Interfaz

Paradigmas de Lenguajes
3er Año LCC– Facultad CEFyN – UNSJ –

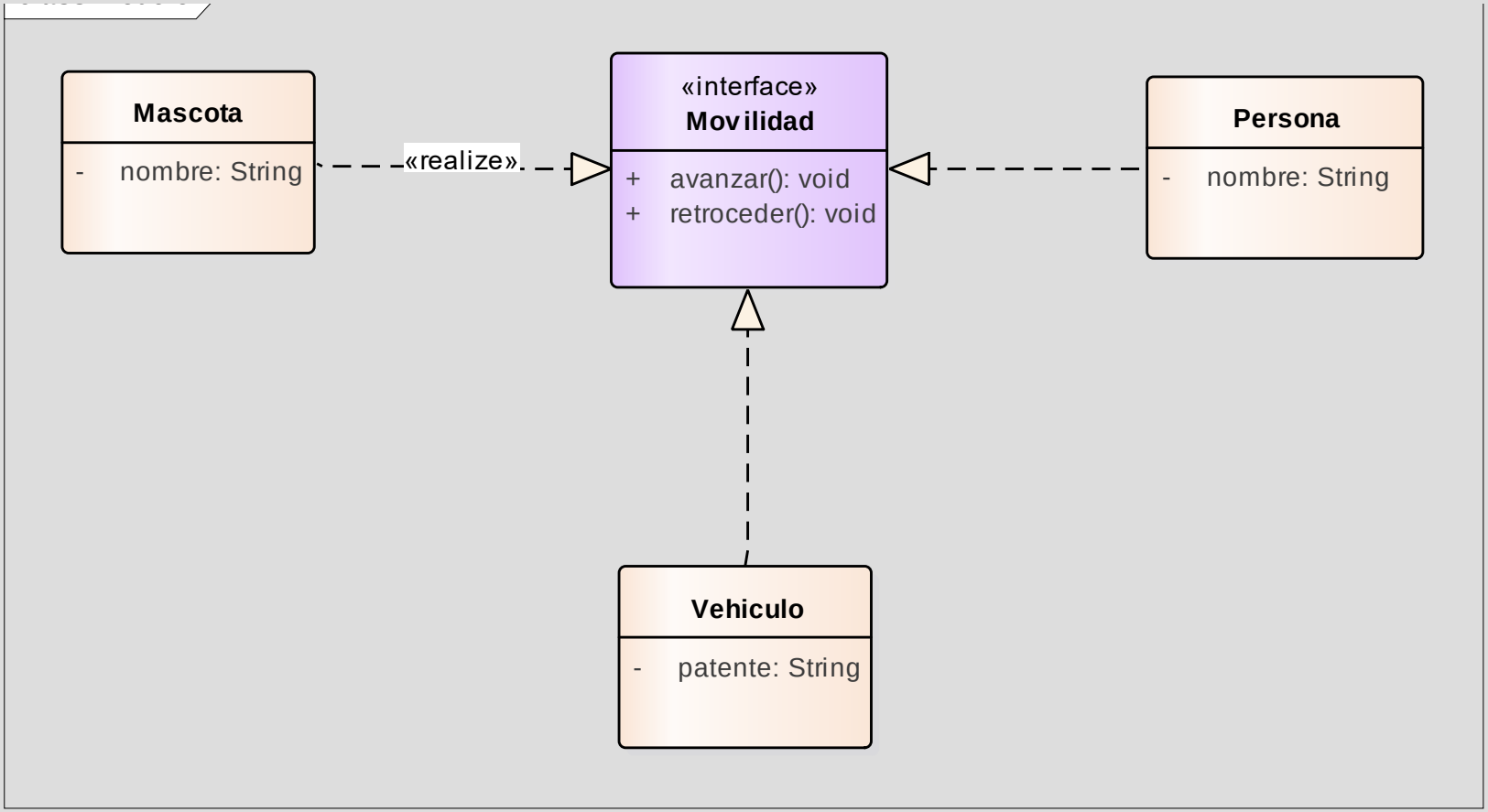
Concepto:

Hay situaciones en la ingeniería de software donde es importante para grupos dispares de programadores acordar un "**contrato**" que explica **a qué debe responder su software.**

Cada grupo debe poder escribir su código sin ningún conocimiento de cómo se escribe el código del otro grupo.

“En términos generales, las interfaces son contratos”

https://docs.oracle.com/javase/tutorial/java/IandI/createinterface.html

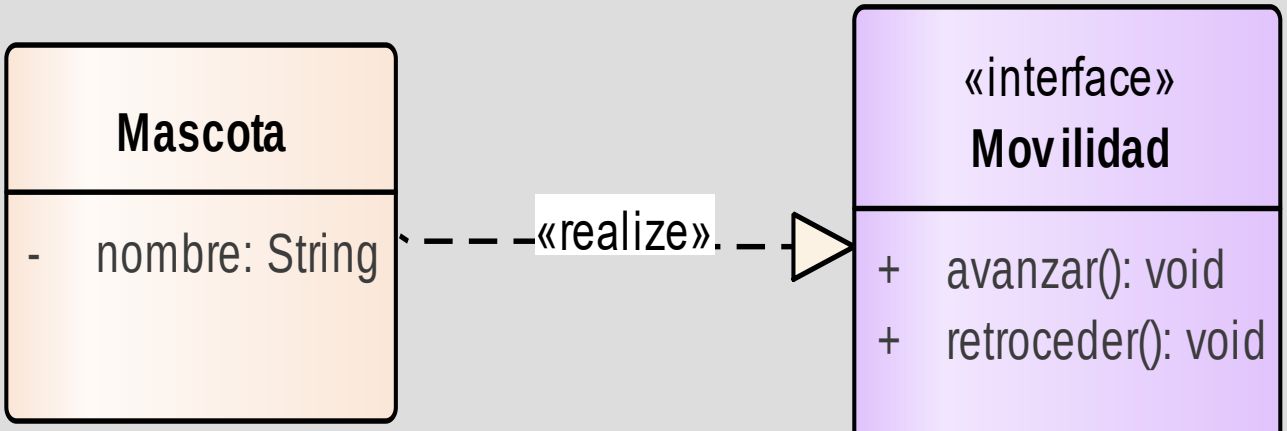

Mascota
- nombre: String
«realize»
«interface»
Movilidad
+ avanzar(): void
+ retroceder(): void
Persona
- nombre: String
Vehiculo
- patente: String

Ejemplo:


```
package clases;

public interface Movilidad {

        void avanzar();
        void retroceder();
}
```

Una Interfaz:

- No puede ser instanciada.
- Sus atributos son implícitamente **public**, **static** y **final**
- Una interfaz especifica **comportamiento** y no el estado de un obj.
- Todos los métodos son **públicos** implícitamente.
  (Se puede omitir el modificador. Solo se escribe por claridad)
- La clase que implemente los métodos de la interface debe declararlos como públicos.


Mascota
- nombre: String
«realize»
«interface»
Movilidad
+ avanzar(): void
+ retroceder(): void

- Todas las clases cuyos objetos se pueden movilizar deben implementar la interfaz **Movilidad.**
- Al implementar una interfaz, una clase **garantiza** que proporcionará una implementación para todos los métodos abstractos declarados en la interfaz **o la clase se declarará abstracta.**
- Una clase puede implementar **una o más** interfaces utilizando la palabra clave **implements** en su declaración.

```
package modelo;

public class Mascota implements Movilidad {
    private String nombre;
      {… constructor, getter y setter …}

   @Override
   public void avanzar() {
      System.out.println(nombre + " avanza
                  caminado en 4 patas.");
   }

   @Override
   public void retroceder() {
      System.out.println("Caminado.");
   }

   public int raza() {…}

}
```

```java
package modelo;

public class Mascota implements Movilidad {
    private String nombre;
      {… constructor, getter y setter …}

   @Override
   public void avanzar() {
      System.out.println(nombre + " avanza
                  caminado en 4 patas.");
   }

   @Override
   public void retroceder() {
      System.out.println("Caminado.");
   }

   public int raza() {…}

}
```

```java
package modelo;

public class Persona implements Movilidad {
    private String nombre;
        {… constructor, getter y setter …}

   @Override
   public void avanzar() {
       System.out.println(nombre + "avanza
                  caminado con dos piernas.");
   }

   @Override
   public void retroceder() {
      System.out.println("Caminado.");
   }

   public String ocupacion() {…}

   public String intereses() {…}
}
```

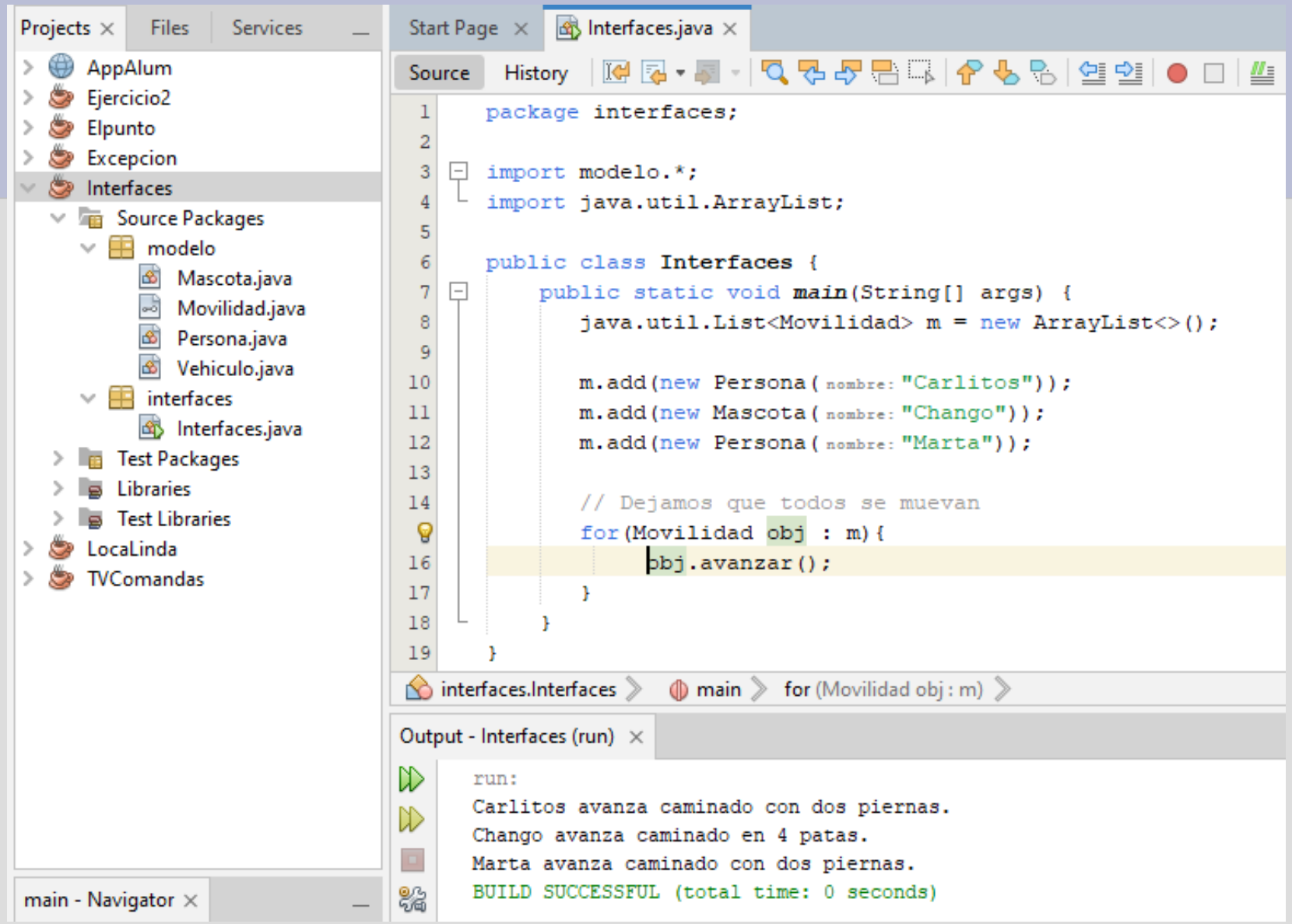
Operaciones:

```
Movilidad m;  // Ok
m = new Movilidad();  // Error
    m = new Persona("Juan");  //Ok
    m = new Mascota("Silvestre");  // Ok
    m = new Vehiculo("AA 170 BB");  // Ok
    m = new Object();  // ....
```

Operaciones:

```
Movilidad m;  // Ok
m = new Movilidad();  // Error
    m = new Persona("Juan");  //Ok
    m = new Mascota("Silvestre");  // Ok
    m = new Vehiculo("AA 170 BB");  // Ok
    m = new Object();  // Error
```

```
package interfaces;
import modelo.*;
import java.util.ArrayList;

public class Interfaces {

public static void main(String[] args) {
        java.util.List<Movilidad> m = new ArrayList<>();

        m.add(new Persona("Juan"));
        m.add(new Mascota("Silvestre"));
        m.add(new Vehiculo("AA 170 BB"));

 //Dejamos que todos se muevan
       mover(m);
}

 private static void mover(Movilidad[] m) {
        for (Movilidad mover : m) {
            mover.avanzar();
        }
    }
}
```


Projects
Files
Services
AppAlum
Ejercicio2
Elpunto
Excepcion
Interfaces
Source Packages
modelo
Mascota.java
Movilidad.java
Persona.java
Vehiculo.java
interfaces
Interfaces.java
Test Packages
Libraries
Test Libraries
LocaLinda
TVComandas
main - Navigator
Start Page
Interfaces.java
Source
History
1 package interfaces;
2
3 import modelo.*;
4 import java.util.ArrayList;
5
6 public class Interfaces {
7 public static void main(String[] args) {
8 java.util.List<Movilidad> m = new ArrayList<>();
9
10 m.add(new Persona(nombre: "Carlitos"));
11 m.add(new Mascota(nombre: "Chango"));
12 m.add(new Persona(nombre: "Marta"));
13
14 // Dejamos que todos se muevan
15 for(Movilidad obj : m){
16 obj.avanzar();
17 }
18 }
19 }
interfaces.Interfaces
main
for (Movilidad obj : m)
Output - Interfaces (run)
run:
Carlitos avanza caminado con dos piernas.
Chango avanza caminado en 4 patas.
Marta avanza caminado con dos piernas.
BUILD SUCCESSFUL (total time: 0 seconds)

Interfaces en Java

- Una interfaz puede tener alcance **public** – o por defecto alcance package.
- Una interfaz es implícitamente **abstract** – Esta obsoleto usar la palabra clave explícitamente.
- Los miembros de una interfaz son tres:
  1. **Atributos** constantes
  2. **Métodos**, que pueden ser: abstractos, estáticos, predeterminados, privados
  3. Clases y/o interfaces **anidadas**.

Para *Java 8* todos los miembros son implícitamente públicos.

***Java 9 ya* permite tener métodos privados**

Ejemplo: Miembros constantes

```
package javax.swing;

public interface SwingConstants {
      int NORTH = 1;
      int NORTH_EAST = 2;
int EAST = 3;
    ...
}
```

```
public static void main(String[] args) {

      System.out.println(javax.swing.SwingConstants.NORTH);

   }
```

Ejemplo: Miembros métodos

```
package clases;

public interface Movilidad {
        void avanzar();
        void retroceder();
}
```

… antes de *java 8*

“Todos los métodos eran abstractos”

El objetivo principal de declarar una interfaz es crear una **especificación abstracta del comportamiento que se espera de algunos objetos**

Nota: pueden incluir parámetros, tipo de retorno y la clausula **throws.**

Ejemplo: Miembros métodos desde Java 8 ...

Métodos estáticos: Antes un método estático no encajaba en la vista de las interfaces como especificaciones abstractas. Ese pensamiento ahora ha evolucionado.

Son implícitamente **públicos**.

Solo se puede invocarse usando el **nombre de la interfaz**:

Porque No se extiende a la clase que la implementa, ni a una subInterface

Metodos por defecto: Agrega funcionalidad a una interfaz y asegura compatibilidad con la versiones existentes. **Provee una implementación predeterminada** para un método de interfaz. Debe etiquetar dicho método con el modificador **default**.

La clase que implementa la interfaz "puede sobrescribir o no el método".

https://docs.oracle.com/javase/tutorial/java/IandI/defaultmethods.html

Ejemplo: Miembros métodos desde Java 9 ...

Métodos privados: Pueden solamente ser usados por lo métodos de la interfaz.

Pueden ser **static** pero… **no** puede ser **default** porque no puede sobrescribirse.

Modificador	Soportado?	Descripción
public static	Si	Soportado desde JDK 8
public abstract	Si	Soportado desde JDK 1
public default	Si	Soportado desde JDK 8
private static	Si	Soportado desde JDK 9
private	Si	Soportado desde JDK 9. Método no abstracto
private abstract	No	Esto no tiene sentido. Un método privado no puede sobrescribirse, y uno abstracto debe ser sobrescrito.
private default	No	Esto no tiene sentido. Un método predeterminado puede sobrescribirse si fuese necesario.

Cuestiones avanzadas

1. **Una clase puede implementar varias interfaces**

   Si una clase implementa dos interfaces, una de las cuales tiene un método predeterminado y la otra un método (predeterminado o no) con el mismo nombre y tipos de parámetros, entonces la clase debe resolver el conflicto.

1. **Una interfaz puede heredar de múltiples interfaces.**

   Si hereda el mismo método de dos interfaces (abstract-default, default - default), debe sobrescribir el método como abstract o default

Ejercicios: LINK

https://docs.oracle.com/javase/tutorial/java/IandI/QandE/interfaces-questions.html

1. ¿Qué métodos debería implementar una clase que implementa la interfaz java.lang.CharSequence?
2. ¿Cuál es el error con la siguiente interfaz?

```
public interface SomethingIsWrong {

    void unMetodo (int aValue) {

        System.out.println("Hola Mundo…");

    }

}
```

1. Que cambios haría en la interfaz anterior?
2. ¿Es válida la siguiente interfaz?

```
public interface Marca {
}
```

5. Considere la siguiente declaración:

```
public interface Saludo{
    default void saludar() {
        System.out.println("Hola");
    }
}
public class Argentina implements Saludo{

}

public class Ingles implements Saludo{
    @Override
    public void saludar() {
        System.out.println(“Hello");
    }
}
```

- Indica posible errores y propone una solución.
- Cual es la salida al ejecutar el código?

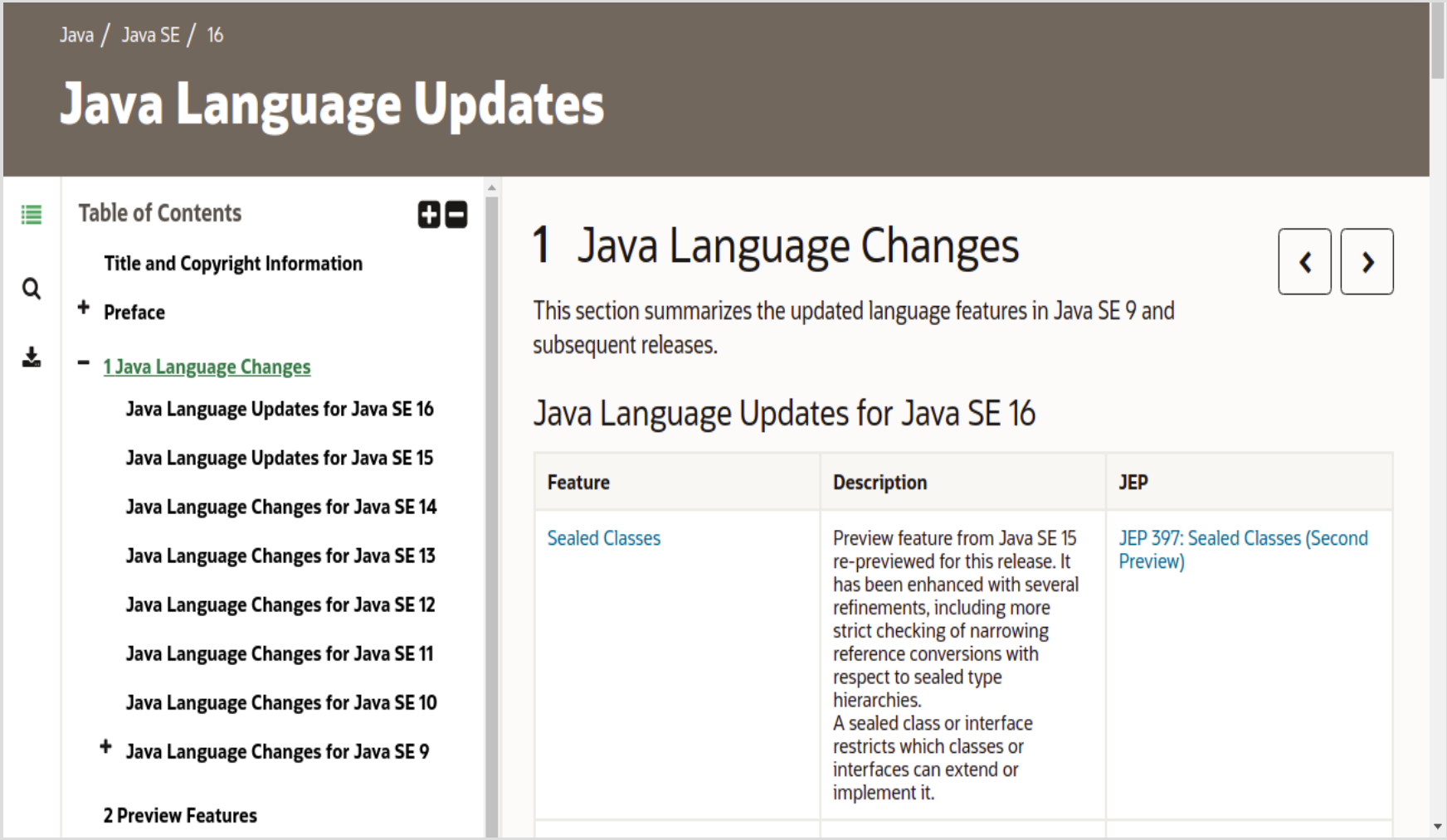
```
Saludo[]  a = {new Argentina(), new Ingles()};
a[0].saludar();
a[1].saludar();
```

6. Escriba una clase que implemente la interfaz CharSequence que se encuentra en el paquete java.lang. Su implementación debe devolver la cadena al revés.

   Escribe un pequeño método principal para evaluar tu clase; asegúrese de llamar a los cuatro métodos.

6. Supongamos que ha escrito un servidor de tiempo que notifica periódicamente a sus clientes sobre la fecha y hora actuales. Escriba una interfaz que el servidor podría usar para imponer un protocolo particular en sus clientes.

Referencia: LINK



Java / Java SE / 16

Java Language Updates

Table of Contents

- Title and Copyright Information
- + Preface
- − 1 Java Language Changes
  - Java Language Updates for Java SE 16
  - Java Language Updates for Java SE 15
  - Java Language Changes for Java SE 14
  - Java Language Changes for Java SE 13
  - Java Language Changes for Java SE 12
  - Java Language Changes for Java SE 11
  - Java Language Changes for Java SE 10
  - + Java Language Changes for Java SE 9
- 2 Preview Features

1 Java Language Changes

This section summarizes the updated language features in Java SE 9 and subsequent releases.

Java Language Updates for Java SE 16

Feature	Description	JEP
Sealed Classes	Preview feature from Java SE 15 re-previewed for this release. It has been enhanced with several refinements, including more strict checking of narrowing reference conversions with respect to sealed type hierarchies. A sealed class or interface restricts which classes or interfaces can extend or implement it.	JEP 397: Sealed Classes (Second Preview)