```java
//AUTHOR:: OSMAN SHAMS, DATE WRITTEN:: 05th DEC 19

//MiniProject : IMPOSSIBLE QUIZ QUESTION

//IMPORTING ALL THE LIBRARIES FORM JAVA UTILITY PACKAGES
import java.util.*;

//IMPORTING I/O LIBRARY SO THAT FILE INPUT AND OUTPUT CAN BE USED IN THE PROGRAM
import java.io.*;


class MiniProject{

 public static void main (String [] args) throws IOException{

 //PASSING DOWN ARGUMENTS TO METHOD createQuiz

 Quiz q1 = createQuiz("Q: Which famous scientist is attributed to being the father of Modern Medicine? (The answer is case sensitive)","Ibn Sina","Alexander Fleming","Charles Darwin");

 Quiz q2 = createQuiz("Q: Which colour is a Banana usually found in? (The answer is case sensitive)","Yellow","Green","Orange");

 Quiz q3 = createQuiz("Q: Which of the following gas is a Noble Gas? (The answer is case sensitive)","Radon", "Oxygen", "Mercury");

 //CREATING AN ARRAY arrayOfQues OF TYPE QUIZ AND ASSIGNING q1, q2, q3 AS ITS ELEMENTS
 Quiz [] arrayOfQues = {q1, q2, q3};

 //ASSIGNING ARRAY arrayOfQues ON OBJECT qb OF RECORD QuestionBank VIA createQuestionBank METHOD
 QuestionBank qb = createQuestionBank(arrayOfQues);

 int totalScore = 0;

 int scoreList [] = new int[3];

//ASKING THE USER IF THE PREVIOUS SCORES HAVE TO BE LOADED
 String response = AskPreviousScore();

 if(response.equalsIgnoreCase("YES")){

   totalScore = LoadScore();

 }

 else{

   totalScore = 0;

 }

 //USING A FOR LOOP IN ORDER TO PRINT ALL 3 QUESTIONS
 for(int i = 0; i < qb.bankQuestion.length; i++){

   //OUTPUTTING QUESTION STORED IN INDEX i OF QUESTIONBANK VIA OutputQuestion METHOD
   OutputQuestion(qb, i);

   //TAKING IN USER ANSWER VIA TakeAnswer METHOD AND ASSIGNING IT TO VARIABLE userAnswer
   String userAnswer = TakeAnswer();

   boolean rightCheck = CheckRightAnswer(userAnswer, qb, i);

   boolean impossibleCheck = CheckImpossibleAnswer(userAnswer, qb, i);

   //CHECKING IF THE ANSWER USER ENTERED IS THE WRONG ANSWER VIA CheckWrongAnswer
```

```
      METHOD AND ASSIGNING IT TO boolean wrongCheck
63    boolean wrongCheck = CheckWrongAnswer(userAnswer, qb, i);
64
65    //THE WHILE LOOP CHECKS IF THE USER ENTERED AN INVALID ANSWER AND IF SO THEN IT
      EPRINTS THE QUESTION UNLESS A VALID INPUT IS TYPED IN
66    while(!rightCheck & !impossibleCheck & !wrongCheck){
67
68      OutputQuestion(qb, i);
69
70      userAnswer = TakeAnswer();
71
72      rightCheck = CheckRightAnswer(userAnswer, qb, i);
73
74      impossibleCheck = CheckImpossibleAnswer(userAnswer, qb, i);
75
76      wrongCheck = CheckWrongAnswer(userAnswer, qb, i);
77
78    }//END WHILE
79
80    scoreList[i] = CalculateScore(rightCheck);
81
82    //AFTER INPUTTING A VALID ANSWER THE SCORE FOR THE USER IS CALCULATED
83    totalScore = totalScore + scoreList[i];
84
85    UploadScore(totalScore);  //UPDATING THE SCORE
86
87  }//END FOR
88
89  System.out.println();
90
91  //AFTER ANSWERING ALL 3 QUESTIONS THE TOTAL SCORE OF THE PLAYER WILL BE DISPLAYED
92  System.out.println("Your Total Score Is "+totalScore);
93
94  System.out.println();
95
96  //ASK USER TO END THE PROGRAM OR PRINT A TABLE
97  String userInput = AskOption();
98
99  System.out.println();
100
101 //IF THE USER TYPES IN AN ANSWER OTHER THAN YES OR NO THEN OPTION WILL BE ASKED AGAIN
102 while(!(userInput.equals("Yes")) || !(userInput.equals("No"))) {
103
104   if (userInput.equalsIgnoreCase("Yes")){
105
106     break;
107   }
108
109   else if (userInput.equalsIgnoreCase("No")){
110
111     PrintTable(scoreList);
112
113     break;
114   }
115
116   userInput = AskOption();
117
118 }//END WHILE
119
120 System.exit(0);
121
122 }//END MAIN
123
124 //LOADING THE PREVIOUS SCORE
125  public static int LoadScore() throws IOException{
126
127      BufferedReader inputStream = new BufferedReader(new FileReader("Scores.txt"));
128
129      String scores = inputStream.readLine();
```

```
130
131         return Integer.parseInt(scores);
132
133     }
134
135     public static String AskPreviousScore(){
136
137         Scanner s = new Scanner(System.in);
138
139         System.out.print("Would You Like To Load Your Previous Score ? : ");
140
141         return s.nextLine();
142     }
143
144     //UPLOADING THE SCORE AFTER THE QUESTIONS HAVE BEEN ASKED
145     public static void UploadScore(int totalScore) throws IOException{
146
147         PrintWriter objectStream = new PrintWriter(new FileWriter("Scores.txt"));
148
149         objectStream.println(totalScore);
150
151         objectStream.close();
152     }
153
154    //SAVING INDIVITUAL QUESTIONSCORES IN A SEPERATE FILE
155     public static void saveScore(int[] scoreList, String filename) {
156
157
158      try {
159          PrintWriter output = new PrintWriter(new FileWriter(filename + ".txt"));
160
161        for (int i = 0; i < scoreList.length; i++) {
162
163          output.println(scoreList[i] + "\n");
164
165        }//END FOR
166
167        output.close();
168
169      }//END TRY
170
171      catch(IOException ex) {
172
173        System.out.println("File not found or file management system is corrupt.");
174
175      }//END CATCH
176
177      return;
178
179     }//END saveScore
180
181     //CREATING AN OBJECT OF QUESTIONBANK AND ASSIGNING ITS FIELD bankQuestion
        ,arrayOfQues
182     public static QuestionBank createQuestionBank(Quiz [] arrayOfQues){
183
184      QuestionBank qs = new QuestionBank();
185
186      qs.bankQuestion = arrayOfQues;
187
188      return qs;
189
190     }//END createQuestionBank
191
192     //TAKING IN ARGUEMENTS FROM THE MAIN METHOD TO SET UP THREE QUESTIONS
193     public static Quiz createQuiz(String aQuestion, String rightAnswer, String
        wrongAnswer, String imposAnswer){
194
195      Quiz q = new Quiz();
196
```

```
197     q.question = aQuestion;
198
199     q.correctAns = rightAnswer;
200
201     q.wrongAns = wrongAnswer;
202
203     q.impossibleAns = imposAnswer;
204
205     return q;
206
207    }//END createQuiz
208
209    //THE ENTIRE QUESTION STORED IN THE INDEX i OF THE ARRAY bankQuestion WILL BE
       DISPLAYED
210    public static void OutputQuestion(QuestionBank qb, int i){
211
212     System.out.println();
213
214     //IN ORDER TO PRINT THE QUESTION ACCESSOR METHODS WERE USED TO GET THE SPECIFIC
        FIELDS OF THE RECORD Quiz
215     System.out.println(getQuestion(qb, i));
216
217     System.out.println(getCorrectAnswer(qb, i));
218
219     System.out.println(getWrongAnswer(qb, i));
220
221     System.out.println(getImpossibleAnswer(qb, i));
222
223     return;
224
225    }//END OutputQuestion
226
227    public static String getQuestion(QuestionBank qb, int i){
228
229     //USING DOT NOTATION TO ACCESS AND RETURN THE PARTICULAR FIELD OF THE RECORD Quiz
230     return qb.bankQuestion[i].question;
231
232    }//END getQuestion
233
234    public static String getCorrectAnswer(QuestionBank qb, int i){
235
236     return qb.bankQuestion[i].correctAns;
237
238    }//END getCorrectAnswer
239
240    public static String getWrongAnswer(QuestionBank qb, int i){
241
242     return qb.bankQuestion[i].wrongAns;
243
244    }//END getWrongAnswer
245
246    public static String getImpossibleAnswer(QuestionBank qb, int i){
247
248     return qb.bankQuestion[i].impossibleAns;
249
250    }//END getImpossibleAnswer
251
252    //AS THE QUESTION WILL ALREADY BE DISPLAYED BEFORE IT THIS METHOD JUST TAKES IN THE
       INPUT
253    public static String TakeAnswer(){
254
255     Scanner s = new Scanner(System.in);
256
257     return s.nextLine();
258
259    }//END TakeAnswer
260
261    //THIS METHOD CHECKS IF THE USER ANSWER ENTERED IS THE RIGHT ANSWER AND RETURNS
       BOOLEAN VALUE TRUE IF THAT IS THE CASE
```

```
262    public static boolean CheckRightAnswer(String userAnswer, QuestionBank qb, int i){
263
264     boolean rAnswer;
265
266     if(userAnswer .equals(getCorrectAnswer(qb, i))){
267
268       rAnswer = true;
269
270       System.out.println("Your Answer Is Correct");
271
272     }//END IF
273
274     else{
275
276       rAnswer = false;
277
278     }//END ELSE
279
280     return rAnswer;
281
282    }//END CheckRightAnswer
283
284    //THIS METHOD CHECKS IF THE USER ANSWER ENTERED IS THE IMPOSSIBLE ANSWER AND
       RETURNS BOOLEAN VALUE TRUE IF THAT IS THE CASE
285    public static boolean CheckImpossibleAnswer(String userAnswer, QuestionBank qb, int
       i){
286
287     boolean iAnswer;
288
289     if(userAnswer .equals(getImpossibleAnswer(qb, i))){
290
291       iAnswer = true;
292
293       System.out.println("Your Answer Is Impossible");
294
295     }//END IF
296
297     else{
298
299       iAnswer = false;
300
301     }//END ELSE
302
303     return iAnswer;
304
305     }//END CheckImpossibleAnswer
306
307    //THIS METHOD CHECKS IF THE USER ANSWER ENTERED IS THE WRONG ANSWER AND RETURNS
       BOOLEAN VALUE TRUE IF THAT IS THE CASE
308    public static boolean CheckWrongAnswer(String userAnswer, QuestionBank qb, int i){
309
310     boolean wAnswer;
311
312     if(userAnswer .equals(getWrongAnswer(qb, i))){
313
314       wAnswer = true;
315
316       System.out.println("Your Answer Is Wrong");
317
318     }//END IF
319
320     else{
321
322       wAnswer = false;
323
324     }//END ELSE
325
326     return wAnswer;
327
```

```
328     }//END CheckWrongAnswer
329
330    //THIS METHOD CALCULATES THE SCORE OF THE USER DEPENDING UPON THE ANSWER
331    public static int CalculateScore(boolean rightCheck){
332
333     int score = 0;
334
335     //IF THE USER ENTERS A CORRECT ANSWER THEN THE A VIRTUAL DICE FROM 1 TO 6 IS
       ROLLED AND A SCORE AT RANDOM IS ASSIGNED TO THE USER
336     if(rightCheck == true){
337
338       Random random = new Random();
339
340       score = random.nextInt(6) + 1;
341
342     }//END IF
343
344     return score;
345     //IF THE USER ENTERS AN IMPOSSIBLE OR A WRONG ANSWER THEN A SCORE 0 IS ASSIGNED TO
       THEM
346
347
348    }//END CalculateScore
349
350    //AN OPTION IS ASKED TO THE USER TO CONTINUE (WHICH WILL PRINT A SCORE TABLE) OR
       STOP THE PROGRAM
351    public static String AskOption(){
352
353     Scanner s = new Scanner(System.in);
354
355     System.out.print("Would You like The Program To End (Yes or No)? : ");
356
357     return s.nextLine();
358
359    }//END AskOption
360
361    //A TABLE OF THE SCORES THAT THE USER SCORED THROUGHOUT THE PROGRAM WILL BE PRINTED
       AFTER SORTING THE SCORES
362    public static void PrintTable(int [] scoreList){
363
364     for(int i = 0; i < scoreList.length - 1; i++){
365
366       for(int j = 0; j < scoreList.length - i - 1; j++){
367
368         if(scoreList[j] > scoreList[j+1]){
369
370           //swap(j,j+1, scoreList);
371           int temp = scoreList[j];
372
373           scoreList[j] = scoreList[j+1];
374
375           scoreList[j+1] = temp;
376
377         }//END IF
378
379       }//END FOR
380
381     }//END FOR
382
383     System.out.println("The Total Scores that you recieved were");
384
385     System.out.println();
386
387     System.out.print(" | ");
388
389     for(int b = 0; b < scoreList.length; b++){
390
391       System.out.print(scoreList[b]+" | ");
392
```

```
393     }//END FOR
394
395     System.out.println();
396
397     return;
398
399    }//END PrintTable
400
401    //THIS METHOD WILL SWAP THE VALUES IN THE ARRAY IF THE PRECEDING VALUE IN THE ARRAY
       IS LARGER THAN THE LATTER
402    public static void swap(int j, int jInc, int [] scoreList){
403
404     int temp = scoreList[jInc];
405
406     scoreList[jInc] = scoreList[j];
407
408     scoreList[j] = temp;
409
410     return;
411
412    }//END swap
413
414   }//END MiniProject
415
416
417   class QuestionBank{ //CREATING AN ABSTRACT DATA TYPE BUILT ON TOP OF RECORD Quiz
418
419     Quiz [] bankQuestion;
420
421   }//END QuestionBank
422
423
424   class Quiz{  //CREATING A RECORD Quiz WITH THE SPECIFIED FIELDS
425
426     String question;
427
428     String correctAns;
429
430     String wrongAns;
431
432     String impossibleAns;
433
434   }//END Quiz
```