CURSO DE PROGRAMACIÓN FULL STACK

HERENCIA

PARADIGMA ORIENTADO A OBJETOS

Objetivos de la Guía

En esta guía aprenderás:

- Qué es la herencia
- Para qué sirve
- Cómo heredar entre clases
- Clases abstractas y finales
- Métodos abstractos y finales
- Qué son interfaces
- Cuándo y cómo usar interfaces

¿QUÉ ES LA 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/madre) y las **subclases** heredan estas propiedades (Clase hija/o). 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**.

```
public class SubClase extends SuperClase{
    //atributos y métodos
}
```

Viendo nuestro ejemplo de la cajonera pensemos que la clase Cajonera podría heredar atributos de MuebleMadera que le otorgue atributos de materiales, durabilidad, resistencia al fuego. Estas serán características que también podría heredar una clase Mesa y clase Silla, compartiendo las 3 esos atributos.

Cuando en nuestro diseño detectamos que más de una clase comparte atributos o métodos podemos buscar puntos en común para diseñar una superclase.

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.

```
public 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 matera que sería especifico 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.

```java
public class Persona {

    protected String nombre;
    protected Integer edad;
    protected Integer documento;

    public Persona(String nombre, Integer edad, Integer documento) {
        this.nombre = nombre;
        this.edad = edad;
        this.documento = documento;
    }

}

class Alumno extends Persona {

    private String materia;

    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 a las clases hijas utilizar un método definido en la superclase.

Una subclase sobreescribe un método de su superclase cuando define un método con las mismas características (nombre, número y tipo de argumentos) que el método de la superclase. Las subclases emplean la sobreescritura de métodos la mayoría de las veces para agregar o modificar la funcionalidad del método heredado de la clase padre.

La sobreescritura permite que las clases hijas sumen sus métodos en torno al funcionamiento y Esto se logra poniendo **la anotación @Override** arriba del método que queremos sobreescribir, el método debe llamarse igual en la subclase como en la superclase.

```java
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");
    }
}
```

Una anotación en Java es una característica que permite incrustar información suplementaria en un archivo fuente. Esta información no cambia las acciones de un programa, pero puede ser utilizada por varias herramientas, tanto durante el desarrollo como durante el despliegue del programa.

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 cuáles 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.

MANOS A LA OBRA!

EJERCICIO ANIMAL

Vamos a crear una clase Animal que tenga un método hacerRuido() que devuelva un saludo "Hola". Luego haremos clase Perro y clase Gato que extiendan de Animal y sobreescriban el método hacerRuido() con el ruido que corresponda a cada uno. Luego, en el main vamos a crear un ArrayList de animales y los siguientes animales

Animal a = new Animal();

Animal b = new Perro();

Animal c = new Gato();

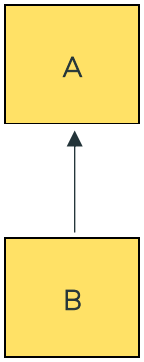
Agregaremos a la lista a cada uno y luego, con un for each, recorreremos la lista llamando al método hacerRuido() de cada ítem.

TIPOS DE HERENCIA

Dentro del concepto de herencia tenemos distintos tipos de herencia, los veremos ahora.

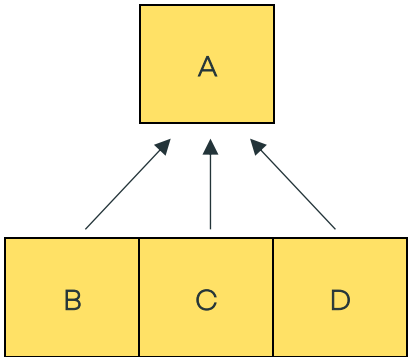
HERENCIA ÚNICA

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.



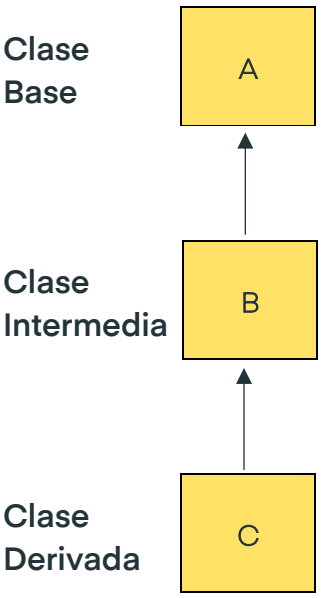
HERENCIA JERÁRQUICA

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.



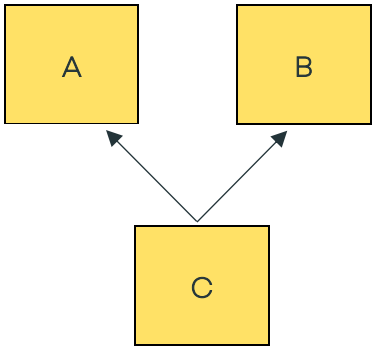
HERENCIA MULTINIVEL

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.



Interfaces las veremos más adelante en esta misma guía.

MODIFICADORES DE CLASES Y MÉTODOS

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(){ }
```

Es útil declarar algo como final cuando no queremos que nada lo pueda modificar.

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 sobreescriben por las clases hijas quienes son las responsables de implementar sus funcionalidades.

```java
abstract class Persona {

    public abstract void codear();
}

class Alumno extends Persona {

    @Override
    public void codear() {
        System.out.println("Está aprendiendo");
    }

}
```

¿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 sobreescriban el método declarado como abstracto.

INTERFACES

Una interfaz es sintácticamente similar a una clase abstracta, en la que puede especificar uno o más métodos que no tienen cuerpo. Esos métodos deben ser implementados por una clase para que se definan sus acciones.

Por lo tanto, **una interfaz especifica qué se debe hacer, pero no cómo hacerlo**. Una vez que se define una interfaz, cualquier cantidad de clases puede implementarla. Además, una clase puede implementar cualquier cantidad de interfaces.

Para implementar una interfaz, una clase debe proporcionar cuerpos (implementaciones) para los métodos descritos por la interfaz. Cada clase es libre de determinar los detalles de su propia implementación. Dos clases pueden implementar la misma interfaz de diferentes maneras, pero **cada clase aún admite el mismo conjunto de métodos**.

Para decirle a Java que estamos trabajando sobre una interfaz vamos a tener que utilizar la palabra **interface**, esto se vería así

```
public interface nombreInterfaz { }
```

Para una interfaz, el modificador de acceso es **public** o no se usa. Cuando no se incluye ningún modificador de acceso, los resultados de acceso serán los predeterminados y la interfaz solo están disponibles para otros miembros de su paquete. Si se declara como public, la interfaz puede ser utilizada por cualquier otra clase. El nombre de la interfaz puede ser cualquier identificador válido.

INSTANCIAR UNA INTERFAZ

Aunque las interfaces van a ser implementadas por clases y van a tener métodos, al igual que una clase abstracta, esta, **no se va a poder instanciar**. La definición de un interfaz no tiene constructor, por lo que no es posible invocar el operador *new* sobre un tipo interfaz, por lo que no podemos crear objetos del tipo interfaz.

MANOS A LA OBRA!

Vamos a crear una interfaz con un método abstracto. Luego desde el main intentaremos instanciar un objeto a partir de la interfaz

IMPLEMENTACIÓN DE INTERFACES

Una vez que se ha definido una interfaz, una o más clases pueden implementar esa interfaz. Para implementar una interfaz, incluya la palabra reservada implements en la definición de clase.

```
public class Clase implements Interfaz { }
```

Los métodos de la interfaz los podemos implementar en nuestra clase y se van a sobrescribir desde la interfaz, métodos que recordemos, no tendrán cuerpo. Nuestra tarea será completar esos métodos que implementamos de la interfaz.

MÉTODOS

Los métodos de una interfaz se declaran utilizando solo su tipo de devolución y firma. Son, esencialmente, métodos abstractos. Por lo tanto, cada clase que incluye dicha interfaz debe implementar todos sus métodos. En una interfaz, los métodos son implícitamente públicos.

```java
public interface Interfaz {

    public void metodo();

    public int sumar();
}

class Clase implements Interfaz {

    @Override
    public void metodo() {
        System.out.println("Implementacion del método");
    }

    @Override
    public int sumar() {
        int suma = 2 + 2;
        return suma;
    }
}
```

Como podemos ver en la interfaz teníamos dos métodos sin cuerpo y al implementar la interfaz en nuestra clase, los sobrescribimos y les dimos una funcionalidad a dichos métodos.

¿Qué es una firma?

En Java dos o más métodos dentro de la misma clase pueden compartir el mismo nombre, siempre que sus declaraciones de parámetros sean diferentes. Cuando esto sucede, se dice que estamos usando sobrecarga de métodos (method overloading). En general sobrecargar un método consiste en declarar versiones diferentes de él. Y aquí es donde el compilador se ocupa del resto y donde el término firma cobra importancia. Una firma es el nombre de un método más su lista de parámetros. Por lo tanto, cada método en una misma clase, en términos de sobrecarga, obtiene una firma diferente.

VARIABLES

Las variables declaradas en una interfaz no son variables de instancia. En cambio, son implícitamente *public* y *final*, además, deben inicializarse. Por lo tanto, son esencialmente constantes. Recordemos que por convención las constantes suelen escribirse en mayúsculas, para separarlas de las variables.

```java
public interface Interfaz {

    public final int CONSTANTE = 10;

    public void metodo();
}

class Clase implements Interfaz {

    @Override
    public void metodo() {
        System.out.println("La constante tiene un valor de " + CONSTANTE);
    }
}
```

Las constantes definidas en nuestra interfaz pueden ser llamadas en la clase, solo con el nombre y de esa manera podemos utilizar los valores definidos en la interfaz.

EJERCICIOS DE APRENDIZAJE

Recordemos que todos los ejercicios debemos realizarlos con las clases de servicio. En los ejercicios se numeran los métodos a escribir, pero no cuáles van en servicio, pueden hablar en el equipo y dar sus opiniones sobre dónde van. **Nos vamos a encontrar con algo nuevo. Vamos a heredar clases con clases y servicios con servicios. No mezclemos las cosas.**

VIDEOS: Te sugerimos ver los videos relacionados con este tema, antes de empezar los ejercicios, los podrás encontrar en tu aula virtual o en nuestro canal de YouTube.

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:

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

    //Declaración del objeto Perro
    Animal perro1 = new Perro("Stich", "Carnivoro", 15, "Doberman");
    perro1.Alimentarse();

    //Declaración de otro objeto Perro
    Animal perro2 = new Perro("Teddy", "Croquetas", 10, "Chihuahua");
    perro2.Alimentarse();

    //Declaración del objeto Gato
    Animal gato1 = new Gato("Pelusa", "Galletas", 15, "Siamés");
    gato1.Alimentarse();

    //Declaración del objeto Caballo
    Animal caballo1 = new Caballo("Spark", "Pasto", 25, "Fino");
    caballo1.Alimentarse();

}
```

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.

- Metodo 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:

LETRA	PRECIO
A	$1000
B	$800
C	$600
D	$500
E	$300
F	$100

PESO	PRECIO
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 los 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 Electrodoméstico 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 los 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 Electrodomésticos 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 Electrodomésticos. 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 circulo: PI * radio ^ 2 / Perímetro circulo: PI * diámetro.

Área rectángulo: base * altura / Perímetro rectángulo: (base + 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, podes 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 edifico.

   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 número de pisos. Esta clase implementará los dos métodos abstractos y los atributos del padre.

   De esta clase nos interesa saber cuántas personas pueden trabajar en todo el edificio, el usuario dirá cuántas personas entran en cada oficina, cuantas oficinas y el número 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 último, 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 cuántos 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 con 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 con la siguiente fórmula: PrecioHabitación = $50 + ($1 x capacidad del hotel) + (valor agregado por restaurante) + (valor agregado por gimnasio) + (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.