Solutions to questions of lecture 7
by Kasper B. Graversen & Kasper Østerbye

version 1 - 8/03-04
version 2 - 28/04-05
---

ex 1.1
substring has the following

preconditions (conjugated):
- beginindex >= 0 and
- endindex >= beginindex
- endindex <= string length

postconditions:
return is string beginning at beginindex, upto, but not including endindex
original string is unchanged

This means that the normal case where you want to extract the substring starting at 3 and ending at 5 m
ust be extracted as "myString.substring(3,5+1)".
A call to extract the one letter substring at position 3, must be done using the call "myString.substri
ng(3,4)".
The call "myString.substring(3,3)" is legal, but return the empty string.
The call "myString.substring(3, myString.length()) return the substring starting at index 3 and includ
es all characters with index above 3.

ex 1.2
Yes it is indeed a good idea. The focus of the paper is that not both the client and the supplier shoul
d check, but only one of them. Although there is no exact rule other than to achieve the simplest archi
tecture (p. 5). personally I find it advantegous to have the check in the supplier, as it then is "reus
ed" when other clients uses it. As noted on page 7, assertion violations are not special cases but are
the result of bugs, which this exception certainly is trying to prevent.

1.3

from the javadoc

true if the character sequence represented by the argument is a prefix of the character sequence repres
ented by this string; false otherwise. Note also that true will be returned if the argument is an empty
 string or is equal to this String object as determined by the equals(Object) method.

```
public boolean startsWith(String prefix) {
   if(prefix == null) throw new NullPointerException(); // throw exception if pre condition not OK.
   if (prefix.length()>this.length()) return false; // a prefix must be shorter
   for(int i = 0; i<prefix.length();i++)
      if (this.charAt(i) != prefix.charAt(i) )
         return false; // each character in prefix must be in this. if not, return false.
   return true; // all charactes in the prefix was in this.
}
```

preconditions:
- prefix string != null

postconditions
- true if prefix.equals(this.substring(0,prefix.length())

(The postcondition shows that the prefix method could be written using the equals and substring methods
.)

2.1
The following method works like this:
First it skips any non-letters to make sure we are at the beginning of a letter
Then we loop, at the beginning of the loop we print out the word that starts there
at the end of the loop, we make sure to skip any leading non-letters, so we are ready
for next round in the loop.

```
void toWords(){
   int i = 0;
   final int N = greet.length();
   // skip leading non letters
   while (i<N && !Character.isLetter(greet.charAt(i)))
      i++;
   while (i<N){
      // print the word
      while (i<N && Character.isLetter(greet.charAt(i))){
         System.out.print(greet.charAt(i));
         i++;
```

```
      }
      System.out.println();
      // skip non letters
      while (i<N && !Character.isLetter(greet.charAt(i))){
         i++;
      }
   }
}
```

2.2

```
class GreetIterator implements Iterator<String>{
   // Exercise 2.3
   // inv: if nextWord.size() = 0, the iterator is empty
   //      else nextWord is the head.
   //      i is either N or the first unprocessed letter in str
   //      N is the length of str
   final String str; // the string we are iterating into words
   final int N;
   int i;
   String nextWord;

   GreetIterator(String str){
      this.str= str;
      i=0;
      N = str.length();
      findNext();
   }

   /* pre: i is first non-processed letter in str or N
      post: if hasNext is true then nextWord is head.
       i is first non-processed letter in str or N
   */
   private void findNext(){
      // skip non letters
      while( i<N && !Character.isLetter(str.charAt(i)) )
         i++;
      // find the word
      int start = i;
      while( i<N && Character.isLetter(str.charAt(i)) )
         i++;
      nextWord = str.substring(start,i);
   }

   public boolean hasNext(){
      return nextWord.length()>0;
   }

   public String next(){
      String ret = nextWord;
      findNext();
      return ret;
   }
}
```

2.4

According to the invariant, i is either str.length or the first unprocessed letter in str.
N is the length of the str, and nextWord is the head of the iterator.
Therefore the pre-condition for the private method findNext is fulfilled.
The post condition of findNext is that the invariant is fullfilled.

2.5

To implement the Iterable<String> interface, you need to declare that in the header of the Haddock clas
s, and then implement the iterator method (see the Java API, it is the only method in that interface).
All we need to do is to create an instance of the greet iterator from 2.3.

```
public class Haddock implements Iterable<String>{
    private String greet
        = "You #!@ Miserable earthworms! Numbskulls! and Carpet-sellers!";


   public Iterator<String> iterator(){
       return new GreetIterator(greet);
   }
}
```

Reg. 2.3 and 2.5, the greet iterator can be implemented as an inner class in Haddock. If we do that, we
 can use the greet string directly in the greet iterator, and need not set it up in the constructor. Th
is solution is not as general, but avoids passing parameters to the constructor.

2.6

Just do it.