7MiniProject - tested

December 14, 2021

1 Mini Project

2 Level 7

2.1 Alireza Rahimi

2.2 11 December 2021

2.3 Version 7

2.4 Summary of the Question

Write a chat bot procedural program that can have realistic conversations with people about a specific topic (eg films, football, musicals, pop music, …).

2.5 The literate program development

2.5.1 inputString()

What it does This method asks the user a given question and returns their answer as a string.

Implementation (how it works) This method uses a string as input and prints that string. then it uses the keyboard as system input and returns the value entered by user as a string.

[ ]:
```
// Shows the entered string (argument) as a message and gets user response from␣
 ↪keyboard and returns it as a string.
public static String inputString(String message)
{
       Scanner scanner = new Scanner(System.in);

       System.out.println(message);
       String response = scanner.nextLine();

       return response;
}
```

Testing

[ ]:
```
String response = inputString("Any Message");
```

2.6 randomInt()

What it does This method gets a bound number as argument and returns a random number between 0 (inclusive) and bound (exclusive)

Implementation (how it works) This method uses a built in type and a built in method to return a randomly generated int from 0 (inclusive) and argument which is bound (exclusive)

[ ]:
```
// Return a random int between 0 (inclusive) and bound (exclusive)
    public static int randomInt(int bound) {
    Random r = new Random(); // built-in type
    return r.nextInt(bound); // built-in method
    }
```

Testing

[ ]:
```
int rand = randomInt(5);
System.out.println(rand);
```

Testing

2.7 introduce()

What it does This method introduces the program as a personal assistant and also introduces its developer.

Implementation (how it works) This method uses screen as output and prints several strings on the screen. It also calls another method to randomly choose a goodbye phrase.

[ ]:
```
// after random hello phrase printed, introduces itself and its developer.
public static void introduce()
{
    final String[] hello = {"Hello!","Greetings!","Hi!","Hola!"};
    final int size = hello.length;
    String message = hello[randomInt(size)];
    System.out.println(message);

    message = "My name is Alleria. I am a personal Assistant developed by␣
 ↪Alireza.";
    System.out.println(message);

    return;
}
```

Testing

[ ]:
```
introduce();
```

2.8 askName()

What it does This method asks the user their name and stores gets the user response as an input and displays a message based on that input.

Implementation (how it works) This method prints some strings and calls another method and uses keyboard as input to get the user response. Then it prints another string based on user response.

[ ]:
```
// asks user for their name and includes it in a nice to meet you phrase and␣
 ↪prints it.
public static void askName()
{
    final String userName = inputString("Please enter your name so I have␣
 ↪something to remember you with.");
    String message = "Glad to know you " + userName + "." + " You have such a␣
 ↪nice name!";
    System.out.println(message);

    return;
}
```

Testing

[ ]:
```
askName();
```

2.9 getUserInterest()

What it does This method asks the user a question and returns their answer.

Implementation (how it works) This method calls another method and uses a string as its argument and get that method's output, stores it, and returns it.

[ ]:
```
// ask user their opinion about games and return the response
public static String getUserInterest()
{
    String question = "What artificially intelligent game do you find most␣
 ↪thrilling?";

    final String userResponse = inputString(question);

    return userResponse;
}
```

Testing

[ ]:
```
String response = getUserInterest();
```

2.10 tellAskInterest()

What it does This method tells its interest to user and asks a specific question about it from user and says a sentence based on the user response.

Implementation (how it works) This method prints a particular string on screen and then calls another method (getUserInterest()) which returns a string. then it stores and prints that in a sentence on screen.

[ ]:
```
// print cahtbot's interest and call another method to know user interest. Then␣
 ↪use the result to generate and print a sentence.
public static void tellAskInterest()
{
    final String interest = "Computer Games";
    final String myInterest = "I am designed to be interested in " + interest +␣
 ↪". Specially in the ones with complicated AIs!";

    System.out.println(myInterest);

    final String userInterest = getUserInterest();

    System.out.println(userInterest + "? I love it! I think it's one of the␣
 ↪bests.");

    return;
}
```

Testing

[ ]:
```
tellAskInterest();
```

2.11 commentOnUserStatus()

What it does This method gets a string as input and based on the argument, decides to do certain tasks.

Implementation (how it works) This method gets a string as argument and uses it to check the if statement condition. If the condition is satisfied It calls another method and uses certain string as argument and then prints another string on the screen. If the condition does not satisfy, is prints a particular string on the screen.

[ ]:
```
// based on input argument, do something.
//if argument is 'Y' ask user something and print another string. else print a␣
 ↪particular string.
public static void commentOnUserStatus(String YorN)
{
    final String yesMessage = "Oh nice! How long have you played it?";
    final String goodWish = "Awesome! Good luck with it.";
```

```
    final String noMessage = "Oh you have no idea what you have missed! I␣
 ↪strongly recommend you to play it.";

    if (YorN.equalsIgnoreCase("Y"))
    {
        final String howLong = inputString(yesMessage);
        System.out.println(goodWish);
    }
    else
    {
        System.out.println(noMessage);
    }

    return;
}
```

Testing

[ ]:
```
commentOnUserStatus("Y");
System.out.println(".");
System.out.println(".");
System.out.println(".");
commentOnUserStatus("N");
```

2.12 userStatus()

What it does This method asks user whether they play the named game or not. Then calls another method using the user response. Possible human errors are handled.

Implementation (how it works) This method prints some strings and calls another method which prints another string and returns user response. Then it uses that response to call another method. A while loop and an id

[ ]:
```
// this method asks about user playing status and uses user response as the␣
 ↪argument to call another method.
// Possible error handling applied using a while loop.
public static void userStatus()
{
    final String question = "Do you personally play this game? (Y/N)";
    final String possibleError = "I could not recognize your answer. Please␣
 ↪answer again using Y or N:";

    boolean looping = true;

    while(looping)
    {
        String playedOrNot = inputString(question);
```

```
        if ( playedOrNot.equalsIgnoreCase("Y") || playedOrNot.
↪equalsIgnoreCase("N") )
        {
            commentOnUserStatus(playedOrNot);

            looping = false;
        }
        else
        {
            System.out.println(possibleError);
        }
    }

    return;
}
```

Testing

[ ]:
```
userStatus();
```

2.13 goodBye()

What it does This method describes the reason it has to go and prints a randomly chosen goodbye phrase.

Implementation (how it works) This method uses screen as output and prints a couple of Strings on the screen. It also calls another method to randomly choose a goodbye phrase.

[ ]:
```
// prints the reason it has to go and prints a random goodbye phrase.
public static void goodBye()
{
    String message = "I just detected overheating on one of my CPUs!␣
↪Unfortunately, I have to shutdown my systems to avoid possible damage. Sorry␣
↪for any inconvenience that may cause.";
    System.out.println(message);

    final String[] goodbye = {"See you later!","Have a nice day!","Bye!","Bye␣
↪bye!","Catch you later!"};
    final int size = goodbye.length;
    message = goodbye[randomInt(size)];
    System.out.println(message);

    return;
}
```

Testing

[ ]:
```
goodBye();
```

2.13.1 MyDatabase

What it does This method creates a new record (user-defined Type) called MyDatabase.

Implementation (how it works) It creates a new record called MyDatabase and defined some values of user-defined type MyDatabase which are called fields. The fields store information about the triggers (string arrays), responses (string arrays), and popularity (int) which will be used to determine the number of times each varibale of this user-defined type is picked by user.

[ ]:
```
// create new record (user-defined type) called MyDatabase and define its␣
↪fields.
class MyDatabase
{
    String[] trigs;
    String[] responses;
    int popularity;
}
```

Testing

[ ]:
```
MyDatabase notInUse = new MyDatabase();
```

2.14 setTrigs()

What it does This is an accessor method for MyDatabase record which assigns the string array input to the field 'trigs' of record variable (with user-defined type: MyDatabase). And returns the same record variable.

Implementation (how it works) This method gets a user-defined type variable and a string array as argument and assigns the string input to one of the fields of given record variable called 'MyDatabase'. And then returns the user-defined type variable.

[ ]:
```
// gets record variable name and triggers array and assigns triggers array to␣
↪the record field called trigs. Then return the record variable.
public static MyDatabase setTrigs(MyDatabase MyData, String[] Trigs)
{
    MyData.trigs = Trigs;

    return MyData;
}
```

Testing

[ ]:
```
String[] sample = {"sample","strings"};
notInUse = setTrigs(notInUse, sample);
```

2.15 setResponses()

What it does This is an accessor method for MyDatabase record which assigns the String array input to the field 'responses' of record variable (with user-defined type: MyDatabase). And returns the same record variable.

Implementation (how it works) This method gets a user-defined type variable and a string array as argument and assigns the integer input to one of the fields of given record variable called 'responses'. And then returns the user-defined type variable.

[ ]:
```
// gets record variable name and responses array and assigns triggers array to␣
↪the record field called responses. Then return the record variable.
public static MyDatabase setResponses(MyDatabase MyData, String[] Responses)
{
    MyData.responses = Responses;

    return MyData;
}
```

Testing

[ ]:
```
String[] sample = {"another sample " , "String"};
notInUse = setResponses(notInUse, sample);
```

2.16 setPopularity()

What it does This is an accessor method for MyDatabase record which assigns the integer input to the field 'popularity' of record variable (with user-defined type: MyDatabase). And returns the same record variable.

Implementation (how it works) This method gets a user-defined type variable and a string as argument and assigns the string input to one of the fields of given record variable called 'MyDatabase'. And then returns the user-defined type variable.

[ ]:
```
// gets record variable name and an integer and assigns the integer to the␣
↪record field called popularity. Then return the record variable.
public static MyDatabase setPopularity(MyDatabase MyData,int amount)
{
    MyData.popularity = amount;

    return MyData;
}
```

Testing

[ ]:
```
notInUse = setPopularity(notInUse, 5);
```

2.17 createTrigNresponse()

What it does This method gets a set of trigger words and a set of response words and creates a new record variable. Then, by calling accessor methods, it assigns received values to the fields of created record. Then it returns the created record variable.

Implementation (how it works) This method gets 2 string arrays as arguments. Then it creates a record variable and uses the input string arrays as arguments of accessor methods when calling them. Using accessor methods it assigns those values to the record field. And then it returns the created record variable.

[ ]:
```
// create and initiate a new record variable and return it taking 2 string␣
 ↪array arguments for trigs and responses.
// for record type MyDatabase
public static MyDatabase createTrigNresponse(String[] Trigs, String[] Responses)
{
    MyDatabase trigNresponse = new MyDatabase();

    trigNresponse = setTrigs(trigNresponse, Trigs);
    trigNresponse = setResponses(trigNresponse, Responses);
    trigNresponse = setPopularity(trigNresponse, 0);

    return trigNresponse;
}
```

Testing

[ ]:
```
String[] sample = {"0" , "1"};
String[] sample1 = {"o" , "i"};

notInUse = createTrigNresponse(sample, sample1);
```

2.18 CreateGenreTrigs()

What it does This method defines, about Gaming Genres, some triggers and responses arrays and passes them through another method which gets 2 arrays as arguments and creates and returns a new record. Then returns the new record.

Implementation (how it works) This method creates some triggers and responses strings individually and places them in corresponding indexes in 2 string arrays. Then it passes them through another method as argument which creates and returns the new record with given information. Then returns the new record.

[ ]:
```
public static MyDatabase CreateGenreTrigs()
{

    final String trig0 = "Shooter";
    final String trig1 = "MOBA";
```

```
    final String trig2 = "Survival";
    final String trig3 = "Strategy";

    final String response0 = "Shooter or Shooting! They come in firs person and␣
↪third person. You have oppurtunity to fight with enemies with a variety of␣
↪guns!";
    final String response1 = "MOBA stands for Multiplayer Online Battle Arena.␣
↪The name determines everything!";
    final String response2 = "You need to roam in the world and look for goods,␣
↪then survive yourself using them!";
    final String response3 = "A strategy game or strategic game is a game in␣
↪which the players' uncoerced, and often autonomous, decision-making skills␣
↪have a high significance in determining the outcome. Almost all strategy␣
↪games require internal decision tree-style thinking, and typically very high␣
↪situational awareness.";

    final String[] MyTrigs = {trig0,trig1,trig2,trig3};
    final String[] MyResponses = {response0,response1,response2,response3};

    MyDatabase trigNresponse = createTrigNresponse(MyTrigs,MyResponses);

    return trigNresponse;

}
```

Testing

[ ]:
```
notInUse = CreateGenreTrigs();
```

2.19 CreatePlatformTrigs()

What it does This method defines, about Gaming Platforms, some triggers and responses arrays and passes them through another method which gets 2 arrays as arguments and creates and returns a new record. Then returns the new record.

Implementation (how it works) This method creates some triggers and responses strings individually and places them in corresponding indexes in 2 string arrays. Then it passes them through another method as argument which creates and returns the new record with given information. Then returns the new record.

[ ]:
```
public static MyDatabase CreatePlatformTrigs()
{

    final String trig0 = "PC";
    final String trig1 = "Console";
    final String trig2 = "Mobile";
```

```
    final String response0 = "Best gaming option, because you can build your␣
↪own PC, you have more control over what games you play and how the system␣
↪operates than you would with a console.";
    final String response1 = "Consoles have advantages over PCs: They are easy␣
↪to use, don't require upgrades, make for simple multiplayer with␣
↪console-owning friends, are generally cheaper, and use wireless controllers␣
↪that allow you to have a more active experience.";
    final String response2 = "Some multiplayer mobile games help children to␣
↪develop qualities such as collaboration and cooperation. It can improve␣
↪their social connections, develop bonds, trust and teach them to play with␣
↪morals that can stay with them forever.";


    final String[] MyTrigs = {trig0,trig1,trig2};
    final String[] MyResponses = {response0,response1,response2};

    MyDatabase trigNresponse = createTrigNresponse(MyTrigs,MyResponses);

    return trigNresponse;

}
```

Testing

[ ]:
```
notInUse = CreatePlatformTrigs();
```

2.20 CreateCompanyTrigs()

What it does This method defines, about Gaming Companies, some triggers and responses arrays and passes them through another method which gets 2 arrays as arguments and creates and returns a new record. Then returns the new record.

Implementation (how it works) This method creates some triggers and responses strings individually and places them in corresponding indexes in 2 string arrays. Then it passes them through another method as argument which creates and returns the new record with given information. Then returns the new record.

[ ]:
```
public static MyDatabase CreateCompanyTrigs()
{

    final String trig0 = "Ubisoft";
    final String trig1 = "Valve";
    final String trig2 = "Rockstar";
    final String trig3 = "Activision";
```

```
    final String response0 = "Ubisoft is a nice one, Everything they make has␣
↪some satisfying gameplay, a decent story and mechanics that work pretty well.
↪";
    final String response1 = "Valve is my developer's favourite, because they␣
↪deliver high-quality games that have great stories, excellent characters,␣
↪and are typically ranked among the best games ever made.";
    final String response2 = "Rockstar Games is a video game publisher␣
↪established under Take-Two Interactive in 1998. It is best known for the␣
↪Grand Theft Auto series.";
    final String response3 = "Activision is a leading worldwide developer,␣
↪publisher, and distributor of interactive entertainment for various gaming␣
↪consoles, handheld platforms, and PC.";

    final String[] MyTrigs = {trig0,trig1,trig2,trig3};
    final String[] MyResponses = {response0,response1,response2,response3};

    MyDatabase trigNresponse = createTrigNresponse(MyTrigs,MyResponses);

    return trigNresponse;

}
```

Testing

[ ]:
```
notInUse = CreateCompanyTrigs();
```

2.21 CreatecontrollerTrigs()

What it does This method defines, about Gaming Controllers, some triggers and responses arrays and passes them through another method which gets 2 arrays as arguments and creates and returns a new record. Then returns the new record.

Implementation (how it works) This method creates some triggers and responses strings individually and places them in corresponding indexes in 2 string arrays. Then it passes them through another method as argument which creates and returns the new record with given information. Then returns the new record.

[ ]:
```
public static MyDatabase CreatecontrollerTrigs()
{

    final String trig0 = "Keyboard";
    final String trig1 = "Joystick";
    final String trig2 = "Touchscreen";
    final String trig3 = "eyetracking";

    final String response0 = "Wow keyboard! That is my developer's favourite :
↪D";
```

```
    final String response1 = "Joystick is a classic one!";
    final String response2 = "It is difficult to have full control using␣
↪touchscreen in games. Good Job!";
    final String response3 = "Unfortunately eyetracking technology has not␣
↪spread too much. Good news is, it is developing rapidly!";

    final String[] MyTrigs = {trig0,trig1,trig2,trig3};
    final String[] MyResponses = {response0,response1,response2,response3};

    MyDatabase trigNresponse = createTrigNresponse(MyTrigs,MyResponses);

    return trigNresponse;
}
```

Testing

[ ]:
```
notInUse = CreatecontrollerTrigs();
```

2.22 getTrigs()

What it does This method is an accessor method for MyDatabase record which gets a record variable name and returns the value of its 'trigs' field.

Implementation (how it works) This accessor method gets a record variable name (with user-defined type) and returns the value of one of its fields called 'trigs'.

[ ]:
```
// get a record variable name and return the value of its 'trigs' field
public static String[] getTrigs(MyDatabase MyData)
{
    String[] Trigs = MyData.trigs;

    return Trigs;
}
```

Testing

[ ]:
```
String[] sampleStr = getTrigs(notInUse);
```

2.23 getResponses()

What it does This method is an accessor method for MyDatabase record which gets a record variable name and returns the value of its 'responses' field.

Implementation (how it works) This accessor method gets a record variable name (with user-defined type) and returns the value of one of its fields called 'responses'.

[ ]:
```
// get a record variable name and return the value of its 'responses' field
public static String[] getResponses(MyDatabase MyData)
{
    String[] Responses = MyData.responses;

    return Responses;
}
```

Testing

[ ]:
```
String[] sampleStr = getResponses(notInUse);
```

2.24 increase1popularity()

What it does This is an accessor method for MyDatabase record which adds 1 unit to the integer stored in the field 'popularity' of record variable (with user-defined type: MyDatabase). And returns the same record variable.

Implementation (how it works) This method gets a user-defined type variable as argument and adds 1 unit to the integer stored in the field of given record variable called 'popularity'. And then returns the user-defined type variable.

[ ]:
```
// get record variable name, increase 1 to its 'popularity' field, then return␣
↪the record variable.
public static MyDatabase increase1popularity(MyDatabase MyData)
{

    MyData.popularity = MyData.popularity + 1;


    return MyData;
}
```

Testing

[ ]:
```
notInUse = increase1popularity(notInUse);
```

2.25 getPopularity()

What it does This method is an accessor method for MyDatabase record which gets a record variable name and returns the value of its 'popularity' field.

Implementation (how it works) This accessor method gets a record variable name (with user-defined type) and returns the value of one of its fields called 'popularity'.

[ ]:
```
// get a record variable name and return the value of its 'popularity' field
public static int getPopularity(MyDatabase MyData)
{
```

```
    int Popularity = MyData.popularity;

    return Popularity;
}
```

Testing

[ ]:
```
int sampleInt = getPopularity(notInUse);
```

2.26 ChckTrigNRespond()

What it does This method gets a set of Triggers and a set of responses and 1 user response as argument. It checks if the user response exists in trigs arrays. If there is, it prints the corresponding response from responses array. If there user response and triggers does not match, it prints an exception message instead.

Implementation (how it works) This method This method gets 2 string arrays (Trigs and responses) and 1 string value (user response) as argument. It uses a for loop to compare each index of trigs array with userresponse. It uses an if statement to print corresponding message from responses if user response matches any trigger. If there is no triggers matched, it prints a exception handling message instead.

[ ]:
```
// Gets triggers array, responses array, and user response as argument. Check␣
 ↪Triggers and print corresponding responses.
// Exception responses handled.
public static void ChckTrigNRespond(String[] Trigs, String[] Responses, String␣
 ↪userResponse)
{
    int arraySize = Trigs.length;

    for (int index = 0 ; index < arraySize ; index++)
    {
        if ( userResponse.equalsIgnoreCase(Trigs[index]) )
            {
                System.out.println(Responses[index]);
                return;
            }
        else
            {
                ;
            }
    }

    System.out.println("I havo no information about that in my databases.␣
 ↪Please try again picking something from the list.");
    return;
}
```

Testing

```
[ ]: String[] T = {"yes",  "no"};
     String[] R = {"good",  "bad"};
     ChckTrigNRespond(T, R,"not known");
```

2.27 RandomQuestion()

What it does This method gets a record variable and a string as argument. It extracts the trigs and responses fields of the record and stores them. Based on the string input, It flows through one of the cases of switch statement. Then it calls another method and uses the generated trigs and responses arrays ans also the user response to a randomly asked question as its arguments. The other method just prints appropriate answers referring to the argument values.

Implementation (how it works) This method gets a record variable and a string as argument. It extracts the trigs and responses fields of the record and stores them. Based on the string input, It flows through one of the cases of switch statement. Then it calls another method and uses the generated trigs and responses arrays ans also the user response to a randomly asked question as its arguments. The other method just prints appropriate answers referring to the argument values. Exception handling provided in the other method called.

```
[ ]: // Extract trigs and responses from given record variable
     // choose a topic to ask a random question about
     // pass the extracted arrays and user response to the asked question to another␣
      ↪method
     // another method prints appropriate answer based on its arguments
     public static void RandomQuestion(MyDatabase MyData, String userResponse)
     {
         String[] Trigs = getTrigs(MyData);
         String[] Responses = getResponses(MyData);
         String response = "";

         final String[] genreQuestions = {"Which one of following computer game␣
      ↪generes do you want to know about? (Shooter , MOBA, Survival, Strategy)",␣
      ↪"Please pick your interest to discuss it. (Shooter , MOBA, Survival,␣
      ↪Strategy)"};
         final String[] platformQuestions = {"Please pick one of the following␣
      ↪gaming platforms to talk about. (PC, Console, Mobile)" , "Which of the␣
      ↪following gaming platforms would you talk about? (PC, Console, Mobile)"};
         final String[] companyQuestions = {"Which company are you curious to know␣
      ↪about? (Ubisoft, Valve, Rockstar, Activision)","Do you wish to know about␣
      ↪one of the following gaming companies? Just State it! (Ubisoft, Valve,␣
      ↪Rockstar, Activision)"};
         final String[] controllerQuestions = {"Which of the following controller␣
      ↪types are you most comfortable with? (Keyboard, Joystick, touchscreen,␣
      ↪eyetracking)","Please pick one of the following to talk about. (Keyboard,␣
      ↪Joystick, touchscreen, eyetracking)"};
```

```
    switch (userResponse)
    {
        case "genre":
            response = inputString( genreQuestions[randomInt(2)] );
            ChckTrigNRespond(Trigs, Responses, response);
            break;

        case "platform":
            response = inputString( platformQuestions[randomInt(2)] );
            ChckTrigNRespond(Trigs, Responses, response);
            break;

        case "company":
            response = inputString( companyQuestions[randomInt(2)] );
            ChckTrigNRespond(Trigs, Responses, response);
            break;

        case "controller":
            response = inputString( controllerQuestions[randomInt(2)] );
            ChckTrigNRespond(Trigs, Responses, response);
            break;
    }
}
```

Testing

[ ]:
```
RandomQuestion(notInUse, "controller");
```

2.28 getPopularity()

What it does This method takes 4 record variable of type MyDatabase and extract their popularity field values. Then it creates and returns an integer array of those values.

Implementation (how it works) This method takes 4 record variables of type MyDatabase and store the value of popularity field of each of them to individual integer variables. Then it assigns the value of those variables to an integer array. Then returns that array.

[ ]:
```
// extract popularity field of 4 record variables of type MyDatabase and assign␣
 ↪them to a integer array and return it.
public static int[] getPopularityDetails(MyDatabase genre, MyDatabase platform,␣
 ↪MyDatabase company, MyDatabase controller)
{
    final int Genre = getPopularity(genre);
    final int Platform = getPopularity(platform);
    final int Company = getPopularity(company);
    final int Controller = getPopularity(controller);
```

```
    final int[] AllPopularities = {Genre, Platform, Company, Controller};

    return AllPopularities;
}
```

Testing

[ ]:
```
int[] sample = getPopularityDetails(notInUse,notInUse,notInUse,notInUse);
```

2.29 BubbleSort()

What it does This method is a bubble sort algorithm which takes 1 integer array and 1 string array with the same size. It sorts the integer array ascending and manipulates the string array indexes corresponding to the integer array. #### Implementation (how it works) This method is a bubble sort algorithm which takes 1 integer array and 1 string array with the same size. It sorts the integer array ascending and manipulates the string array indexes corresponding to the integer array. It uses temporary values to swap each element of arrays.

[ ]:
```
// take an integer array and a string array.
// sort the integer array ascending.
// manipulate String array in which way corresponding to the integer array␣
 ↪values.
public static void BubbleSort(int[] array, String[] array1)
{
    int size = array.length;
    int temp = 0;
    String temp1 = "";
    for (int pass = 0 ; pass < size ; pass++)
    {
        for (int position = 1 ; position < (size - pass) ; position++)
        {
            if (array[position - 1] > array[position])
            {
                // swapping using temporary values (integer array)
                temp = array[position - 1];
                array[position - 1] = array[position];
                array[position] = temp;
                // // swapping using temporary values (string array)
                temp1 = array1[position - 1];
                array1[position - 1] = array1[position];
                array1[position] = temp1;
            }
        }
    }
}
```

Testing

```
[ ]: int[] arr = {1, 9, 6, 1};
     String[] arr1 = {"a","d","c","b"};
     BubbleSort(arr,arr1);
     System.out.println(arr1[0] + arr1[1] + arr1[2] + arr1[3]);
```

2.30 repeatedQuestions()

What it does This method asks a question repeatedly. By calling other methods, it defines and uses 4 record variables about 4 different topics. Using a switch statement it chooses appropriate case based on user response and calls appropriate method using appropriate arguments. The popularity field of each topic's record variable increases by 1 unit by calling an accessor method. If the user enters skip or stop, The question asking section is skipped. Then by calling other methods it gets the popularity array of all topics. using a sort method, The popularity array and its corresponding topic names are sorted. Then it prints a string showing the sorted order of popularity of topics as well as number of times each topic is discussed.

Implementation (how it works) This method asks a question repeatedly using a while loop until 'stop' or 'skip' is entered. By calling other methods, it defines and uses 4 record variables about 4 different topics. Using a switch statement it chooses appropriate case based on user response and calls appropriate method using appropriate arguments. The popularity field of each topic's record variable increases by 1 unit by calling an accessor method. If the user enters skip or stop, The question asking section is skipped. Then by calling other methods it gets the popularity array of all topics. using a sort method, The popularity array and its corresponding topic names are sorted descending. Then it prints a string showing the sorted order of popularity of topics as well as number of times each topic is discussed. Unexpected user input is handled.

```
[ ]: // create 4 records for topics
     // ask questions and repond to answers by calling other methods (skip/stop) to␣
      ↪skip question asking.
     // get popularity of each record variable.
     // sort ascending but store in a string, descending popular topic names by␣
      ↪sorting the popularity array and manipulating correspondence.
     // return the final string
     public static String repeatedQuestions()
     {
         MyDatabase GenreTrigNResponse = CreateGenreTrigs();
         MyDatabase PlatformTrigNResponse = CreatePlatformTrigs();
         MyDatabase CompanyTrigNResponse = CreateCompanyTrigs();
         MyDatabase controllerTrigNResponse = CreatecontrollerTrigs();

         String case1 = "genre";
         String case2 ="platform";
         String case3 = "company";
         String case4 = "controller";
```

```
   final String Question = "\nPlease choose one of following topics about␣
↪gaming to discuss by entering its number.(enter 'skip' to skip these␣
↪discussions)\n1) Game Geners\n2) Gaming Platforms\n3) Gaming Companies\n4)␣
↪Gaming Controllers";
   final String ExceptionMessage = "I could not recognize your response.␣
↪Please try again entering numbers 1 to 4 or 'stop' to skip question asking.";

   String userResponse = inputString(Question);

   while(!userResponse.equalsIgnoreCase("stop") && !userResponse.
↪equalsIgnoreCase("skip"))
   {
       switch (userResponse)
       {
           case "1":
              GenreTrigNResponse = increase1popularity(GenreTrigNResponse);
              RandomQuestion(GenreTrigNResponse, case1);
              break;

           case "2":
               PlatformTrigNResponse =␣
↪increase1popularity(PlatformTrigNResponse);
               RandomQuestion(PlatformTrigNResponse, case2);
               break;

           case "3":
               CompanyTrigNResponse =␣
↪increase1popularity(CompanyTrigNResponse);
               RandomQuestion(CompanyTrigNResponse, case3);
               break;

           case "4":
               controllerTrigNResponse =␣
↪increase1popularity(controllerTrigNResponse);
               RandomQuestion(controllerTrigNResponse, case4);
               break;

           default:
               System.out.println(ExceptionMessage);
       }

       userResponse = inputString(Question);
   }

   final int[] popularitydetails = getPopularityDetails(GenreTrigNResponse,␣
↪PlatformTrigNResponse, CompanyTrigNResponse, controllerTrigNResponse);
```

```
    final String[] PopularityNames = {"Genre", "Platform", "Company",␣
↪"Controller"};
    BubbleSort(popularitydetails,PopularityNames);

    final String[] SortedOrder = PopularityNames;
    final String finalAnswer = "\nThe following list shows your interest rank␣
↪in each topic about computer gaming:\n1) " + SortedOrder[3] + "␣
↪--------------------------- " + popularitydetails[3] +" Time(s)\n2) " +␣
↪SortedOrder[2] + " --------------------------- " + popularitydetails[2] +"␣
↪Time(s)\n3) " + SortedOrder[1] + " --------------------------- " +␣
↪popularitydetails[1] + " Time(s)\n4) " + SortedOrder[0] + "␣
↪--------------------------- " + popularitydetails[0] + " Time(s)\n";
    return finalAnswer;
}
```

Testing

[ ]:
```
repeatedQuestions();
```

2.30.1 Running the program

Run the following call to simulate running the complete program.

[ ]:
```
// levels 1 to 3 inclusive
introduce();
askName();
tellAskInterest();
userStatus();

// levels 4 to 7 inclusive
final String sortedAnswer = repeatedQuestions();
System.out.println(sortedAnswer);

goodBye();
```

2.31 The complete program

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

[ ]:
```
/* ************************************
  @author    Alireza Rahimi
  @date      11 December 2021
  @version   7

        (mini Project level 7)
```

```
   This program is a chatbot intended to be a personal assistant talking to the␣
 ↪user about a specific topic
   which is computer games.

   Overall, it asks personal and general questions about this topic several␣
 ↪times
   and uses its records to respond them in a limited way. Then sorts the user␣
 ↪interest
   based on how many times they answered each topic. Possible human error␣
 ↪handled.

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

import java.util.Scanner;
import java.util.Random;
import java.util.Arrays;

class AlleriaAssistant
{
    public static void main (String [] a)
    {
        // levels 1 to 3 inclusive
        introduce();
        askName();
        tellAskInterest();
        userStatus();

        // levels 4 to 7 inclusive
        final String sortedAnswer = repeatedQuestions();
        System.out.println(sortedAnswer);

        goodBye();

        System.exit(0);
    }



    // Shows the entered string (argument) as a message and gets user response␣
 ↪from keyboard and returns it as a string.
    public static String inputString(String message)
    {
           Scanner scanner = new Scanner(System.in);

           System.out.println(message);
           String response = scanner.nextLine();
```

```
        return response;
    }



    // Return a random int between 0 (inclusive) and bound (exclusive)
    public static int randomInt(int bound)
    {
    Random r = new Random(); // built-in type
    return r.nextInt(bound); // built-in method
    }



    // after random hello phrase printed, introduces itself and its developer.
    public static void introduce()
    {
        final String[] hello = {"Hello!","Greetings!","Hi!","Hola!"};
        final int size = hello.length;
        String message = hello[randomInt(size)];
        System.out.println(message);

        message = "My name is Alleria. I am a personal Assistant developed by␣
↪Alireza.";
        System.out.println(message);

        return;
    }



    // asks user for their name and includes it in a nice to meet you phrase␣
↪and prints it.
    public static void askName()
    {
        final String userName = inputString("Please enter your name so I have␣
↪something to remember you with.");
        String message = "Glad to know you " + userName + "." + " You have such␣
↪a nice name!";
        System.out.println(message);

        return;
    }



    // ask user their opinion about games and return the response
```

```
   public static String getUserInterest()
   {
       String question = "What artificially intelligent game do you find most␣
↪thrilling?";

       final String userResponse = inputString(question);

       return userResponse;
   }



   // print cahtbot's interest and call another method to know user interest.␣
↪Then use the result to generate and print a sentence.
   public static void tellAskInterest()
   {
       final String interest = "Computer Games";
       final String myInterest = "I am designed to be interested in " +␣
↪interest + ". Specially in the ones with complicated AIs!";

       System.out.println(myInterest);

       final String userInterest = getUserInterest();

       System.out.println(userInterest + "? I love it! I think it's one of the␣
↪bests.");

       return;
   }



   // based on input argument, do something.
   //if argument is 'Y' ask user something and print another string. else␣
↪print a particular string.
   public static void commentOnUserStatus(String YorN)
   {
       final String yesMessage = "Oh nice! How long have you played it?";
       final String goodWish = "Awesome! Good luck with it.";
       final String noMessage = "Oh you have no idea what you have missed! I␣
↪strongly recommend you to play it.";

       if (YorN.equalsIgnoreCase("Y"))
       {
           final String howLong = inputString(yesMessage);
           System.out.println(goodWish);
```

```
        }
        else
        {
            System.out.println(noMessage);
        }

        return;
    }



    // this method asks about user playing status and uses user response as the␣
↪argument to call another method.
    // Possible error handling applied using a while loop.
    public static void userStatus()
    {
        final String question = "Do you personally play this game? (Y/N)";
        final String possibleError = "I could not recognize your answer. Please␣
↪answer again using Y or N:";

        boolean looping = true;

        while(looping)
        {
            String playedOrNot = inputString(question);

            if ( playedOrNot.equalsIgnoreCase("Y") || playedOrNot.
↪equalsIgnoreCase("N") )
            {
                commentOnUserStatus(playedOrNot);

                looping = false;
            }
            else
            {
                System.out.println(possibleError);
            }
        }

        return;
    }



    // prints the reason it has to go and prints a random goodbye phrase.
    public static void goodBye()
    {
```

```
        String message = "I just detected overheating on one of my CPUs! Unfortunately, I have to shutdown my systems to avoid possible damage. Sorry for any inconvenience that may cause.";
        System.out.println(message);

        final String[] goodbye = {"See you later!","Have a nice day!","Bye!","Bye bye!","Catch you later!"};
        final int size = goodbye.length;
        message = goodbye[randomInt(size)];
        System.out.println(message);

        return;
    }


    // gets record variable name and triggers array and assigns triggers array to the record field called trigs. Then return the record variable.
    public static MyDatabase setTrigs(MyDatabase MyData, String[] Trigs)
    {
        MyData.trigs = Trigs;

        return MyData;
    }


    // gets record variable name and responses array and assigns triggers array to the record field called responses. Then return the record variable.
    public static MyDatabase setResponses(MyDatabase MyData, String[] Responses)
    {
        MyData.responses = Responses;

        return MyData;
    }


    // gets record variable name and an integer and assigns the integer to the record field called popularity. Then return the record variable.
    public static MyDatabase setPopularity(MyDatabase MyData,int amount)
    {
        MyData.popularity = amount;

        return MyData;
    }


    // create and initiate a new record variable and return it taking 2 string array arguments for trigs and responses.
```

```
   // for record type MyDatabase
   public static MyDatabase createTrigNresponse(String[] Trigs, String[]␣
↪Responses)
   {
       MyDatabase trigNresponse = new MyDatabase();

       trigNresponse = setTrigs(trigNresponse, Trigs);
       trigNresponse = setResponses(trigNresponse, Responses);
       trigNresponse = setPopularity(trigNresponse, 0);

       return trigNresponse;
   }

   public static MyDatabase CreateGenreTrigs()
   {

       final String trig0 = "Shooter";
       final String trig1 = "MOBA";
       final String trig2 = "Survival";
       final String trig3 = "Strategy";

       final String response0 = "Shooter or Shooting! They come in firs person␣
↪and third person. You have oppurtunity to fight with enemies with a variety␣
↪of guns!";
       final String response1 = "MOBA stands for Multiplayer Online Battle␣
↪Arena. The name determines everything!";
       final String response2 = "You need to roam in the world and look for␣
↪goods, then survive yourself using them!";
       final String response3 = "A strategy game or strategic game is a game␣
↪in which the players' uncoerced, and often autonomous, decision-making␣
↪skills have a high significance in determining the outcome. Almost all␣
↪strategy games require internal decision tree-style thinking, and typically␣
↪very high situational awareness.";

       final String[] MyTrigs = {trig0,trig1,trig2,trig3};
       final String[] MyResponses = {response0,response1,response2,response3};

       MyDatabase trigNresponse = createTrigNresponse(MyTrigs,MyResponses);

       return trigNresponse;

   }


   public static MyDatabase CreatePlatformTrigs()
   {
```

```
        final String trig0 = "PC";
        final String trig1 = "Console";
        final String trig2 = "Mobile";


        final String response0 = "Best gaming option, because you can build␣
↪your own PC, you have more control over what games you play and how the␣
↪system operates than you would with a console.";
        final String response1 = "Consoles have advantages over PCs: They are␣
↪easy to use, don't require upgrades, make for simple multiplayer with␣
↪console-owning friends, are generally cheaper, and use wireless controllers␣
↪that allow you to have a more active experience.";
        final String response2 = "Some multiplayer mobile games help children␣
↪to develop qualities such as collaboration and cooperation. It can improve␣
↪their social connections, develop bonds, trust and teach them to play with␣
↪morals that can stay with them forever.";


        final String[] MyTrigs = {trig0,trig1,trig2};
        final String[] MyResponses = {response0,response1,response2};

        MyDatabase trigNresponse = createTrigNresponse(MyTrigs,MyResponses);

        return trigNresponse;

    }


    public static MyDatabase CreateCompanyTrigs()
    {

        final String trig0 = "Ubisoft";
        final String trig1 = "Valve";
        final String trig2 = "Rockstar";
        final String trig3 = "Activision";

        final String response0 = "Ubisoft is a nice one, Everything they make␣
↪has some satisfying gameplay, a decent story and mechanics that work pretty␣
↪well.";
        final String response1 = "Valve is my developer's favourite, because␣
↪they deliver high-quality games that have great stories, excellent␣
↪characters, and are typically ranked among the best games ever made.";
        final String response2 = "Rockstar Games is a video game publisher␣
↪established under Take-Two Interactive in 1998. It is best known for the␣
↪Grand Theft Auto series.";
```

```
        final String response3 = "Activision is a leading worldwide developer,␣
↪publisher, and distributor of interactive entertainment for various gaming␣
↪consoles, handheld platforms, and PC.";

        final String[] MyTrigs = {trig0,trig1,trig2,trig3};
        final String[] MyResponses = {response0,response1,response2,response3};

        MyDatabase trigNresponse = createTrigNresponse(MyTrigs,MyResponses);

        return trigNresponse;

    }


    public static MyDatabase CreatecontrollerTrigs()
    {

        final String trig0 = "Keyboard";
        final String trig1 = "Joystick";
        final String trig2 = "Touchscreen";
        final String trig3 = "eyetracking";

        final String response0 = "Wow keyboard! That is my developer's␣
↪favourite :D";
        final String response1 = "Joystick is a classic one!";
        final String response2 = "It is difficult to have full control using␣
↪touchscreen in games. Good Job!";
        final String response3 = "Unfortunately eyetracking technology has not␣
↪spread too much. Good news is, it is developing rapidly!";

        final String[] MyTrigs = {trig0,trig1,trig2,trig3};
        final String[] MyResponses = {response0,response1,response2,response3};

        MyDatabase trigNresponse = createTrigNresponse(MyTrigs,MyResponses);

        return trigNresponse;
    }


    // get a record variable name and return the value of its 'trigs' field
    public static String[] getTrigs(MyDatabase MyData)
    {
        String[] Trigs = MyData.trigs;

        return Trigs;
    }
```

```
   // get a record variable name and return the value of its 'responses' field
   public static String[] getResponses(MyDatabase MyData)
   {
       String[] Responses = MyData.responses;

       return Responses;
   }


   // get record variable name, increase 1 to its 'popularity' field, then␣
↪return the record variable.
   public static MyDatabase increase1popularity(MyDatabase MyData)
   {

       MyData.popularity = MyData.popularity + 1;


       return MyData;
   }


   // get a record variable name and return the value of its 'popularity' field
   public static int getPopularity(MyDatabase MyData)
   {
       int Popularity = MyData.popularity;

       return Popularity;
   }


   // Gets triggers array, responses array, and user response as argument.␣
↪Check Triggers and print corresponding responses.
   // Exception responses handled.
   public static void ChckTrigNRespond(String[] Trigs, String[] Responses,␣
↪String userResponse)
   {
       int arraySize = Trigs.length;

       for (int index = 0 ; index < arraySize ; index++)
       {
           if ( userResponse.equalsIgnoreCase(Trigs[index]) )
               {
                   System.out.println(Responses[index]);
                   return;
               }
           else
```

```
                {
                    ;
                }
        }

        System.out.println("I havo no information about that in my databases.␣
↪Please try again picking something from the list.");
        return;
    }

    // Extract trigs and responses from given record variable
    // choose a topic to ask a random question about
    // pass the extracted arrays and user response to the asked question to␣
↪another method
    // another method prints appropriate answer based on its arguments
    public static void RandomQuestion(MyDatabase MyData, String userResponse)
    {
        String[] Trigs = getTrigs(MyData);
        String[] Responses = getResponses(MyData);
        String response = "";

        final String[] genreQuestions = {"Which one of following computer game␣
↪generes do you want to know about? (Shooter , MOBA, Survival, Strategy)",␣
↪"Please pick your interest to discuss it. (Shooter , MOBA, Survival,␣
↪Strategy)"};
        final String[] platformQuestions = {"Please pick one of the following␣
↪gaming platforms to talk about. (PC, Console, Mobile)" , "Which of the␣
↪following gaming platforms would you talk about? (PC, Console, Mobile)"};
        final String[] companyQuestions = {"Which company are you curious to␣
↪know about? (Ubisoft, Valve, Rockstar, Activision)","Do you wish to know␣
↪about one of the following gaming companies? Just State it! (Ubisoft, Valve,␣
↪Rockstar, Activision)"};
        final String[] controllerQuestions = {"Which of the following␣
↪controller types are you most comfortable with? (Keyboard, Joystick,␣
↪touchscreen, eyetracking)","Please pick one of the following to talk about.␣
↪(Keyboard, Joystick, touchscreen, eyetracking)"};

        switch (userResponse)
        {
            case "genre":
                response = inputString( genreQuestions[randomInt(2)] );
                ChckTrigNRespond(Trigs, Responses, response);
                break;

            case "platform":
                response = inputString( platformQuestions[randomInt(2)] );
```

```
                ChckTrigNRespond(Trigs, Responses, response);
                break;

            case "company":
                response = inputString( companyQuestions[randomInt(2)] );
                ChckTrigNRespond(Trigs, Responses, response);
                break;

            case "controller":
                response = inputString( controllerQuestions[randomInt(2)] );
                ChckTrigNRespond(Trigs, Responses, response);
                break;
        }
    }


    // extract popularity field of 4 record variables of type MyDatabase and␣
↪assign them to a integer array and return it.
    public static int[] getPopularityDetails(MyDatabase genre, MyDatabase␣
↪platform, MyDatabase company, MyDatabase controller)
    {
        final int Genre = getPopularity(genre);
        final int Platform = getPopularity(platform);
        final int Company = getPopularity(company);
        final int Controller = getPopularity(controller);

        final int[] AllPopularities = {Genre, Platform, Company, Controller};

        return AllPopularities;
    }


    // take an integer array and a string array.
    // sort the integer array ascending.
    // manipulate String array in which way corresponding to the integer array␣
↪values.
    public static void BubbleSort(int[] array, String[] array1)
    {
        int size = array.length;
        int temp = 0;
        String temp1 = "";
        for (int pass = 0 ; pass < size ; pass++)
        {
            for (int position = 1 ; position < (size - pass) ; position++)
            {
                if (array[position - 1] > array[position])
                {
```

```
                    // swapping using temporary values (integer array)
                    temp = array[position - 1];
                    array[position - 1] = array[position];
                    array[position] = temp;
                    // // swapping using temporary values (string array)
                    temp1 = array1[position - 1];
                    array1[position - 1] = array1[position];
                    array1[position] = temp1;
                }
            }
        }
    }


    // create 4 records for topics
    // ask questions and repond to answers by calling other methods (skip|stop)␣
↪to skip question asking.
    // get popularity of each record variable.
    // sort ascending but store in a string, descending popular topic names by␣
↪sorting the popularity array and manipulating correspondence.
    // return the final string
    public static String repeatedQuestions()
    {
        MyDatabase GenreTrigNResponse = CreateGenreTrigs();
        MyDatabase PlatformTrigNResponse = CreatePlatformTrigs();
        MyDatabase CompanyTrigNResponse = CreateCompanyTrigs();
        MyDatabase controllerTrigNResponse = CreatecontrollerTrigs();

        String case1 = "genre";
        String case2 ="platform";
        String case3 = "company";
        String case4 = "controller";

        final String Question = "\nPlease choose one of following topics about␣
↪gaming to discuss by entering its number.(enter 'skip' to skip these␣
↪discussions)\n1) Game Geners\n2) Gaming Platforms\n3) Gaming Companies\n4)␣
↪Gaming Controllers";
        final String ExceptionMessage = "I could not recognize your response.␣
↪Please try again entering numbers 1 to 4 or 'stop' to skip question asking.";

        String userResponse = inputString(Question);

        while(!userResponse.equalsIgnoreCase("stop") && !userResponse.
↪equalsIgnoreCase("skip"))
        {
            switch (userResponse)
            {
```

```
                case "1":
                   GenreTrigNResponse = increase1popularity(GenreTrigNResponse);
                   RandomQuestion(GenreTrigNResponse, case1);
                   break;

                case "2":
                    PlatformTrigNResponse =␣
↪increase1popularity(PlatformTrigNResponse);
                    RandomQuestion(PlatformTrigNResponse, case2);
                    break;

                case "3":
                    CompanyTrigNResponse =␣
↪increase1popularity(CompanyTrigNResponse);
                    RandomQuestion(CompanyTrigNResponse, case3);
                    break;

                case "4":
                    controllerTrigNResponse =␣
↪increase1popularity(controllerTrigNResponse);
                    RandomQuestion(controllerTrigNResponse, case4);
                    break;

                default:
                    System.out.println(ExceptionMessage);
            }

            userResponse = inputString(Question);
        }

        final int[] popularitydetails =␣
↪getPopularityDetails(GenreTrigNResponse, PlatformTrigNResponse,␣
↪CompanyTrigNResponse, controllerTrigNResponse);
        final String[] PopularityNames = {"Genre", "Platform", "Company",␣
↪"Controller"};
        BubbleSort(popularitydetails,PopularityNames);

        final String[] SortedOrder = PopularityNames;
        final String finalAnswer = "\nThe following list shows your interest␣
↪rank in each topic about computer gaming:\n1) " + SortedOrder[3] + "␣
↪--------------------------- " + popularitydetails[3] +" Time(s)\n2) " +␣
↪SortedOrder[2] + " --------------------------- " + popularitydetails[2] +"␣
↪Time(s)\n3) " + SortedOrder[1] + " --------------------------- " +␣
↪popularitydetails[1] + " Time(s)\n4) " + SortedOrder[0] + "␣
↪--------------------------- " + popularitydetails[0] + " Time(s)\n";
        return finalAnswer;
```

```
    }

}



// create new record (user-defined type) called MyDatabase and define its␣
 ↪fields.
class MyDatabase
{
    String[] trigs;
    String[] responses;
    int popularity;
}
```

[ ]:

END OF LITERATE DOCUMENT