Programowanie współbieżne
Lista 2

1. Zdefiniuj swoją klasę dla modyfikowalnej pary polimorficznej MyPair[A, B]. Ma ona mieć dwa pola modyfikowalne fst i snd z odpowiednimi akcesorami i mutatorami, oraz metodę toString, zwracającą napis w formacie (*fst, snd*).

2. Klasa BankAccount jest zdefiniowana następująco:

```
class BankAccount(initialBalance : Double) {
  private[this] var balance = initialBalance
  def checkBalance = balance
  def deposit(amount : Double) = { balance += amount; balance}
  def withdraw(amount : Double) = { balance -= amount; balance}
}
```

a) Zdefiniuj klasę CheckingAccount, rozszerzającą klasę BankAccount, w której pobierana jest opłata w wysokości 1$ za każdą wpłatę i wypłatę z konta. Zmodyfikuj odpowiednio metody deposit i withdraw.

b) Zdefiniuj klasę SavingsAccount, rozszerzającą klasę BankAccount, w której co miesiąc do konta dodawane jest oprocentowanie 1% . Nie chodzi tu o korzystanie z kalendarza. Dodaj metodę earnMonthlyInterest; jej użycie oznacza, że upłynął miesiąc.
Trzy transakcje miesięcznie są bezpłatne, za pozostałe pobierana jest opłata w wysokości 1$. Zmodyfikuj odpowiednio metody deposit i withdraw.

3. Jedna z pętli w języku Scala ma następującą składnię: while (*warunek*) *wyrażenie*, np.

```
var count = 0
while (count < 5) {
  println(count)
  count += 1
}
```

Napisz funkcję whileLoop (**bez używania efektów obliczeniowych**), która pobiera dwa argumenty: warunek oraz wyrażenie i dokładnie symuluje działanie pętli while (również składniowo). Jakiego typu (i dlaczego) muszą być argumenty i wynik funkcji?

4. Polimorficzne leniwe drzewa binarne można zdefiniować następująco:

```
sealed trait lBT[+A]
case object LEmpty extends lBT[Nothing]
case class LNode[+A](elem: A, left: () => lBT[A], right: () => lBT[A]) extends lBT[A]
```

a) Napisz funkcję lBreadth: [A](ltree: lBT[A])Stream[A], tworzącą strumień zawierający wszystkie wartości węzłów leniwego drzewa binarnego.
Wskazówka: zastosuj obejście drzewa wszerz, reprezentując kolejkę jako zwykłą listę.

b) Napisz funkcję lTree: (n: Int)lBT[Int] , która dla zadanej liczby naturalnej n konstruuje nieskończone leniwe drzewo binarne z korzeniem o wartości n i z dwoma poddrzewami lTree (2*n) oraz lTree(2*n+1).
To drzewo jest przydatne do testowania funkcji z poprzedniego podpunktu.