

Programowanie funkcyjne i współbieżne

Lista 8

1. Dana jest następująca klasa abstrakcyjna dla generycznych kolejek modyfikowalnych.

```
class FullException(msg: String) extends Exception(msg)
```

```
abstract class MyQueue[E] {  
  @throws[FullException]  
  def enqueue(x: E): Unit  
  def dequeue: Unit  
  @throws[NoSuchElementException]  
  def first: E  
  def isEmpty: Boolean  
  def isFull: Boolean  
}
```

- a) Napisz klasę generyczną `QueueMut`, rozszerzającą powyższą klasę abstrakcyjną, w której kolejka jest implementowana przez **tablicę cykliczną** (wszystkie operacje na indeksach tablicy cyklicznej są wykonywane modulo rozmiar tablicy). Implementacja ma być zgodna z poniższym rysunkiem, czyli rozmiar tablicy musi być o 1 większy od pojemności kolejki (dzięki temu indeksy `f` oraz `r` wystarczą do sprawdzenia, czy kolejka jest pusta czy pełna). Metoda `dequeue` dla pustej kolejki ma pozostawiać pustą kolejkę.
- b) Zdefiniuj obiekt towarzyszący z metodami:
- ```
def apply[E: ClassTag](xs: E*): QueueMut[E] = ???
def empty[E: ClassTag](capacity: Int = 1000) : QueueMut[E] = ???
```

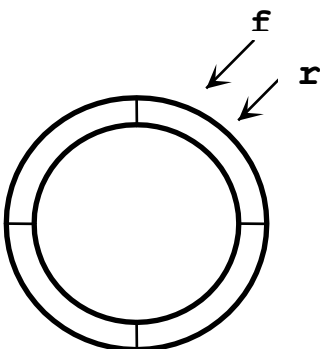
Przeprowadź testy na **małej** kolejce, którą **całkowicie** zapelnisz. Przetestuj przejście przez „sklejenie” tablicy. Wszystkie definicje oraz proste testy w obiekcie singletonowym `Lista8` z metodą `main` umieść w pliku `Lista8.scala`.

**Uwaga.** Tablice w języku Java nie mogą być generyczne (wykład 7, str. 36). W Scali jest to jednak możliwe, ale w czasie tworzenia tablicy generycznej potrzebna jest informacja o wytartym typie elementów tej tablicy. Można to zrobić, wykorzystując znacznik typu (ang. class tag), który sam jest typu `scala.reflect.ClassTag` (wykład 7, str. 38), co jest zilustrowane poniżej:

```
import reflect.ClassTag
class QueueMut[E: ClassTag] private (val capacity: Int = 1000)
 extends MyQueue[E]
```

### *Kolejka reprezentowana przez tablicę cykliczną*

*kolejka pusta*



*kolejka pełna*

