Aufgabe 6 (Stacks)

Gegeben sei die Implementierung eines Stacks ganzer Zahlen mit folgender Schnittstelle:

```
3  import java.util.Stack;
4
5  /**
6   * Um schnell einen lauffähigen Stack zu bekommen, verwenden wir den Stack aus
7   * der Java Collection.
8   */
9  public class IntStack {
10   private Stack<Integer> stack = new Stack<Integer>();
11
12   /**
13    * Legt Element i auf den Stack.
14    *
15    * @param i Eine Zahl, die auf dem Stack gelegt werden soll.
16    */
17   public void push(int i) {
18     stack.push(i);
19   }
20
21   /**
22    * Gibt oberstes Element vom Stack.
23    *
24    * @return Das oberste Element auf dem Stapel.
25    */
26   public int pop() {
27     return stack.pop();
28   }
29
30   /**
31    * Fragt ab, ob Stack leer ist.
32    *
33    * @return Wahr, wenn der Stapel leer ist.
34    */
35   public boolean isEmpty() {
36     return stack.empty();
37   }
38 }
```

Betrachten Sie nun die Realisierung der folgenden Datenstruktur `Mystery`, die zwei Stacks benutzt.

```
3  public class Mystery {
4    private IntStack a = new IntStack();
5    private IntStack b = new IntStack();
6
7    public void foo(int item) {
8      a.push(item);
9    }
10
11   public int bar() {
12     if (b.isEmpty()) {
13       while (!a.isEmpty()) {
14         b.push(a.pop());
15       }
16     }
17     return b.pop();
18   }
```

(a) Skizzieren Sie nach jedem Methodenaufruf der im folgenden angegebenen Befehlssequenz den Zustand der beiden Stacks eines Objekts `m` der Klasse `Mystery`. Geben Sie zudem bei jedem Aufruf der Methode `bar` an, welchen Wert diese zurückliefert.

```
21  Mystery m = new Mystery();
22  m.foo(3);
23  m.foo(5);
24  m.foo(4);
25  m.bar();
26  m.foo(7);
27  m.bar();
28  m.foo(2);
29  m.bar();
30  m.bar();
```

Code	Stack b	Stack b	Rückgabewert
`m.foo(3);`	{ 3 }	{ }	
`m.foo(5);`	{ 5, 3 }	{ }	
`m.foo(4);`	{ 4, 5, 3 }	{ }	
`m.bar();`	{ }	{ 5, 4 }	3
`m.foo(7);`	{ 7 }	{ 5, 4 }	
`m.bar();`	{ 7 }	{ 4 }	5
`m.foo(2);`	{ 2, 7 }	{ }	
`m.bar();`	{ 2, 7 }	{ }	4
`m.bar();`	{ }	{ 2 }	7

(b) Sei n die Anzahl der in einem Objekt der Klasse `Mystery` gespeicherten Werte. Im folgenden wird gefragt, wieviele Aufrufe von Operationen der Klasse `IntStack` einzelne Aufrufe von Methoden der Klasse `Mystery` verursachen. Begründen Sie jeweils Ihre Antwort.

(i) Wie viele Aufrufe von Operationen der Klasse `IntStack` verursacht die Methode `foo(x)` im besten Fall?

> Einen Aufruf, nämlich `a.push(i)`

(ii) Wie viele Aufrufe von Operationen der Klasse `IntStack` verursacht die Methode `foo(x)` im schlechtesten Fall?

> Einen Aufruf, nämlich `a.push(i)`

(iii) Wie viele Aufrufe von Operationen der Klasse `IntStack` verursacht die Methode `bar()` im besten Fall?

> Wenn der Stack `b` nicht leer ist, dann werden zwei Aufrufe benötigt, nämlich `b.isEmpty()` und `b.pop()`

(iv) Wie viele Aufrufe von Operationen der Klasse `IntStack` verursacht die Methode `bar()` im schlechtesten Fall?

Wenn der Stack `b` leer ist, dann liegen all n Objekte im Stack `a`. Die Objekte im Stack `a` werden in der `while`-Schleife nach `b` verschoben. Pro Objekt sind drei Aufrufe nötig, also $3 \cdot n$. `b.isEmpty()` (erste Zeile in der Methode) und `b.pop()` (letzte Zeile in der Methode) wird immer aufgerufen. Wenn alle Objekt von `a` nach `b` verschoben wurden, wird zusätzlich noch einmal in der Bedingung der `while`-Schleife `a.isEmpty()` aufgerufen. Im schlechtesten Fall werden also $3 \cdot n + 3$ Operationen der Klasse `IntStack` aufgerufen.

(c) Welche allgemeinen Eigenschaften werden durch die Methoden `foo` und `bar` realisiert? Unter welchem Namen ist diese Datenstruktur allgemein bekannt?

`foo()` Legt das Objekt auf den Stack `a`. Das Objekt wird in die Warteschlange eingereiht. Die Methode müsste eigentlich `enqueue()` heißen.

`bar()` Verschiebt alle Objekte vom Stack `a` in umgekehrter Reihenfolge in den Stack `b`, aber nur dann, wenn Stack `b` leer ist. Entfernt dann den obersten Wert aus dem Stack `b` und gibt ihn zurück. Das zuerst eingereihte Objekt wird aus der Warteschlange entnommen. Die Methode müsste eigentlich `dequeue()` heißen.

Die Datenstruktur ist unter dem Namen Warteschlange oder Queue bekannt