

CURSO DE PROGRAMACIÓN FULL STACK

HERENCIA

PARADIGMA ORIENTADO A OBJETOS



GUÍA DE HERENCIA

HERENCIA

La herencia es una relación fuerte entre dos clases donde una clase es padre de otra.

La herencia es un pilar importante de la POO. Es el mecanismo mediante el cual una clase es capaz de heredar todas las características (atributos y métodos) de otra clase.

Las propiedades comunes se definen en la superclase (clase padre) y las subclases heredan estas propiedades (Clase hija). En esta relación, la frase "Un objeto es un tipo de una superclase" debe tener sentido, por ejemplo: un perro es un tipo de animal, o también, una heladera es un tipo de electrodoméstico.

La herencia apoya el concepto de "reutilización", es decir, cuando queremos crear una nueva clase y ya existe una clase que incluye parte del código que queremos, podemos utilizar esa clase que ya tiene el código que queremos y hacer de la nueva clase una subclase. Al hacer esto, estamos reutilizando los campos y métodos de la clase existente.

La manera de usar herencia es a través de la palabra **extends**.

```
class subclase extends superclase {  
    // atributos y métodos  
}
```

HERENCIA Y ATRIBUTOS

La subclase (Hija) como hemos dicho recibe todos los atributos de la superclase (Madre), y además la subclase puede tener atributos propios.

```
Class Persona {  
    protected String nombre;  
    protected Integer edad;  
    protected Integer documento;  
}  
  
Class Alumno extends Persona {  
    private String materia;  
}
```

El siguiente programa crea una superclase llamada Persona, que crea personas según su nombre, edad y documento, y una subclase llamada Alumno, que recibe todos los atributos de Persona. De esta manera se piensa que los atributos de alumno son nombre, edad y documento, que son propios de cualquier Persona y materia que sería específico de cada Alumno. Usualmente la superclase suele ser un concepto muy general y abstracto, para que pueda utilizarse para varias subclases.

En la superclase podemos observar que los atributos están creados con el modificador de acceso **protected** y no **private**. Esto es porque el modificador de acceso **protected** permite que las subclases puedan acceder a los atributos sin la necesidad de getters y setters.

Los atributos se trabajan como protected también, porque una subclase no hereda los miembros privados de su clase principal. Sin embargo, si la superclase tiene métodos públicos o protegidos (como getters y setters) para acceder a sus campos privados, estos también pueden ser utilizados por la subclase. Entonces, para evitar esto, usamos atributos protected.

Visibilidad	Public	Private	Protected	Default
Desde la misma Clase	SI	SI	SI	SI
Desde cualquier Clase del mismo Paquete	SI	NO	SI	SI
Desde una Subclase del mismo Paquete	SI	NO	SI	SI
Desde una Subclase fuera del mismo Paquete	SI	NO	SI, a través de la herencia	NO
Desde cualquier Clase fuera del Paquete	SI	NO	NO	NO

HERENCIA Y CONSTRUCTORES

Una diferencia entre los constructores y los métodos es que los constructores no se heredan, pero los métodos sí. Todos los constructores definidos en una superclase pueden ser usados desde constructores de las subclases a través de la palabra clave super. La palabra clave super es la que me permite elegir qué constructor, entre los que tiene definida la clase padre, es el que debo usar. Si la superclase tiene definido el constructor vacío y no colocamos una llamada explícita super, se llamará el constructor vacío de la superclase.

```
Class Persona {  
    public Persona(String nombre, Integer edad, Integer documento){  
        this.nombre = nombre;  
        this.edad = edad;  
        this.documento = documento;  
    }  
}  
  
Class Alumno extends Persona {  
    public Alumno(String materia, String nombre, Integer edad, Integer documento){  
        super.(nombre, edad, documento);  
        this.materia = materia;  
    }  
}
```

En el ejemplo podemos ver que el constructor de la clase Alumno utiliza la palabra clave super para llamar al constructor de la superclase y de esa manera utilizarlo como constructor propio y además sumarle su atributo materia.

La palabra clave super nos sirve para hacer referencia o llamar a los atributos, métodos y constructores de la superclase en las subclases.

```
super.atributoClasePadre;
```

```
super.metodoClasePadre;
```

HERENCIA Y MÉTODOS

Todos los métodos accesibles o visibles de una superclase se heredan a sus subclases. Pero, ¿qué ocurre si una subclase necesita que uno de sus métodos heredados funcione de manera diferente?

Los métodos heredados pueden ser redefinidos en las clases hijas. Este mecanismo se lo llama sobreescritura. La sobreescritura permite que las clases hijas sumen sus particulares en torno al funcionamiento y agrega coherencia al modelo. Esto se logra poniendo la anotación @Override arriba del método que queremos sobrecribir, el método debe llamarse igual en la subclase como en la superclase.

```
Class Persona {  
    public void codear(){  
        System.out.println("Una persona comun no codea");  
    }  
}  
  
Class Alumno extends Persona {  
    @Override  
    public void codear(){  
        System.out.println("Está aprendiendo");  
    }  
}
```

En el ejemplo podemos ver que tenemos el mismo método en la clase Persona, que en la clase Alumno, el método nos va a informar cuales son sus capacidades para codear según la clase que llamemos. Por lo que en la clase Alumno, cuando heredamos el método lo sobreescribimos para cambiar su funcionamiento, y hacer que diga algo distinto al método de Persona.

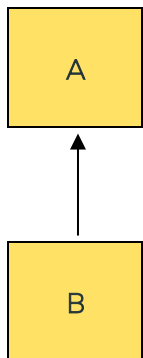
En los métodos de la superclase, también podemos hacer que tengan un modificador de acceso protected, esto hace que los únicos que puedan invocar a ese método sean las subclases.

POLIMORFISMO

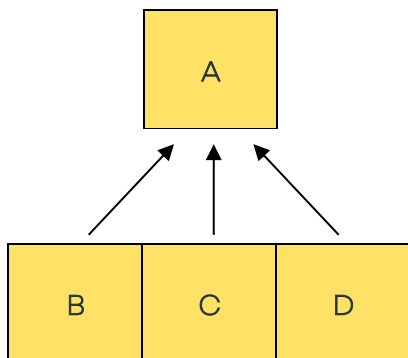
El término polimorfismo es una palabra de origen griego que significa “muchas formas”. Este término se utiliza en POO para referirse a la propiedad por la que es posible enviar mensajes sintácticamente iguales a objetos de tipos distintos, es decir, que la misma operación se realiza en las clases de diferente forma. Estas operaciones tienen el mismo significado y comportamiento, pero internamente, cada operación se realiza de diferente forma. El único requisito que deben cumplir los objetos que se utilizan de manera polimórfica es saber responder al mensaje que se les envía. Esto hace referencia a la idea de que podemos tener un método definido en la superclase y que las subclases tengan el mismo método, pero con distintas funcionalidades

TIPOS DE HERENCIA

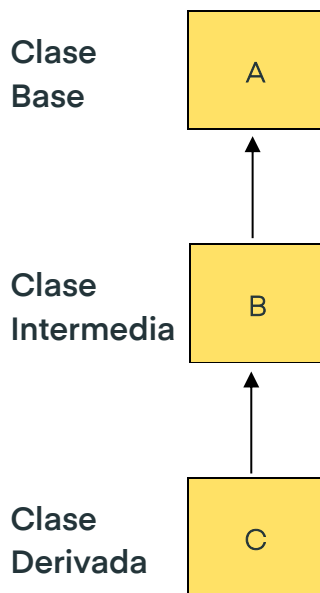
Herencia única: en la herencia única, las subclases heredan las características de solo una superclase. En la imagen a continuación, la clase A sirve como clase base para la clase derivada B.



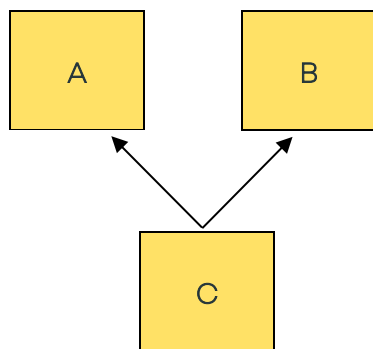
Herencia Jerárquica: en la herencia jerárquica, una clase sirve como una superclase (clase base) para más de una subclase. En la imagen inferior, la clase A sirve como clase base para la clase derivada B, C y D.



Herencia Multinivel: en la herencia multinivel, una clase derivada heredará una clase base y, además, la clase derivada también actuará como la clase base de otra clase. En la imagen inferior, la clase A sirve como clase base para la clase derivada B, que a su vez sirve como clase base para la clase derivada C. En Java, una clase no puede acceder directamente a los miembros de los “abuelos”.



Herencia Múltiple (a través de interfaces): en Herencia múltiple, una clase puede tener más de una superclase y heredar características de todas las clases principales. Tenga en cuenta que Java no admite herencia múltiple con clases. En Java, podemos lograr herencia múltiple **solo a través de Interfaces**. En la imagen a continuación, la Clase C se deriva de la interfaz A y B.



MODIFICADORES DE CLASES Y METODOS

Clases Finales

El modificador final puede utilizarse también como modificador de clases. Al marcar una clase como final impedimos que se generen hijos a partir de esta clase, es decir, cortamos la jerarquía de herencia.

```
public final class Animal{ }
```

Métodos Finales

El modificador final puede utilizarse también como modificador de métodos. La herencia nos permite reutilizar el código existente y uno de los mecanismos es la crear una subclase y sobrescribir alguno de los métodos de la clase padre. Cuando un método es marcado como final en una clase, evitamos que sus clases hijas puedan sobrescribir estos métodos.

```
public final void método(){ }
```

Clases Abstractas

En java se dice que una clase es abstracta cuando no se permiten instancias de esa clase, es decir que no se pueden crear objetos. Nosotros haríamos una clase abstracta por dos razones. Usualmente las clases abstractas suelen ser las superclases, esto lo hacemos porque creemos que la superclase o clase padre, no debería poder instanciarse. Por ejemplo, si tenemos una clase Animal, el usuario no debería poder crear un Animal, sino que solo debería poder instanciar solo objetos de las subclases

```
public abstract class Animal { }
```

Otra razón es porque decidimos hacer métodos abstractos en nuestra superclase. Cuando una clase posee al menos un método abstracto esa clase necesariamente debe ser marcada como abstracta.

Métodos Abstractos

Un método abstracto es un método declarado, pero no implementado, es decir, es un método del que solo se escribe su nombre, parámetros, y tipo devuelto, pero no su código de implementación. Estos métodos se heredan y se sobrescriben por las clases hijas quienes son las responsables de implementar sus funcionalidades. ¿Qué utilidad tiene un método abstracto? Podemos ver un método abstracto como una palanca que fuerza dos cosas: la primera, que no se puedan crear objetos de una clase. La segunda, que todas las subclases sobrescriban el método declarado como abstracto.

```
public abstract void codear();
```

Hija:

```
@Override
```

```
Public void codear(){
```

```
System.out.println("Está aprendiendo");
```

```
}
```

INTERFACES

Una interfaz en Java es una clase que contiene todos sus métodos abstractos y sus atributos son sólo constantes. En las interfaces se especifica qué se debe hacer, pero no su implementación o funcionalidad. Las clases que implementen una interface serán las responsables de describir la lógica del comportamiento de los métodos, las interfaces se implementan usando la palabra **implements**. La principal diferencia entre interface y abstract es que una interface proporciona un mecanismo de encapsulación de los métodos sin forzar a la utilización de una herencia.

```
public interface Interfaz {
```

```
public final int constante = 10;
```

```
public void metodo();
```

```
}
```

```
public class Clase implements Interfaz {  
    @Override  
    Public void metodo(){  
        System.out.println("Implementacion del método");  
        System.out.println("La constante tiene un valor de " + constante);  
    }  
}
```


PREGUNTAS DE APRENDIZAJE

- 1) ¿Qué palabra se usa para generar la herencia entre clases?
 - a) implements
 - b) super
 - c) this
 - d) extends

- 2) ¿Qué código de los siguientes tiene que ver con la herencia?
 - a) public class Componente extends Producto
 - b) public class Componente inherit Producto
 - c) public class Componente implements Producto
 - d) public class Componente belong to Producto

- 3) ¿La superclase es la clase?
 - a) Madre
 - b) Hija
 - c) Nieta
 - d) Es una interfaz

- 4) Cual de estos componentes de la superclase no hereda la subclase
 - a) Atributos
 - b) Métodos
 - c) Getter/Setters
 - d) Constructores

- 5) La superclase debe ser abstracta
 - a) Siempre
 - b) Nunca
 - c) Depende de la situación
 - d) Cuando implementa una interfaz

- 6) Podemos acceder a los atributos de una superclase desde una subclase gracias al modificador de acceso:
 - a) public
 - b) private
 - c) protected
 - d) final

- 7) ¿De cuántas clases se puede derivar en Java?
 - a) Tres clases
 - b) Dos clases
 - c) Una clase
 - d) Cinco clases

8) Una clase que termina la cadena de una herencia se la declarara como:

- a) final
- b) abstract
- c) public
- d) Ninguna de las anteriores

9) ¿Qué palabra se usa para implementar una interfaz?

- a) implements
- b) super
- c) this
- d) extends

10) ¿Qué código asociarías a una Interfaz en Java?

- a) public class Componente interface Product
- b) Componente cp = new Componente (interfaz)
- c) public class Componente implements Producto
- d) Componente cp = new Componente.interfaz

EJERCICIOS DE APRENDIZAJE

En este módulo de POO, vamos a empezar a ver cómo dos o más clases pueden relacionarse entre sí mediante una herencia de clases.

VER VIDEOS:

- A. [Herencia 1](#)
- B. [Herencia 2](#)

1. Tenemos una clase padre Animal junto con sus 3 clases hijas Perro, Gato, Caballo. La clase Animal tendrá como atributos el nombre, alimento, edad y raza del Animal.

Crear un método en la clase Animal a través del cual cada clase hija deberá mostrar luego un mensaje por pantalla informando de que se alimenta. Generar una clase Main que realice lo siguiente:

```
1 public class Main {
2
3     public static void main(String[] args) {
4
5         //-->Declaracion del objeto PERRO
6         Animal perro = new Perro("Stich", "Carnivoro", 15, "Doberman");
7         perro.Alimentarse();
8         //-->Declaracion de otro objeto PERRO
9         Perro perro1 = new Perro("Teddy", "Croquetas", 10, "Chihuahua");
10        perro1.Alimentarse();
11
12
13        //-->Declaracion del objeto Gato
14        Animal gato = new Gato("Pelusa", "Galletas", 15, "Siames");
15        gato.Alimentarse();
16        //-->Declaracion del objeto Caballo
17        Animal caballo = new Caballo("Spark", "Pasto", 25, "Fino");
18        caballo.Alimentarse();
19
20    }
21 }
```

VER VIDEOS:

- A. [Interfaces](#)
- B. [Clases Abstractas](#)
- C. [Polimorfismo](#)

2. Crear una superclase llamada Electrodoméstico con los siguientes atributos: precio, color, consumo energético (letras entre A y F) y peso.

Los constructores que se deben implementar son los siguientes:

- Un constructor vacío.
- Un constructor con todos los atributos pasados por parámetro.

Los métodos a implementar son:

- Métodos getters y setters de todos los atributos.

- Método `comprobarConsumoEnergetico(char letra)`: comprueba que la letra es correcta, sino es correcta usara la letra F por defecto. Este método se debe invocar al crear el objeto y no será visible.
- Método `comprobarColor(String color)`: comprueba que el color es correcto, y si no lo es, usa el color blanco por defecto. Los colores disponibles para los electrodomésticos son blanco, negro, rojo, azul y gris. No importa si el nombre está en mayúsculas o en minúsculas. Este método se invocará al crear el objeto y no será visible.
- Método `crearElectrodomestico()`: le pide la información al usuario y llena el electrodoméstico, también llama los métodos para comprobar el color y el consumo. Al precio se le da un valor base de \$1000.
- Método `precioFinal()`: según el consumo energético y su tamaño, aumentará el valor del precio. Esta es la lista de precios:

<u>LETRA</u>	<u>PRECIO</u>
A	\$1000
B	\$800
C	\$600
D	\$500
E	\$300
F	\$100

<u>PESO</u>	<u>PRECIO</u>
Entre 1 y 19 kg	\$100
Entre 20 y 49 kg	\$500
Entre 50 y 79 kg	\$800
Mayor que 80 kg	\$1000

A continuación se debe crear una subclase llamada Lavadora, con el atributo carga, además de los atributos heredados.

Los constructores que se implementarán serán:

- Un constructor vacío.
- Un constructor con la carga y el resto de atributos heredados. Recuerda que debes llamar al constructor de la clase padre.

Los métodos que se implementara serán:

- Método get y set del atributo carga.
- Método `crearLavadora ()`: este método llama a `crearElectrodomestico()` de la clase padre, lo utilizamos para llenar los atributos heredados del padre y después llenamos el atributo propio de la lavadora.

- Método `precioFinal()`: este método será heredado y se le sumará la siguiente funcionalidad. Si tiene una carga mayor de 30 kg, aumentará el precio en \$500, si la carga es menor o igual, no se incrementará el precio. Este método debe llamar al método padre y añadir el código necesario. Recuerda que las condiciones que hemos visto en la clase `Electrodomestico` también deben afectar al precio.

Se debe crear también una subclase llamada `Televisor` con los siguientes atributos: resolución (en pulgadas) y sintonizador TDT (booleano), además de los atributos heredados.

Los constructores que se implementarán serán:

- Un constructor vacío.
- Un constructor con la resolución, sintonizador TDT y el resto de atributos heredados. Recuerda que debes llamar al constructor de la clase padre.

Los métodos que se implementara serán:

- Método `get` y `set` de los atributos resolución y sintonizador TDT.
- Método `crearTelevisor()`: este método llama a `crearElectrodomestico()` de la clase padre, lo utilizamos para llenar los atributos heredados del padre y después llenamos los atributos del televisor.
- Método `precioFinal()`: este método será heredado y se le sumará la siguiente funcionalidad. Si el televisor tiene una resolución mayor de 40 pulgadas, se incrementará el precio un 30% y si tiene un sintonizador TDT incorporado, aumentará \$500. Recuerda que las condiciones que hemos visto en la clase `Electrodomestico` también deben afectar al precio.

Finalmente, en el `main` debemos realizar lo siguiente:

Vamos a crear una `Lavadora` y un `Televisor` y llamar a los métodos necesarios para mostrar el precio final de los dos electrodomésticos.

3. Siguiendo el ejercicio anterior, en el `main` vamos a crear un `ArrayList` de `Electrodomesticos` para guardar 4 electrodomésticos, ya sean lavadoras o televisores, con valores ya asignados.

Luego, recorrer este array y ejecutar el método `precioFinal()` en cada electrodoméstico. Se deberá también mostrar el precio de cada tipo de objeto, es decir, el precio de todos los televisores y el de las lavadoras. Una vez hecho eso, también deberemos mostrar, la suma del precio de todos los `Electrodomesticos`. Por ejemplo, si tenemos una lavadora con un precio de 2000 y un televisor de 5000, el resultado final será de 7000 (2000+5000) para electrodomésticos, 2000 para lavadora y 5000 para televisor.

4. Se plantea desarrollar un programa que nos permita calcular el área y el perímetro de formas geométricas, en este caso un círculo y un rectángulo. Ya que este cálculo se va a repetir en las dos formas geométricas, vamos a crear una `Interfaz`, llamada `calculosFormas` que tendrá, los dos métodos para calcular el área, el perímetro y el valor de PI como constante.

Desarrollar el ejercicio para que las formas implementen los métodos de la interfaz y se calcule el área y el perímetro de los dos. En el main se crearán las formas y se mostrará el resultado final.

Área círculo: $\text{PI} * \text{radio}^2$ / Perímetro círculo: $\text{PI} * \text{diámetro}$.

Área rectángulo: $\text{base} * \text{altura}$ / Perímetro rectángulo: $(\text{base} + \text{altura}) * 2$.

EJERCICIOS DE APRENDIZAJE EXTRA

Estos van a ser ejercicios para reforzar los conocimientos previamente vistos. Estos pueden realizarse cuando hayas terminado la guía y tengas una buena base sobre lo que venimos trabajando. Además, si ya terminaste la guía y te queda tiempo libre en las mesas, puedes continuar con estos ejercicios extra, recordando siempre que no es necesario que los termines para continuar con el tema siguiente. Por último, recordá que la prioridad es ayudar a los compañeros de la mesa y que cuando tengas que ayudar, lo más valioso es que puedas explicar el ejercicio con la intención de que tu compañero lo comprenda, y no sólo mostrarlo. ¡Muchas gracias!

1. En un puerto se alquilan amarres para barcos de distinto tipo. Para cada Alquiler se guarda: el nombre, documento del cliente, la fecha de alquiler, fecha de devolución, la posición del amarre y el barco que lo ocupará.
Un Barco se caracteriza por: su matrícula, su eslora en metros y año de fabricación.

Sin embargo, se pretende diferenciar la información de algunos tipos de barcos especiales:

- Número de mástiles para veleros.
- Potencia en CV para barcos a motor.
- Potencia en CV y número de camarotes para yates de lujo.

Un alquiler se calcula multiplicando el número de días de ocupación (calculado con la fecha de alquiler y devolución), por un valor módulo de cada barco (obtenido simplemente multiplicando por 10 los metros de eslora).

En los barcos de tipo especial el módulo de cada barco, se calcula sacando el módulo normal y sumándole el atributo particular de cada barco. En los veleros se suma el número de mástiles, en los barcos a motor se le suma la potencia en CV y en los yates se suma la potencia en CV y el número de camarotes.

Utilizando la herencia de forma apropiada, deberemos programar en Java, las clases y los métodos necesarios que permitan al usuario elegir el barco que quiera alquilar y mostrarle el precio final de su alquiler.

2. Crear una superclase llamada Edificio con los siguientes atributos: ancho, alto y largo. La clase edificio tendrá como métodos:

- Método calcularSuperficie(): calcula la superficie del edificio.
- Método calcularVolumen(): calcula el volumen del edificio.

Estos métodos serán abstractos y los implementarán las siguientes clases:

- Clase Polideportivo con su nombre y tipo de instalación que puede ser Techado o Abierto, esta clase implementará los dos métodos abstractos y los atributos del padre.

- Clase EdificioDeOficinas con sus atributos número de oficinas, cantidad de personas por oficina y numero de pisos. Esta clase implementará los dos métodos abstractos y los atributos del padre.
De esta clase nos interesa saber cuantas personas pueden trabajar en todo el edificio, el usuario dirá cuantas personas entran en cada oficina, cuantas oficinas y el numero de piso (suponiendo que en cada piso hay una oficina). Crear el método cantPersonas(), que muestre cuantas personas entrarían en un piso y cuantas en todo el edificio.

Por ultimo, en el main vamos a crear un ArrayList de tipo Edificio. El ArrayList debe contener dos polideportivos y dos edificios de oficinas. Luego, recorrer este array y ejecutar los métodos calcularSuperficie y calcularVolumen en cada Edificio. Se deberá mostrar la superficie y el volumen de cada edificio.

Además de esto, para la clase Polideportivo nos interesa saber cuantos polideportivos son techados y cuantos abiertos. Y para la clase EdificioDeOficinas deberemos llamar al método cantPersonas() y mostrar los resultados de cada edificio de oficinas.

3. Una compañía de promociones turísticas desea mantener información sobre la infraestructura de alojamiento para turistas, de forma tal que los clientes puedan planear sus vacaciones de acuerdo a sus posibilidades. Los alojamientos se identifican por un nombre, una dirección, una localidad y un gerente encargado del lugar. La compañía trabaja con dos tipos de alojamientos: Hoteles y Alojamientos Extrahoteleros.

Los Hoteles tienen como atributos: Cantidad de Habitaciones, Número de Camas, Cantidad de Pisos, Precio de Habitaciones. Y estos pueden ser de cuatro o cinco estrellas. Las características de las distintas categorías son las siguientes:

- Hotel **** Cantidad de Habitaciones, Número de camas, Cantidad de Pisos, Gimnasio, Nombre del Restaurante, Capacidad del Restaurante, Precio de las Habitaciones.
- Hotel ***** Cantidad de Habitaciones, Número de camas, Cantidad de Pisos, Gimnasio, Nombre del Restaurante, Capacidad del Restaurante, Cantidad Salones de Conferencia, Cantidad de Suites, Cantidad de Limosinas, Precio de las Habitaciones.

Los gimnasios pueden ser clasificados por la empresa como de tipo “A” o de tipo “B”, de acuerdo a las prestaciones observadas. Las limosinas están disponibles para cualquier cliente, pero sujeto a disponibilidad, por lo que cuanto más limosinas tenga el hotel, más caro será.

El precio de una habitación debe calcularse de acuerdo a la siguiente fórmula:

$$\text{PrecioHabitación} = \$50 + (\$1 \times \text{capacidad del hotel}) + (\text{valor agregado por restaurante}) + (\text{valor agregado por gimnasio}) + (\text{valor agregado por limosinas}).$$

Donde:

Valor agregado por el restaurante:

- \$10 si la capacidad del restaurante es de menos de 30 personas.
- \$30 si está entre 30 y 50 personas.
- \$50 si es mayor de 50.

Valor agregado por el gimnasio:

- \$50 si el tipo del gimnasio es A.
- \$30 si el tipo del gimnasio es B.

Valor agregado por las limosinas:

- \$15 por la cantidad de limosinas del hotel.

En contraste, los Alojamientos Extra hoteleros proveen servicios diferentes a los de los hoteles, estando más orientados a la vida al aire libre y al turista de bajos recursos. Por cada Alojamiento Extrahotelero se indica si es privado o no, así como la cantidad de metros cuadrados que ocupa. Existen dos tipos de alojamientos extrahoteleros: los Camping y las Residencias. Para los Camping se indica la capacidad máxima de carpas, la cantidad de baños disponibles y si posee o no un restaurante dentro de las instalaciones. Para las residencias se indica la cantidad de habitaciones, si se hacen o no descuentos a los gremios y si posee o no campo deportivo. Realizar un programa en el que se representen todas las relaciones descriptas.

Realizar un sistema de consulta que le permite al usuario consultar por diferentes criterios:

- todos los alojamientos.
- todos los hoteles de más caro a más barato.
- todos los campings con restaurante
- todos las residencias que tienen descuento.

4. Sistema Gestión Facultad. Se pretende realizar una aplicación para una facultad que gestione la información sobre las personas vinculadas con la misma y que se pueden clasificar en tres tipos: estudiantes, profesores y personal de servicio. A continuación, se detalla qué tipo de información debe gestionar esta aplicación:

- Por cada persona, se debe conocer, al menos, su nombre y apellidos, su número de identificación y su estado civil.
- Con respecto a los empleados, sean del tipo que sean, hay que saber su año de incorporación a la facultad y qué número de despacho tienen asignado.
- En cuanto a los estudiantes, se requiere almacenar el curso en el que están matriculados.
- Por lo que se refiere a los profesores, es necesario gestionar a qué departamento pertenecen (lenguajes, matemáticas, arquitectura, ...).
- Sobre el personal de servicio, hay que conocer a qué sección están asignados (biblioteca, decanato, secretaría, ...).

El ejercicio consiste, en primer lugar, en definir la jerarquía de clases de esta aplicación. A continuación, debe programar las clases definidas en las que, además de los constructores, hay que desarrollar los métodos correspondientes a las siguientes operaciones:

- Cambio del estado civil de una persona.
- Reasignación de despacho a un empleado.
- Matriculación de un estudiante en un nuevo curso.
- Cambio de departamento de un profesor.
- Traslado de sección de un empleado del personal de servicio.
- Imprimir toda la información de cada tipo de individuo. Incluya un programa de prueba que instancie objetos de los distintos tipos y pruebe los métodos desarrollados.