Übungsblatt zu Vorlesung 11— Diverses

1 Modellierung einer Geldbörse (Altklausuraufgabe)

Im folgenden Programm gibt es einige Probleme. Nennen Sie 4 Stellen, die im Gegensatz zu in dieser Vorlesung gelernten Prinzipien stehen und nennen Sie jeweils eine mögliche Lösung:

```
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 Modellierung eines Kunden (Altklausuraufgabe)

a) Modellieren sie einen Typ `Kunde`, der die folgende Businesslogik möglichst genau abbildet: Ein Kunde ist entweder ein privater oder ein geschäftlicher Kunde. Beide haben einen Namen. Ein Geschäftskunde muss allerdings mindestens eine Telefonnummer angegeben haben. Ein Privatkunde hingegen darf maximal eine Telefonnummer angeben. Telefonnummern und Namen werden als `String` gespeichert.

Modellieren Sie das „mindestens eins“ als eigenen Typ und benutzen Sie diesen dann.

b) Welches Problem kann bei der Verwendung der in der Businesslogik verwendeten Typen auftretenund wie würden Sie dieses lösen?

3 Wiederholung: Parametrizität (Altklausuraufgabe)

Gegeben sei folgende Funktionssignatur:

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

a) Geben Sie nur anhand der Typen eine Implementation der Funktion an, die einen gültigen Wert zurückgibt.

b) Warum reicht hier die Signatur aus, um Aussagen über das Verhalten zu treffen, solange sich die Implementation referentiell transparent verhält (also keine Exception wirft o.ä.)?

4 Wiederholung: Lazy Evaluation (Altklausuraufgabe)

a) Erläutern Sie, wie sich die Auswertung von strikten Parametern, „Call by name“-Parametern und `lazy val`s unterscheidet.

b) Die fold-Methoden von List durchlaufen immer die vollständige Liste. In manchen Fällen führt dies zu unnötigen Berechnungen, z.B. könnte beim Multiplizieren von Ints nach

einer 0 abgebrochen werden. Verändern Sie folgende Implementation von `List.foldRight` so, dass ein vorzeitiges Abbrechen der Iteration möglich ist:

```
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) Geben Sie einen Aufruf dieser `foldRight`-Methode an, der eine Liste von `Int`s multipliziert und bei Auftreten einer 0 direkt 0 zurückgibt, ohne den Rest der Liste durchzuiterieren.