Interfaces

Interfaces

- Define the behavior of their implementers
- Contain only constants and declarations of operations (and nested types)
- Classes can implement interfaces
- A class can implement several interfaces, eventhough it can only extend one classe
- An interface can extend another interface

Interfaces

```
public interface Drawable {
    void draw();
}

public Square implements Drawable {
    public void draw() {
        …
    }
}
```

Operations are public by default

Operations are only declared. No definitions. `Abstract` qualifier not required.

Defnition is mandatory.

Clonable

- No members defined
- Is a placeholder to indicate that a class support cloning (full copy)
- Problem (of the programmer): Copy references or values?
- Redefinition of clone() is protected in class Object and usually redefined as public
- In some cases definition is not requires, only necessary to declare:

```
public class MyObject implements Clonable {
}
```

Generic Interfaces

Generic interface. T is a parameter. The corresponding argument can be a type.

Note: The Queue interface is slightly different in the Java API!

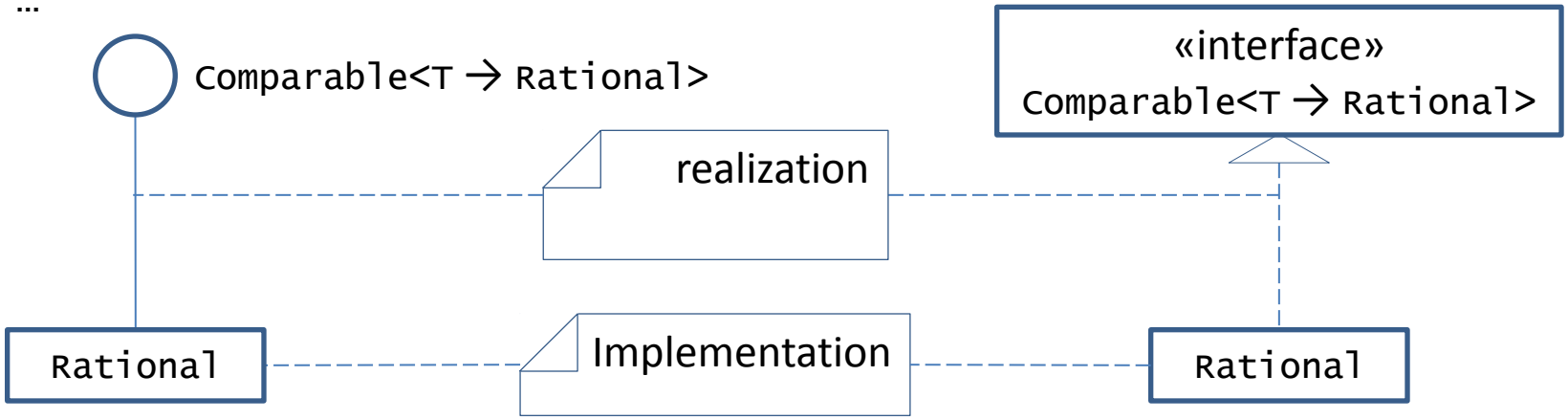
```
public interface Comparable<T> {
    int compareTo(T object);
}

public interface Queue<E> {
    E element();
    void add(E e);
    void remove();
}
```

Interfaces: implementation

```
public class Rational implements Comparable<Rational> {
    private int numerator;
    private int denominator;
    …
    public int compareTo(final Rational another) {
        int leftNumerator =
            getNumerator() * another.getDenominator();
        int rightNumerator =
            another.getNumerator() * getDenominator();

        if (leftNumerator > rightNumerator)
            return 1;
        if (leftNumerator < rightNumerator)
            return -1;
        return 0;
    }
    …
}
```

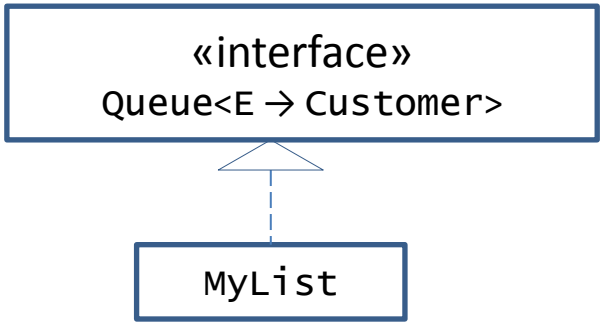
Interfaces: implementation

```
public class Rational implements Comparable<Rational> {
    private int numerator;
    private int denominator;
    …
    public int compareTo(final Rational another){
        return getNumerator() * another.getDenominator()
            - another.getNumerator() * getDenominator();
    }
    …
}
```



Interfaces: polimorphism

```
public class MyList implements Queue<Customer> {

    …

}

public class MyListTester {

    …

    public static void main(final String[] arguments) {
        Queue<Customer> customerQueue =
            new MyList();

        …
    }

    …

}
```



Comparator

- To compare classes that are not value-types* the *interface Comparator* and not *Comparable* should be used.

```
public class ComparadorPorNome implements Comparator<Student> {
   public int compare(final Student one, final Student another){
       return one.getName().compareTo(another.getName());
   }
}
```

*classes where two objects are equal if the content is equal, even if they have different references

Interfaces: names

- Adjective meaning that it is possible to execute (a) certain operation(s) (e.g., `Comparable`)

- Name meaning of the implemented concept (e.g., `Queue`)

Similar to abstract classes, but
1. There is no implementation inheritance, and
2. a class can implement several interfaces.

References

- Y. Daniel Liang, *Introduction to Java Programming*, 7.ª edição, Prentice-Hall, 2010.

Summary

- Interfaces