

An abstract, light blue geometric pattern consisting of interconnected lines and star-like shapes, resembling a complex network or a stylized molecular structure, located in the upper left corner of the slide.

# Interfaces

# Interfaces

- Definem um comportamento que as classes podem implementar
- Contêm apenas constantes e declarações de operações (e tipos embutidos)
- Classes podem **implementar** interfaces
- Uma classe pode **implementar** várias interfaces, embora possa derivar apenas de uma outra classe
- Uma interface pode estender (derivar de) outra

# Interfaces

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

Operações públicas por omissão.

Operações apenas declaradas. Não se define qualquer método. Não é necessário usar o qualificador `abstract`.

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

Definição obrigatória

# Clonable

- Não define membros
- Serve de indicador que a classe suporta clonagem (cópia integral)
- Problema (a resolver pelo programador): Cópia de referências
- Redefinição do método clone() é protegido na classe
- Object e normalmente é redefinido como público
- Em alguns casos não é necessário redefinir, basta a declaração:

```
public class MeuObjeto implements Clonable {  
}
```

# Interfaces genéricas

Interface genérica.  $T$  é um parâmetro. O correspondente argumento tem de ser um tipo.

Nota: A interface queue é um pouco diferente na biblioteca do Java!

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

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

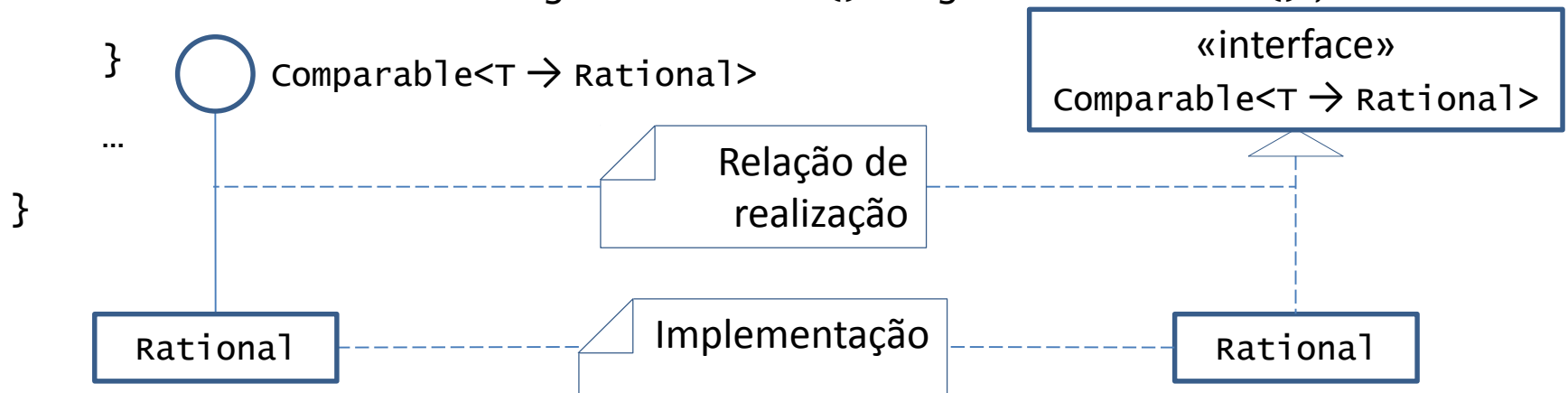
# Interfaces: implementação

```
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;
    }
    ...
}
```

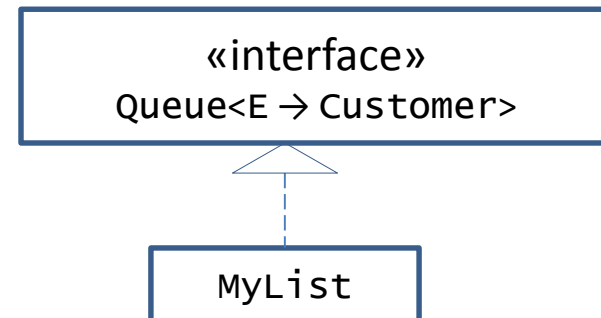
# Interfaces: implementação

```
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: polimorfismo

```
public class MyList implements Queue<Customer> {  
    ...  
}  
  
public class MyListTester {  
    ...  
  
    public static void main(final String[] arguments) {  
        Queue<Customer> customerQueue =  
            new MyList();  
    }  
    ...  
}
```





# Comparator

- Para comparar classes que não são de valor\* devem usar-se implementações do *interface Comparator* e não de *Comparable*.

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

\*classes em que dois objetos são iguais se o conteúdo é igual, mesmo que tenham referências diferentes

# Interfaces: nomes

- Adjectivo denotando possibilidade de realizar determinadas operações (e.g., Comparable)
- Nome denotando conceito cujo comportamento será implementado (e.g., Queue)

Caso semelhante ao das classes abstractas, mas

1. não há qualquer herança de implementação (i.e., de atributos não constantes e métodos) e
2. uma classe que implemente a interface pode simultaneamente implementar outras interfaces.

# Referências

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

# Sumário

- Interfaces