Questions and Exercises

These questions and exercises is an opportunity to see what you've learnt from the lecture as well as practice the new things we've been talking about. In other words, these questions and exercises are completely optional but it's recomended to do them. In the end of the document you will find the answers to the questions as well as possible solutions to the exercises, note that one can solve an exercise in different ways. There will also be some suggestions about what one could code if one want to continue with some more advanced things. These suggestions will not come with a possible solution and might include things that haven't been covered in the lecture.

Question 1

What is overloading of methods? And what's the requirement for it to work?

Question 2

The following code is supposed to print out multiple strings on the screen and also convert them to uppercase if the boolean parameter is true.

```
static void printStrings(String... strings, boolean upperCase) {
        for (String str : strings) {
                if (upperCase) {
                        System.out.println(str.toUpperCase());
                }else{
                        System.out.println(str);
                }
        }
}
```

Question 3

Recursion is when a method is calling itself. What's important when doing recursion to make it all work?

Question 4

What would the following code print out on the screen? Answer the question before you run the code.

```
public static void main(String[] args) {
        int myVariable = 2;
        myMethod(myVariable);
        System.out.println(myVariable);
}

static void myMethod(int myParameter) {
        myParameter += 10;
        System.out.println(myParameter);
}
```

Exercise 1
Write a method that you can call with two or more strings and one boolean value. Print out the strings on the screen, if the boolean value is true print them out in the opposite order instead((don't reverse the strings themselves). Write a simple test program to see that your method works.

Exercise 2
The first fibonacci numbers are 0, 1, 1, 2, 3, 5, 8... To calculate the next one just calculate the sum of 5 and 8 which is 13. The following one is the sum of 8 and 13 which is 21 etc, Write an iterative method that calculates the N:th fibonacci number. Then write a recursive one. Which one is best?

For example, the 4th fibonacci number is 2, and the 8th is 13.

Further explorations
This is the last lecture in this course, do the assignment to continue.

Answers and solutions

Answer to Question 1
Overloading of methods are when you have multiple methods with the same name. The requirement for this to work is that the different methods have a different parameter list. In other words, the parameters have to be different. Then one can call the different methods by giving them the values that match their parameter list when calling them. The return types of overloaded methods can be different. However, the return types themselves are not enough for overloading, the parameter lists have to be different.

Answer to Question 2
When using VarArgs one can not have any parameters after it. You can however have as many parameters as you want it front of the VarArgs. So the code that wasn't working can be rewritten as the following

```
static void printStrings(boolean upperCase, String... strings) {
        for (String str : strings) {
                if (upperCase) {
                        System.out.println(str.toUpperCase());
                }else{
                        System.out.println(str);
                }
        }
}
```

Answer to Question 3
When using recusion it's very important to have a base case that allows the recursion to stop. If the

method always calls itself it will continue with that forever and it will never be done. An example of recursion can be seen below

```
static String reverse(String str) {
        if (str.length() == 0) {
                return "";
        }else{
                return reverse(str.substring(1)) + str.charAt(0);
        }
}
```

Answer to Question 4
The given code will print out the following

```
12
2
```

When calling a method with parameters you will send them by value. This means that there's no reference back to the variable that is used to call the method with. If the parameter itself is changed it won't affect anything outside the method.

Possible solution to Exercise 1

```
public class Exercise1 {
  public static void main(String[] args) {
    printStrings(false, "Hello", "World");
    printStrings(false, "Just", "Testing", "My", "Method");
    printStrings(true, "This should", "be in the reversed", "order");
  }

  static void printStrings(boolean reversedOrder, String string1, String string2, String... strings) {
    if (reversedOrder) {
      for (int i = strings.length - 1; i >= 0; i--) {
        System.out.println(strings[i]);
      }
      System.out.println(string2);
      System.out.println(string1);
    }else{
      System.out.println(string1);
      System.out.println(string2);
      for (int i = 0; i < strings.length; i++) {
        System.out.println(strings[i]);
      }
    }

  }

}
```

Possible solution to Exercise 2

```
/**
 * When calculating fibonacci numbers the iterative solution is much better.
 * The reason is that in the iterative version we have the two last values stored
 * and can therefore just add them together. In the recursive one we're calculating
 * the old numbers old the time. For instance the 45th fibonacci number is the
 * 44th + the 43rd while the 44th is the 43rd + the 42nd. This means that we will have
 * to calculate the 43rd fibonacci number twice, once when calculating the 45th and one
 * when calulating the 44th. Depending on your comptuer the recursive method might take
 * a lot longer than the itertive one when calculating the 45th fibonacci number.
 **/

public class Exercice2 {
   public static void main(String[] args) {
      System.out.println(getFibonacciIterative(4));
      System.out.println(getFibonacciRecursive(4));
      System.out.println(getFibonacciIterative(8));
      System.out.println(getFibonacciRecursive(8));
      System.out.println(getFibonacciIterative(45));
      System.out.println(getFibonacciRecursive(45));
   }

   static int getFibonacciIterative(int n) {
      if (n <= 1) {
         return 0;
      }else{
         int last = 0;
         int current = 1;
         for (int i = 2; i < n; i++) {
            //calculate the next one
            int next = current + last;

            //move the values 'forward'
            last = current;
            current = next;
         }
         return current;
      }
   }

   static int getFibonacciRecursive(int n) {
      if (n <= 1) {
         return 0;
      }else if(n == 2) {
         return 1;
```

```
    }else{
      //add the two previous ones together
      return getFibonacciRecursive(n - 1) + getFibonacciRecursive(n - 2);
    }
  }

}
```