Software Engineering Exercise 6

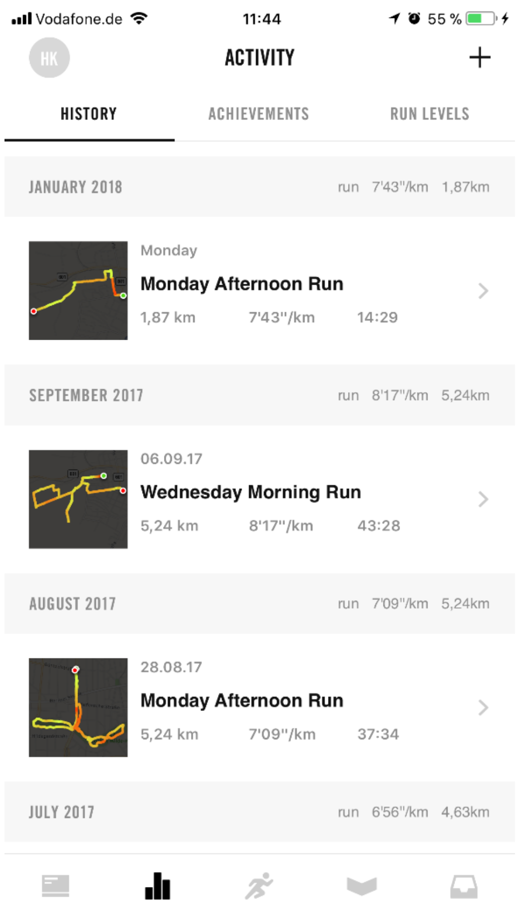
15 . 01 . 2018

Hiyeon Kim 118654
Evangelist Eirini Koktsidou 118884

Part1

1. Illustrated Choices

   a. Nike+ Run Club
   The application tracks the route and numeric details when jogging. In the HISTORY section, a user can check such data by specific dates, but also can preview the picture of jogging route on the map.


Vodafone.de 11:44 55 %

ACTIVITY

HISTORY ACHIEVEMENTS RUN LEVELS

JANUARY 2018 run 7'43''/km 1,87km

Monday
Monday Afternoon Run
1,87 km 7'43''/km 14:29

SEPTEMBER 2017 run 8'17''/km 5,24km

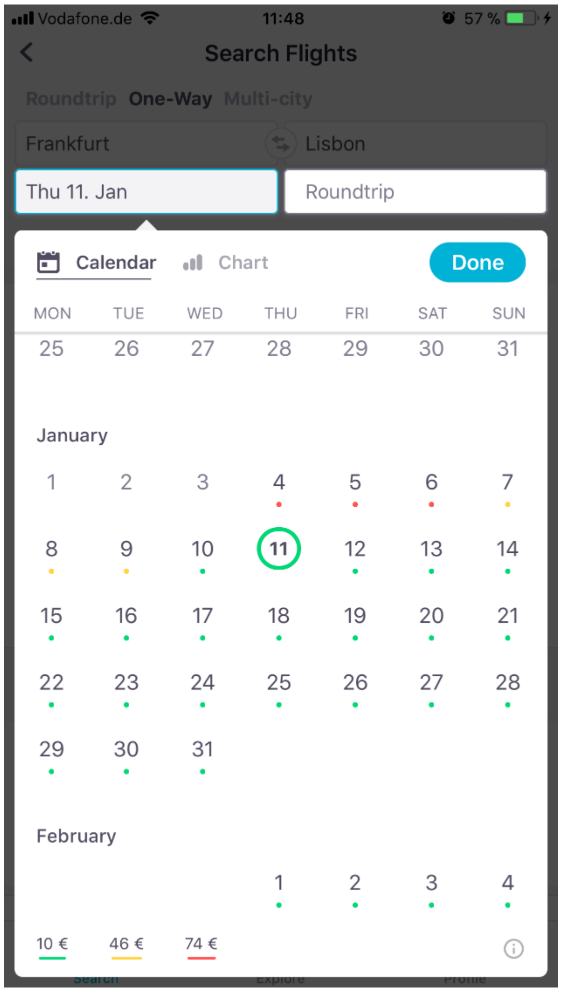
06.09.17
Wednesday Morning Run
5,24 km 8'17''/km 43:28

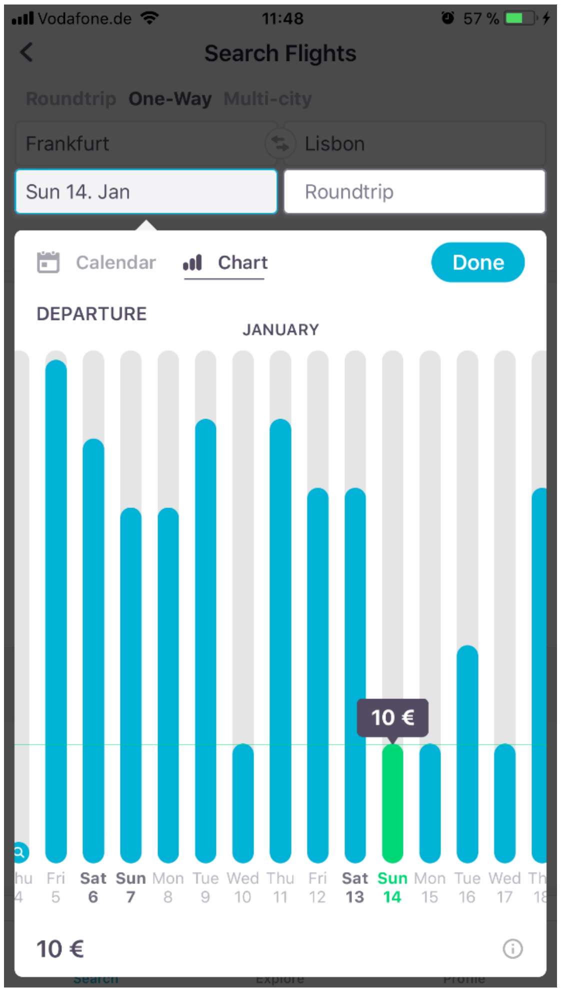
AUGUST 2017 run 7'09''/km 5,24km

28.08.17
Monday Afternoon Run
5,24 km 7'09''/km 37:34

JULY 2017 run 6'56''/km 4,63km

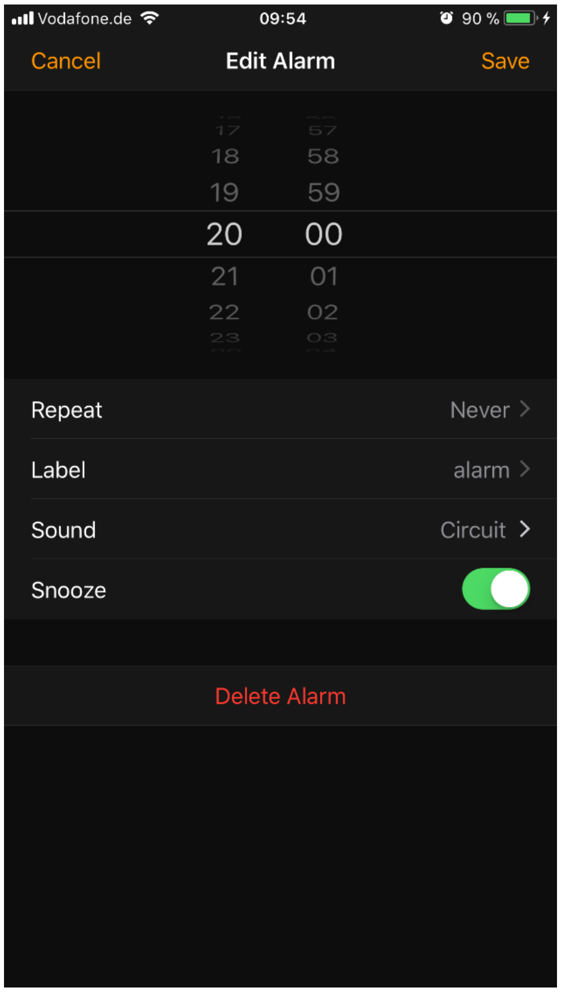
b. Skyscanner
This application helps search for flight tickets. To save time of waiting for price loadings each search, it provides color dots or bar charts indicating the lowest flight ticket when choosing the date.





2. Extras On Demands

   a. Alarm
      In an alarm application in most phones, there is an option for a user to choose a custom alarm sound, as well as some basic built-in sounds.

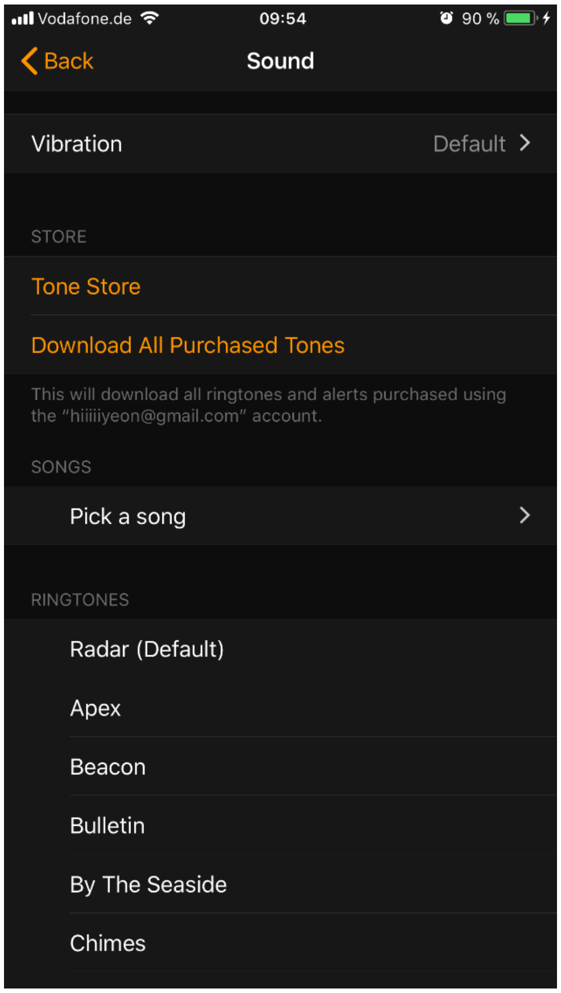

Vodafone.de 09:54 90 %

Cancel Edit Alarm Save

17 57
18 58
19 59
20 00
21 01
22 02
23 03

Repeat Never
Label alarm
Sound Circuit
Snooze

Delete Alarm


Vodafone.de 09:54 90 %

Back Sound

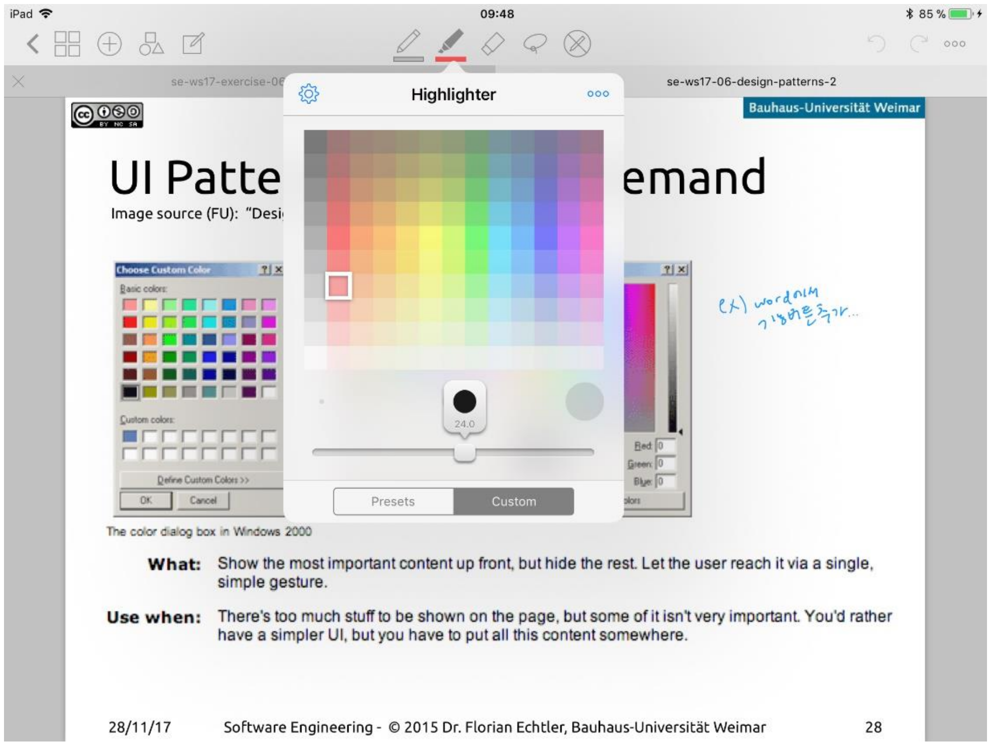
Vibration Default

STORE
Tone Store
Download All Purchased Tones
This will download all ringtones and alerts purchased using the "hiiiiiyeon@gmail.com" account.

SONGS
Pick a song

RINGTONES
Radar (Default)
Apex
Beacon
Bulletin
By The Seaside
Chimes

b. GoodNotes (Ipad)
It is a note-taking application for an Ipad. When using a pen or a highlighter, along with some default settings, users can add their own pen settings with different colors and sizes. This goes the same with the size of an eraser.


iPad 09:48 85 %

se-ws17-exercise-06 se-ws17-06-design-patterns-2

Highlighter

24.0

Presets Custom

Bauhaus-Universität Weimar

UI Patte… …emand

Image source (FU): "Desi…

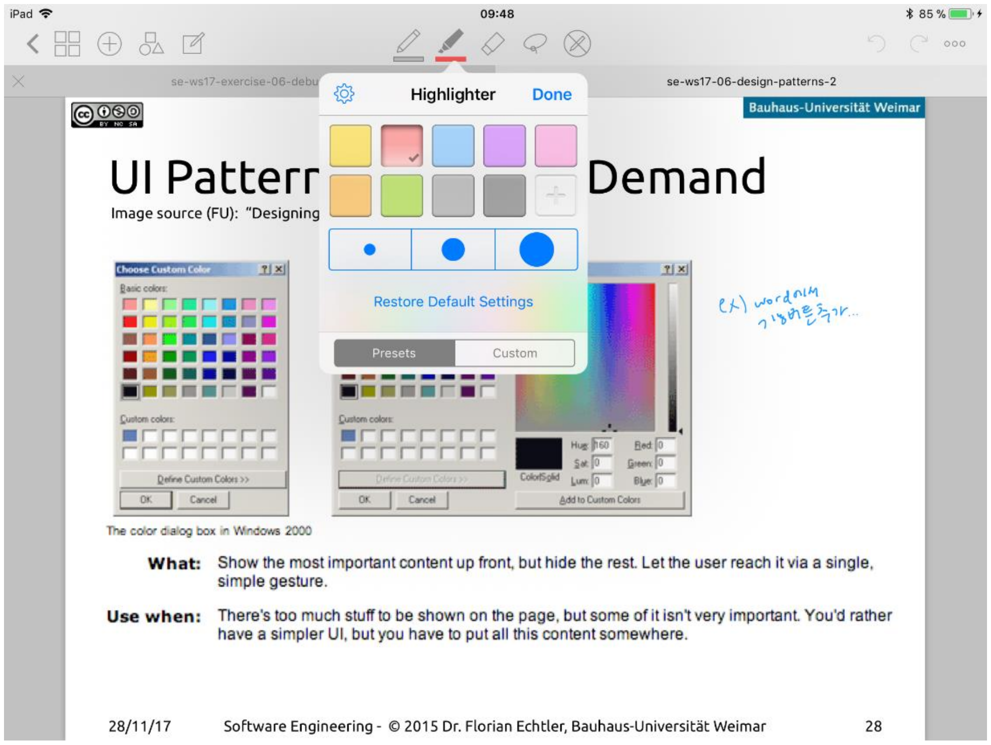
The color dialog box in Windows 2000

What: Show the most important content up front, but hide the rest. Let the user reach it via a single, simple gesture.

Use when: There's too much stuff to be shown on the page, but some of it isn't very important. You'd rather have a simpler UI, but you have to put all this content somewhere.

ex) word에서 기능을 추가...

28/11/17 Software Engineering - © 2015 Dr. Florian Echtler, Bauhaus-Universität Weimar 28


iPad 09:48 85 %

se-ws17-exercise-06-debu… se-ws17-06-design-patterns-2

Highlighter Done

Restore Default Settings

Presets Custom

Bauhaus-Universität Weimar

UI Patterr… Demand

Image source (FU): "Designing…

The color dialog box in Windows 2000

What: Show the most important content up front, but hide the rest. Let the user reach it via a single, simple gesture.

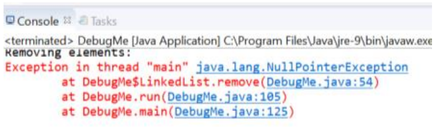
Use when: There's too much stuff to be shown on the page, but some of it isn't very important. You'd rather have a simpler UI, but you have to put all this content somewhere.

ex) word에서 기능을 추가...

28/11/17 Software Engineering - © 2015 Dr. Florian Echtler, Bauhaus-Universität Weimar 28

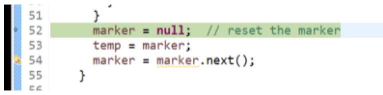
Part3

1. Method *remove* – NullPointerException
   When first running the program, *NullPointerException* occurs when removing the value 2.

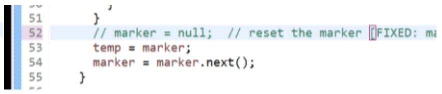


```
<terminated> DebugMe [Java Application] C:\Program Files\Java\jre-9\bin\javaw.exe
Removing elements:
Exception in thread "main" java.lang.NullPointerException
	at DebugMe$LinkedList.remove(DebugMe.java:54)
	at DebugMe.run(DebugMe.java:105)
	at DebugMe.main(DebugMe.java:125)
```

   By setting the breakpoint at the *remove* and stepping into the method, we found that the marker is set to null before its value should be used for *temp* and next *marker* value.



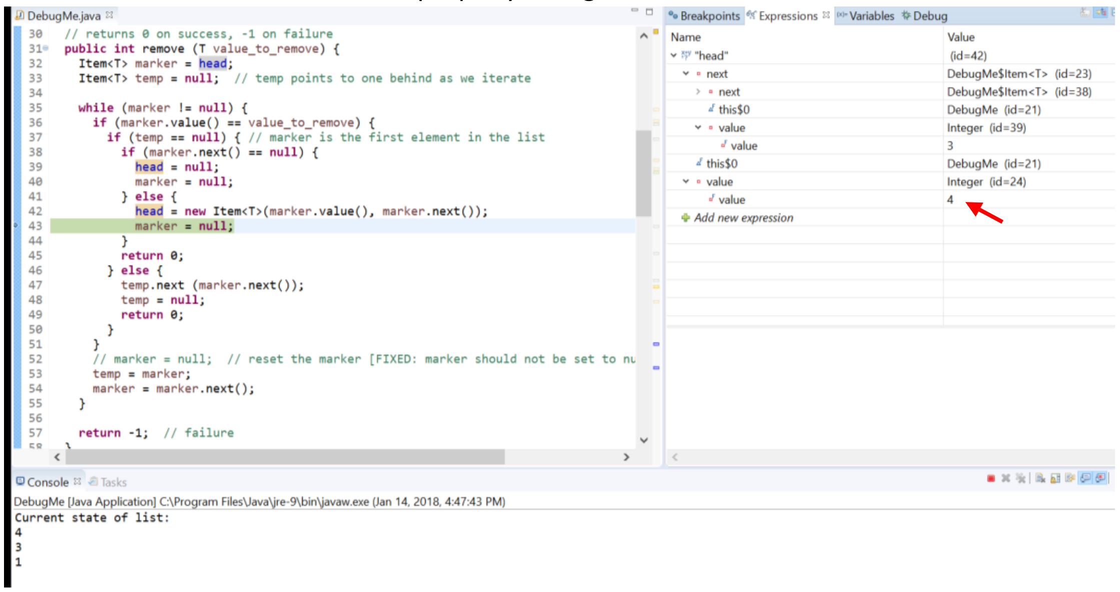
```
51        }
52        marker = null;  // reset the marker
53        temp = marker;
54        marker = marker.next();
55      }
```



```
51        }
52        // marker = null;  // reset the marker [FIXED: ma
53        temp = marker;
54        marker = marker.next();
55      }
```

   Deleting the line out, and running the next line (step over: printing the list) shows that the removal of 2 has been successful.

2. Method *remove* – header changing
   Removing the value 4 doesn't work as is shown with the *print* method. At line 43, the head value is still 4. The head value should be properly changed to the next value.

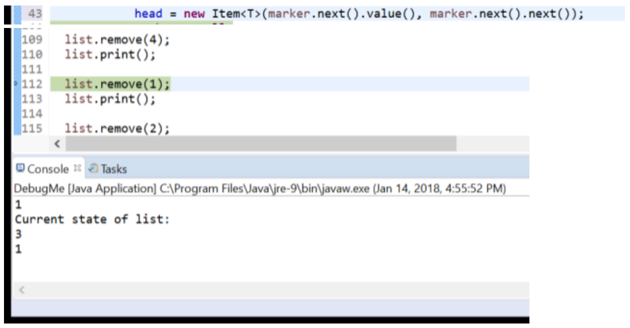


```
30    // returns 0 on success, -1 on failure
31    public int remove (T value_to_remove) {
32      Item<T> marker = head;
33      Item<T> temp = null;  // temp points to one behind as we iterate
34
35      while (marker != null) {
36        if (marker.value() == value_to_remove) {
37          if (temp == null) { // marker is the first element in the list
38            if (marker.next() == null) {
39              head = null;
40              marker = null;
41            } else {
42              head = new Item<T>(marker.value(), marker.next());
43              marker = null;
44            }
45            return 0;
46          } else {
47            temp.next (marker.next());
48            temp = null;
49            return 0;
50          }
51        }
52        // marker = null;  // reset the marker [FIXED: marker should not be set to nu
53        temp = marker;
54        marker = marker.next();
55      }
56
57      return -1;  // failure
```

Name	Value
"head"	(id=42)
next	DebugMe$Item<T> (id=23)
next	DebugMe$Item<T> (id=38)
this$0	DebugMe (id=21)
value	Integer (id=39)
value	3
this$0	DebugMe (id=21)
value	Integer (id=24)
value	4

```
DebugMe [Java Application] C:\Program Files\Java\jre-9\bin\javaw.exe (Jan 14, 2018, 4:47:43 PM)
Current state of list:
4
3
1
```

   Changing the line to the head's next value as such solves the problem.



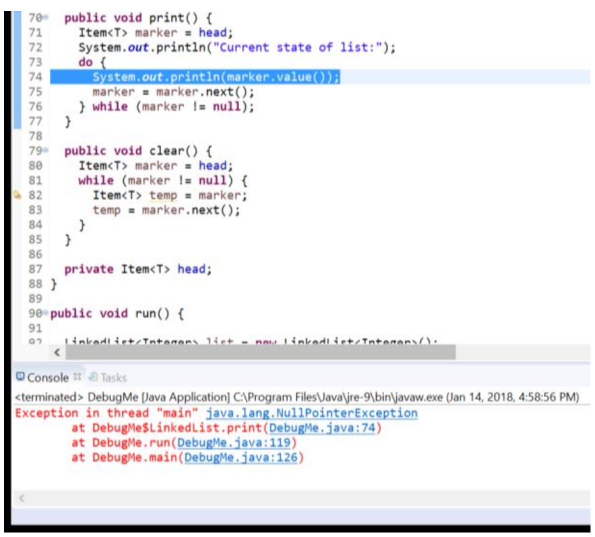
```
43              head = new Item<T>(marker.next().value(), marker.next().next());
```

```
109   list.remove(4);
110   list.print();
111
112   list.remove(1);
113   list.print();
114
115   list.remove(2);
```

```
DebugMe [Java Application] C:\Program Files\Java\jre-9\bin\javaw.exe (Jan 14, 2018, 4:55:52 PM)
1
Current state of list:
3
1
```

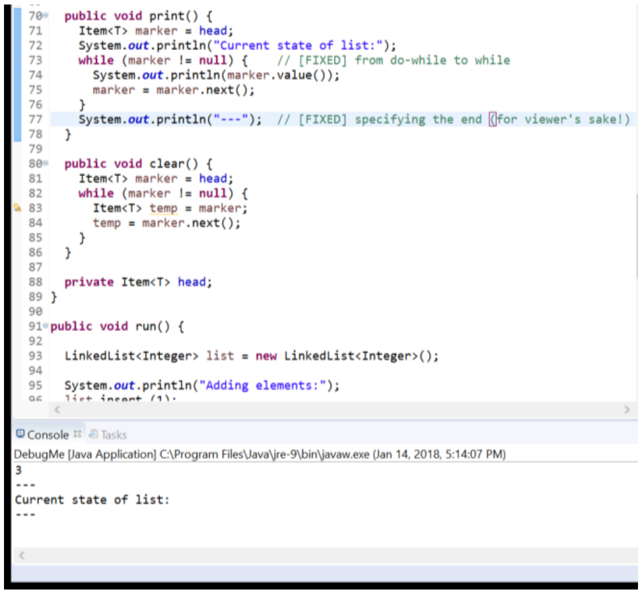
3. Method *print* – NullPointerException

Printing the list after removing 3 shows an error. By stepping into the method, we can see that the print() method is trying to print empty value, *marker.value()*. Fixing line 73 from do-while to while works, in case the list is empty. For viewer's sake, we specified the ending.



```
70   public void print() {
71     Item<T> marker = head;
72     System.out.println("Current state of list:");
73     do {
74       System.out.println(marker.value());
75       marker = marker.next();
76     } while (marker != null);
77   }
78
79   public void clear() {
80     Item<T> marker = head;
81     while (marker != null) {
82       Item<T> temp = marker;
83       temp = marker.next();
84     }
85   }
86
87   private Item<T> head;
88 }
89
90 public void run() {
91
92   LinkedList<Integer> list = new LinkedList<Integer>();
```

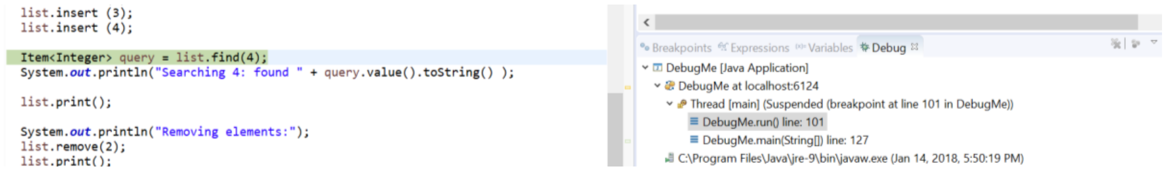
```
Console
<terminated> DebugMe [Java Application] C:\Program Files\Java\jre-9\bin\javaw.exe (Jan 14, 2018, 4:58:56 PM)
Exception in thread "main" java.lang.NullPointerException
	at DebugMe$LinkedList.print(DebugMe.java:74)
	at DebugMe.run(DebugMe.java:119)
	at DebugMe.main(DebugMe.java:126)
```



```
70   public void print() {
71     Item<T> marker = head;
72     System.out.println("Current state of list:");
73     while (marker != null) {    // [FIXED] from do-while to while
74       System.out.println(marker.value());
75       marker = marker.next();
76     }
77     System.out.println("---");  // [FIXED] specifying the end (for viewer's sake!)
78   }
79
80   public void clear() {
81     Item<T> marker = head;
82     while (marker != null) {
83       Item<T> temp = marker;
84       temp = marker.next();
85     }
86   }
87
88   private Item<T> head;
89 }
90
91 public void run() {
92
93   LinkedList<Integer> list = new LinkedList<Integer>();
94
95   System.out.println("Adding elements:");
96   list.insert (1);
```

```
Console
DebugMe [Java Application] C:\Program Files\Java\jre-9\bin\javaw.exe (Jan 14, 2018, 5:14:07 PM)
3
---
Current state of list:
---
```

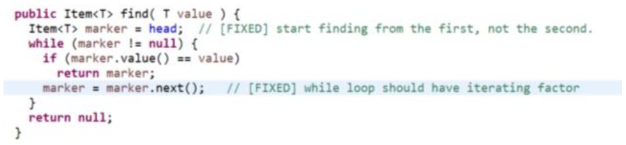
4. Method *find* – marker iteration
   When the finding value at line 101 changes from 3 to 4, the program does not proceed.



```
  list.insert (3);
  list.insert (4);

  Item<Integer> query = list.find(4);
  System.out.println("Searching 4: found " + query.value().toString() );

  list.print();

  System.out.println("Removing elements:");
  list.remove(2);
  list.print();
```

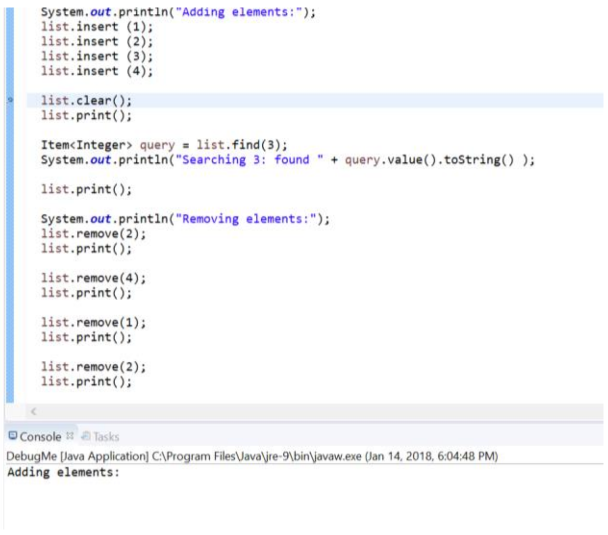
```
Breakpoints  Expressions  Variables  Debug
DebugMe [Java Application]
  DebugMe at localhost:6124
    Thread [main] (Suspended (breakpoint at line 101 in DebugMe))
      DebugMe.run() line: 101
      DebugMe.main(String[]) line: 127
  C:\Program Files\Java\jre-9\bin\javaw.exe (Jan 14, 2018, 5:50:19 PM)
```

Stepping into the find method, two problems can be found. First, marker should start from the beginning, not the second item in the list. Also, while loop doesn't have the iterating factor. For this reason, value 1 and 2 also cannot be found.



```
public Item<T> find( T value ) {
  Item<T> marker = head;  // [FIXED] start finding from the first, not the second.
  while (marker != null) {
    if (marker.value() == value)
      return marker;
    marker = marker.next();   // [FIXED] while loop should have iterating factor
  }
  return null;
}
```

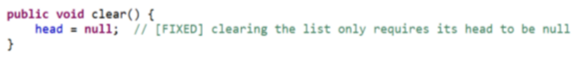
5. Method *clear*
   When testing the *clear* method when the list is not empty, the program does not proceed as well.



```
System.out.println("Adding elements:");
list.insert (1);
list.insert (2);
list.insert (3);
list.insert (4);

list.clear();
list.print();

Item<Integer> query = list.find(3);
System.out.println("Searching 3: found " + query.value().toString() );

list.print();

System.out.println("Removing elements:");
list.remove(2);
list.print();

list.remove(4);
list.print();

list.remove(1);
list.print();

list.remove(2);
list.print();
```

```
DebugMe [Java Application] C:\Program Files\Java\jre-9\bin\javaw.exe (Jan 14, 2018, 6:04:48 PM)
Adding elements:
```

By simply fixing the code as such solves the problem.



```
public void clear() {
    head = null;  // [FIXED] clearing the list only requires its head to be null
}
```

Final Code

Fixed parts are highlighted with black.

```
public class DebugMe {

class Item<T> {
  public Item( T _value, Item<T> _next ) {
    value = _value;
    next = _next;
  }

  public Item<T> next() { return next; }
  public void next(Item<T> _next) { next = _next; }
  public T value() { return value; }
  public void value(T _value) { value = _value; }

  private T value;
  private Item<T> next;
}

class LinkedList<T> {

  public LinkedList() { head = null; }

  // returns 0 on success, -1 on failure
  public int insert(T new_value) {
    head = new Item<T>(new_value, head);
    return (head != null) ? 0 : -1;
```

```
}

// returns 0 on success, -1 on failure
public int remove (T value_to_remove) {
  Item<T> marker = head;
  Item<T> temp = null;  // temp points to one behind as we iterate

  while (marker != null) {
    if (marker.value() == value_to_remove) {
      if (temp == null) { // marker is the first element in the list
        if (marker.next() == null) {
          head = null;
          marker = null;
        } else {
          // [FIXED] the value should be changed to the next one
          head = new Item<T>(marker.next().value(), marker.next().next());
          marker = null;
        }
        return 0;
      } else {
        temp.next (marker.next());
        temp = null;
        return 0;
      }
    }
    // [FIXED] marker should not be set to null before being used
    temp = marker;
    marker = marker.next();
  }

  return -1;  // failure
}

public Item<T> find( T value ) {
  Item<T> marker = head;  // [FIXED] start finding from the first, not the second.
  while (marker != null) {
    if (marker.value() == value)
      return marker;
    marker = marker.next(); // [FIXED] while loop should have iterating factor
  }
  return null;
}

public void print() {
  Item<T> marker = head;
```

```
    System.out.println("Current state of list:");
    while (marker != null) {  // [FIXED] from do-while to while
      System.out.println(marker.value());
      marker = marker.next();
    }
    System.out.println("---");  // [FIXED] specifying the end (for viewer's sake!)
  }

  public void clear() {
    head = null;  // [FIXED] clearing the list only requires its head to be null
  }

  private Item<T> head;
}

public void run() {

  LinkedList<Integer> list = new LinkedList<Integer>();

  System.out.println("Adding elements:");
  list.insert (1);
  list.insert (2);
  list.insert (3);
  list.insert (4);

  Item<Integer> query = list.find(3);
  System.out.println("Searching 3: found " + query.value().toString() );

  list.print();

  System.out.println("Removing elements:");
  list.remove(2);
  list.print();

  list.remove(4);
  list.print();

  list.remove(1);
  list.print();

  list.remove(2);
  list.print();

  list.remove(3);
  list.print();
  list.clear();
}

public static void main(String args[]) {
  DebugMe dbm = new DebugMe();
  dbm.run();
}


}
```