Blatt 6

Punktetabelle

1	2	3	$\sum$
5	5	5	15

1 Aufgabe 1

Gegeben sei eine Stack Implementierung (z.B.: *java.util.Stack*). Implementieren Sie in einem ersten Schritt (a) das Queue Interface (siehe *IQueue.java*) als FIFO Queue. Diese Implementierung des Queue Interface soll jedoch keine eigene Funktionalität implementieren, sondern durch Delegation jene der Klasse Stack wiederverwenden. Erweitern sie Ihre Implementierung in einem zweiten Schritt (b) um aussagekräftige Exceptions. Berücksichtigen Sie dabei, das sinnvolle Durchreichen von Exceptions.

```
public interface IQueue {

    /** Adds a new entry to the back of this queue.
     *  @param item an item to be added
     */
    public boolean enqueue(Object item);

    /** Removes and returns the entry at the front of this queue.
     *  @return either the element at the front of the queue or, if the
     *       queue is empty before the operation, null
     */
    public Object dequeue();

    /** Returns the size of this queue.
     *  @return the number of elements in the queue, if the
     *       queue is not empty, 0 otherwise
     */
    public int size();

    /** Detects whether this queue is empty.
     *  @return true if the queue is empty, or false otherwise
     */
    public boolean isEmpty();

    /** Removes all entries from this queue.
     */
    public void clear();
}
```

Listing 1: IQueue.java

2 Aufgabe 2

Als Teil der `Object` Klasse stellt Ihnen Java rudimentäre Implementierungen von `equals(...)` und `clone(...)` zum Vergleich zweier Objekte bzw. zum Kopieren eines Objektes zur Verfügung. Erstellen Sie jeweils eine Implementierung dieser Methoden, welche füre zyklische Datenstrukturen, wie z.B.: *Teacher.java* aus der VL.

3 Aufgabe 3

Betrachten Sie folgende Java Klassen und deren Verwendung. Was ist die vollständige Ausgabe der `main(...)` Methode in der Klasse `Test.java`? Erklären Sie die Ergebnisse!

```
class Moe {
    public void print(Moe p) {
        System.out.println("Moe 1\n");
    }
}
```

Listing 2: Moe.java

```
class Larry extends Moe {
    public void print(Moe p) {
        System.out.println("Larry 1\n");
    }
    public void print(Larry l) {
        System.out.println("Larry 2\n");
    }
}
```

Listing 3: Larry.java

```
class Curly extends Larry {
    public void print(Moe p) {
        System.out.println("Curly 1\n");
    }
    public void print(Larry l) {
        System.out.println("Curly 2\n");
    }
    public void print(Curly b) {
        System.out.println("Curly 3\n");
    }
}
```

Listing 4: Curly.java

```
public class Test {
    public static void main (String [] args) {
       Larry stooge1 = new Curly();
        Moe stooge2 = new Larry();
        Moe stooge3 = new Curly();
        Curly stooge4 = new Curly();
        Larry stooge5 = new Larry();
        Object stooge6 = new Curly();
        Object stooge7 = new Moe();
        Object stooge8 = new Larry();
        stooge1.print(new Moe());
        ((Curly)stooge1).print(new Larry());
        ((Larry)stooge2).print(new Moe());
        stooge2.print(new Curly());
        stooge3.print(new Curly());
        stooge3.print(new Moe());
        stooge3.print(new Larry());
        ((Curly)stooge3).print(new Larry());
        ((Curly)stooge3).print(new Curly());
        stooge4.print(new Curly());
        stooge4.print(new Moe());
        stooge4.print(new Larry());
        stooge5.print(new Curly());
        stooge5.print(new Larry());
        stooge5.print(new Moe());
        ((Curly)stooge6).print(new Larry());
        ((Moe)stooge7).print(new Curly());
        ((Larry)stooge8).print(new Moe());
        ((Moe)stooge6).print(new Moe());
```

```
        ((Larry)stooge6).print(new Larry());
        ((Moe)stooge8).print(new Moe());
    }
}
```

Listing 5: Test.java