HERRAMIENTAS PARA EL DESARROLLO DE SOFTWARE

Sesión 08

Interfaces

Computación e Informática

2017 - I

Eric Gustavo Coronel Castillo

gcoronelc.blogspot.com

Encuéntranos en:

INSTITUTO DE EMPRENDEDORES

LOGRO DE LA SESIÓN

Utilizar interfaces para diseñar y programar soluciones a requerimientos de software.

Computación e Informática

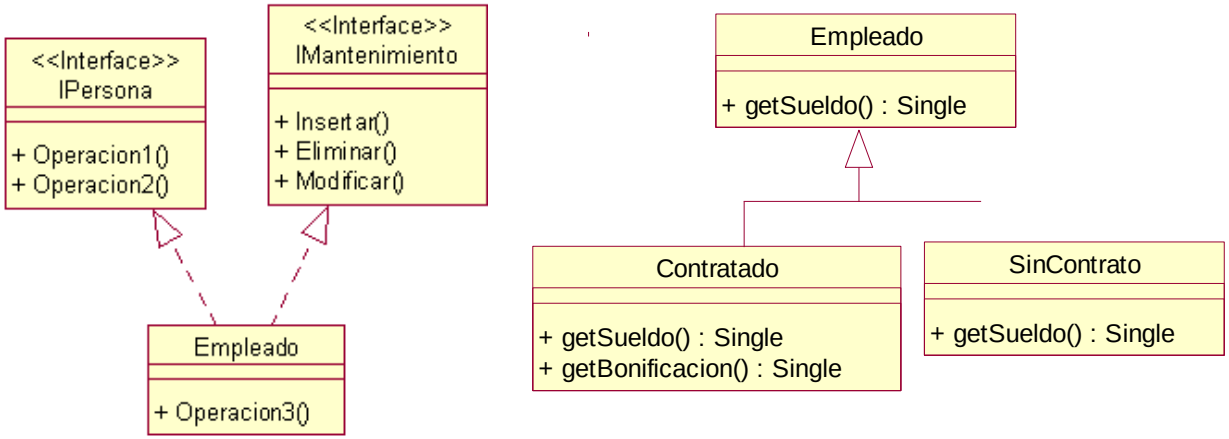
2017 - I

Eric Gustavo Coronel Castillo

gcoronelc.blogspot.com

OBJETIVOS

- Aplicar interfaces en el diseño de componentes software.
- Aplicar el polimorfismo en el diseño de componentes software

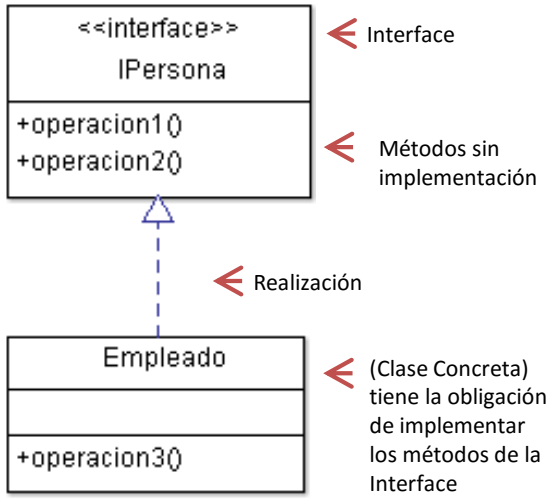


INTERFACE

- Solo contienen operaciones (métodos) sin implementación, es decir solo la firma (signature).
- Las clases son las encargadas de implementar las operaciones (métodos) de una o varias Interfaces (*Herencia múltiple*).
- Se dice que se crean Interface cuando sabemos que queremos y no sabemos como hacerlo y lo hará otro o lo harán de varias formas (*polimorfismo*).

```
public interface IPersona {
   public void operacion1();
   public void operacion2();
}
```

```
public class Empleado implements IPersona {
   public void operacion1() {
      //implementa el método de la interface
   }
   public void operacion2() {
      //implementa el método de la interface
   }
   public void operacion3() {
      //implementación
   }
}
```



INTERFACE

Ejemplo de Herencia múltiple de Interface.

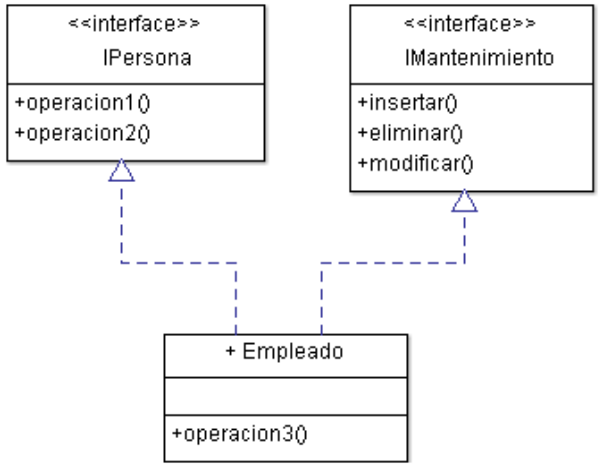
```
public interface IPersona {
   public void operacion1();
   public void operacion2();
}
```

```
public interface IMantenimiento {
   public void insertar();
   public void eliminar();
   public void modificar();
}
```

```
public class Empleado
implements Ipersona, IMantenimiento {

    // Implementa los métodos de las interfaces
    // . . .
    // . . .
    // . . .

}
```



CLASE CONCRETA, ABSTRACTA E INTERFACE

CARACTERISTICA	CLASE CONCRETA	CLASE ABSTRACTA	INTERFACE
HERENCIA	extends (simple)	extends (simple)	implements (múltiple)
INSTANCIABLE	Si	No	No
IMPLEMENTA	Métodos	Algunos métodos	Nada
DATOS	Se permite	Se permite	No se permite*

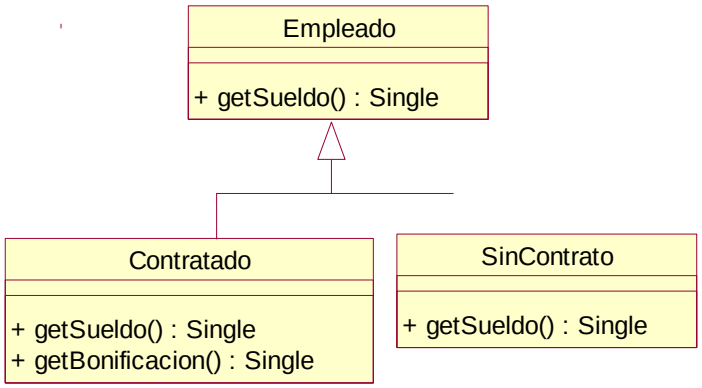
* Las variables que se declaran en una interface son implícitamente estáticas, finales y publicas.

POLIMORFISMO

- Se dice que existe polimorfismo cuando un método de una clase es implementado de varias formas en otras clases.
- Algunos ejemplos de Polimorfismos de herencia son: *sobre-escritura*, *implementación* de métodos abstractos (clase abstracta e interface).
- Es posible apuntar a un objeto con una variable de tipo de *clase padre* (supercalse), esta sólo podrá acceder a los miembros (campos y métodos) que le pertenece.

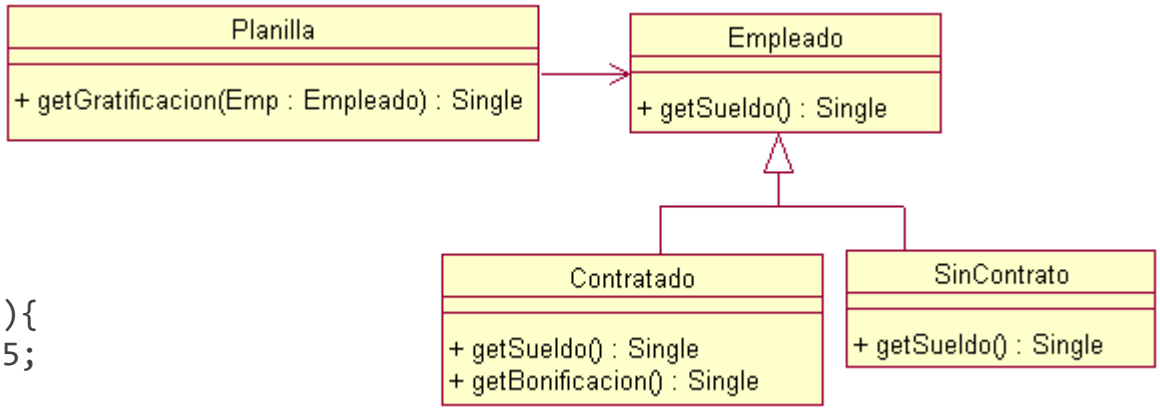
```
// Variable de tipo Empleado y apunta a un
// objeto de tipo Contratado.
Empleado objEmp = new Contratado();

// Invocando sus métodos
double S = objEmp.getSueldo();        //OK
double B = objEmp.getBonificacion();
//Error
```



POLIMORFISMO

- El método **getGratificacion** puede recibir objetos de tipo **Empleado** o subtipos a este.
- Cuando invoque el método **getSueldo** se ejecutará la versión correspondiente al objeto referenciado.

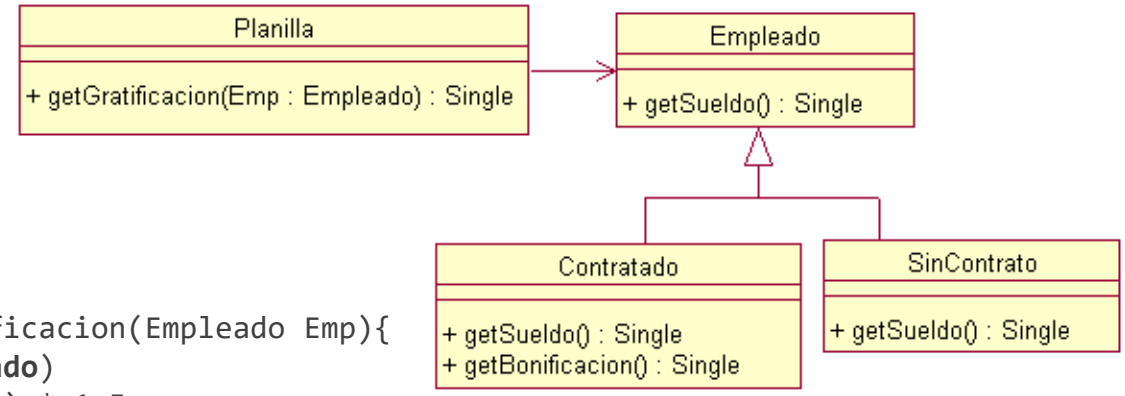



```
public class Planilla {
  public static double
  getGratificacion(Empleado Emp){
    return Emp.getSueldo() * 1.5;
  }
}
```

```
// Usando la clase Planilla
double G1 = Planilla.getGratificacion(new Contratado());
double G2 = Planilla.getGratificacion(new SinContratado());
```

OPERADOR INSTANCEOF

Este operador permite verificar si el objeto es instancia de un tipo especifico.




```
public class Planilla {
   public static double getGratificacion(Empleado Emp){
      if (Emp instanceof Contratado)
            return Emp.getSueldo() * 1.5;
      if (Emp instanceof SinContratado)
            return Emp.getSueldo() * 1.2;
   }
}
```

```
//Usando la clase Planilla
double G1 = Planilla.getGratificacion(new Contratado());
double G2 = Planilla.getGratificacion(new SinContratado());
```

CASTING

- Para restablecer la funcionalidad completa de un objeto, que es de un tipo y hace referencia a otro tipo, debe realizar una conversión (Cast).
- **UpCasting:** Conversión a clases superiores de la jerarquía de clases (Herencia), es automático (conversión implícita), basta realizar la asignación.
- **DownCasting:** Conversión hacia abajo, es decir hacia las subclases de la jerarquía (Herencia), es recomendable realizar Cast (conversión explicita), si no es compatible genera un error (Excepción).

```
// UpCasting (Conversión implícita)
Contratado a = new Contratado();
Empleado b = a;

// DownCasting (Conversión explícita)
Empleado a = new Contrtado();
Contratado b = (Contratado)a;

// Error de compilación
SinContrato a = new SinContrato();
Contratado b = (Contratado)a;
```

CONTROL DE ACCESO

- Se conoce 4 formas de controlar el acceso a los campos (atributos) y métodos (operaciones) de las clases.
  - **private** (**-**): Acceso sólo dentro de la clase.
  - **package (~) :** Acceso sólo dentro del paquete.
  - **protected** (#): Acceso en la clase, dentro del paquete y en subclases (herencia dentro o fuera del paquete).
  - **public** (**+**): Acceso desde cualquier parte.

Acceso / Visibilidad	Misma Clase	Mismo Paquete	SubClases y Mismo Paquete	Universal
public (+)	Sí	Sí	Sí	Sí
protected (#)	Sí	Sí	Sí	No
package (~)	Sí	Sí	No	No
private (-)	Sí	No	No	No

PROYECTO EJEMPLO

La institución educativa **EduTec** cuenta con dos tipos de trabajadores: Empleados y Docentes.

Los empleados cuentan con un sueldo fijo y depende del cargo que ocupa, según la siguiente tabla:

CARGO	SUELDO
Coordinador	5,000.00
Asistente	4,000.00
Secretaria	3,000.00

El sueldo del docente está en función de las horas que dicta, el pago por hora es de 120 Nuevos Soles.

El departamento de recursos humanos necesita una aplicación para calcular la bonificación que se debe pagar a cada trabajador según el siguiente cuadro:

TRABAJADOR	BONIFICACIÓN
Empleado	100% del Sueldo
Docente	70% del Sueldo

INSTITUTO DE
EMPRENDEDORES

USIL