* This method takes an argument of a string representing the key to the map.

\* If the movie searched for is in the map it will return a list of the actors in the movie.

\* If the movie searched for is not in the map it will return a message saying the movie is not in the libary.

\* @param String

\* @return void

\*/

public void printMapValue(String aString)

{

boolean movieTest = listOfMovies.containsKey(aString);

if (movieTest == true)

{

System.out.println("The movie you searched for stars " + listOfMovies.get(aString));

}

else

{

System.out.println("The movie you searched for is not in our libary");

}

}

/\*\*

\* 3.C.iii

\* This method takes a string and a set of strings as an argument.

\* The string used as an argument is used as the movie name, and the set of strings is used as the actors names.

\* The new entry is then added to the map.

\* @param String, Set<String>

\* @return void

\*/

public void addMapEntry(String aMovieName, Set<String> actorsNames)

{

listOfMovies.put(aMovieName, actorsNames);

}

**3.d.**

/\*\*

\* Deletes the key and value from the map, if the string taken as an argument is in the map.

\* If the map entry is deleted the method returns true.

\* If the movie is not in the map, the method returns false.

\* @param String

\* @return boolean

\*/

public boolean deleteEntry(String movieToBeDeleted)

{

if (listOfMovies.containsKey(movieToBeDeleted))

{

listOfMovies.remove(movieToBeDeleted);

return true ;

}

else

{

return false;

}

}

/\*\*

\* This method returns a collection of keys that contain the actor used as the argument.

\* @param String

\* @return Set<String>

\*/

public Set<String> selectKeys(String actorsName)

{

Set<String> sortedMovieList = new HashSet<>();

for (String aMovie: listOfMovies.keySet())

{

if (listOfMovies.get(aMovie).contains(actorsName))

{

sortedMovieList.add(aMovie);

}

}

return sortedMovieList;

}

/\*\*

\* This method adds a new actor to a movie that is already in the map.

\* @param String, String

\* @return void

\*/

public void addValue(String movieName, String newActorsName)

{

listOfMovies.get(movieName).add(newActorsName);

}

/\*\*

\* This method removes an actor from a movie that is already in the map.

\* @param String, String

\* @return void

\*/

public void deleteValue(String movieName, String actorToBeRemoved)

{

listOfMovies.get(movieName).remove(actorToBeRemoved);

}

**4.a.i.**

/\* static variables \*/

static int nextNumber = 1;

**4.a.ii**

/\*\*

\* Constructor for objects of class Entrant.

\*/

public Entrant()

{

super();

this.name = "";

this.category = "";

this.time = 0.0;

nextNumber ++;

this.number = nextNumber;

}

**4.b.i.**

private List <Entrant> entrants;

/\*\*

\* Constructor for objects of class WCCR

\*/

public WCCR()

{

entrants = new ArrayList<>();

}

**4.b**

/\*\*

\* This method prompts the user to select a text file.

\* It then scans the contens of the file, and adds the information to the map of entrants.

\* @param void

\* @return void

\*/

public void readInEntrants()

{

String pathName = OUFileChooser.getFilename();

File entrantsList = new File(pathName);

BufferedReader bufferedFileReader = null;

try

{

Scanner scanner = new Scanner(new BufferedReader(new FileReader(entrantsList)));

scanner.useDelimiter(",");

while (scanner.hasNext())

{

String name = scanner.next();

String category = scanner.nextLine().substring(1);

Entrant entrant = new Entrant();

entrant.setName(name);

entrant.setCategory(category);

entrants.add(entrant);

}

}

catch (Exception anException)

{

System.out.println("Error: " + anException);

}

}

**4.b.iii**

/\*\*

\* Generates a random result between the start(30.00) and end times (60.00)

\* @param void

\* @return double

\*/

private double generateTime()

{

double start = 30.00;

double end = 60.00;

double random = new Random().nextDouble();

double result = start + (random \* (end - start));

return result;

}

**4.b.iv**

/\*\*

\* This method loops over the entrants assigning each one a randomly generated time.

\* @param void

\* @return void

\*/

public void runRally()

{

for (Entrant eachEntrant: entrants)

{

eachEntrant.setTime(generateTime());

}

}

**4.c.i**

/\*\*

\* This method compares the times of two entrants.

\* @param Entrant

\* @return integer

\*/

public int compareTo(Entrant aEntrant)

{

int result = Double.compare(this.getTime(), aEntrant.getTime());

return result;

}

**4.c.ii**

/\*\*

\* This method sorts the entrants by their times

\* @param void

\* @return void

\*/

public void sortEntranceList()

{

Collections.sort(entrants);

}

**4.d.i & ii**

public class WCCR

{

// instance variables - replace the example below with your own

private List <Entrant> entrants;

private Map <String, Double> e1Results;

private Map <String, Double> e2Results;

private Map <String, Double> e3Results;

/\*\*

\* Constructor for objects of class WCCR

\*/

public WCCR()

{

entrants = new ArrayList<>();

e1Results = new HashMap<>();

e2Results = new HashMap<>();

e3Results = new HashMap<>();

}

**4.d.iii**

/\*\*

\* This method sorts the entrance by their catagory and added them to the appropriate map.

\* @param void

\* @return void

\*/

public void categorise()

{

for (Entrant eachEntrant: entrants)

{

if (eachEntrant.getCategory().equals("E1"))

{

e1Results.put(eachEntrant.getName(), eachEntrant.getTime());

}

if (eachEntrant.getCategory().equals("E2"))

{

e2Results.put(eachEntrant.getName(), eachEntrant.getTime());

}

if (eachEntrant.getCategory().equals("E3"))

{

e3Results.put(eachEntrant.getName(), eachEntrant.getTime());

}

}

}