Java Interfaces

Interfaces beschreiben eine Menge definierter Verhaltensweisen oder Dienste.

Beispiel: Bibliothek …. Dinge ausleihen, Dinge verlängern, Dinge zurückgeben

Geschäft …. Produkt kaufen, Produkt reklamieren

VersandGeschäft … Produkte bestellen, Produkt kaufen, Produkt reklamieren

Definition von Interfaces

claimable.java

```
public interface claimable {
    public boolean seize(Product product, Person person);
    public boolean release(Product product, Person person);
}
```

buyable.java

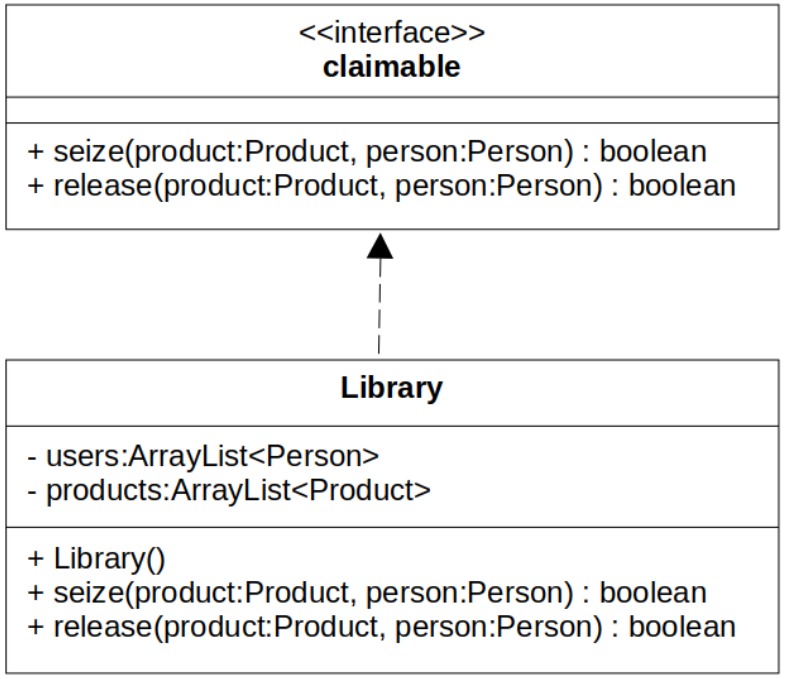
```
public interface buyable {
    public Receipt order(Product product, Person person);
    public boolean pay(Receipt receipt);
    public boolean reject(Receipt receipt);
}
```

Java Interfaces

Implementierung/Umsetzung/Realisierung von Interfaces

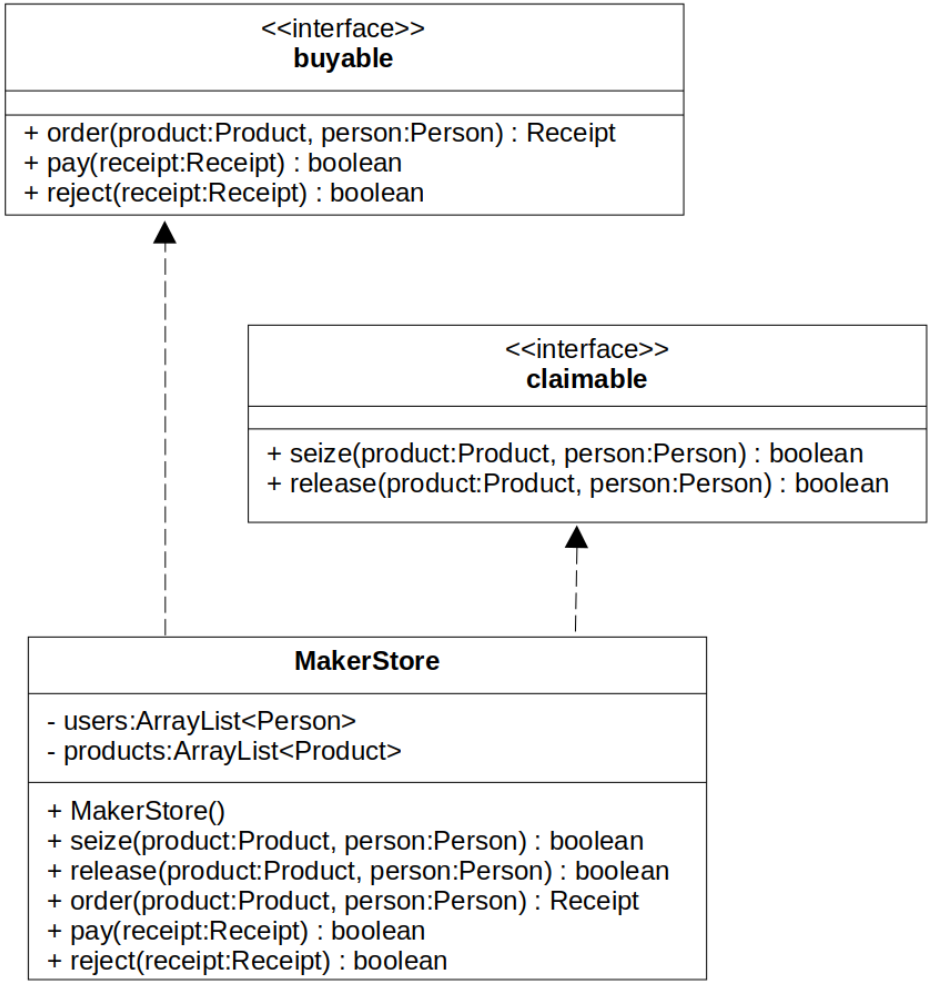
Library.java

```java
public class Library implements claimable {

    private ArrayList<Person> users = new ArrayList<Person>();
    private ArrayList<Product> products = new ArrayList<Product>();

    public Library(){ }

    // ....

    @Override
    public boolean seize(Product product, Person person) {
        // TODO Auto-generated method stub
        System.out.println(person + " seizes " + product);
        // ...
        return false;
    }

    @Override
    public boolean release(Product product, Person person) {
        // TODO Auto-generated method stub
        System.out.println(person + " releases " + product);
        // ...
        return false;
    }
}
```



Java Interfaces

Implementierung von Interfaces



MakerStore.java

```java
public class MakerStore implements claimable, buyable {

    private ArrayList<Person> users = new ArrayList<Person>();
    private ArrayList<Product> products = new ArrayList<Product>();

    public MakerStore(){ }

    // ....

    @Override
    public Receipt order(Product product, Person person) {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public boolean pay(Receipt receipt) {
        // TODO Auto-generated method stub
        return false;
    }

    @Override
    public boolean reject(Receipt receipt) {
        // TODO Auto-generated method stub
        return false;
    }

    @Override
    public boolean seize(Product product, Person person) {
        // TODO Auto-generated method stub
        return false;
    }

    @Override
    public boolean release(Product product, Person person) {
        // TODO Auto-generated method stub
        return false;
    }

}
```

Zusammenfassung: Java Interfaces

Interfaces beschreiben eine Menge definierter Verhaltensweisen.

Interfaces enthalten Definitionen von Methoden und ggf. auch Variable.

Interfaces werden:
- definiert
- implementiert

Interfaces können vererbet werden.

Eine Klasse kann mehrere Interfaces implementieren.

Interfaces sind damit ein weiteres wichtiges Entwurfsinstrument objektorientierter Programmierung.

Praktische Bedeutung? … wird auf den nächsten Seiten gezeigt.

BEISPIEL COMPARABLE

Interface Comparable – Beispiel 1

Standard-Interface

demo008.interfaces.comparable

- in Package java.lang
- ermöglicht Vergleich nach einem „**natürlichen**“ Kriterium
- enthält nur die Methode **`public int compareTo(Object obj)`**
- Standardisierte Rückgabewerte:
  - **0** – falls aktuelles Objekt gleich dem anderen Objekt ist
  - **positiver Integer** – falls aktuelles Objekt größer als das andere Objekt ist
  - **negativer Integer** – falls aktuelles Objekt kleiner als das andere Objekt ist

Hinweis:
Das Interface Comparable ist in der Klasse Integer bereits implementiert und führt zum Vergleich mit dem erforderlichen Ergebnis.
Natürliche Ordnung -> der Größe nach

Beispiel 1 (SimpelComparableExample):

```
Integer x = 4;
Integer y = 7;
Integer z = 7;

int compareErgebnis1 = x.compareTo(y);
int compareErgebnis2 = y.compareTo(x);
int compareErgebnis3 = y.compareTo(z);

System.out.println("x.compareTo(y) : "+compareErgebnis1);
System.out.println("y.compareTo(x) : "+compareErgebnis2);
System.out.println("y.compareTo(z) : "+compareErgebnis3);
```

Ausgabe auf Kommandozeile:

```
x.compareTo(y) : -1
y.compareTo(x) : 1
y.compareTo(z) : 0
```

weil x kleiner als y ist

weil y größer als x ist

weil y gleich z ist

Interface Comparable – Beispiel 2

Natürliches Kriterium soll für Student die id sein

Natürliche Ordnung → entsprechend der Größe der id sortiert

Interface Comparable wird implementiert → Methode compareTo() muss implementiert werden

```
public class Student implements Comparable<Student> {

    private int id;
    private String name;

    public Student(int id, String name) {
        this.id=id;
        this.name=name;
    }

    @Override
    public int compareTo(Student student) {
        if(this.id==student.id) {
            return 0;
        } else if(this.id>student.id) {
            return 1;
        } else {
            return -1;
        }
    }
    ...
}
```

Implementierung der Methode compareTo, sodass „natürliches“ Kriterium id ausgewertet wird.

Größenvergleich der id führt zu größer/kleiner/gleich hinsichtlich dieses Kriteriums

demo008.interfaces.comparable

Interface Comparable – Beispiel 2

Natürliches Kriterium soll für Student die id sein

Natürliche Ordnung → entsprechend der Größe der id sortiert

demo008.interfaces.comparable

in Klasse StudentTest:

```
List<Student> studentList = new ArrayList<Student>();

studentList.add(new Student(100, "Miller"));
studentList.add(new Student(43, "Smith"));
studentList.add(new Student(67, "Brooks"));
studentList.add(new Student(23, "Adams"));

Collections.sort(studentList);

for (int i = 0; i < studentList.size(); i++) {
     System.out.println(studentList.get(i).getId()
          + " : "+studentList.get(i).getName());
}
```

Erzeugen der Student-Objekte in der Liste, nicht der „natürlichen“ Ordnung entsprechend

In der Methode sort() wird das implementierte Interface aufgerufen, um die „natürliche“ Ordnung herzustellen

Ausgabe auf Kommandozeile:

```
23 : Adams
43 : Smith
67 : Brooks
100 : Miller
```