# Demo 10 / 17.11

## Tasks

1. **Ville:** Run Ville web application (look: https://trac.cc.jyu.fi/projects/ohj1/wiki/villeEn) and carry out task round: Sorting algorithms. Scale the amount of tasks done to the range [1,0] and round the point you get with precision of 0.2. (An example: if you did 120/195 tasks (=0.615) mark 0.6 as you final point. If 140/195 (=0.72) mark 0.8.)

2. **Integer Lists**: Write a lottery program that randomly chooses 7 numbers and 3 additional numbers from 39. Hint:

```
public static void main(String[] args) {
     List<Integer> balls = new ArrayList<Integer>();
     // fill balls with numbers 1-39, example: balls.add(4);
     // scrable balls (see Collections)
     // print 7 first "balls"
     // print the 3 additional balls
}
```

3. **Loops**: The number of the 7 different lottery numbers can be calculated with the following formula:

   See: http://en.wikipedia.org/wiki/Combination

```
      formula 1    formula 2             formula 3

 39           39!            33*34*35*36*37*38*39
(   )  = -------------  =  ---------------------- = 15380937
  7       7! * (39-7)!           1*2*3*4*5*6*7
```

   The result can be calculated using `long` type numbers. `Write a function`

```
long nOverK(int n,int k)
```

   that gives the aforementioned result with tha call `nYliK(39,7)`. The result cannot be calculated from the middle formula 2 because `39!` would greatly exceed the numberical capacity of even the `long` type.

4. **Lists:** Even with the previous formula (3) the capacity of the `long` type may be exceeded if one tries to use much larger numbers (the limit is now 515). One way of handling this is simplifying the formula as much as possible before the calculation. First simplify formula 3 by hand as much as you can. Think about what you did. In the file Amount.java you have a draft how to make the calculation if both the numerator and the denominator are kept as list of multipliers. Carry out the methods `fill`, `multiply` and `remove`. The methods have to function as written in the tests in the comments.

5. **Lists:** Implement the following method to the previous

```
int cancel(List<Integer> numerator, List<Integer> denominator, int
cancel)
```

so that it does as the documentation/commentation explains. Like this up to

```
nOverK(1733,7) = 9202167919706100768L
```

can be calculated.

6. **Exceptions:** Write using `Double.parseDouble` and exceptions a function `convertString(String s,double assumption)` that can be used as follows:

```
public static void main(String args[]) {
    double d1 = convertString("12.3",0.0);
    double d2 = convertString("12.3e",0.0);
    System.out.printf("%5.2f %5.2f",d1,d2); // 12.30 0.00
}
```

B1. Examine how the `BigInteger` class works in Java and write `nOverK(int n,int k)` using it.

# GURU-tasks

G1. Figure out how to make `nOverK(int n,int k)` ( henceforth known as `C(n,k)` ) work recursively, so:

```
C(n+1,k) = C(n,k) * f(n,k)  //think of what the expression of f(n,k) is
C(k,k)   = 1
```

Then using this write an iterative (a loop, not recursion) based solution where the multiplications don't ”easily” exceed the capacity. With this algorithm we can get to results `< Long.MAX_VALUE/n`. With formula 3 we got to results`< Long.MAX_VALUE/(k!)`.

G2. Implement and test the following methods to the class `Amount.java` :

```
public static int simplify(List<Integer> numerator, List<Integer>
denominator)
public static int cancel(List<Integer> numerator, List<Integer>
denominator)
```