Interfaces

Que si o si debes aprender para el examen

JoeDayz | 2024

Examples of Java Interfaces: `java.lang.Comparable`

Comparable interface describes a way of comparing current object (this) to another object.

- It defines a single abstract method `int compareTo(T o)`.
- It compares current object (this) with the specified object (parameter) to establish their order.
- The `compareTo` method returns an `int` value.

current object	less than	equal to	greater than	parameter object
	negative	zero	positive	

```
public class Product
       implements Comparable<Product> {
  public int compareTo(Product p) {
    return this.name.compareTo(p.name);
  }
  // other variable and methods
}
```

```
package java.lang;
public interface Comparable<T> {
  int compareTo(T o);
}
```

```
Product[] products = {new Product("Tea"),
                      new Product("Coffee"),
                      new Product("Cake")};
Arrays.sort(products);
```

Examples of Java Interfaces: `java.util.Comparator`

The Comparator interface describes a way of comparing a pair of objects.

- Defines a single abstract method `int compare(T o1, T o2)`
- Compares one object with another to establish their order by returning an `int` value from the `compare` method

first object	**less than**	**equal to**	**greater than**	**second object**
	negative	**zero**	**positive**	

```
public class ProductNameSorter
       implements Comparator<Product> {
  public int compare(Product p1, Product p2) {
    return p1.getName().compareTo(p2.getName());
  }
}
```

```
package java.lang;
public interface Comparator<T>{
  int compare(T o1, T o2);
}
```

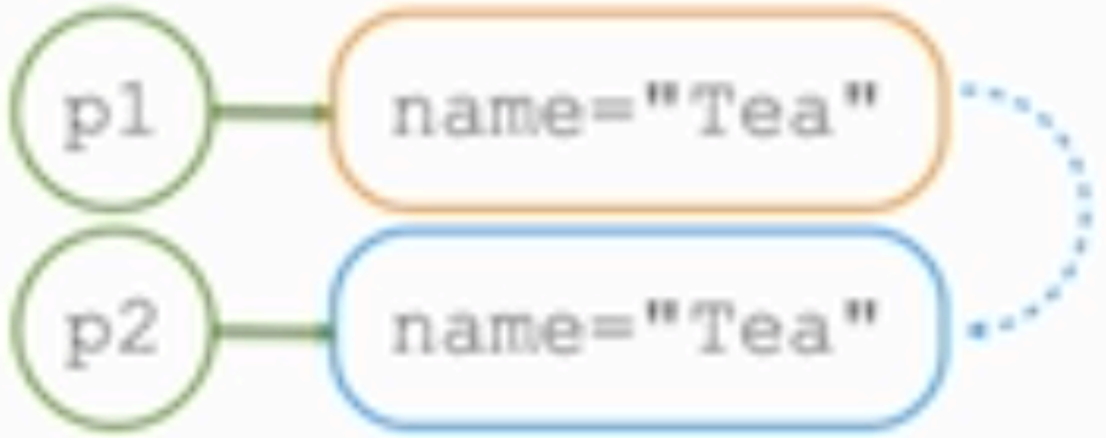
```
Product[] products = {new Product("Tea"),
                      new Product("Coffee"),
                      new Product("Cake")};
Arrays.sort(products, new ProductNameSorter());
```

```
public class Product {
  // variables and methods
}
```

Examples of Java Interfaces: `java.lang.Cloneable`

Cloneable is an example of an interface used as a "type-marker" or "tag-interface."

- The interface does not have to define any methods.
- It can still be used with the `instanceof` operator to validate the object type.
- Cloning an object means creating a replica of the objects memory.
- The `java.lang.Cloneable` interface indicates a permission that an object can be cloned.



```
package java.lang;
public interface Cloneable { }
```

```
public class Product
       implements Cloneable {
 protected Object clone()
 throws CloneNotSupportedException {
  return super.clone();
 }
}
```

```
Product p1 = new Product("Tea");
Product p2 = (Product)p1.clone();
```

```
package java.lang;
public class Object {
 protected Object clone()
 throws CloneNotSupportedException {
   if (!(this instanceof Cloneable)) {
    throw new CloneNotSupportedException();
   }
   // clone object
  }
}
```

Composition Pattern

A Class may represent a composition of features implemented by different other classes.

- Interfaces describe capabilities.
- Classes implement these capabilities.
- Capabilities are aggregated.

```
public interface Withdrawing {
  BigDecimal withdraw();
}
```

```
public interface Depositing {
  void deposit(BigDecimal amount);
}
```

```
public interface Authentication {
  void authenticate();
}
```

```
public class Bank
       implements Withdrawing,
                  Depositing,
                  Authentication {
  private Account a;
  private Security s;
  public BigDecimal withdraw() {
    authenticate();
    return a.withdraw();
  }
  public void deposit(BigDecimal amount) {
    authenticate();
    a.deposit(amount);
  }
  public void authenticate() {
    s.authenticate();
  }
}
```

```
public class Account
       implements Withdrawing,
                  Depositing {
  public BigDecimal withdraw() { }
  public void deposit(BigDecimal amount) { }
}
```

```
public class Security
       implements Authentication {
  public void authenticate() { }
}
```

Summary

In this lesson, you should have learned how to:

- Describe Java interfaces
- Implement an interface
- Describe nonabstract interface methods
- Explain generics
- Utilize some of the commonly used Java Interfaces
- Implement the Composition design pattern