Instructions

- You have 90 minutes from the start of the exam.
- This exam has four problems, each with parts a and b.
- You can use any books, files, or web sites that existed before the start of the exam.
- You may NOT communicate with anyone during the exam.
- You can use your Scala IDE, codecheck, or both, or neither.
- You can turn in IDE worksheets, codecheck zips, or paper, or any mixture thereof.
- Email your electronic submissions to borran.fatemeh@heig-vd.ch **and copy yourself**, so you have a proof that you sent the email on time.

Divisors

You are given the function

```
def divisors(n: Int) = (1 to n).filter(n % _ == 0).toSet
```

that returns the set of divisors for a given integer.

a. **Using `flatMap`**, write a function that, for a set of integers, yields the union of the divisors of the elements (in other words, the numbers that divide at least one of the elements). For example,
`divisorsOfAny(Set(10, 12, 18))`
is
`Set(1, 2, 3, 4, 5, 6, 9, 10, 12, 18)`

```
def divisorsOfAny(numbers: Set[Int]) : Set[Int] = ... flatMap ...
```

http://cs14.cs.sjsu.edu:8080/codecheck/files?repo=heigvdcs1&problem=divisors

b. **Using `foldLeft`**, write a function that, for a set of integers, yields the intersection of the divisors of the elements (in other words, the numbers that divide all of the elements). For example,
`divisorsOfAll(Set(10, 12, 18))`
is
`Set(1, 2)`

```
def divisorsOfAll(numbers: Set[Int]) : Set[Int] = ... foldLeft ...
```

http://cs14.cs.sjsu.edu:8080/codecheck/files?repo=heigvdcs1&problem=divisors2

Structured Text

Structured text (such as HTML or XML) is made up of nodes that are either plain text or elements. Each element has a tag and a list of child nodes. (For simplicity, we ignore attributes.) Plain text just has a string contents.

Provide an abstract class `Node` and case classes `Text`, `Elem`, so that one can one define structured text like this:

```
val sample = Elem("body", List(
  Text("Goodbye "),
  Elem("b", List(Text("cruel"))),
  Text(" World")))
```

a. Define a method `toXML` of the `Node` class that renders a node in XML, by surrounding element content with <tag>...</tag>. Text content is included as is. (For simplicity, don't escape < characters.)
   For example, `sample.toXML` is `<body>Goodbye <b>cruel</b> World</body>`. Use pattern matching.

   http://cs14.cs.sjsu.edu:8080/codecheck/files?repo=heigvdcs1&problem=structuredText

b. Repeat this using inheritance. Define a method `toXML2` that yields the same result. It should be an abstract method of `Node`.

   http://cs14.cs.sjsu.edu:8080/codecheck/files?repo=heigvdcs1&problem=structuredText2

Do This And Maybe That

a) Write a function `doThisAndMaybeThat` that has three parameters, each a function. The first and third function have type `String => String`, and the second function has type `String => Boolean`. Return a function that, when given a string, applies `f1`. If the result of `f1`, when passed to `f2`, yields true, then `f3` is applied to the result of `f1`, and that result is returned. Otherwise, the result of `f1` is returned.

For example,

```
val f = doThisAndMaybeThat(s => s.replace("a", ""), s => s.length <= 5,
     s => s.toUpperCase)

f("macadamia") is "MCDMI"
f("mustard") is "mustrd"
```

http://cs14.cs.sjsu.edu:8080/codecheck/files?repo=heigvdcs1&problem=doThisAndMaybeThat

b) Write `genericDoThisAndMaybeThat` as a generic function with three type parameters. Note that the result types of the first and third function need not be the same. The result type of `f2` remains `Boolean`. Provide an appropriate constraint so that the result type of the function returned by `genericDoThisAndMaybeThat` is the same as the result type of `f1`. Use Currying so that the parameter types of `f2` and `f3` can be inferred.

http://cs14.cs.sjsu.edu:8080/codecheck/files?repo=heigvdcs1&problem=doThisAndMaybeThat2

Grades Database

Consider this database of students' grades:

```
case class Student(studentId: Int, name: String, firstname: String)
case class Grade(studentId: Int, courseId: String, grade: Int)

val students :Set[Student] =  Set(
 Student(1, "Pit", "Brad"),
 Student(2, "Cage", "Nicolas"),
 Student(3, "Winslet", "Kate"),
 Student(4, "Hanks", "Tom"),
 Student(5, "Dicaprio", "Leonardo"),
 Student(6, "Portman", "Nathalie"),
 Student(7, "Kidman", "Nicole")
)

val grades :Set[Grade] = Set(
 Grade(1, "SCALA", 5),
 Grade(2, "SCALA", 4),
 Grade(3, "SCALA", 6),
 Grade(4, "SCALA", 3),
 Grade(6, "SCALA", 1),
 Grade(1, "POO", 4),
 Grade(2, "POO", 6),
 Grade(3, "POO", 3 ),
 Grade(4, "POO", 4),
 Grade(7, "POO", 4)
)
```

a. **Using a `for` expression**, write a query on our database: "For each student that has taken a course, give the student's name, the course ID, and the corresponding grade."

   http://cs14.cs.sjsu.edu:8080/codecheck/files?repo=heigvdcs1&problem=grades

b. **Using the `groupBy` method and another `for` expression**, transform the result of the first query into a map associating each course ID with a set of names of the students in that course.

   http://cs14.cs.sjsu.edu:8080/codecheck/files?repo=heigvdcs1&problem=grades2

Expected results:
Part a) Set[(String, String, Int)] = Set((Cage, SCALA,4), (Kidman,POO,4), (Portman,SCALA,1), (Cage,POO,6), (Pitt,POO,4), (Hanks,SCALA,3), (Winslet,POO,3), (Pitt,SCALA,5), (Winslet,SCALA,6), (Hanks,POO ,4)

Part b) Map(SCALA -> Set(Portman, Winslet, Pitt, Hanks, Cage), POO -> Set(Winslet, Kidman, Pitt, Hanks, Cage)