Orientación a Objetos II

Sesión Conceptual 01

{desafío} latam_

- Objetivo de la sesión
- Activación de conceptos clave

- Conceptualización
- Ejercicios
- Quiz

- Cierre

Inicio

{desafío}
latam_

Objetivos

- Comprender conceptos de polimorfismo para utilizar en listas.
- Reconocer cuándo realizar un casteo para transformar variables o clases.

Desarrollo

{desafío}
latam_

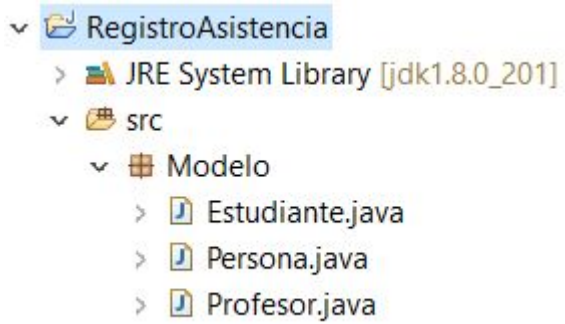
Polimorfismo

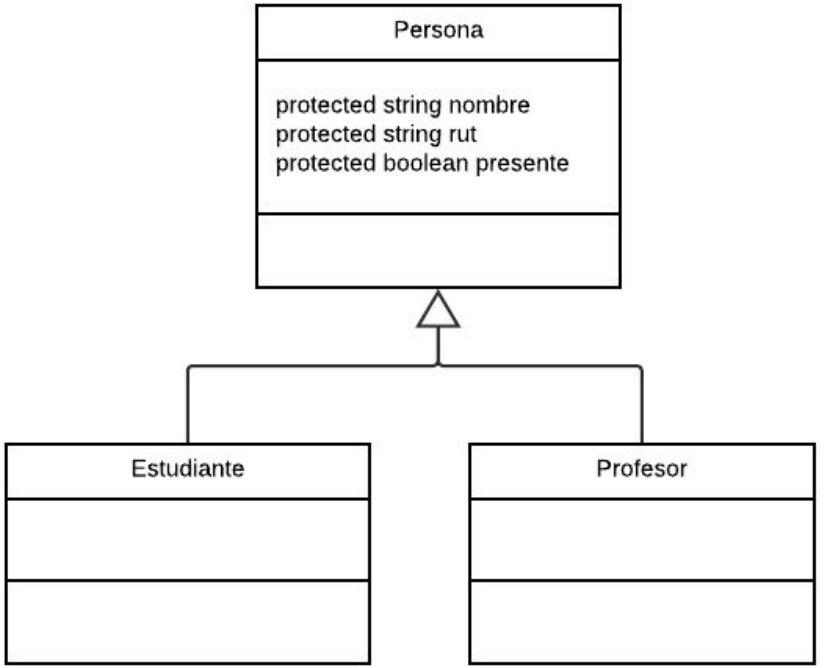
¿Qué es el polimorfismo?

Aplicación Registro de asistencia

{desafío} latam_

Las estructura de la aplicación





La clase Persona

```
package Modelo;

public class Persona {
	protected String rut;
	protected String nombre;
	protected boolean presente;

	public Persona(String rut, String nombre,
boolean presente) {
		super();
		this.rut = rut;
		this.nombre = nombre;
		this.presente = presente;
	}
public String getRut() {
		return rut;
	}
	public void setRut(String rut) {
		this.rut = rut;
	}
```

```
	public String getNombre() {
		return nombre;
	}
	public void setNombre(String nombre) {
		this.nombre = nombre;
	}
	public boolean isPresente() {
		return presente;
	}
	public void setPresente(boolean presente) {
		this.presente = presente;
	}

	@Override
	public String toString() {
		return "Persona [rut=" + rut + ", nombre="
+ nombre + ", presente=" + presente + "]";
	}
}
```

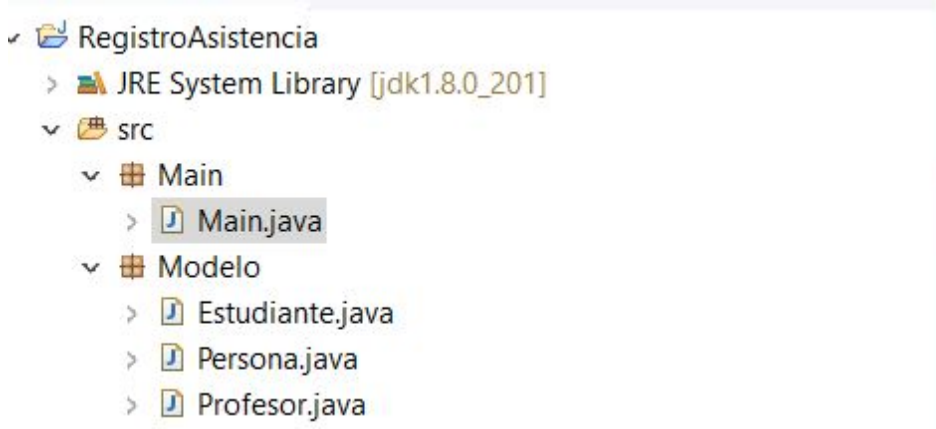
{desafío} latam_

Las clases Profesor y Estudiante

```
public class Profesor extends Persona {
     public Profesor(String rut, String nombre, boolean presente) {
          super(rut, nombre, presente);
     }
}
```

```
public class Estudiante extends Persona {
     public Estudiante(String rut, String nombre, boolean presente) {
          super(rut, nombre, presente);
     }
}
```

Clase Main




```java
package Main;

public class Main {

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

}
```

Método main

```
ArrayList<Estudiante> listaEstudiantes = new ArrayList<>();
ArrayList<Profesor> listaProfesores = new ArrayList<>();

listaEstudiantes.add(new Estudiante("1", "Juan", true));
listaEstudiantes.add(new Estudiante("2", "Andres", true));
listaEstudiantes.add(new Estudiante("3", "Juan", false));
listaProfesores.add(new Profesor("10", "Jose", true));

for(Profesor profesor : listaProfesores) {
      System.out.println(profesor.toString());
}

for(Estudiante estudiante : listaEstudiantes) {
      System.out.println(estudiante.toString());
}
```

Output

```
Persona [rut=10, nombre=Jose, presente=true]
Persona [rut=1, nombre=Juan, presente=true]
Persona [rut=2, nombre=Andres, presente=true]
Persona [rut=3, nombre=Juan, presente=false]
```

Mejorando el método main

```
ArrayList<Persona> lista = new ArrayList<>();

lista.add(new Estudiante("1", "Juan", true));
lista.add(new Estudiante("2", "Andres", true));
lista.add(new Estudiante("3", "Juan", false));
lista.add(new Profesor("10", "Jose", true));

for(Persona individuo : lista) {
     System.out.println(individuo.toString());
}
```

Output

```
Persona [rut=10, nombre=Jose, presente=true]
Persona [rut=1, nombre=Juan, presente=true]
Persona [rut=2, nombre=Andres, presente=true]
Persona [rut=3, nombre=Juan, presente=false]
```

Casteo de clases

Casteo de clases

```
for(Persona individuo : lista) {
     System.out.println(individuo.getClass().getSimpleName());
}
```

Output

```
Estudiante
Estudiante
Estudiante
Profesor
```

Uso de getClass()

```
public class Estudiante extends Persona {

	private double deuda;

	public Estudiante(double deuda, String rut, String nombre, boolean presente) {
		super(rut, nombre, presente);
		this.deuda = deuda;
	}


	public Estudiante(String rut, String nombre, boolean presente) {
		super(rut, nombre, presente);
	}

	public double getDeuda() {
		return deuda;
	}

	public void setDeuda(double deuda) {
		this.deuda = deuda;
	}
}
```

Uso de getClass()

```
public static void main(String[] args) {
     ArrayList<Persona> lista = new ArrayList<>();
     lista.add(new Estudiante(1500, "1", "Juan", true));
     lista.add(new Estudiante(2000, "2", "Andres", true));
     lista.add(new Estudiante(3500, "3", "Juan", false));
     lista.add(new Profesor("10", "Jose", true));
     for(Persona individuo : lista) {
          System.out.println(individuo.getClass().getSimpleName());
     }
}
```

Sintaxis para el casteo de clases

```
Estudiante estudiante = (Estudiante) instanciaPersona;
```

Casteo de clases

```
Integer.parseInt("a");
```

Output

```
Exception in thread "main" java.lang.NumberFormatException: For input string: "a"
     at java.lang.NumberFormatException.forInputString(Unknown Source)
     at java.lang.Integer.parseInt(Unknown Source)
     at java.lang.Integer.parseInt(Unknown Source)
```

Casteo de clases

```
for(Persona individuo : lista) {
      Estudiante estudiante = (Estudiante) individuo;
      System.out.println(individuo.getClass().getSimpleName());
      System.out.println(estudiante.getDeuda());
}
```

Output

```
Estudiante
1500.0
Estudiante
2000.0
Estudiante
3500.0
Exception in thread "main" java.lang.ClassCastException: Modelo.Profesor
cannot be cast to Modelo.Estudiante
      at Main.Main.main(Main.java:18)
```

Casteo de clases

Desde	Hacia	¿ Es posible ?
Superclase	Subclase	Si
Subclase	Superclase	Si
Subclase 1	Subclase 2	No

Comprobando si es posible castear

```
for(Persona p : lista) {
      System.out.println(p.getClass().getSimpleName());
      if(p.getClass() == Estudiante.class) {
            Estudiante est = (Estudiante) p;
            System.out.println("Deuda: "+ est.getDeuda());
      }
}
```

Output

```
Estudiante
Deuda: 1500.0
Estudiante
Deuda: 2000.0
Estudiante
Deuda: 3500.0
Profesor
```

Abstracción I

Las interfaces

- Proveen de una lista de prototipos de métodos, lo que significa que sólo se declara tipo de retorno, nombre y parámetros de entrada.
- Permiten conocer la lista de métodos que tendrán las clases que las implementen sin conocer el comportamiento específico de cada una.
- Los métodos que se declaran en la interfaz deben existir en todas las clases que la implementen, por ende, ayudan a establecer la forma de las clases.

Las interfaces

```
public interface nombreInterfaz{
      void imprimirHola();
}

//Implementación:
public class nombreClase implements nombreInterfaz[, nombreOtraInterfaz]{
      @Override
      public void imprimirHola(){
            System.out.println("hola");
      }
}
```

Polimorfismo con interfaces

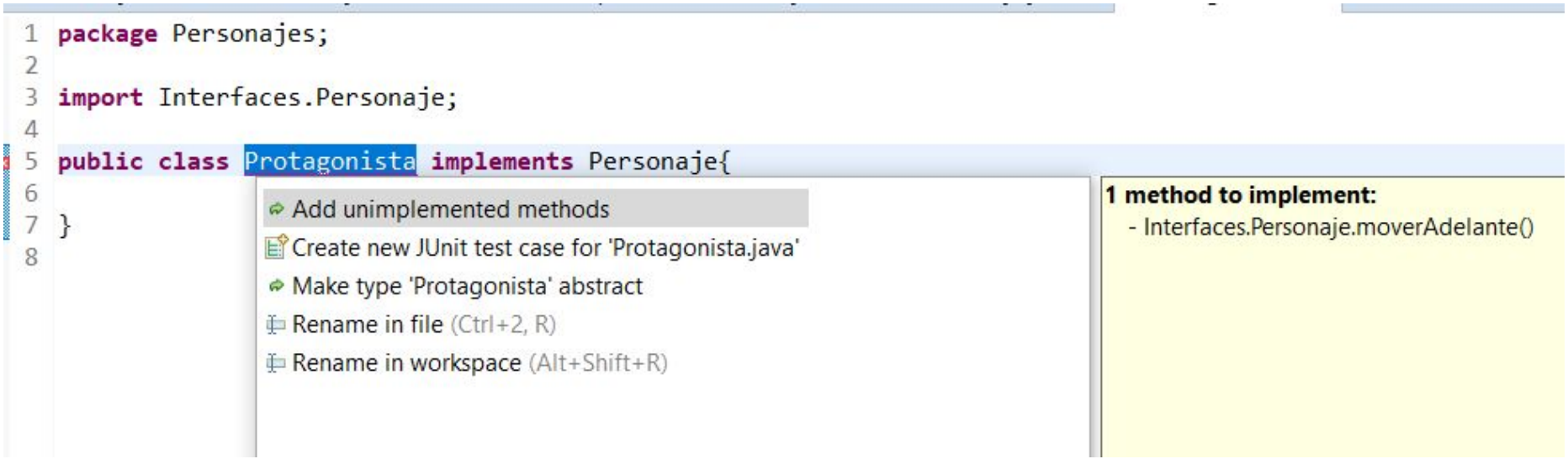
Creamos un proyecto en Eclipse y creamos la siguiente interfaz dentro de un paquete Interfaces:

```
public interface Personaje {
      void mover(int x);
}
```

Luego creamos un paquete Personajes y creamos una implementación de Personaje llamada Protagonista.

```
public class Protagonista implements Personaje{
}
```

Polimorfismo con interfaces



Polimorfismo con interfaces

- Como puedes ver, la implementación del método no es más que una sobre-escritura del mismo.

```
public class Protagonista implements Personaje{

	@Override
	public void mover(int x) {
		// TODO Auto-generated method stub

	}
}
```

Polimorfismo con interfaces

- Ahora agregamos una variable llamada xActual, para indicar la posición del personaje y hacemos que `mover()` modifique esa variable.

```
public class Protagonista implements Personaje{

     private int xActual;

     @Override
     public void mover(int x){
          xActual = xActual + x;
     }
}
```

Polimorfismo con interfaces

- Ahora creamos otra clase llamada Enemigo en el paquete de Protagonista, que implemente la interfaz Personaje:

```
public class Enemigo implements Personaje{

     private int xActual;

     @Override
     public void mover(int x){
          while(xActual < x){
               xActual++;
          }
     }
}
```

Polimorfismo con interfaces

- Interfaz jugador

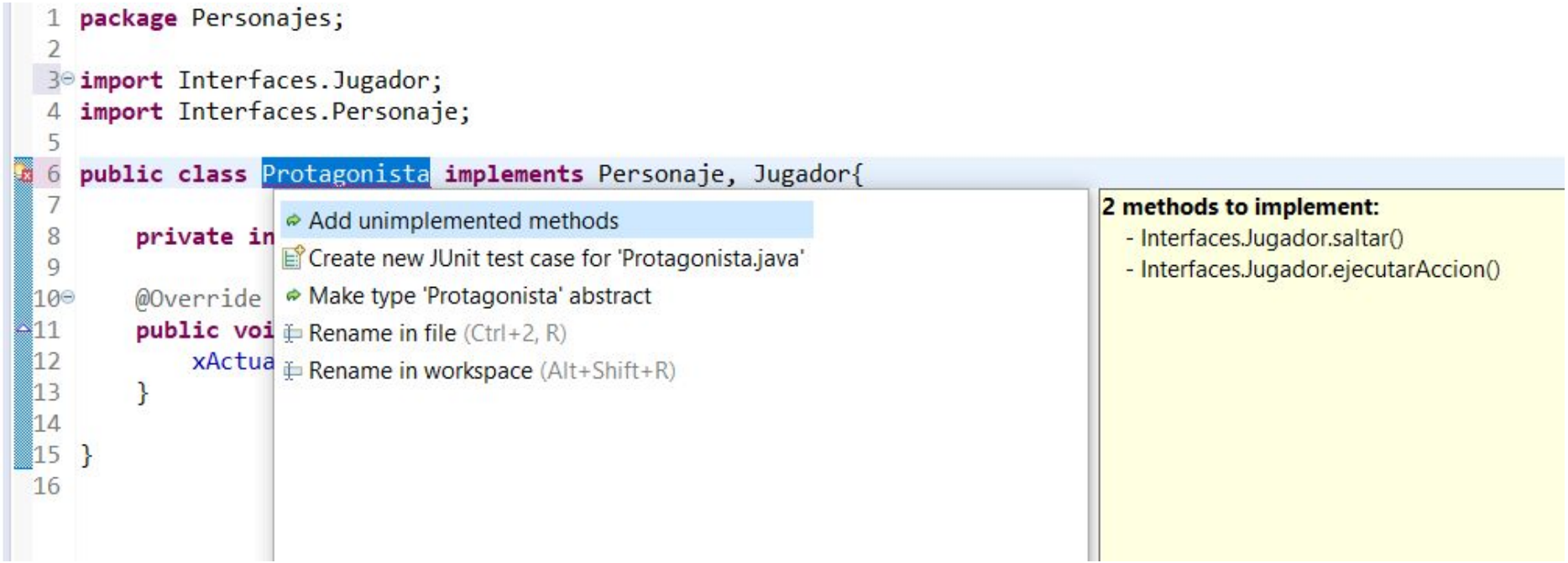
```
package Interfaces;

public interface Jugador {

     void saltar();
     void ejecutarAccion(String accion);

}
```

Polimorfismo con interfaces



{desafío} latam_

Polimorfismo con interfaces

```
    private int xActual;

    @Override
    public void mover(int x){
        xActual = xActual + x;
    }

    @Override
    public void saltar() {
        // TODO Auto-generated method stub

    }

    @Override
    public void ejecutarAccion(String accion) {
        // TODO Auto-generated method stub

    }
```

Modificando el método saltar de Protagonista

```
private int yActual = 1;

@Override
public void saltar() {
      //Aumentamos hasta 5
      while(yActual < 5){
            yActual++;
      }
      //Cuando sea 5, disminuimos a 1 nuevamente
      while(yActual > 1){
            yActual--;
      }
}
```

Modificando el método saltar de Protagonista

```
@Override
public void ejecutarAccion(String accion) {
     if(accion.equals("saltar") && yActual == 1){
          saltar()
     }else if(accion.equals("avanzar")){
          mover(1);
     }
}
```

¡ Polimorfismo implementado !

Cierre

{desafío}
latam_

¿Existe algún concepto que no hayas comprendido?

Volvamos a revisar los conceptos que más te hayan costado antes de seguir adelante

Reflexionemos

{desafío} latam_

{desafío} latam_

Academia de talentos digitales