Laboration 6 – länkad lista, stack och kö

Grundläggande

1. Lös Exercise 15.1 i Java Foundations. Hur ser stackens innehåll ut efter instruktionerna i 15.1?

   EX 15.1 Hand trace a queue X through the following operations:

   ```
   X.enqueue(new Integer(4));
   X.enqueue(new Integer(1));
   Object Y = X.dequeue();
   X.enqueue(new Integer(8));
   X.enqueue(new Integer(2));
   X.enqueue(new Integer(5));
   X.enqueue(new Integer(3));
   Object Y = X.dequeue();
   X.enqueue(new Integer(4));
   X.enqueue(new Integer(9));
   ```

2. Skriv klassen ***LinkedStack<E>*** vilken ska vara en stack som implementerar ***Stack<E>*** (F6 / L5). *LinkedStack* ska ha en instansvariabel av typen

   ```
   List<E> stack = LinkedList<E>; // från F6 och L5
   ```

   Det är en god idé att utgå från en stack-implementeringen när du löser. Det är endast mindre modifieringar som är nödvändiga.
   Om du testar din klass med *TestLinkedStack* ska du få utskriften nedan:

   ```
   size=10
   Senast placerad i stacken:
   9
   9 8 7 6 5 4 3 2 1 0
   ```

3. Ändra i uppgift två så att objekten lagras i en *ArrayList<E> (från L7)*. Spara den nya klassen som ***ArrayListStack<E>***. Bör elementen placeras/tas bort först eller sist i listan?
   Om du testar din klass med *TestArrayListStack* får du samma utskrift som i uppgift 2.
4. Skriv klassen ***LinkedQueue<E>*** vilken ska vara en kö som implementerar ***Queue<E>***. *LinkedQueue* ska ha en instansvariabel av typen *LinkedList<E>*.
   Det är en god idé att utgå från en queue-implementeringen när du löser uppgiften. Det är endast mindre modifieringar som är nödvändiga.
   Om du testar din klass med *TestLinkedQueue* ska du få utskriften nedan:

   ```
   size=10
   Först placerad i kön:
   0
   0 1 2 3 4 5 6 7 8 9
   ```

5. PP 15.9 Create a system using a stack and a queue to test whether a given string is a palindrome (i.e., the characters read the same forward or backward).

   Exempel på palindrom: olasalo, sirapiparis, anna

   Ditt program ska fungera så här:
   1. Användaren ska uppmanas skriva in en sekvens tecken via en dialog.
   2. Programmet kontrollerar om den inmatade sekvensen är ett palindrom eller ej.
      Vid undersökningen ska du använda en *LinkedStack* och en *LinkedQueue*. Placera samtliga tecken i den inmatade strängen i båda objektsamlingarna (som Character-objekt). Jämför sedan tecknen som plockas ur samlingarna.
   3. Programmet meddelar resultatet

Fördjupande

6. Skriv klassen ***PriorityQueue<E>*** vilken ska implementera en prioritetskö för objekt.
   Klassen ska ha en konstruktor vilken tar emot en *Comparator<E>*-implementering. Med hjälp av *Comparator*-implementeringen ska objekten ordnas i kön.
   Använd en *List*-implementering från L7 (*LinkedList* eller *ArrayList*) för att lagra objekten i kön.
   Klassen ska implementera interfacet Queue<E>.

   Om du kör *TestPriorityQueue* ska du få körresultatet nedan.

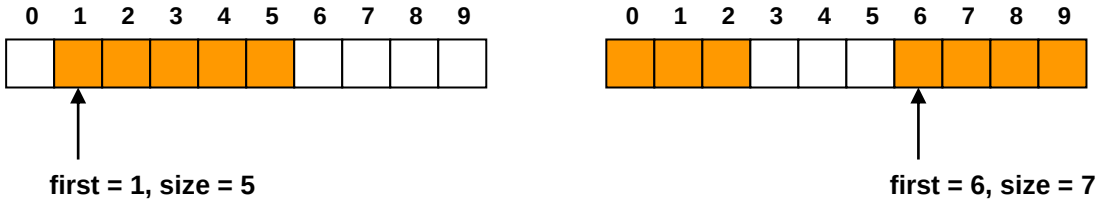
```
Size = 6
Första element: C, 28 år, förmögenhet: 200
Element: C, 28 år, förmögenhet: 200, size = 5
Element: A, 20 år, förmögenhet: 100, size = 4
Element: D, 28 år, förmögenhet: 100, size = 3
Element: F, 24 år, förmögenhet: 100, size = 2
Element: E, 24 år, förmögenhet: 50, size = 1
Element: B, 28 år, förmögenhet: 50, size = 0
----------------------------------------
Size = 6
Första element: A, 20 år, förmögenhet: 100
Element: A, 20 år, förmögenhet: 100, size = 5
Element: E, 24 år, förmögenhet: 50, size = 4
Element: F, 24 år, förmögenhet: 100, size = 3
Element: C, 28 år, förmögenhet: 200, size = 2
Element: B, 28 år, förmögenhet: 50, size = 1
Element: D, 28 år, förmögenhet: 100, size = 0
```

7. Lägg till konstruktorerna ***public PriorityQueue()*** i klassen **PriorityQueue**. Om den nya konstruktorn används ska objekten ordnas med hjälp av *Comparable*-implementeringen i objekten..
   Om du aktiverar de avmarkerade raderna i main-metoden i *TestPriorityQueue* ska du få nedanstående körresultat.

```
Size = 6
Första element: A, 20 år, förmögenhet: 100
Element: A, 20 år, förmögenhet: 100, size = 5
Element: B, 28 år, förmögenhet: 50, size = 4
Element: C, 28 år, förmögenhet: 200, size = 3
Element: D, 28 år, förmögenhet: 100, size = 2
Element: E, 24 år, förmögenhet: 50, size = 1
Element: F, 24 år, förmögenhet: 100, size = 0
----------------------------------------
Size = 6
Första element: C, 28 år, förmögenhet: 200
Element: C, 28 år, förmögenhet: 200, size = 5
Element: A, 20 år, förmögenhet: 100, size = 4
Element: D, 28 år, förmögenhet: 100, size = 3
Element: F, 24 år, förmögenhet: 100, size = 2
Element: E, 24 år, förmögenhet: 50, size = 1
Element: B, 28 år, förmögenhet: 50, size = 0
----------------------------------------
Size = 6
Första element: A, 20 år, förmögenhet: 100
Element: A, 20 år, förmögenhet: 100, size = 5
Element: E, 24 år, förmögenhet: 50, size = 4
Element: F, 24 år, förmögenhet: 100, size = 3
Element: C, 28 år, förmögenhet: 200, size = 2
Element: B, 28 år, förmögenhet: 50, size = 1
Element: D, 28 år, förmögenhet: 100, size = 0
```

8. Skriv klassen ***ArrayQueue<E>*** vilken ska implementera ***Queue<E>.*** Skillnaderna mot *ArrayQueue<E>* i *F9* är att kön ska flytta sig i arrayen för att slippa kopieringar vid *dequeue*.

   En svårighet i denna uppgift är att hantera arrayen på ett effektivt sätt. Bästa sättet är att låta början respektive slutet på kön flytta sig allteftersom element läggs till och tas bort. Ett sätt att klara detta är att ha en markör till första elementet i kön (*first*) och en räknare som håller reda på antalet element i kön (*size*).



   Test din kö så den fungerar väl.

9. Skriv metoden ***public boolean check(String expression)***. Metoden ska kontrollera om parenteser i strängen *expression* parvis matchar varandra. En vänsterparentes ska alltid följas av samma typ av högerparentes. Parenteser som ska behandlas av check är: (,), {, }, [och].

   Exempel:
   * I en korrekt skriven metod eller klass i java matchar parenteserna varandra (bortsett från parenteser som förekommer i String-litteraler).
   * Korrekt: `"((({[[{}]]}))))"`, mellan parenteserna kan du placera godtyckliga tecken (naturligtvis ej parenteser)
   * exempel på felaktiga: `"(}", "{(})", "}", "())", "(", "(()"`
     Även i dessa strängar kan du fylla på med godtyckliga tecken där du önskar.

   För att lösa problemet ska du använda en stack.
   * Kontrollera tecken för tecken från vänster i strängen.
   * Varje gång du stöter på en vänsterparentes så ska denna pushas på stacken (använd t.ex. klassen Character)
   * Varje gång du stöter på en högerparentes så ska du poppa ett element från stacken och kontrollera att parenteserna matchar varandra. Är så inte fallet så ska metoden returnera false. Finns det inget element att poppa så ska metoden också returnera false.
   * När samtliga tecken har kontrollerats är resultatet sant om alla parenteser matchat varandra och stacken är tom.

Lösningar

Uppgift2

```
package laboration9;
import f6.Stack;
import laboration7.LinkedList;
import java.util.EmptyStackException;

public class LinkedStack<E> implements Stack<E> {
    private LinkedList<E> list = new LinkedList<E>();

    public void push(E element) {
        list.add(0, element);
    }

    public E pop() {
        if(empty()) {
            throw new EmptyStackException();
        }
        return list.remove(0);
    }

    public E peek() {
        if(empty()) {
            throw new EmptyStackException();
        }
        return list.get(0);
    }

    public boolean empty() {
        return (list.size()==0);
    }

    public int size() {
        return list.size();
    }
}
```

Uppgift3

```
package laboration9;
import java.util.EmptyStackException;
import laboration7.ArrayList;
import f6.Stack;

public class ArrayListStack<E>  implements Stack<E> {
    private ArrayList<E> list = new ArrayList<E>();

    public void push(E element) {
        list.addLast( element );
    }

    public E pop() {
        if(empty()) {
            throw new EmptyStackException();
        }
        return list.removeLast();
    }

    public E peek() {
        if(empty()) {
```

```
            throw new EmptyStackException();
        }
        return list.get(size()-1);
    }

    public boolean empty() {
        return (size()==0);
    }

    public int size() {
        return list.size();
    }
}
```

Uppgift4

```
package laboration9;
import f9.Queue;
import f9.QueueException;
import laboration7.LinkedList;

public class LinkedQueue<E> implements Queue<E> {
    private LinkedList<E> list = new LinkedList<E>();

    public void enqueue(E data) {
        list.add(size(), data);
    }

    public E dequeue() {
        if(empty()) {
            throw new QueueException("dequeue: Queue is empty");
        }
        return list.remove(0);
    }

    public E peek() {
        if(empty()) {
            throw new QueueException("peek: Queue is empty");
        }
        return list.get(0);
    }

    public boolean empty() {
        return list.size()==0;
    }

    public int size() {
        return list.size();
    }
}
```

Uppgift5

```
public static boolean palindrom( String str ) {
    LinkedStack<Character> stack = new LinkedStack<Character>();
    LinkedQueue<Character> queue = new LinkedQueue<Character>();
    boolean res = true;
    for( int i=0; i<str.length(); i++ ) {
        stack.push( str.charAt( i ) );
        queue.enqueue( str.charAt( i ) );
    }
    while( !stack.empty() && res ) {
        res = stack.pop().equals(queue.dequeue());
```

```
    }
    return res;
}
```

Uppgift6

```
package laboration9;
import java.util.Comparator;
import laboration7.LinkedList;
import f9.Queue;
import f9.QueueException;

public class PriorityQueue<E> implements Queue<E> {
    private LinkedList<E> queue = new LinkedList<E>();
    private Comparator<E> comparator;

    public PriorityQueue(Comparator<E> comparator) {
        this.comparator = comparator;
    }

    public int size() {
        return queue.size();
    }

    public boolean empty() {
        return size()==0;
    }

    public void enqueue( E data ) {
        int index = 0;
        int size = size();
        while (index<size && comparator.compare(queue.get(index), data) <=
0) {
            index++;
        }
        queue.add(index, data);
    }

    public E dequeue() {
        if( empty() ) {
            throw new QueueException("dequeue: Kön är tom");
        }
        return queue.removeFirst();
    }

    public E peek() {
        if( empty() ) {
            throw new QueueException("peek: Kön är tom");
        }
        return queue.get(0);
    }
}
```

Uppgift7

```
package laboration9;
import java.util.Comparator;
import laboration7.LinkedList;
import f9.Queue;
import f9.QueueException;

public class PriorityQueue<E> implements Queue<E> {
    private LinkedList<E> queue = new LinkedList<E>();
```

```
    private Comparator<E> comparator;

    public PriorityQueue() {
        this.comparator = new Comp();
    }

    // som tidigare

    private class Comp implements Comparator<E> {
        public int compare(E e1, E e2) {
            return ((Comparable<E>)e1).compareTo(e2);
        }
    }
}
```

Uppgift8

```
package laboration9;
import f9.Queue;
import f9.QueueException;

public class ArrayQueue<E> implements Queue<E> {
    private E[] elements;
    private int front = 0;
    private int size = 0;

    public ArrayQueue(int capacity) {
        elements = (E[])new Object[ capacity ];
    }

    public void enqueue( E elem ) {
        if( size==elements.length) {
            throw new QueueException("enqueue: Queue is full");
        }
        elements[ (front+size) % elements.length ] = elem;
        size++;
    }

    public E dequeue() {
        if(size==0) {
            throw new QueueException("dequeue: Queue is empty");
        }
        E value = elements[ front ];
        elements[ front ] = null;
        size--;
        front = (front+1) % elements.length;
        return value;
    }

    public E peek() {
        if( size==0 ) {
            throw new QueueException("peek: Queue is empty");
        }
        return elements[ front ];
    }

    public boolean empty() {
        return (size==0);
    }

    public int size() {
        return size;
    }
```

```
}
```

Uppgift 9

```
public static boolean check( String str ) {
    String leftP = "({[", rightP = ")}]";
    char left, chr;
    LinkedStack<Character> stack = new LinkedStack<Character>();
    boolean res = true;
    try {
        for( int i=0; i<str.length() && res==true; i++ ) {
            chr = str.charAt(i);
            if(leftP.indexOf(chr)>-1)
                stack.push(new Character(chr));
            if(rightP.indexOf(chr)>-1) {
                left = ((Character)stack.pop()).charValue();
                res = (left=='(' && chr==')') ||
                      (left=='{' && chr=='}') ||
                      (left=='[' && chr==']');
            }
        }
    }catch(EmptyStackException e) {
        res = false;
    }
    return res && ( stack.size() == 0 );
}
```