Exercise sheet for lecture 11—Diverses

1 Modelling a wallet (old exam question)

The following program has several problems. Point out 4 things, that are in contradiction to the principles presented in the lecture and give possible solutions for each.

```scala
final case class Wallet(
  amountMoney: Int,
  numberOfDocuments: Int,
)
// --- later ---
def transformWallet(w: Wallet): Wallet =
  if w == null then
    changeWallet(w)
  else
    throw new RuntimeException("no wallet");
```

2 Modelling a customer (old exam question)

a) Model a type `Customer`, that describes the following business logic as accurately as possible: a customer is either a private or a business customer. Both have a name. Business customers are required to have at least one phone number. A private customer on the other hand is only allowed to have at most one phone number. Phone numbers and names are stored as `String`.

Model the “at least one” requirement as its own type and use it.

b) What problem can occur when using the types specified in the business logic and how would you solve it?

3 Recap: Parametricity (old exam question)

Given the following function signature:

```scala
def p2[A,B,C,D](a: A, b: B)(f: (A, B) => C, g: (A, C) => D): D
```

a) Give an implementation for the function based only on the types, which returns a valid value.

b) Why is the signature sufficient here to make assertions about the function’s behaviour, as long as the implementation behaves referentially transparent (i.e. doesn’t throw exceptions etc.)?

4 Recap: Lazy Evaluation (old exam exercise)

a) Explain the difference in evaluation for strict parameters, by-name parameters and `lazy vals`.

b) The fold methods for List always iterate through the the whole list. In some cases this leads to unneccessary calculations, e.g. when multiplying ints we could stop when one is a

0. Change the following implementation of `List.foldRight`, such that early stopping the iteration is possible:

```scala
def foldRight[B](z: B)(f: (A, B) => B): B = this match {
  case Cons(x, xs) => f(x, xs.foldRight(z)(f))
  case Nil => z
}
```

c) Give a call for this `foldRight` method, which multiplies a list of `Int`s and directly returns 0 when encountering a 0, without iterating through the rest of the list.