LAB REPORT NO. 6

Technical Support

PROF. DR. WEBER-WULFF / INFORMATIK 1 GROUP 1

Hussein Omar, Pavel Tsvyatkov
Collaborator: Lotte Unckell

May 31, 2019

Lab plan

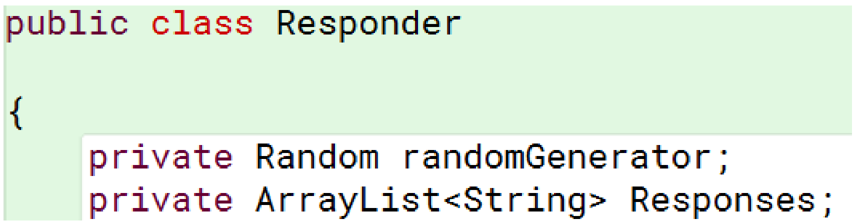
1. **Assignment introduction**

The exercises in this week's lab are intended to let us practice how to use the Random class and HashMaps. Hussein and Pavel worked together on this assignment. In the lab, before we began with the exercises, we talked about the pre-lab and discussed about HashMap and some of its methods. We managed to complete exercise 1 and 2 in the lab. Our program was working well for the first task, but we didn't have enough time to run tests for the second task. We also collaborated with our tutor Lotte Unckell. She helped us and showed us a better way to solve exercise 5.

2. **Technical Support**

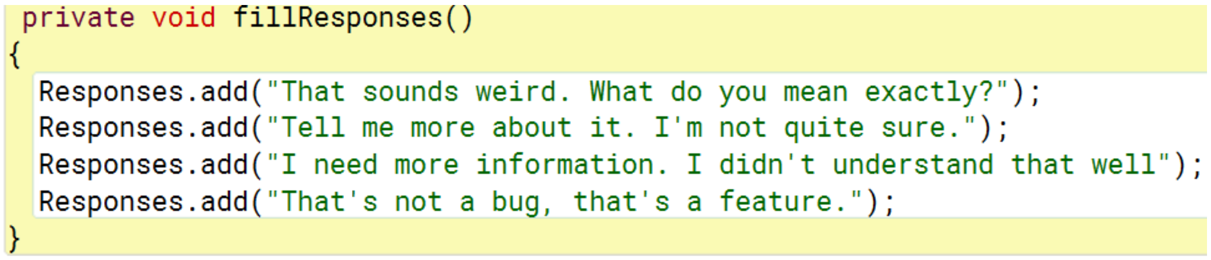
2.1 Download the basic TechSupport project that we used in the lecture. Expand it to give random responses as we did in class.

After we sat down, we downloaded the TechSupport project and opened it in BlueJ. We wanted to test the program and it was repeating the same response over and over. We then looked at the code of each class and discussed for a while what we did in class with the TechSupport project. We both had an idea about what we are supposed to do, but we both weren't sure how to implement it ourselves exactly. That's why the first exercise had us thinking a lot. We spent the first minutes trying to create a HashMap in the Responder class, but then soon we realized that we just want to give random responses and not ones that are based on keywords. We thought for a bit about it and started over. We both agreed that an ArrayList would be more suitable for this exercise. First we checked our notes and then checked the BlueJ book as well to gain a better understanding of the whole process. We discussed about how to handle the exercise and started writing the code. In the Responder class we declared fields for the random generator and our array list. We made sure not to forget importing the java.util.ArrayList and java.util.Random in the very beginning.



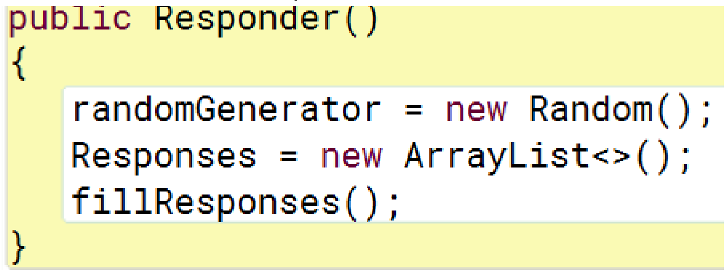
```
public class Responder

{
    private Random randomGenerator;
    private ArrayList<String> Responses;
```

In our constructor we created a new Random() called "randomGenerator" and a new ArrayList() with the name "Responses". After that we created a new method called "fillResponses()". This method fills up our array list with all the random responses that we want to use.



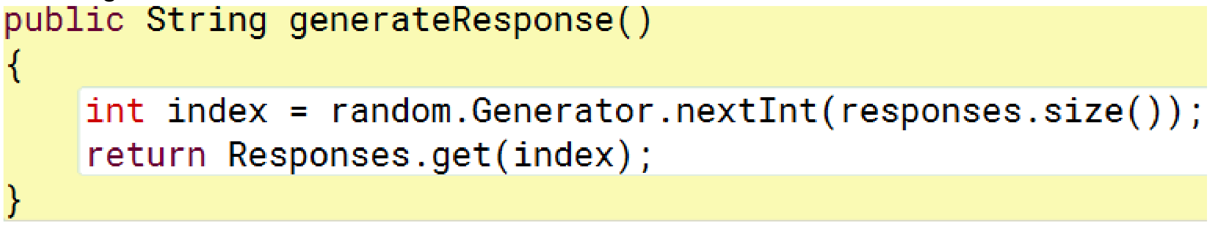
```
 private void fillResponses()
{
  Responses.add("That sounds weird. What do you mean exactly?");
  Responses.add("Tell me more about it. I'm not quite sure.");
  Responses.add("I need more information. I didn't understand that well");
  Responses.add("That's not a bug, that's a feature.");
}
```

Then we saw in our notes that it’s better to call the fillResponses() method from the constructor, so that the array list gets filled as soon as the Responder object is created so our constructor at this point looked like this.



```
public Responder()
{
    randomGenerator = new Random();
    Responses = new ArrayList<>();
    fillResponses();
}
```

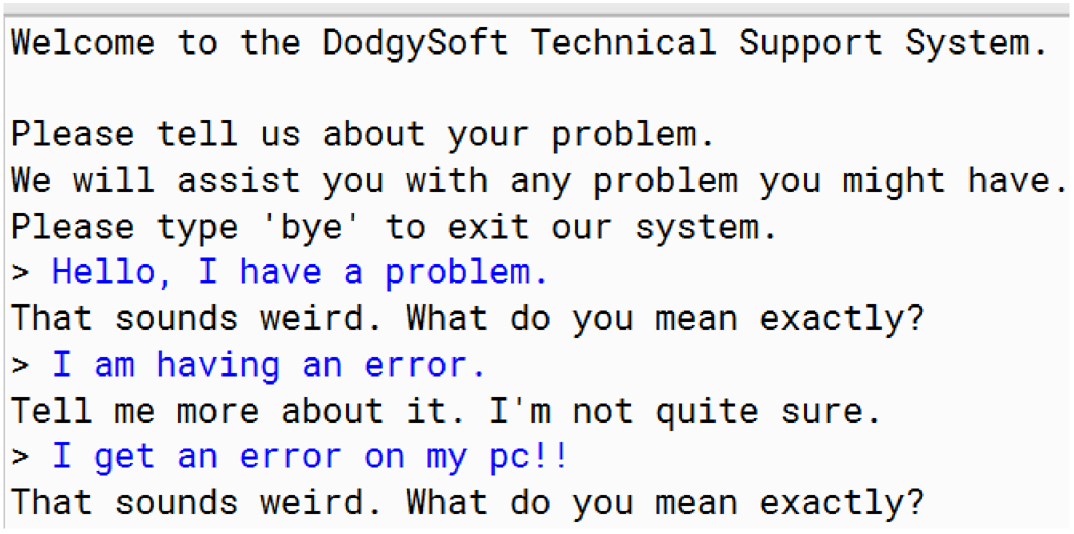
After that we had to create a new method that returns one of the random responses. Since we return a string we started writing the method “public String generateResponse()”. We knew we can get the size of an array list with the method size(). We created a local variable "index" of type int. We looked at our notes, discussed about what to do next and wrote the following.



```
public String generateResponse()
{
    int index = random.Generator.nextInt(responses.size());
    return Responses.get(index);
}
```

We get the size of our ArrayList and generate a random number, which gets stored in our local variable "index". Then we return a random response at the "index" position.

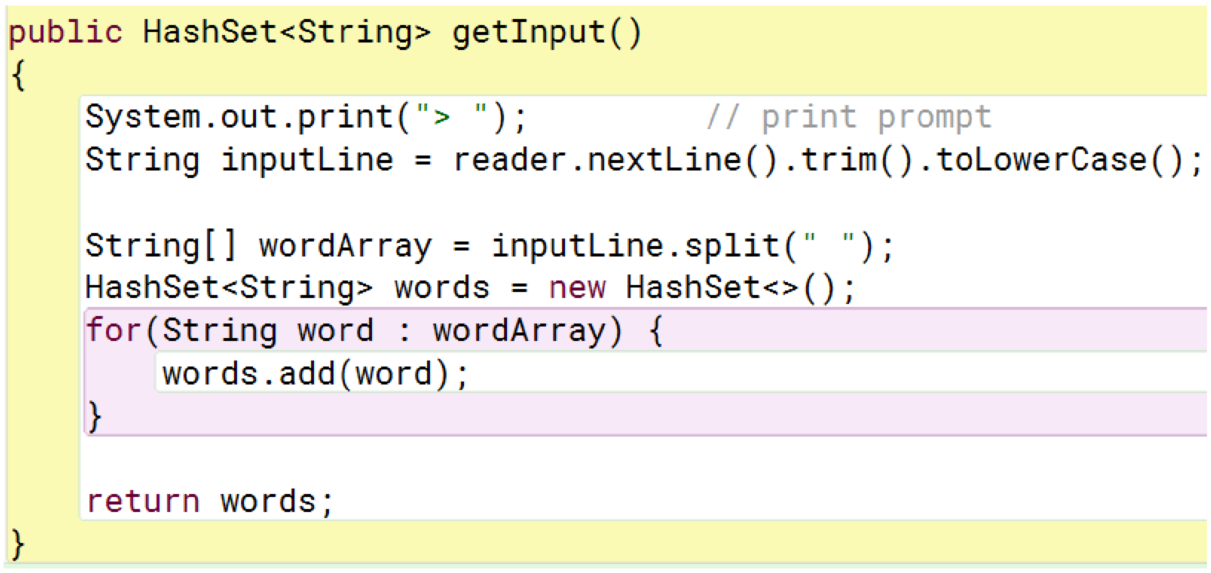
After that we tested our program and saw that it works correctly and we get different random responses. This exercise took us around 30 minutes, because we weren’t sure what to do in the beginning, but we were happy we could focus and make it work in the end.



```
Welcome to the DodgySoft Technical Support System.

Please tell us about your problem.
We will assist you with any problem you might have.
Please type 'bye' to exit our system.
> Hello, I have a problem.
That sounds weird. What do you mean exactly?
> I am having an error.
Tell me more about it. I'm not quite sure.
> I get an error on my pc!!
That sounds weird. What do you mean exactly?
```

2.2 Use the method split from String and a HashMap to give appropriate answers, depending on the input.

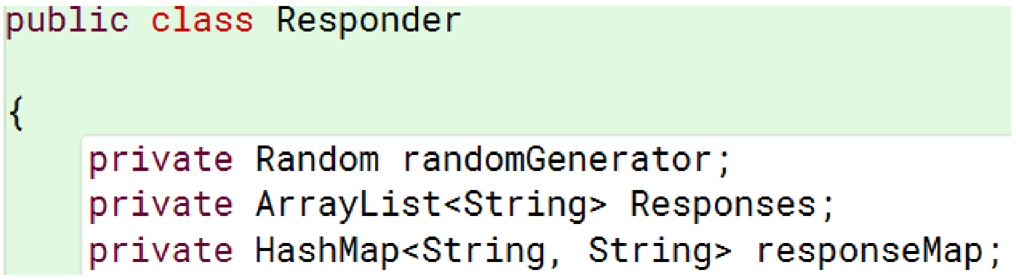
We looked at the exercise and discussed together how we should work to solve it. The BlueJ book was also very helpful here, because there were things that we didn't cover in class yet, like Arrays. When we looked at our notes we remembered that we need to change the code in the InputReader class first and we also needed to use a HashSet. First we renamed the method to public HashSet<String> getInput. We wanted to remove the leading and trailing whitespace from the inputLine so we used the method trim(). After that we also call the toLowerCase() method to make sure the input is in lowercase. Then we use the split() method on the String inputLine, which divides the string into substrings on every whitespace and returns them in an array of strings. In the end we create a new HashSet called words and use a for loop to add the words stored in the array to the HashSet "words" and return the set. At this point our code was looking like this.



```java
public HashSet<String> getInput()
{
    System.out.print("> ");         // print prompt
    String inputLine = reader.nextLine().trim().toLowerCase();

    String[] wordArray = inputLine.split(" ");
    HashSet<String> words = new HashSet<>();
    for(String word : wordArray) {
        words.add(word);
    }

    return words;
}
```

It was important to really understand every line of code before moving forward. That's why we took few minutes to discuss about what exactly happens in this piece of code and moved on.

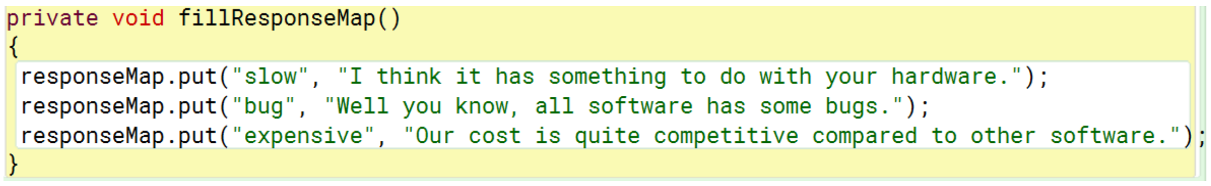
Next thing we had to do was to open up the Responder class and declare our HashMap in the class field.



```java
public class Responder

{
    private Random randomGenerator;
    private ArrayList<String> Responses;
    private HashMap<String, String> responseMap;
```

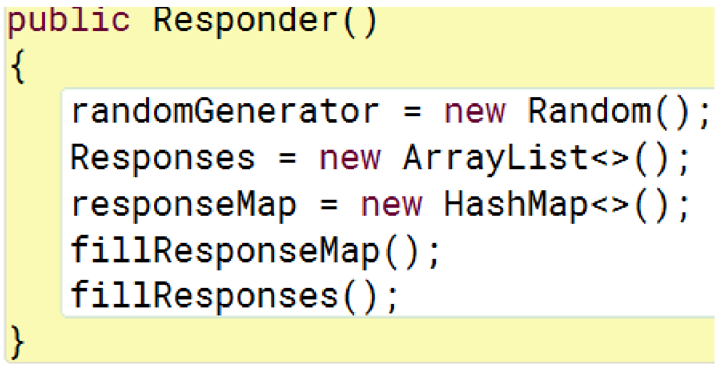
Then we both agreed we had to import the java.util.HashMap to be able to work with it.

After that we created our new HashMap called responseMap in the constructor "responseMap = new HashMap<>();". We discussed shortly and our first thought was that we need to fill our HashMap with proper responses based on keywords. The exercises from the pre-lab helped us a lot, because we felt more comfortable using some of the methods of a HashMap after searching about them online. First we created our fillResponseMap() method. Then by calling the put(Key,Value) method on our responseMap, we used some of the known keywords to look for in a user's input and associated them with proper responses.



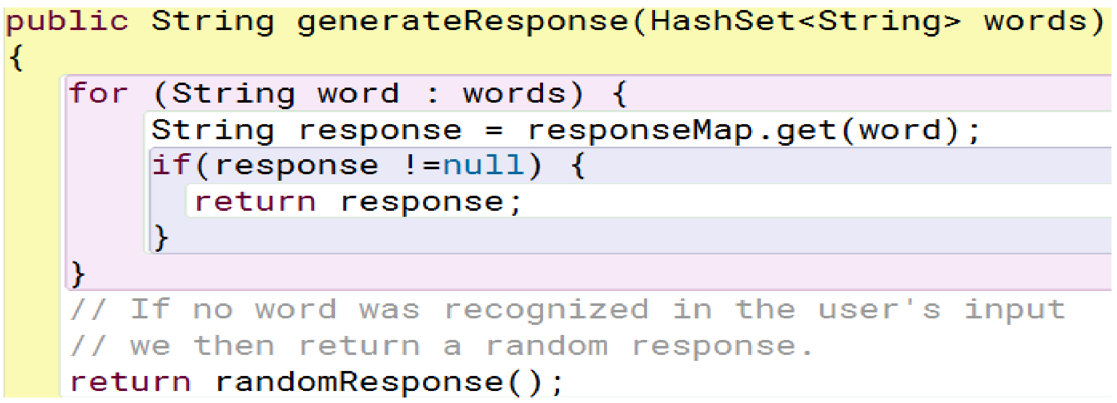
```
private void fillResponseMap()
{
 responseMap.put("slow", "I think it has something to do with your hardware.");
 responseMap.put("bug", "Well you know, all software has some bugs.");
 responseMap.put("expensive", "Our cost is quite competitive compared to other software.");
}
```

We both agreed that we need to call this method from the constructor like we did earlier for the random responses. At this point our constructor looked like this.



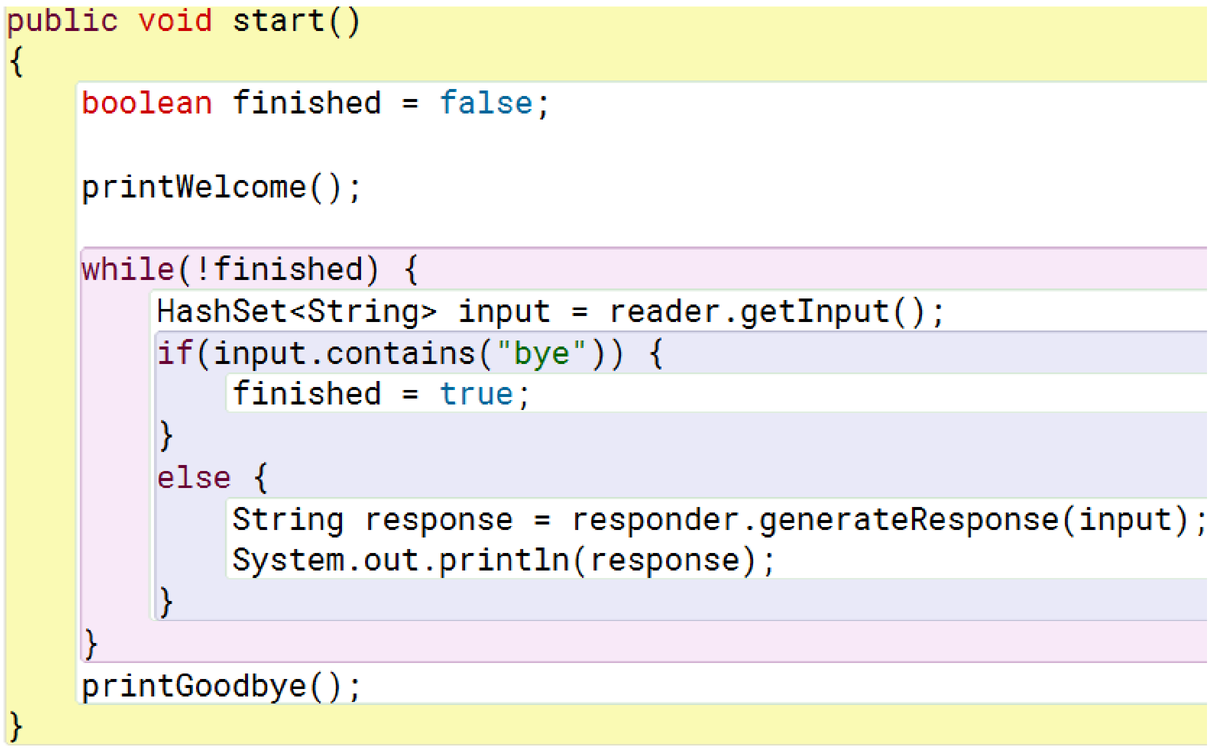
```
public Responder()
{
    randomGenerator = new Random();
    Responses = new ArrayList<>();
    responseMap = new HashMap<>();
    fillResponseMap();
    fillResponses();
}
```

We knew we were on the right way, but there was still something missing. We checked our Responder class and noticed that we need to change our generateResponse() method. Here we had to take a look at our notes again and talk about the way we should do that. First we rewrote our generateResponse() method to take a set of words as parameter generateResponse(HashSet<String> words). In the method body we used a for loop that goes over the words and checks if any of the words is in the responseMap, if there is a recognized keyword we give the response related to it by using the get() method. In case no word is recognized we had to return one of our random responses. To do that we created a new method "public String randomResponse()", which now holds the code from the generateResponse method. At this point our generateResponse method looked like that.



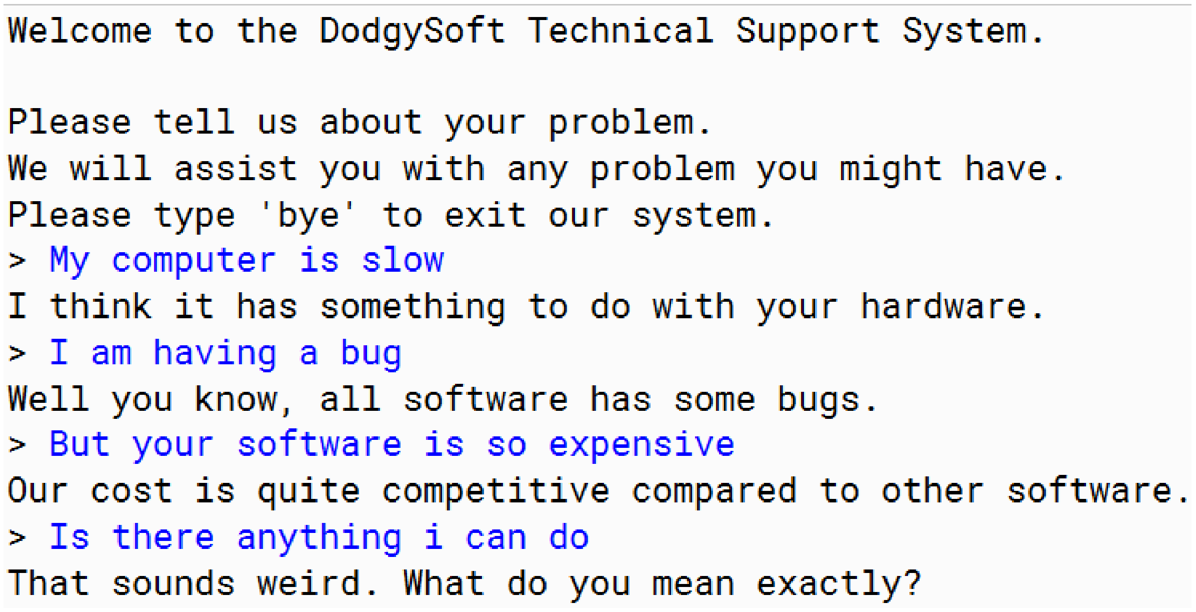
```
public String generateResponse(HashSet<String> words)
{
    for (String word : words) {
        String response = responseMap.get(word);
        if(response !=null) {
            return response;
        }
    }
    // If no word was recognized in the user's input
    // we then return a random response.
    return randomResponse();
```

At this point we had few minutes left in the lab. We decided to quickly test how our program works, but we noticed something was wrong in the SupportSystem class and we couldn't compile. We exchanged e-mails and agreed to solve the issue and work on the rest of the report at home. Later on the same day we both found our error, we had to change our start() method a bit so that it works correctly with the changes we made in the other classes. Input had to be of type HashSet<String> and in the "if" statement we needed to call the contains() method instead of startsWith(). The final thing to do was that now our generateResponse method was taking the input as parameter, so our code looked like this.



```
public void start()
{
    boolean finished = false;

    printWelcome();

    while(!finished) {
        HashSet<String> input = reader.getInput();
        if(input.contains("bye")) {
            finished = true;
        }
        else {
            String response = responder.generateResponse(input);
            System.out.println(response);
        }
    }
    printGoodbye();
}
```

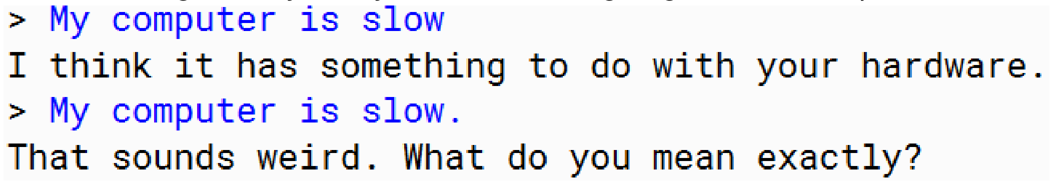
We tested our program and it was now working correctly. When a keyword was found, our program was giving a proper response to it and when there wasn't any recognized word, it just gave a random response.



```
Welcome to the DodgySoft Technical Support System.

Please tell us about your problem.
We will assist you with any problem you might have.
Please type 'bye' to exit our system.
> My computer is slow
I think it has something to do with your hardware.
> I am having a bug
Well you know, all software has some bugs.
> But your software is so expensive
Our cost is quite competitive compared to other software.
> Is there anything i can do
That sounds weird. What do you mean exactly?
```

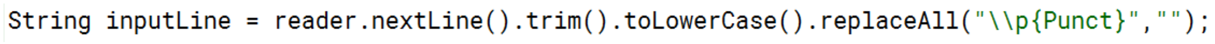
2.3 How can you deal with punctuation marks? What if there is more than one space between words? Can your tech support system deal with this?

We talked about this exercise and both assumed that more than one space wouldn't be a problem for the system, but we weren't sure about punctuation marks. We decided to test it and saw that more than one space didn't affect our system, but punctuation marks were an issue. If there was a punctuation mark before or after a keyword, the specific keyword was then not recognized by our system and it was giving a random response.



```
> My computer is slow
I think it has something to do with your hardware.
> My computer is slow.
That sounds weird. What do you mean exactly?
```

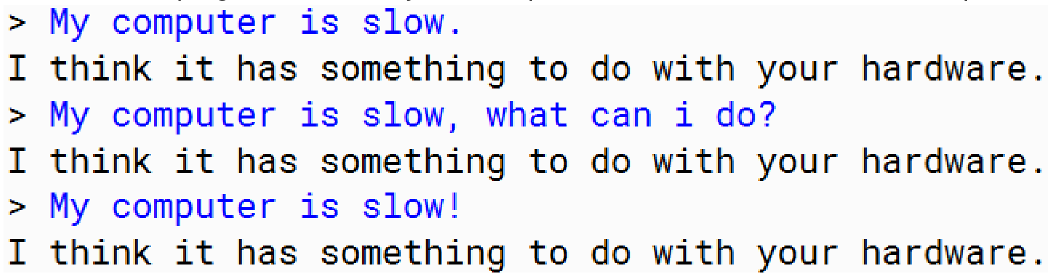
We decided to try and deal with the punctuation marks and we found a method called replaceAll() that takes two parameters – one for the thing we look for and another for the thing to replace it to - and can be called on a String. By calling this method on the "String input" in the InputReader class, we could deal with any punctuation mark. We then added it to the method and we were curious if our program can deal with punctuation marks.



```
String inputLine = reader.nextLine().trim().toLowerCase().replaceAll("\\p{Punct}","");
```

The exact syntax how to replace all punctuation marks we found here: https://www.geeksforgeeks.org/removing-punctuations-given-string/ [1].

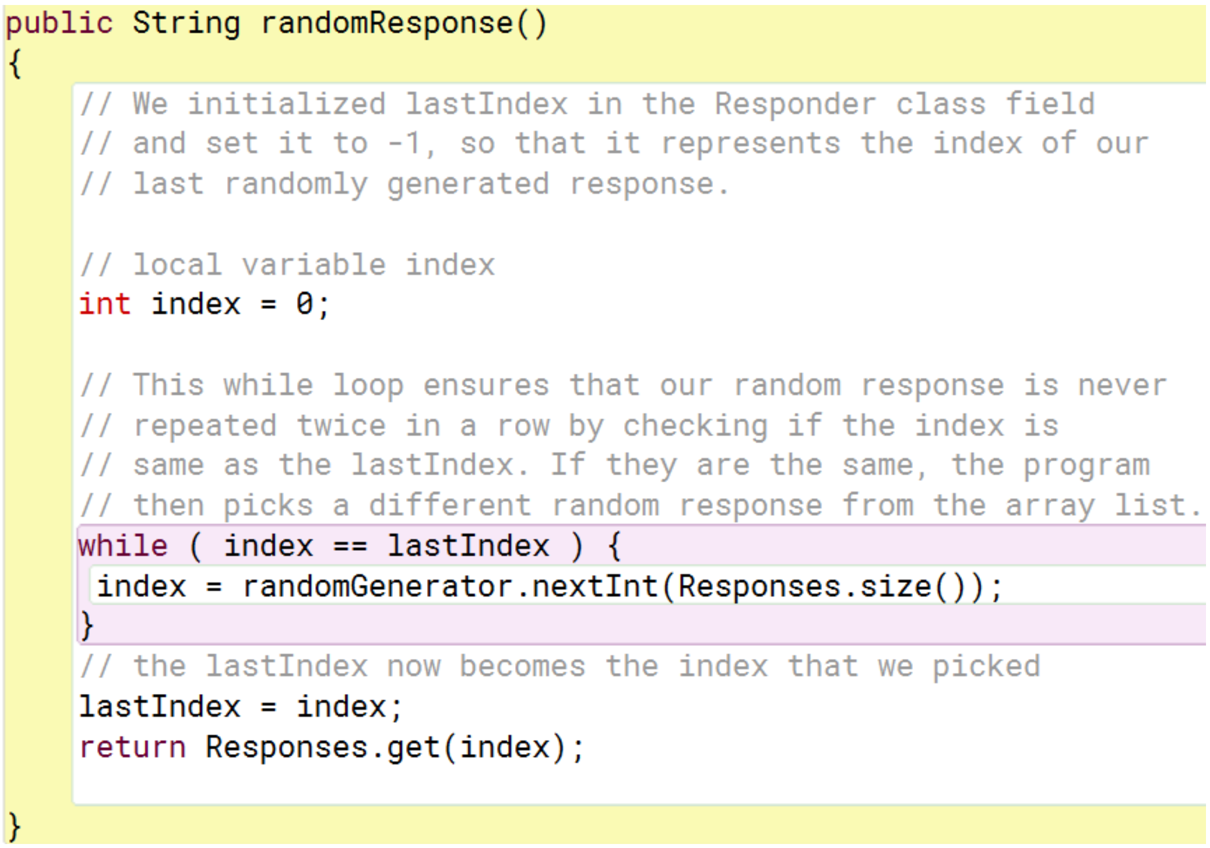
We tested the program and it really worked, punctuation marks didn't affect the input.



```
> My computer is slow.
I think it has something to do with your hardware.
> My computer is slow, what can i do?
I think it has something to do with your hardware.
> My computer is slow!
I think it has something to do with your hardware.
```

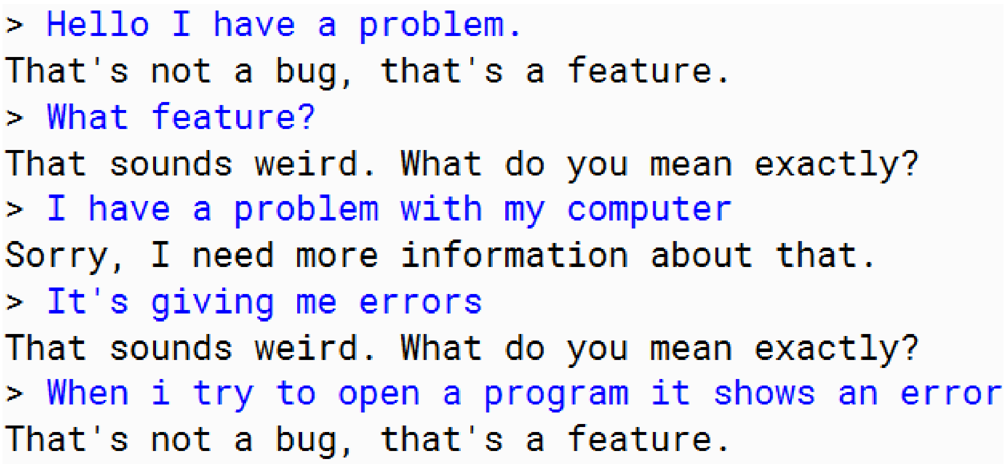
2.4 Ensure that the same default response is never repeated twice in a row.

We talked about this exercise and we were both sure that we can do that by changing our randomResponse() method. We had the same idea that since we use an array list for our random responses we could refer to the responses with an index. Our array list "Responses" was picking an index between 0 and size()-1. We both agreed to put comments, so that we knew exactly what every line of code does. It was also important to initialize a variable that we called "lastIndex" and gave it the value -1 so it represents the index of our last response. After rewriting our randomResponse() method it looked like that.



```java
public String randomResponse()
{
    // We initialized lastIndex in the Responder class field
    // and set it to -1, so that it represents the index of our
    // last randomly generated response.

    // local variable index
    int index = 0;

    // This while loop ensures that our random response is never
    // repeated twice in a row by checking if the index is
    // same as the lastIndex. If they are the same, the program
    // then picks a different random response from the array list.
    while ( index == lastIndex ) {
     index = randomGenerator.nextInt(Responses.size());
    }
    // the lastIndex now becomes the index that we picked
    lastIndex = index;
    return Responses.get(index);

}
```

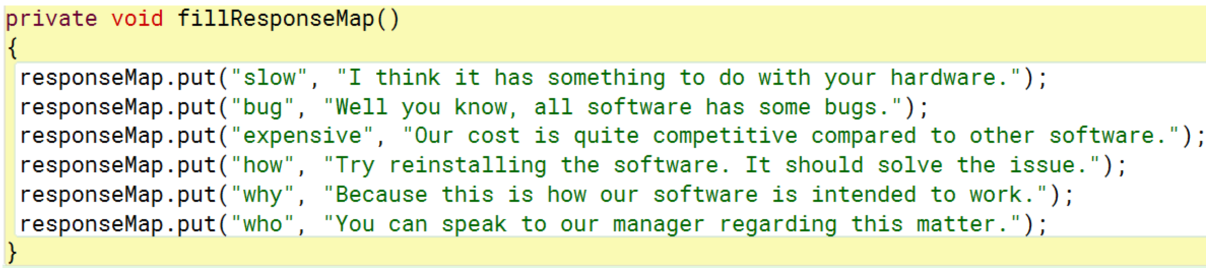
We tested our support system and it was never repeating the same response twice in a row.



```
> Hello I have a problem.
That's not a bug, that's a feature.
> What feature?
That sounds weird. What do you mean exactly?
> I have a problem with my computer
Sorry, I need more information about that.
> It's giving me errors
That sounds weird. What do you mean exactly?
> When i try to open a program it shows an error
That's not a bug, that's a feature.
```

2.5 When no word is recognized, use other words from the user's input to pick a well-fitting default response: for example, words such as "why", "how", and "who" might be useful.

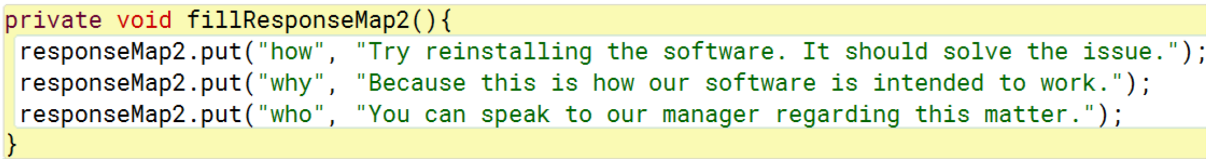
After thinking about this exercise, we were both not sure what would be the proper way to solve this exercise. We both had the same thought, that we can use a second HashMap and create "if" cases when giving a response, which check if a user's input contains any of the known keywords. However, we weren't sure how to structure our code exactly and weren't able to solve it that way. We then read the exercise again and thought that adding more keywords to the HashMap we already have would also be a correct way to do it, so we added some more keywords for which our support system should look in a user's input. We added them to the fillResponseMap() method.



```
private void fillResponseMap()
{
 responseMap.put("slow", "I think it has something to do with your hardware.");
 responseMap.put("bug", "Well you know, all software has some bugs.");
 responseMap.put("expensive", "Our cost is quite competitive compared to other software.");
 responseMap.put("how", "Try reinstalling the software. It should solve the issue.");
 responseMap.put("why", "Because this is how our software is intended to work.");
 responseMap.put("who", "You can speak to our manager regarding this matter.");
}
```

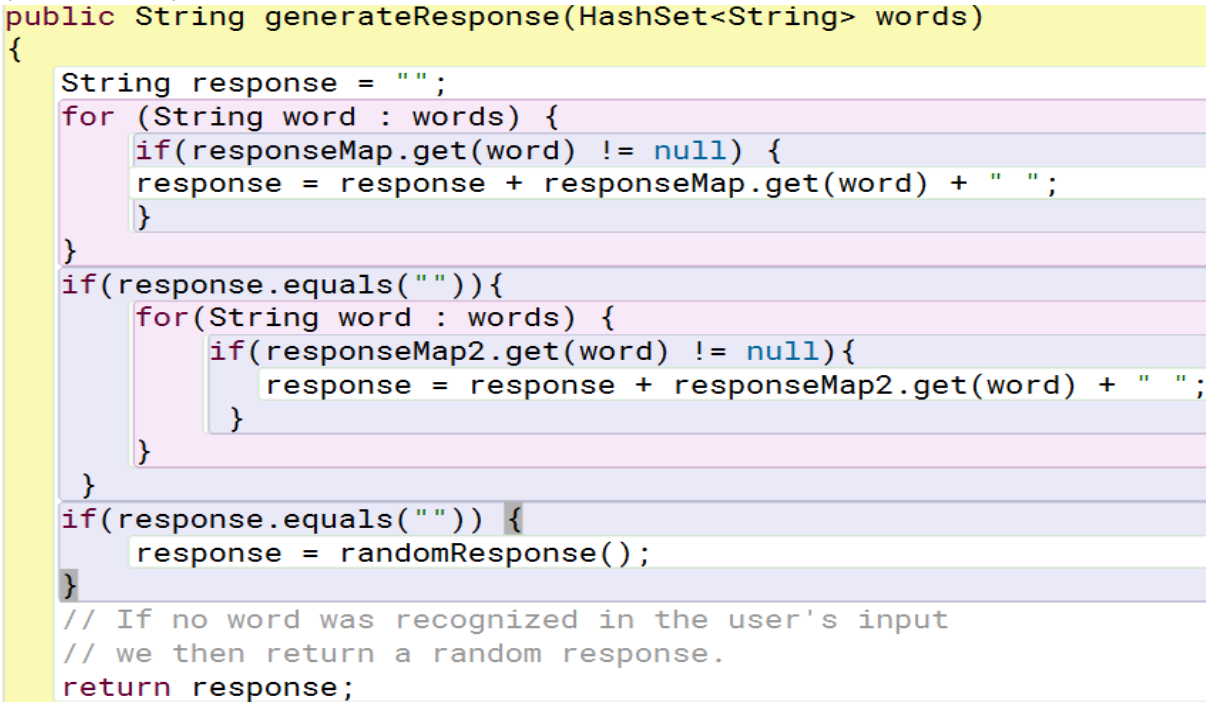
Then we tested our program and it seemed to work fine, even though we weren't sure if the way we solved the exercise was correct. We decided to ask our tutor Lotte about the initial thoughts we had about solving this exercise.

We collaborated with our tutor Lotte and she explained to us how we could go about this exercise and solve it with two HashMaps, the way we first thought about it. We created a new HashMap called responseMap2. We created a method called fillResponseMap2() and added keywords to it. We also didn't forget to call the fillResponseMap2() method from the constructor. Our fillResponseMap2() method looked like this.



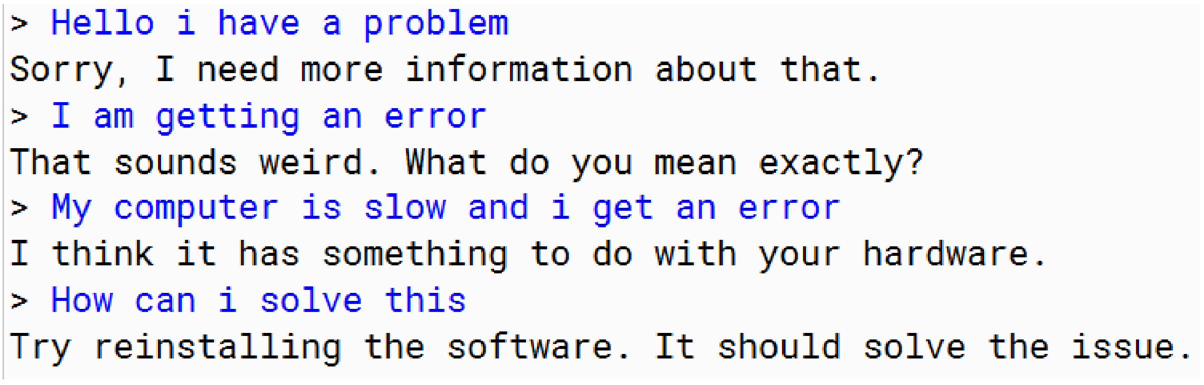
```
private void fillResponseMap2(){
 responseMap2.put("how", "Try reinstalling the software. It should solve the issue.");
 responseMap2.put("why", "Because this is how our software is intended to work.");
 responseMap2.put("who", "You can speak to our manager regarding this matter.");
}
```

Then we had to rewrite our generateResponse method so that it checks about any recognized keywords in responseMap first, then in responseMap2 and if no keyword was found in either of them, then it generates a random response. After getting help from our tutor, our generateResponse method looked like that.



```
public String generateResponse(HashSet<String> words)
{
    String response = "";
    for (String word : words) {
        if(responseMap.get(word) != null) {
        response = response + responseMap.get(word) + "  ";
        }
    }
    if(response.equals("")){
        for(String word : words) {
            if(responseMap2.get(word) != null){
                response = response + responseMap2.get(word) + "  ";
            }
        }
     }
    if(response.equals("")) {
        response = randomResponse();
    }
    // If no word was recognized in the user's input
    // we then return a random response.
    return response;
```

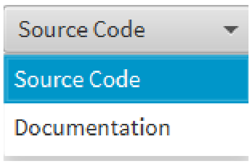
We were excited to see it working and were thankful that our tutor showed us how we can solve the exercise with two HashMaps.



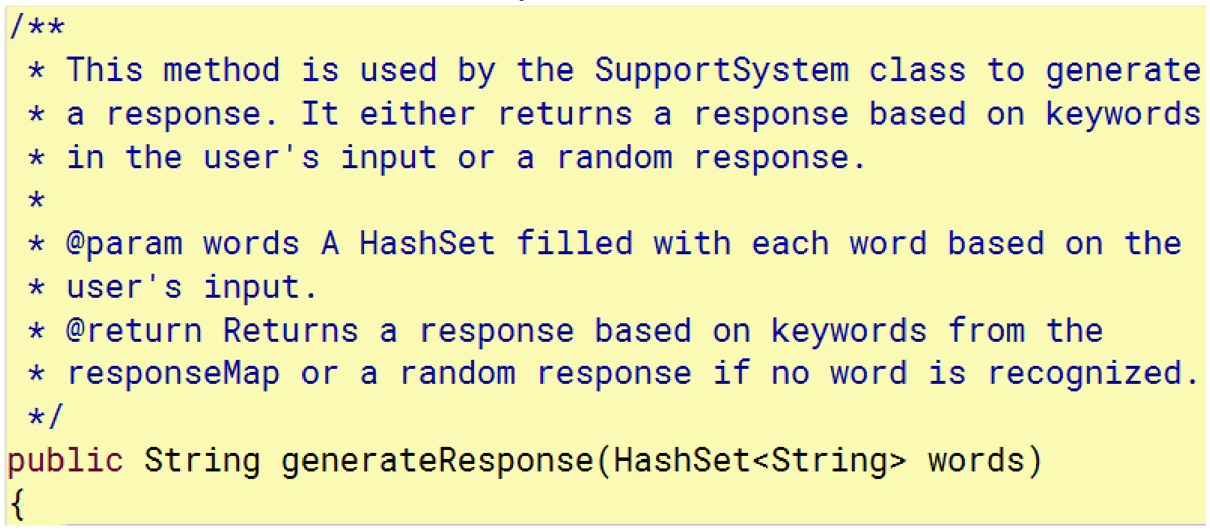
```
> Hello i have a problem
Sorry, I need more information about that.
> I am getting an error
That sounds weird. What do you mean exactly?
> My computer is slow and i get an error
I think it has something to do with your hardware.
> How can i solve this
Try reinstalling the software. It should solve the issue.
```

2.6 Document your code thoroughly using javadoc!

This exercise made us think more about the code that we write and helped us understand it better by explaining what each method does exactly. It was very helpful to do it properly, because in BlueJ we could change the view between Source Code and Documentation any time we wanted to and it looked exactly like when we search in the Java API.

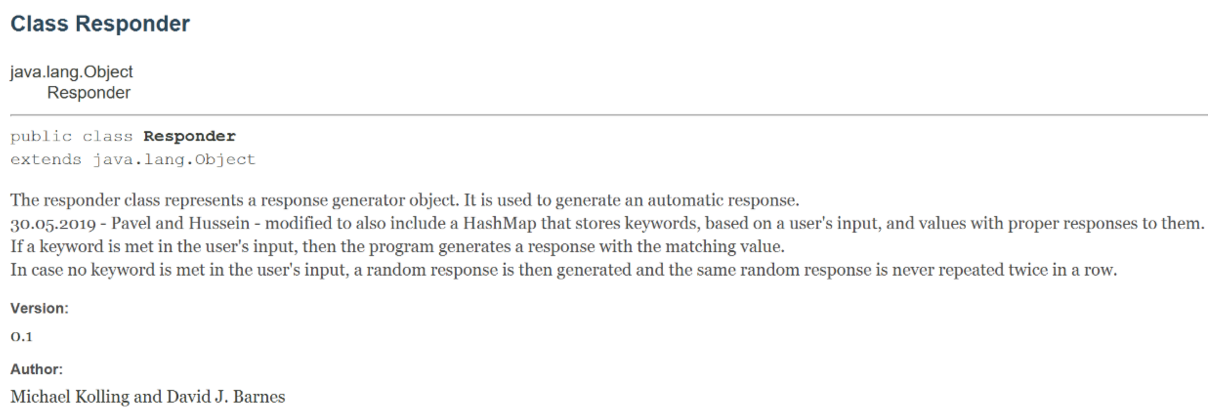


There were certain key symbols we could use like @param and @return to make our documentation understandable and easy to read.



```
/**
 * This method is used by the SupportSystem class to generate
 * a response. It either returns a response based on keywords
 * in the user's input or a random response.
 *
 * @param words A HashSet filled with each word based on the
 * user's input.
 * @return Returns a response based on keywords from the
 * responseMap or a random response if no word is recognized.
 */
public String generateResponse(HashSet<String> words)
{
```

The last exercise in our pre-lab was asking us to search where we can find more information about javadoc online and we also found that we can use HTML tags inside a javadoc comment. We weren't sure if we can use @author, since we only modified the program and we weren't the authors, so we decided to not use it, but write that we modified the program. We decided to include some example screenshots in our report.



Class Responder

java.lang.Object
    Responder

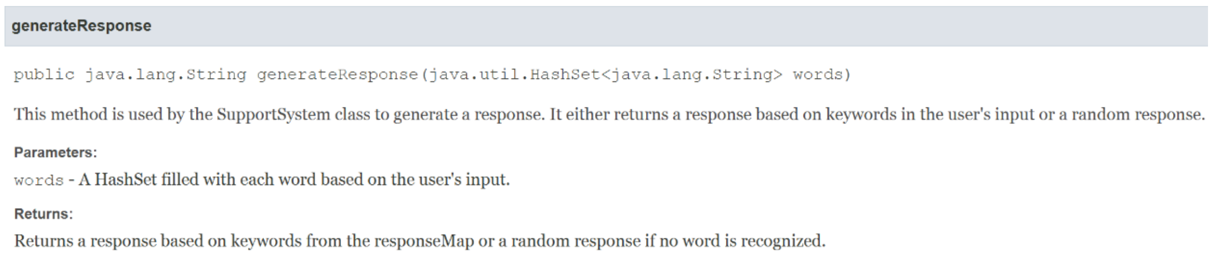
```
public class Responder
extends java.lang.Object
```

The responder class represents a response generator object. It is used to generate an automatic response.
30.05.2019 - Pavel and Hussein - modified to also include a HashMap that stores keywords, based on a user's input, and values with proper responses to them.
If a keyword is met in the user's input, then the program generates a response with the matching value.
In case no keyword is met in the user's input, a random response is then generated and the same random response is never repeated twice in a row.

Version:
0.1

Author:
Michael Kolling and David J. Barnes

It was very interesting to read it, because we could now easily see a description about each method that we wrote.



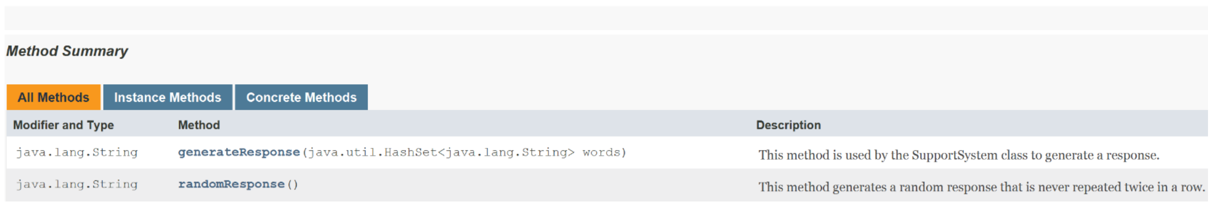
generateResponse

```
public java.lang.String generateResponse(java.util.HashSet<java.lang.String> words)
```

This method is used by the SupportSystem class to generate a response. It either returns a response based on keywords in the user's input or a random response.

Parameters:
`words` - A HashSet filled with each word based on the user's input.

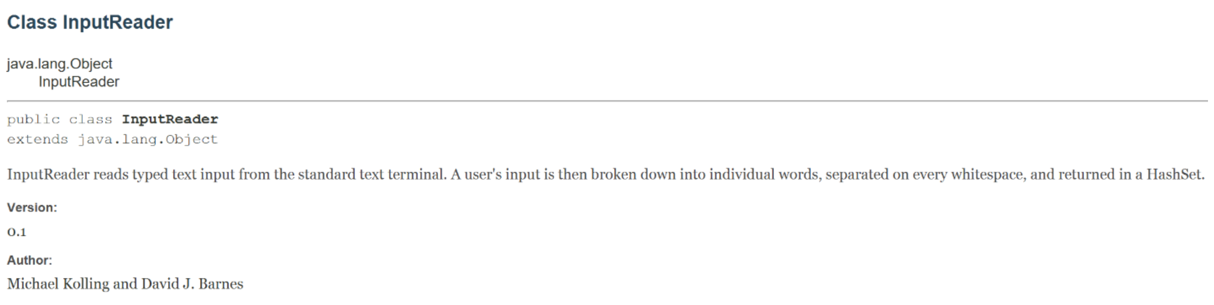
Returns:
Returns a response based on keywords from the responseMap or a random response if no word is recognized.



Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method	Description
`java.lang.String`	`generateResponse(java.util.HashSet<java.lang.String> words)`	This method is used by the SupportSystem class to generate a response.
`java.lang.String`	`randomResponse()`	This method generates a random response that is never repeated twice in a row.



Class InputReader

java.lang.Object
    InputReader

```
public class InputReader
extends java.lang.Object
```

InputReader reads typed text input from the standard text terminal. A user's input is then broken down into individual words, separated on every whitespace, and returned in a HashSet.

Version:
0.1

Author:
Michael Kolling and David J. Barnes

3. After the lab

After the lab we weren't ready with the exercises so we exchanged e-mails and sent the BlueJ file and the notes we took to each other. We both agreed to continue working on the exercises at home. We were discussing about every exercise and looking for working solutions. To finish the exercises and write our report we spent around 4 hours in total.

4. What I learned

4.1 Hussein:

Unfortunately I was sick the whole week before the Lab exercise so I missed the class where we talked about HashMaps. But fortunately my lab partner was of great help as he explained to me some of the stuff I didn't fully understand. This exercise was also a good way to understand how to go about using a HashMap and how it can be implemented in a smart way. At first we had a rough start and the pressure got to us because we didn't know exactly where to start. But after a while we started to make progress and the book was also of great help to us.

4.2 Pavel:

While working on the exercises I learned a lot about generating responses based on user input. I needed some time to understand the code and how the three classes work together. I learned how to give random response based on the user's input and how to make the program never repeat the same random response twice. Then by using a HashMap, I learned how to give a response based on keywords in the user's input. I also learned about a new method called "replaceAll()", that can be called on a String and we were able to deal with the punctuation marks in exercise 3 with it. In this lab exercise I also learned more about javadoc and how to properly document my code. Documenting the code properly was very helpful, because it helped me understand the code better and made me think more about what exactly each method does.

5.Appendix

5.1 Reference links

[1] https://www.geeksforgeeks.org/removing-punctuations-given-string/

5.2 Pre-Lab

- Hussein

P1. Using the Java documentation on Map and HashMap, write a short paragraph explaining what a HashMap is, what its purpose is and how it is used.

It's like an arrayList but uses a key to find a Value which is stored in the Map. However you can't find a key using a Value. The HashMap cannot contain duplicate keys and is an unordered collection which means that the order of the elements isn't guaranteed. It also allows null values and the null key. Java HashMap is not thread-safe. You must explicitly synchronize concurrent modifications to the HashMap.

P2. HashMap is a parameterized class. List the methods that depend on the types used to parameterize it. Do you think the same type could be used for both parameters?

get(Object key)
Parameters:
key - the key whose associated value is to be returned
containsKey(Object key)
Parameters:
key - The key whose presence in this map is to be tested
remove(Object key)
Parameters:
key - key whose mapping is to be removed from the map
put(K key, V value)
Parameters:
key - key with which the specified value is to be associated
value - value to be associated with the specified key
keySet()
size()

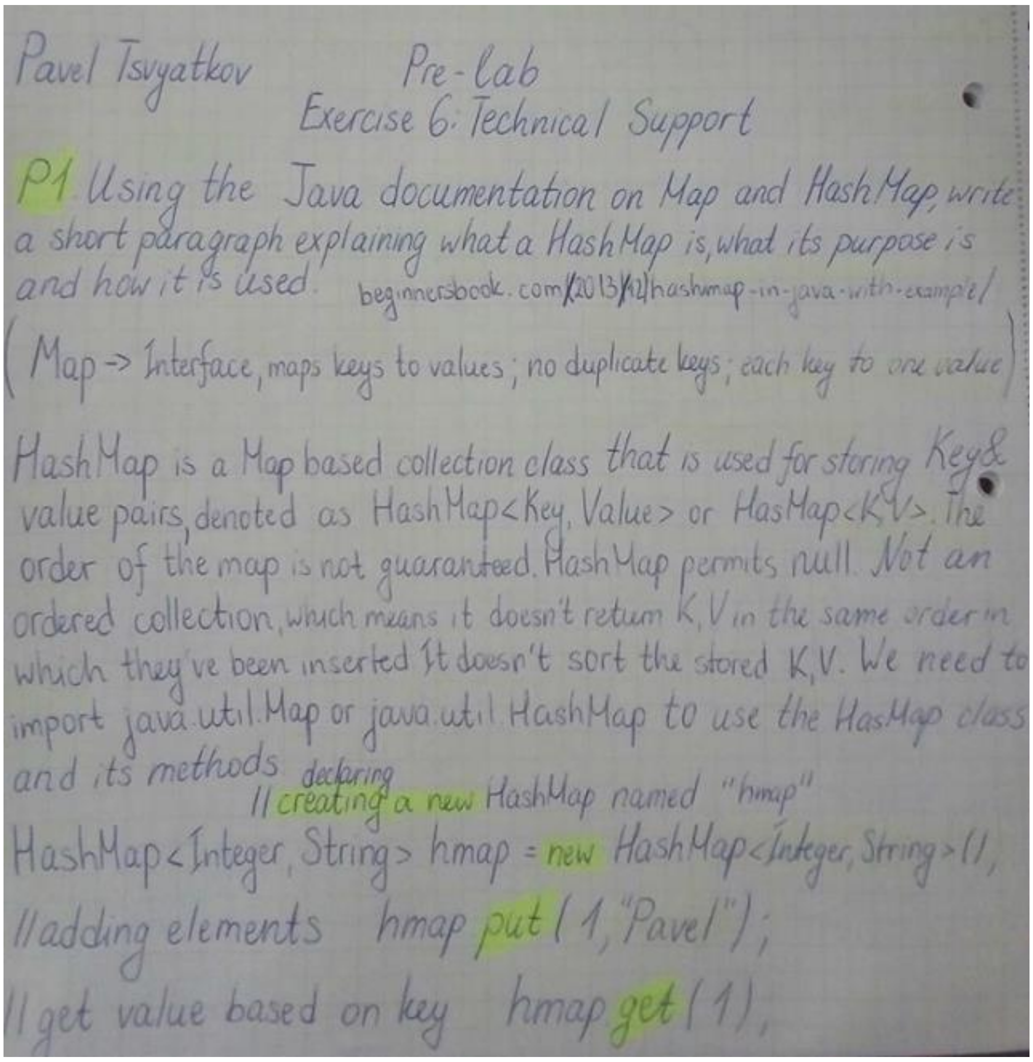
P3. How do you check how many entries are contained in a map? What happens when you add an entry to a map with a key that already exists in the map? How do you print out all the keys currently stored in a map?

- the number gets replaced with the new number
- Using the size() method from the HashMap

P4. Where can you find information about javadoc on the Internet?

-doc.oracle.com
-cs.duke.edu

- Pavel



Pavel Tsvyatkov Pre-lab

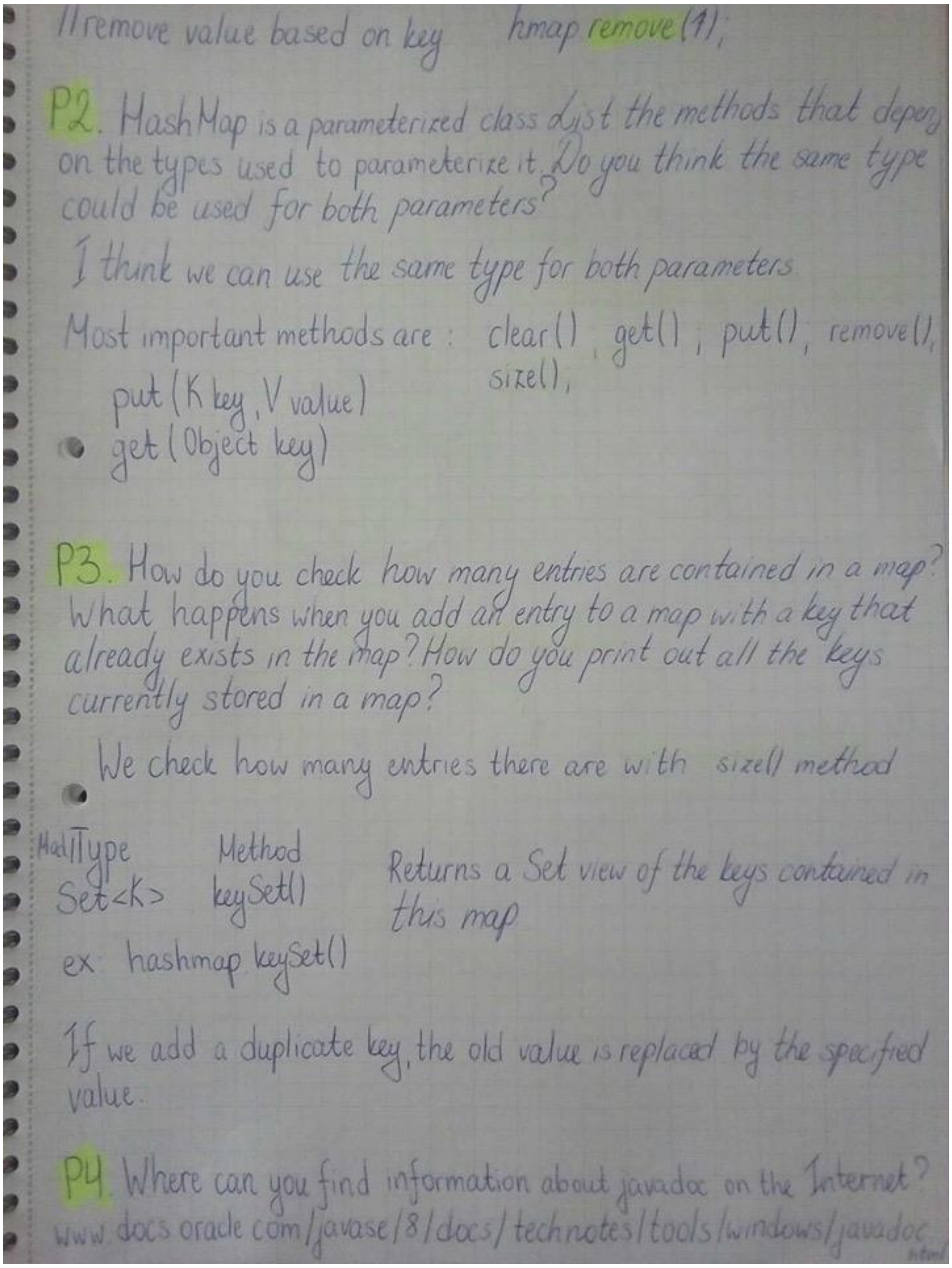
Exercise 6: Technical Support

P1 Using the Java documentation on Map and HashMap, write a short paragraph explaining what a HashMap is, what its purpose is and how it is used. beginnersbook.com/2013/12/hashmap-in-java-with-example/

(Map -> Interface, maps keys to values; no duplicate keys; each key to one value)

HashMap is a Map based collection class that is used for storing Key & value pairs, denoted as HashMap<Key, Value> or HasMap<K,V>. The order of the map is not guaranteed. HashMap permits null. Not an ordered collection, which means it doesn't return K,V in the same order in which they've been inserted. It doesn't sort the stored K,V. We need to import java.util.Map or java.util.HashMap to use the HasMap class and its methods.

```
         declaring
        //creating a new HashMap named "hmap"
HashMap<Integer, String> hmap = new HashMap<Integer, String>();
//adding elements    hmap.put(1, "Pavel");
//get value based on key    hmap.get(1);
```



//remove value based on key   hmap.remove(1);

P2. HashMap is a parameterized class. List the methods that depen on the types used to parameterize it. Do you think the same type could be used for both parameters?

I think we can use the same type for both parameters.

Most important methods are : clear() , get() ; put(); remove(), size();

put (K key, V value)
get (Object key)

P3. How do you check how many entries are contained in a map? What happens when you add an entry to a map with a key that already exists in the map? How do you print out all the keys currently stored in a map?

We check how many entries there are with size() method

MapType	Method	
Set<K>	keySet()	Returns a Set view of the keys contained in this map

ex: hashmap.keySet()

If we add a duplicate key, the old value is replaced by the specified value.

P4. Where can you find information about javadoc on the Internet?
www.docs.oracle.com/javase/8/docs/technotes/tools/windows/javadoc.html

5.3 Our Code

5.3.1 SupportSystem

```
import java.util.Map;
import java.util.HashMap;
import java.util.HashSet;

/**
 * This class implements a technical support system. It is the top
 * level class in this project. The support system communicates via
 * text input/output in the text terminal.
 *
 * This class uses an object of class InputReader to read input from
 * the user, and an object of class Responder to generate responses.
 * It contains a loop that repeatedly reads input and generates output
 * until the users wants to leave.
 *
 * @author     Michael Kolling and David J. Barnes
 * @version    0.1
 */
public class SupportSystem
{
    private InputReader reader;
    private Responder responder;

    /**
     * Creates a technical support system.
     */
    public SupportSystem()
    {
        reader = new InputReader();
        responder = new Responder();
    }

    /**
     * Start the technical support system. This will print a welcome
     * message and enter into a dialog with the user, until the user
     * ends the dialog.
     */
    public void start()
    {
        boolean finished = false;

        printWelcome();
        while(!finished) {
            HashSet<String> input = reader.getInput();
            if(input.contains("bye")) {
                finished = true;
            }
            else {
                String response = responder.generateResponse(input);
                System.out.println(response);
            }
        }
        printGoodbye();
    }

    /**
     * Print a welcome message to the screen.
     */
    private void printWelcome()
    {
        System.out.println("Welcome to the DodgySoft Technical Support System.");
        System.out.println();
        System.out.println("Please tell us about your problem.");
        System.out.println("We will assist you with any problem you might have.");
        System.out.println("Please type 'bye' to exit our system.");
    }

    /**
     * Print a good-bye message to the screen.
     */
    private void printGoodbye()
    {
        System.out.println("Nice talking to you. Bye...");
    }
}
```

5.3.2 InputReader

```
import java.util.HashSet;
import java.util.Scanner;

/**
 * InputReader reads typed text input from the standard text terminal.
 * A user's input is then broken down into individual words, separated on every whitespace,
 * and returned in a HashSet.
 * @author     Michael Kolling and David J. Barnes
 * @version    0.1
 */
```

```
public class InputReader
{
    private Scanner reader;

    /**
     * Create a new InputReader that reads text from the text terminal.
     */
    public InputReader()
    {
        reader = new Scanner(System.in);
    }

    /**
     * The getInput method takes no parameters. It reads
     * a line of text from standard input (the text terminal), we
     * then use trim() to remove leading and trailing whitespace, then
     * by using toLowerCase() the input is now lowercase. In the end
     * we replace all punctuation marks. We use the .split() method to
     * split the user's input on every whitespace. We use a for loop to
     * add every word from the input into a HashSet called words.
     *
     * @return  A set of Strings, but now each String is every single word
     * typed by the user.
     */
    public HashSet<String> getInput()
    {
        System.out.print("> ");         // print prompt
        String inputLine = reader.nextLine().trim().toLowerCase().replaceAll("\\p{Punct}","");

        String[] wordArray = inputLine.split(" ");
        HashSet<String> words = new HashSet<>();
        for(String word : wordArray) {
            words.add(word);
        }

        return words;
    }
}
```

5.3.3 Responder

```
import java.util.Map;
import java.util.HashMap;
import java.util.ArrayList;
import java.util.Random;
import java.util.HashSet;
```

```
/**
 * The responder class represents a response generator object.
 * It is used to generate an automatic response.<br>
 * 30.05.2019 - Pavel and Hussein - modified to also include a HashMap that
 * stores keywords, based on a user's input, and values with proper responses
 * to them. <br>If a keyword is met in the user's input, then the program generates
 * a response with the matching value. <br>In case no keyword is met in the user's
 * input, a random response is then generated and the same random response is
 * never repeated twice in a row.
 *
 * @author     Michael Kolling and David J. Barnes
 * @version    0.1
 */
public class Responder

{
    private Random randomGenerator;
    private ArrayList<String> Responses;
    private HashMap<String, String> responseMap;
    private HashMap<String, String> responseMap2;
    private int lastIndex = -1;
    /**
     * Construct a Responder - it consists of a random generator,
     * an array list called Responses, which holds our random responses,
     * a HashMap called responseMap, which holds the responses based on
     * keywords in the user's input and two methods - fillResponses
     * and fillResponseMap, where we add our responses to display.
     */
    public Responder()
    {
        randomGenerator = new Random();
        Responses = new ArrayList<>();
        responseMap = new HashMap<>();
        responseMap2 = new HashMap<>();
        fillResponseMap();
        fillResponseMap2();
        fillResponses();
    }

    /**
     * This method generates a random response that is never
     * repeated twice in a row. It compares the current and the last index,
     * the lastIndex is kept at class level. If they are the same it picks
     * a different current index, which now becomes the last index.
     *
     * @return   A string that should be displayed as the response
     */
```

```
public String randomResponse()
{
    // We initialized lastIndex in the Responder class field
    // and set it to -1, so that it represents the index of our
    // last randomly generated response.

    // local variable index
    int index = 0;

    // This while loop ensures that our random response is never
    // repeated twice in a row by checking if the index is
    // same as the lastIndex. If they are the same, the program
    // then picks a different random response from the array list.
    while ( index == lastIndex ) {
     index = randomGenerator.nextInt(Responses.size());
    }
    // the lastIndex now becomes the index that we picked
    lastIndex = index;
    return Responses.get(index);

}

/**
 * This method is used by the SupportSystem class to generate
 * a response. It either returns a response based on keywords
 * in the user's input or a random response.
 *
 * @param words A HashSet filled with each word based on the
 * user's input.
 * @return Returns a response based on keywords from the
 * responseMap or a random response if no word is recognized.
 */
public String generateResponse(HashSet<String> words)
{
   String response = "";
   for (String word : words) {
       if(responseMap.get(word) != null) {
       response = response + responseMap.get(word) + " ";
       }
   }
   if(response.equals("")){
       for(String word : words) {
           if(responseMap2.get(word) != null){
              response = response + responseMap2.get(word) + " ";
            }
       }
    }
   if(response.equals("")) {
```

```
            response = randomResponse();
        }

        // If no word was recognized in the user's input
        // we then return a random response.
        return response;
    }

    /**
     * Create a list of random responses. The program picks a random
     * response in case there is no matching keyword in the user's input.
     */
     private void fillResponses()
    {
      Responses.add("That sounds weird. What do you mean exactly?");
      Responses.add("Tell me more about it. I'm not quite sure.");
      Responses.add("Sorry, I need more information about that.");
      Responses.add("That's not a bug, that's a feature.");
    }

    /**
     * By using the put(Key,Value) method, we add all the keywords and
     * their corresponding values to our response map.
     */
    private void fillResponseMap()
    {
     responseMap.put("slow", "I think it has something to do with your hardware.");
     responseMap.put("bug", "Well you know, all software has some bugs.");
     responseMap.put("expensive", "Our cost is quite competitive compared to other
software.");
    }

    private void fillResponseMap2(){
     responseMap2.put("how", "Try reinstalling the software. It should solve the issue.");
     responseMap2.put("why", "Because this is how our software is intended to work.");
     responseMap2.put("who", "You can speak to our manager regarding this matter.");
    }
}
```