Pointless-Mini Project Level 7

December 1, 2020

1 Pointless-Mini Project

2 Level 7

2.1 Dilkush Punja

2.2 Tuesday 1st December 2020

2.3 Version 7.0

2.4 Summary of the Question

Write a program that resembles the Pointless game show. In which a user is asked a question and the player is awarded points based on the most correct unique answer. The aim is to choose an answer that is correct but is the least popular response. To win the game you need to gain as few points as possible.

2.5 The Literate Program Development

2.5.1 Method 1-main()

What it does Method 1 is the main method, it prints out a user menu in which the user can confirm if they want to continue. If the user enters 'TARDIS' they can access admin mode. Otherwise it prints out a question alongside potential answers. The user is able to enter their answer and their score is saved within an array. After all players have been asked questions. The program then prints out the Pointless leaderboard.

Implementation (how it works) Method 1 utilises function calls in order to carry out the steps out above. If the user enters admin mode they have the option to create new questions, in which they are prompted to enter a topic, questions and answers this is then saved into a .csv file. Or they can enter a filename manually. Upon exiting admin mode it creates a player record storing the number of players and their names. It will also store each player's score. The program then continues to use a for loop to ask each player a number of questions. The questions are printed using the getQuestion method, potential answers are printed using the getPotentialAns method and the users response is marked using the marking method. After each player has been asked their final question their total score is saved within the player record using the setPlayerScores method. The method then calls the printPlayerScores method printing each player's score.

[2]:

```
/** The main method uses function and procedure calls to ask the user set␣
 ↪questions in a
random order. The method prints suitable answer and the user is prompted to␣
 ↪enter
their "pointless" answer.
**/
public static void main(String[] args) throws IOException
{

// Below is the start menu:
//************************************************************************
print("Welcome to Pointless!");
String response=inputString("Would you like to start? Enter Y/N");

String filename="Random.csv";

if(response.equals("N") || response.equals("n"))
{
        print("It looks like you do not want play Pointless!");
        print("Goodbye!");
        System.exit(0);
}
else if(response.equals("TARDIS"))   // If the user enters 'TARDIS' they are␣
 ↪given access to admin mode.
{
        print("Your administrator options are below:");
        print("A- Create new questions");
        print("B- Enter a filename");
        print("C- Play Pointless!");
        String response2=inputString("");

        if(response2.equals("A"))
        {
                filename=designQuestions();
        }
        else if (response2.equals("B"))
        {
                filename=inputString("Please enter a file name, remember to␣
 ↪include '.csv'.");
        }
        else if (response2.equals("C"))
        {

        }
        else
        {
                System.exit(0);
        }

}

//**************************************************************************

/** Declaration of the variable 'question1' of the 'Data' type. Values are␣
 ↪passed to
function as an argument in the function call.
**/
        Random randomQuestion = new Random(); // Variable of the Random type is␣
 ↪created.
        int score=0;
        int totalScore=0;
        final int numberOfQuestions=2;

        Player players=createPlayers();

        for (int k=1; k<=getNumberOfPlayers(players); k++)
        {

                for (int i=1; i<=numberOfQuestions; i++) // Counter controlled␣
 ↪for loop is used to ask multiple questions.
                {
                        int questionNumber = randomQuestion.nextInt(5)+0; //␣
 ↪The variable 'questionNumber' is assigned a randomly generated number to␣
 ↪randomise the order of questions.
                        Data question=initialisation(questionNumber,filename);
                        print("\n" + getPlayerNames(players,k)+ ", here is␣
 ↪question number " + i + ":");
                        print(getQuestion(question));

                        print("\n"+"Your options are below:" + "\n"); // The 4␣
 ↪potential answers are printed.

                                for (int j=1; j<=4; j++)
                                {
                                        print(getPotentialAns(question,j));
 ↪
                                }

                        print("\n");
                        String answer=inputAnswer();
                        score = marking(question, answer); // This variable␣
 ↪contains the score awarded to the player's answer.
```

```
                        print("\n"+"You received " + score + " points."); //␣
↪The user's score is printed.
                        print("");
                        totalScore=totalScore + score; // This variable␣
↪contains the player's total score.


                }

                setPlayerScores(players,k,totalScore); // The player's total␣
↪score is saved in the player record.
                totalScore=0;
        }

        printPlayerScores(players);
/** An exit command has been used below to quit the program after the␣
↪instructions within
the main method have been completed.
**/
        System.exit(0);
}
```

2.5.2 Primitive Operations-‘Data’ Abstract Data Type

initialisation() is a primitive operation method.

Data is an abstract data type as the way we have actually implemented it has been hidden by using methods for each of these operations. The program only manipulates the values of this type by using these operations.

The ‘Data’ abstract data type includes operations such as: Returning records of the ‘Data’ type. Creating records of the ‘Data’ type. Accessing the question field of a question record. Accessing the potentialAnswers field of a question record. Accessing the answerScores field of a question record.

2.5.3 Method 2 -initialisation()

What it does This function creates an array containing question records. The record contains the question itself and the potential answers as well as their scores. The record is created by reading the .csv file based on the filename passed to the method. Once the records have been made, the function returns a record based on the randomly generated number passed to it.

Implementation (how it works) Method 2 uses the createQuestion method which is given array arguments such as the potential answers and their scores. The method then returns a record containing all that information. This occurs for each question.

[2]:
```
/** This function accesses the csv file from the given filename and is used to␣
↪create an
array of question records. Once the records have been made, the function␣
↪returns a record
based on the randomly generated number passed to it.
**/

public static Data initialisation(int questionNumber,String filename) throws␣
↪IOException
{
        Data [] qBank= new Data [6]; // An array of question records.

        BufferedReader inputStream = new BufferedReader (new␣
↪FileReader(filename));
        for (int i=0; i<qBank.length; i++)
        {

                String question= inputStream.readLine();

                String s= inputStream.readLine();
                String [] answers=s.split(",");

                s= inputStream.readLine();

                String [] scoresInString=s.split(",");

                int [] answerScores= new int [4];

                for(int j=0; j<4; j++)
                {
                        answerScores[j]=Integer.parseInt(scoresInString[j]);
                }

                qBank[i]=createQuestion(question,answers,answerScores);
        }

        inputStream.close();
        return qBank[questionNumber];
}
```

2.5.4 Primitive Operations-‘Data’ Abstract Data Type

createQuestion() is a primitive operation method.

Data is an abstract data type as the way we have actually implemented it has been hidden by using methods for each of these operations. The program only manipulates the values of this type by using these operations.

2.5.5 Method 3- createQuestion()

What it does Method 3 creates a new record of the 'Data' type which stores a question's information. This includes the actual question, alongside potential answers and the score for each potential answer.

Implementation (how it works) The function works by being passed values stored in arrays containing potential answers and their scores. Values such as the question are stored in a local String variable, answers and scores are stored in separate String and integer arrays. A record 'x' is created which contains the fields .questions, .potentialAnswers, .answerScores. The fields are then assigned the previously created arrays containing the values passed to the method. 'x' is used to enable this method to be used multiple times with the only difference being the values passed to the method to create a new question record. Once assignment is complete the method returns the 'question' record to where the method was called.

[4]:
```
// This function is used to create a record for a question. It is passed array␣
↪arguments.
public static Data createQuestion(String question, String possibleAnswers [],␣
↪int answerPoints []) // Arguments passed to the method in the function call␣
↪are used to assign values in the appropriate array.
{
        Data x = new Data();

        x.questions = question;
        x.potentialAnswers = possibleAnswers;
        x.answerScores = answerPoints;

        return x;
}
```

2.5.6 Method 4- marking()

What it does This function is used to 'mark' the users response according to the values stored in the corresponding question record. The method prints whether the user entered a Pointless answer which is worth 0 points or an incorrect answer which is worth 100. The method then returns the player's score to where the function was called.

Implementation (how it works) The method works by utilising boolean expressions to check whether the user's response matches any of the potential answers. The potential answers are accessed using accessor methods which return the value stored in that particular position of the field array. If there is a match then the score variable is assigned the value stored in the AnswerScore field array. If the player's response is incorrect the score is set to 100. The method then returns the score value to wherever the method was called.

[6]:
```
// This function is used to 'mark' the user's response.
public static int marking(Data question, String answer)
{
        int score=0;
```

```
/** If the user's response matches a answer in the record they are given the␣
↪score
associated with that answer.
**/

        if (answer.equals(getPotentialAns(question,1))) /** Boolean expression␣
↪used to verify
                                                                                the␣
↪user's answer.**/
                {
                        score=getAnswerScores(question,1);
                }

        else if (answer.equals(getPotentialAns(question,2)))
                {
                        score=getAnswerScores(question,2);
                }

        else if (answer.equals(getPotentialAns(question,3)))
                {
                        score=getAnswerScores(question,3);
                }

        else if (answer.equals(getPotentialAns(question,4)))
                {
                        score=getAnswerScores(question,4);
                }

        else
                {
                        score=100; // If the given answer is incorrect the␣
↪user's score is set to 100.
                        print("Unfortunately your response was not a correct␣
↪answer.");

                        return score; // The score is returned to the function␣
↪call.
                }

        if (score==0) // If a users enters a pointless answer they are awarded␣
↪0 points.
        {
                print("Congratulations, you entered a Pointless answer!");
        }
        return score; // The score is returned to the function call.
}
```

2.5.7 Primitive Operations-'Player' Abstract Data Type

Below are the set of primitive operations which will be performed on the Player data type.

The 'Data' abstract data type includes operations such as: Firstly, creating a player which involves:Storing the number of players and Setting the name of players .

Getting the number of players. Getting the name of players.

Setting player scores. Getting player scores.

Ordering the array containing the player scores from lowest to highest.

These accessor methods are the only way for the rest of the program to access values in the array fields.

2.5.8 Method 5- createPlayer()

What it does This function is used to create a record containing player information.

Implementation (how it works) This method works by creating a new variable called players of the Player type. The user then enters the number of players which is stored in the .numberOfPlayers field. Input validation is used, so if the users enter a value less than 1 the program quits because you cannot play with 0 players. An array of the same length as the number of players is created and is assigned to the playerNames field. This also occurs for the playerScores field. A counter controlled loop based on the number of players allows players to enter their names. Their names are saved in the playerNames field. The method then returns the 'players' record to where the method was called.

```
[ ]: // This function is used to create a record for player information.
public static Player createPlayers()
{
        Player players=new Player(); // A new variable called 'player' is␣
 ↪declared of the Player type.

        players.numberOfPlayers=inputInt("Please enter the number of players.");

        if (players.numberOfPlayers <1) // Input validation
        {
                print("\n" + "It seems like you do not want to play Pointless!
 ↪");
                print("Goodbye!");
                System.exit(0);
        }

        players.playerNames= new String [players.numberOfPlayers];
        players.playerScores= new int [players.numberOfPlayers];

/** The counter controlled loop below is used to allow players to enter their␣
 ↪names, the
loop is repeated until all players have done so. **/
        for (int i=0; i<=players.numberOfPlayers-1; i++)
        {
                players.playerNames[i]=inputString("Enter player " + (i+1) +␣
 ↪"'s name.");
        }

        return players;
}
```

2.5.9 Method 6- getNumberOfPlayers()

What it does This function returns the number of players.

Implementation (how it works) The function is passed to it the player record as argument. It then access the .numberOfPlayers field and returns the value stored inside to where the method was called.

[ ]:
```
// This function is an accessor method to access the total number of players.
public static int getNumberOfPlayers(Player players)
{
        return players.numberOfPlayers;
}
```

2.5.10 Method 7- getPlayerNames()

What it does This function returns a player's name.

Implementation (how it works) This method is passed the players record and a player number. 1 is subtracted from the player number as an array's first element is designated as compartment 0. The program then accesses the .playerNames field and the array inside and in particular the compartment of the given player number.

[ ]:
```
// This function is an accessor method to access a player's name.
public static String getPlayerNames(Player players, int playerNumber)
{
        playerNumber=playerNumber-1;
        return players.playerNames[playerNumber];
}
```

2.5.11 Method 8- getPlayerScores()

What it does This function returns a player's total score.

Implementation (how it works) This method is passed the players record and a player number. 1 is subtracted from the player number as an array's first element is designated as compartment 0. The program then accesses the .playerScores field and the array inside and in particular the compartment of the given player number.

[ ]:
```
// This function is an accessor method to access a player's total score.
public static int getPlayerScores(Player players, int playerNumber)
{
        playerNumber=playerNumber-1;
        return players.playerScores[playerNumber];
}
```

2.5.12 Method 9- setPlayerScores()

What it does This function assigns the total score for a player. The value is saved in the .playerScore field.

Implementation (how it works) This method is passed the players record, the player number and the score. 1 is subtracted from the player number as an array's first element is designated as compartment 0. The program then accesses the .playerScores field and the array inside and in particular the compartment of the given player Number and assigns the compartment the score value.

[ ]:
```
// This function stores a player's total score into the array in the record␣
 ↪player.
public static Player setPlayerScores(Player players, int playerNumber, int␣
 ↪score)
{
        playerNumber=playerNumber-1;
        players.playerScores[playerNumber]=score;

        return players;
}
```

2.5.13 Method 10- arraySort()

What it does This function is used to order player scores so they can be printed in a leaderboard.

Implementation (how it works) This method is passed the players record and accesses the playerNames and playerScores fields. It orders each score from lowest to highest. Based upon this order the player names are also ordered so the correct score corresponds to the correct player.

[ ]:
```
/** This function uses 'bubble sort' to order player's scores so that they can␣
 ↪be printed
in a leaderboard, the scores are arranged from low to high within the␣
 ↪playerScores array.
**/
public static Player arraySort(Player players)
```

```
{
        for (int pass=1; pass <=getNumberOfPlayers(players)-1; pass++)
        {
                for (int i=0; i< getNumberOfPlayers(players)-pass; i++)
                {
                        if (players.playerScores[i] > players.playerScores[i+1])
                        {
                                        int tmp1= players.playerScores[i+1];
                                        players.playerScores[i+1]=players.
 ↪playerScores[i];
                                        players.playerScores[i]= tmp1;

                                        String tmp2=players.playerNames[i+1];
                                        players.playerNames[i+1]=players.
 ↪playerNames[i];
                                        players.playerNames[i]=tmp2;
                        }
                }
        }

        return players;
}
```

2.5.14 Primitive Operations-'Data' Abstract Data Type

Below are the set of primitive operations which will be performed on the 'Data' data type. The operations include, getting a question, potential answers and answer scores.

Data is an abstract data type as the way we have actually implemented it has been hidden by using methods for each of these operations. The program only manipulates the values of this type by using these operations.

2.5.15 Method 11- getQuestion()

What it does This function returns the question from the question record.

Implementation (how it works) This method is passed a question record, it then accesses the .questions field of the record. The value stored in that compartment is then returned.

[ ]:
```
// This function is an accessor method to access a question stored in the␣
 ↪question record.
public static String getQuestion(Data question)
{
        return question.questions;
}
```

2.5.16 Method 12- getPotentialAns()

What it does This function returns one of the potential answers saved in the question record.

Implementation (how it works) This method is passed the question record and a answer number. 1 is subtracted from the answer number as an array's first element is designated as compartment 0. The program then accesses the .potentialAnswers field and the array inside and in particular the compartment of the given answer number and returns the value.

[ ]:
```
// This function is an accessor method to access appropriate answers to a␣
 ↪particular question.
public static String getPotentialAns(Data question, int ansNumber)
{
        ansNumber=ansNumber-1;

        return question.potentialAnswers[ansNumber];
}
```

2.5.17 Method 13- getAnswerScores()

What it does This function returns the score of one of the potential answers.

Implementation (how it works) This method is passed the question record and a score number. 1 is subtracted from the score number as an array's first element is designated as compartment 0. The program then accesses the .answerScores field and the array inside and in particular the compartment of the given score number and returns the value.

[ ]:
```
// This function is an accessor method to access answer scores to a particular␣
 ↪question.
public static int getAnswerScores(Data question, int scoreNumber)
{
        scoreNumber=scoreNumber-1;
        return question.answerScores[scoreNumber];
}
```

2.5.18 Method 14- printPlayerScores()

What it does This procedure prints out all the player names and well as each player's score.

Implementation (how it works) This method is passed the players record. Using a counter controlled loop based upon the number of players it prints out the players names and their score using the appropriate accessor method.

[ ]:
```
// This procedure is used to print out the scores for each player at the end.
public static void printPlayerScores(Player players)
{
        print("\n" + "Pointless Leaderboard:");
        for (int j=1; j<=getNumberOfPlayers(players); j++)
        {
                print(getPlayerNames(players,j)+ " " +␣
 ↪getPlayerScores(players,j));
        }
}
```

2.5.19 Method 15- designQuestions()

What it does This function allows the user to create their own questions.

Implementation (how it works) This method is only called from the admin mode. The method creates a .csv file called the name of a topic the user enters. They are then prompted to enter a question and the potential answers along with their scores. The method prints what the user enters to the .csv file, this occurs 6 times and then the method closes the .csv file and returns the name of the file created.

[ ]:
```
/** This function enables the user to enter their own questions and answers␣
 ↪which are
saved in a csv file. The filename is then returned so subsequent methods can
access the file and print accordingly.
**/
public static String designQuestions() throws IOException
{

        String filename=inputString("What topic would you like the questions to␣
 ↪be on?")+ ".csv";
        PrintWriter outputStream= new PrintWriter(new FileWriter(filename));

        for(int i=1; i<=6; i++)
        {
                outputStream.println(inputString("\nPlease enter a question you␣
 ↪would like to ask."));

                outputStream.println(inputString("Enter the suitable answers.␣
 ↪In this format: Blue,Red,Green,Yellow"));
                outputStream.println(inputString("Please assign the answers a␣
 ↪score. In this format: 34,76,23,41"));
        }
        outputStream.close();

        return filename;
}
```

2.5.20 File Format:

The method will create a .csv file, it will be called the topic entered by the user. The method will print 18 lines in total and the information for each question will be on 3 lines. For example:

Geography.csv

A country in Europe.

UK Germany France Norway

90 34 65 12

Name a continent.

Africa Europe Asia South America

78 99 56 21

Name US states.

North Dakota Indiana Texas Michigan

34 76 23 41

Name a process of the water cycle.

Transpiration Evaporation Run-off Precipitation

45 32 2 21

Name a process of the carbon cycle.

Decomposition Respiration Photosynthesis Combustion

32 2 3 4

Countries beginning with the letter M.

Malawi Mali Marshall Islands Malta

34 12 45 67

2.5.21 Method 16- inputAnswer()

What it does This function enables the user to enter their answer to the question given.

Implementation (how it works) This method works by using the scanner utility to allow the user to enter their answer via the keyboard and returns the string to the function call.

[10]:
```
// This function is used to enable a user to enter their answer.
public static String inputAnswer()
{
        String text_input;
        Scanner scanner= new Scanner(System.in);

        text_input=scanner.nextLine();

        return text_input;
}
```

[11]:
```
inputAnswer();
```

```
Test 1
```

```
[11]: Test 1
```

2.5.22 Method 17- inputString()

What it does This function enables a user to enter to enter a String value.

Implementation (how it works) This method works by using the scanner utility which enables the user to enter a sequence of characters to form a string. The string is then returned to wherever the method was called.

[3]:
```
// This function enables the user to enter String values.
public static String inputString(String message)
{
        String text_input;
        Scanner scanner= new Scanner(System.in);

        System.out.println(message);
        text_input=scanner.nextLine();

        return text_input;
}
```

[4]:
```
inputString("Testing");
```

```
Testing
Hello
```

```
[4]: Hello
```

2.5.23 Method 18- inputInt()

What it does This function enables a user to enter to enter an integer value.

Implementation (how it works) This method works by calling the inputString method which returns a String value, this value is then converted to an integer value by the method 'Integer.parseInt'.

[7]:
```
// This function enables the user to enter integer values.
public static int inputInt(String message)
{
        System.out.println(message);
        String text_input=inputAnswer();
        int x=Integer.parseInt(text_input);

        return x;
}
```

[12]:
```
inputInt("What number would you like me to store?");
```

```
What number would you like me to store?
76
```

[12]:
```
76
```

2.5.24 Method 19- print()

What it does A simple print procedure to print variables or text.

Implementation (how it works) This procedure uses abstraction to simplify the printing process. It works by printing the value passed to it by the procedure call.

[5]:
```
// This procedure uses abstraction, to simplify the use of 'System.out.println'.
public static void print(String message)
{
        System.out.println(message);
        return;

}
```

[8]:
```
print("Hi!");
```

```
Hi!
```

2.5.25 Running the program

Run the following call to simulate running the complete program.

[ ]:
```
main();
```

2.6 The complete program

This version will only compile here. To run it copy it into a file called initials.java on your local computer and compile and run it there.

[1]:
```
// Pointless Quiz

/** Program Credentials:

 @Author        Dilkush Punja
 @Date                1 December 2020
 @Version        7.0

The purpose of the program is to resemble the Pointless game show where the␣
 ↪player is
```

```
awarded points based on the most correct unique answer. The aim is to choose an␣
 ↪answer
that is correct but is the least popular response. To win the game you need to␣
 ↪gain as
few points as possible.
**/

import java.util.Scanner; // Import the scanner class
import java.util.Random; // Import the random number class
import java.io.*; // Import file input and output class


/** Below is a record called 'Data' which includes fields containing questions,␣
 ↪possible
answers and their scores.
**/

class Data
{
        String questions;
        String [] potentialAnswers;
        int [] answerScores;
}

class Player
{
        int numberOfPlayers;
        String [] playerNames;
        int [] playerScores;
}

// Below the class 'Pointless' has been defined. It contains all the programs␣
 ↪methods.
public class Pointless
{

/** The main method uses function and procedure calls to ask the user set␣
 ↪questions in a
random order. The method prints suitable answer and the user is prompted to␣
 ↪enter
their "pointless" answer.
**/
public static void main(String[] args) throws IOException
{

// Below is the start menu:
```

```
//*************************************************************************
print("Welcome to Pointless!");
String response=inputString("Would you like to start? Enter Y/N");

String filename="Random.csv";

if(response.equals("N") || response.equals("n"))
{
        print("It looks like you do not want play Pointless!");
        print("Goodbye!");
        System.exit(0);
}
else if(response.equals("TARDIS"))   // If the user enters 'TARDIS' they are␣
 ↪given access to admin mode.
{
        print("Your administrator options are below:");
        print("A- Create new questions");
        print("B- Enter a filename");
        print("C- Play Pointless!");
        String response2=inputString("");

        if(response2.equals("A"))
        {
                filename=designQuestions();
        }
        else if (response2.equals("B"))
        {
                filename=inputString("Please enter a file name, remember to␣
 ↪include '.csv'.");
        }
        else if (response2.equals("C"))
        {

        }
        else
        {
                System.exit(0);
        }

}

//*************************************************************************

/** Declaration of the variable 'question1' of the 'Data' type. Values are␣
 ↪passed to
function as an argument in the function call.
**/
```

```
        Random randomQuestion = new Random(); // Variable of the Random type is␣
↪created.
        int score=0;
        int totalScore=0;
        final int numberOfQuestions=2;

        Player players=createPlayers();

        for (int k=1; k<=getNumberOfPlayers(players); k++)
        {

                for (int i=1; i<=numberOfQuestions; i++) // Counter controlled␣
↪for loop is used to ask multiple questions.
                {
                        int questionNumber = randomQuestion.nextInt(5)+0; //␣
↪The variable 'questionNumber' is assigned a randomly generated number to␣
↪randomise the order of questions.
                        Data question=initialisation(questionNumber,filename);
                        print("\n" + getPlayerNames(players,k)+ ", here is␣
↪question number " + i + ":");
                        print(getQuestion(question));

                        print("\n"+"Your options are below:" + "\n"); // The 4␣
↪potential answers are printed.

                                for (int j=1; j<=4; j++)
                                {
                                        print(getPotentialAns(question,j));
↪
                                }

                        print("\n");
                        String answer=inputAnswer();
                        score = marking(question, answer); // This variable␣
↪contains the score awarded to the player's answer.
                        print("\n"+"You received " + score + " points."); //␣
↪The user's score is printed.
                        print("");
                        totalScore=totalScore + score; // This variable␣
↪contains the player's total score.


                }

                setPlayerScores(players,k,totalScore); // The player's total␣
↪score is saved in the player record.
```

```
                totalScore=0;
        }

        printPlayerScores(players);
/** An exit command has been used below to quit the program after the␣
 ↪instructions within
the main method have been completed.
**/
        System.exit(0);
}



//
 ↪*************************************************************************************

//Primitive Operations-'Data' Abstract Data Type:

/**The 'Data' abstract data type includes operations such as:
 Returning records of the 'Data' type.
 Creating records of the 'Data' type.
 Accessing the question field of a question record.
 Accessing the potentialAnswers field of a question record.
 Accessing the answerScores field of a question record.
**/

/** This function accesses the csv file from the given filename and is used to␣
 ↪create an
array of question records. Once the records have been made, the function␣
 ↪returns a record
based on the randomly generated number passed to it.
**/

public static Data initialisation(int questionNumber,String filename) throws␣
 ↪IOException
{
        Data [] qBank= new Data [6]; // An array of question records.

        BufferedReader inputStream = new BufferedReader (new␣
 ↪FileReader(filename));
        for (int i=0; i<qBank.length; i++)
        {

                String question= inputStream.readLine();

                String s= inputStream.readLine();
                String [] answers=s.split(",");

                s= inputStream.readLine();

                String [] scoresInString=s.split(",");

                int [] answerScores= new int [4];

                for(int j=0; j<4; j++)
                {
                        answerScores[j]=Integer.parseInt(scoresInString[j]);
                }

                qBank[i]=createQuestion(question,answers,answerScores);
        }

        inputStream.close();
        return qBank[questionNumber];
}

// This function is used to create a record for a question. It is passed array␣
 ↪arguments.
public static Data createQuestion(String question, String possibleAnswers [],␣
 ↪int answerPoints []) // Arguments passed to the method in the function call␣
 ↪are used to assign values in the appropriate array.
{
        Data x = new Data();

        x.questions = question;
        x.potentialAnswers = possibleAnswers;
        x.answerScores = answerPoints;

        return x;
}

//
 ↪**************************************************************************************

// This function is used to 'mark' the user's response.
public static int marking(Data question, String answer)
{
        int score=0;

/** If the user's response matches a answer in the record they are given the␣
 ↪score
associated with that answer.
**/
```

```
        if (answer.equals(getPotentialAns(question,1))) /** Boolean expression␣
↪used to verify
                                                                                    the␣
↪user's answer.**/
                {
                        score=getAnswerScores(question,1);
                }

        else if (answer.equals(getPotentialAns(question,2)))
                {
                        score=getAnswerScores(question,2);
                }

        else if (answer.equals(getPotentialAns(question,3)))
                {
                        score=getAnswerScores(question,3);
                }

        else if (answer.equals(getPotentialAns(question,4)))
                {
                        score=getAnswerScores(question,4);
                }

        else
                {
                        score=100; // If the given answer is incorrect the␣
↪user's score is set to 100.
                        print("Unfortunately your response was not a correct␣
↪answer.");

                        return score; // The score is returned to the function␣
↪call.
                }

        if (score==0) // If a users enters a pointless answer they are awarded␣
↪0 points.
        {
                print("Congratulations, you entered a Pointless answer!");
        }

        return score; // The score is returned to the function call.
}

//
↪************************************************************************************
```

```
// Primitive Operations-'Player' Abstract Data Type:

/** Below are the set of primitive operations which will be performed on the␣
 ↪Player data
type. The operations include:

Firstly, creating a player which involves: Storing the number of players and␣
 ↪Setting the name of players .

 Getting the number of players.
 Getting the name of players.

 Setting player scores.
 Getting player scores.

 Ordering the array containing the player scores from lowest to highest.

These accessor methods are the only way for the rest of the program to access␣
 ↪values
in the array fields. **/

// This function is used to create a record for player information.
public static Player createPlayers()
{
        Player players=new Player(); // A new variable called 'player' is␣
 ↪declared of the Player type.

        players.numberOfPlayers=inputInt("Please enter the number of players.");

        if (players.numberOfPlayers <1) // Input validation
        {
                print("\n" + "It seems like you do not want to play Pointless!
 ↪");
                print("Goodbye!");
                System.exit(0);
        }

        players.playerNames= new String [players.numberOfPlayers];
        players.playerScores= new int [players.numberOfPlayers];

/** The counter controlled loop below is used to allow players to enter their␣
 ↪names, the
loop is repeated until all players have done so. **/
        for (int i=0; i<=players.numberOfPlayers-1; i++)
        {
                players.playerNames[i]=inputString("Enter player " + (i+1) +␣
 ↪"'s name.");
        }

        return players;
}

// This function is an accessor method to access the total number of players.
public static int getNumberOfPlayers(Player players)
{
        return players.numberOfPlayers;
}

// This function is an accessor method to access a player's name.
public static String getPlayerNames(Player players, int playerNumber)
{
        playerNumber=playerNumber-1;
        return players.playerNames[playerNumber];
}

// This function is an accessor method to access a player's total score.
public static int getPlayerScores(Player players, int playerNumber)
{
        playerNumber=playerNumber-1;
        return players.playerScores[playerNumber];
}

// This function stores a player's total score into the array in the record␣
 ↪player.
public static Player setPlayerScores(Player players, int playerNumber, int␣
 ↪score)
{
        playerNumber=playerNumber-1;
        players.playerScores[playerNumber]=score;

        return players;
}

/** This function uses 'bubble sort' to order player scores so that they can be␣
 ↪printed
in a leaderboard, the scores are arranged from low to high within the␣
 ↪playerScores array.
**/
public static Player arraySort(Player players)
{
        for (int pass=1; pass <=getNumberOfPlayers(players)-1; pass++)
        {
                for (int i=0; i< getNumberOfPlayers(players)-pass; i++)
                {
                        if (players.playerScores[i] > players.playerScores[i+1])
                        {
                                        int tmp1= players.playerScores[i+1];
                                        players.playerScores[i+1]=players.
 ↪playerScores[i];
                                        players.playerScores[i]= tmp1;

                                        String tmp2=players.playerNames[i+1];
                                        players.playerNames[i+1]=players.
 ↪playerNames[i];
                                        players.playerNames[i]=tmp2;
                        }
                }
        }

        return players;
}

//
 ↪*************************************************************************************

// Primitive Operations-'Data' Abstract Data Type:

/**Below are the set of primitive operations which will be performed on the␣
 ↪'Data' data type.
The operations include, getting a question, potential answers and answer scores.
 ↪ **/

// This function is an accessor method to access a question stored in the␣
 ↪question record.
public static String getQuestion(Data question)
{
        return question.questions;
}

// This function is an accessor method to access appropriate answers to a␣
 ↪particular question.
public static String getPotentialAns(Data question, int ansNumber)
{
        ansNumber=ansNumber-1;

        return question.potentialAnswers[ansNumber];
}
```

```
// This function is an accessor method to access answer scores to a particular␣
 ↪question.
public static int getAnswerScores(Data question, int scoreNumber)
{
        scoreNumber=scoreNumber-1;
        return question.answerScores[scoreNumber];
}

//
 ↪*************************************************************************************

// This procedure is used to print out the scores for each player at the end.
public static void printPlayerScores(Player players)
{
        print("\n" + "Pointless Leaderboard: \n");
        arraySort(players);

        for (int j=1; j<=getNumberOfPlayers(players); j++)
        {
                print(getPlayerNames(players,j)+ " " +␣
 ↪getPlayerScores(players,j));
        }
}

/** This function enables the user to enter their own questions and answers␣
 ↪which are
saved in a csv file. The filename is then returned so subsequent methods can
access the file and print accordingly.
**/
public static String designQuestions() throws IOException
{

        String filename=inputString("What topic would you like the questions to␣
 ↪be on?")+ ".csv";
        PrintWriter outputStream= new PrintWriter(new FileWriter(filename));

        for(int i=1; i<=6; i++)
        {
                outputStream.println(inputString("\nPlease enter a question you␣
 ↪would like to ask."));

                outputStream.println(inputString("Enter the suitable answers.␣
 ↪In this format: Blue,Red,Green,Yellow"));
                outputStream.println(inputString("Please assign the answers a␣
 ↪score. In this format: 34,76,23,41"));
        }
```

```java
        outputStream.close();

        return filename;
}


// This function is used to enable a user to enter their answer.
public static String inputAnswer()
{
        String text_input;
        Scanner scanner= new Scanner(System.in);

        text_input=scanner.nextLine();

        return text_input;
}

// This function enables the user to enter String values.
public static String inputString(String message)
{
        String text_input;
        Scanner scanner= new Scanner(System.in);

        System.out.println(message);
        text_input=scanner.nextLine();

        return text_input;
}

// This function enables the user to enter integer values.
public static int inputInt(String message)
{
        System.out.println(message);
        String text_input=inputAnswer();
        int x=Integer.parseInt(text_input);

        return x;
}

// This procedure uses abstraction, to simplify the use of 'System.out.println'.
public static void print(String message)
{
        System.out.println(message);
        return;

}
```

```
} // End of the 'Pointless' class.
```

END OF LITERATE DOCUMENT