7.C. Herencia.

1. Herencia.

1.5. Redefinición de métodos heredados.

Una clase puede **redefinir** algunos de los métodos que ha heredado de su **clase base**. En tal caso, el nuevo método (**especializado**) sustituye al **heredado**. Este procedimiento también es conocido como de **sobrescritura de métodos**.

En cualquier caso, aunque un método sea **sobrescrito** o **redefinido**, aún es posible acceder a él a través de la referencia `super`, aunque sólo se podrá acceder a métodos de la **clase padre** y no a métodos de clases superiores en la **jerarquía de herencia**.

Los **métodos redefinidos** pueden **ampliar su accesibilidad** con respecto a la que ofrezca el método original de la **superclase**, pero **nunca restringirla**. Por ejemplo, si un método es declarado como `protected` o **de paquete** en la clase base, podría ser redefinido como `public` en una clase derivada.

Los **métodos estáticos** o de clase no pueden ser sobrescritos. Los originales de la clase base permanecen inalterables a través de toda la **jerarquía de herencia**.

En el ejemplo de la clase `Alumno`, podrían redefinirse algunos de los métodos heredados. Por ejemplo, imagina que el método `getApellidos` devuelva la cadena "Alumno:" junto con los apellidos del alumno. En tal caso habría que rescribir ese método para realizara esa modificación:

```
public String getApellidos () {
```

```
    return "Alumno: " + apellidos;
```

```
}
```

Cuando sobrescribas un método heredado en Java puedes incluir la **anotación** `@Override`. Esto indicará al compilador que tu intención es **sobrescribir el método de la clase padre**. De este modo, si te equivocas (por ejemplo, al escribir el nombre del método) y no lo estás realmente sobrescribiendo, el compilador producirá un error y así podrás darte cuenta del fallo. En cualquier caso, no es necesario indicar `@Override`, pero puede resultar de ayuda a la hora de localizar este tipo de errores (crees que has sobrescrito un **método heredado** y al confundirte en una letra estás realmente creando un nuevo método diferente). En el caso del ejemplo anterior quedaría:

```
@Override
```

```
public String getApellidos ()
```

Autoevaluación

Dado que el método `finalize()` de la clase `Object` es `protected`, el método `finalize()` de cualquier clase que tú escribas podrá ser `public`, `private` o `protected`. ¿Verdadero o Falso?

- ○ Verdadero
- ○ Falso

Ejercicio resuelto

Dadas las clases `Persona`, `Alumno` y `Profesor` que has utilizado anteriormente, redefine el método `getNombre` para que devuelva la cadena "`Alumno:` ", junto con el nombre del alumno, si se trata de un objeto de la clase `Alumno` o bien "`Profesor` ", junto con el nombre del profesor, si se trata de un objeto de la clase `Profesor`.

Solución:

1. **Clase `Alumno`.**

Al heredar de la clase `Persona` tan solo es necesario escribir métodos para los nuevos atributos (**métodos especializados** de acceso a los **atributos especializados**), pues los **métodos genéricos** (de acceso a los **atributos genéricos**) ya forman parte de la clase al haberlos heredado. Esos son los métodos que se implementaron en el ejercicio anterior (`getGrupo`, `setGrupo`, etc.).

Ahora bien, hay que escribir otro método más, pues tienes que redefinir el método `getNombre` para que tenga un comportamiento un poco

diferente al **getNombre** que se hereda de la clase base **Persona**:

```
// Método getNombre

@Override

public String getNombre (){

            return “Alumno: “ + this.nombre;

}
```

En este caso podría decirse que se “renuncia” al método heredado para redefinirlo con un comportamiento más especializado y acorde con la clase derivada.

2. Clase **Profesor**.

Seguimos exactamente el mismo procedimiento que con la clase **Alumno** (redefinición del método **getNombre**).

```
// Método getNombre

@Override

public String getNombre (){

            return “Profesor: “ + this.nombre;

}
```