Assignment Scala 1

Erik Myhre, Anders Hagen Ottersland, Nicolai Molstad

Task 1

a)

```scala
def a(): Array[Int] = {
    val array = new Array[Int](50)
    for (i <- 1 to 50){
        array(i - 1) = i
    }
    return array
}
```

b)

```scala
def b(list: Array[Int]): Int = {
    var sum = 0
    for (i <- list) {
        sum += i
    }

    return sum
}
```

c)

```scala
def c(list: Array[Int]): Int = {
    if (list.length == 0) 0
    else list.head + c(list.tail)
}
```

d)

```scala
def fibonacci(number: Int): BigInt = {
    if (number <= 2) 1
    else fibonacci(number - 1) + fibonacci(number - 2)
}
```

The difference between Int and BigInt is the number of bits a number can be stored ass, 4 bytes vs 8 bytes.

Task 2

a)

```
def makeThread(function: => Unit): Thread = {
    val thread = new Thread {
        override def run = function
    }

    thread
}
```

b)

```
def printCounter(): Unit = counter.synchronized {
    println(counter)
}
```

The run of three threads

```
val threads = Array[Thread](
    makeThread(increaseCounter()),
    makeThread(increaseCounter()),
    makeThread(printCounter()),
)

for (i <- threads) {
    i.start()
}

for (i <- threads) {
    i.join()
}
```

The it should print 2 every time the code is executed. However because of raceconditions we cannot ensure that this happens every time. When different threads are running with the same data, raceconditions can make it so that a later thread coud access the data before anther thread.

This could be problematic if there are threads dependant on eachoter, for instance in a banking system.

c)

```
private var counter: AtomicInteger = AtomicInteger(0)

def increaseCounter(): Unit = counter.synchronized {
    counter.getAndAdd(1)
}
```

d)

Deadlock is if a program waits for something to happen, but because of the structure never happens.

The best way of preventing deadlock is by eleminating one of the four deadlock conditions:

- Mutual exclution
- Hold and Wait
- No preemtion
- Circular wait

```
object DeadLock extends App { self =>
    lazy val x: Int = {
        val thread = new Thread (){
            override def run() = self.synchronized {}
        }
        thread.start();
        thread.join();
        1
    }

    println(DeadLock.x);
}
```