Seminar 3

1. `Iterable` und `Iterator`

Jede Klasse, die `Iterable` entsprechend implementiert, kann in for-each-Schleifen verwendet werden. Um eine iterierbare Datenstruktur zu implementieren, muss man:

1. `Iterable` implementieren
2. eine `Iterator` Klasse erstellen, die `Iterator` und die entsprechenden Methoden implementieren.

```
class CustomDataStructure implements Iterable<> {

    // code for data structure
    public Iterator<> iterator() {
        return new CustomIterator<>(this);
    }
}
class CustomIterator<> implements Iterator<> {

    // constructor
    CustomIterator<>(CustomDataStructure obj) {
        // initialize cursor
    }

    // Checks if the next element exists
    public boolean hasNext() {
    }

    // moves the cursor/iterator to next element
    public T next() {
    }

    // Used to remove an element. Implement only if needed
    public void remove() {
        // Default throws UnsupportedOperationException.
    }
}
```

2. Übungen

1. Implementieren Sie eine Klasse `Spielkarte` mit zwei Attribute: *`farbe`* und *`wert`*. Farbe stellt bei Spielkarten die Sorte dar. Also Pik, Kreuz, Herz, Karo sind Farben. Implementieren Sie eine Klasse `Deck`, damit der folgende Codefragment valid ist:

   ```
   for (Spielkarte c : deck)
   ```

2. Implementieren Sie dieselbe Funktionalität von Punkt 1 aber mit Enum.
3. Implementieren Sie eine Klasse TV mit einer Liste von Kanäle als Attribut. Sie soll auch zwei Methoden `channel_up` und `channel_down` bereitstellen, die die TV-Kanäle entsprechend wechseln. Implementieren Sie eine Klasse Remote mit Methoden für Kanalsteuerung. Eine Fernbedienung kann nur mit einem TV funktionieren.