Trabajo práctico Nro. 2



| Logo Departamento 2.jpg | **Asignatura: Programación II** | |
| --- | --- | --- |
|  | |
| **Cursado:** Segundo semestre | **Horas semanales**: |
|  | **Horas semestrales:**  *Cantidad estimada de horas*  *semestrales/anuales.* |
| **Carrera**: *Tecnicatura Universitaria en*  *Programación* | **Nivel (Año):**  1°  2°  3° |
| **Ciclo Lectivo: 2024** |

**Integrantes de la Cátedra:** Lima Pablo, Andres Bercih, Ulises Guzmán, Santiago Comeglio y Federico Cappello.

* **DOCENTES:**

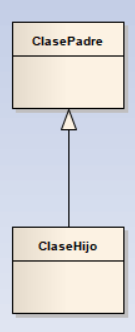
| **Nombre del Profesor** | **Periodo** | **Cantidad horas materia** |
| --- | --- | --- |
| **Cinthia Rigoni** |  | 6 horas |

* 1. ¿Qué es la herencia en java? ¿Cuáles son sus beneficios?

La herencia en Java es un concepto fundamental de la programación orientada a objetos (POO) que permite crear nuevas clases a partir de clases existentes. La nueva clase, llamada clase hija o subclase, hereda los atributos y métodos de la clase existente, llamada clase padre o superclase. Esto permite la reutilización del código y la creación de jerarquías de clases.

Los beneficios de la herencia en Java son:

* Reutilización de código: Al heredar de una clase, la subclase obtiene todos los campos y métodos de la superclase, lo que permite reutilizar el código existente.
* Organización y estructura: La herencia crea una estructura lógica y organizada en la que las clases se agrupan en jerarquías. Esto facilita la comprensión y el mantenimiento del código.
* Polimorfismo: La herencia permite que objetos de una subclase sean tratados como objetos de la superclase, lo que permite una mayor flexibilidad y facilidad de uso.
* Extensibilidad: Al heredar de una clase, se puede agregar nuevos campos y métodos a la subclase sin afectar a la superclase. Esto permite extender la funcionalidad de una clase sin modificar su código fuente.
* Abstracción: La herencia permite ocultar los detalles de implementación de una superclase y exponer sólo los detalles relevantes a la subclase. Esto promueve la abstracción y la modularidad del código.
  1. ¿Cómo se representa la herencia en un diagrama UML?



* 1. ¿Cuándo es recomendable usar la herencia?

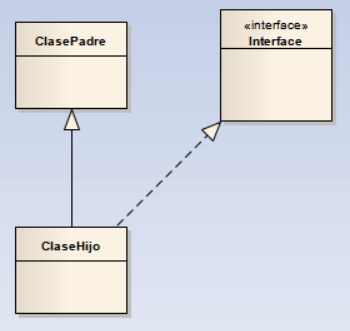
En general, la herencia debe usarse para modelar relaciones "es-un" naturales en el mundo real, como una relación entre un coche y un vehículo, o entre un empleado y una persona. La herencia debe usarse para compartir comportamiento común y para reutilizar código, no para compartir propiedades o para controlar el flujo de la aplicación.

* 1. ¿Qué es una interfaz en java?

Una interfaz en Java es un tipo de referencia que se utiliza para declarar que una clase proporciona un conjunto de métodos que pueden ser implementados por otras clases. Una interfaz no proporciona implementación de los métodos, sólo la firma del método.

Las interfaces en Java tienen las siguientes características:

* Solo declaración de métodos: Una interfaz sólo puede contener métodos sin cuerpo (es decir, solo la declaración del método, sin la implementación).
* Implementación de múltiples interfaces: Una clase puede implementar múltiples interfaces.
* Herencia de interfaces: Una interfaz puede heredar de otra interfaz, similar a la herencia de clases.
* Campos estáticos y finales: Una interfaz puede contener campos estáticos y finales.
* Métodos por defecto: A partir de Java 8, las interfaces pueden contener métodos con implementación por defecto.
* Métodos privados: A partir de Java 9, las interfaces pueden contener métodos privados con implementación.
  1. ¿Cómo se representa una interfaz en un diagrama UML?



* 1. ¿Cuándo es recomendable usar una interfaz?

La principal característica de una interfaz es que solo declara métodos, pero no proporciona implementaciones para ellos. Es decir, no define el “cómo” de la funcionalidad, solo especifica el “qué”. Las clases que implementan una interfaz deben proporcionar sus propias implementaciones para cada uno de los métodos declarados en la interfaz. Las interfaces en la programación orientada a objetos son útiles cuando quieres definir un conjunto de métodos que deben ser implementados por varias clases diferentes, pero no necesariamente tienen una relación de herencia común. Aquí hay algunas situaciones en las que es recomendable usar interfaces: 1.Definir contratos: Las interfaces se utilizan para definir contratos que las clases deben seguir. Si tienes un conjunto de métodos que varias clases deben implementar, puedes definir una interfaz que contenga esos métodos y hacer que las clases implementen esa interfaz.

* 1. ¿Cuál es la diferencia entre una interfaz y una clase abstracta?

Existen varias diferencias entre una clase abstracta y una interfaz:

Una clase abstracta puede heredar o extender cualquier clase (independientemente de que esta sea abstracta o no), mientras que una interfaz solamente puede extender o implementar otras interfaces.

Una clase abstracta puede heredar de una sola clase (abstracta o no) mientras que una interfaz puede extender varias interfaces de una misma vez.

Una clase abstracta puede tener métodos que sean abstractos o que no lo sean, mientras que las interfaces sólo y exclusivamente pueden definir métodos abstractos.

En java concretamente (ya que has puesto la etiqueta Java), en las clases abstractas la palabra abstract es obligatoria para definir un método abstracto (así como la clase). Cuando defines una interfaz, esta palabra es opcional ya que se infiere en el concepto de interfaz.

En una clase abstracta, los métodos abstractos pueden ser public o protected. En una interfaz solamente puede haber métodos públicos.

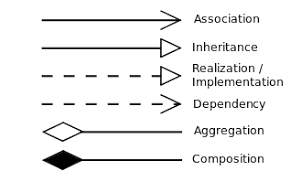
En una clase abstracta pueden existir variables static, final o static final con cualquier modificador de acceso (public, private, protected o default). En una interfaz sólo puedes tener constantes (public static final).

* 1. ¿Qué tipos de relaciones pueden existir entre clases y objetos?

Hay seis tipos principales de relaciones entre clases: herencia, realización/implementación, composición, agregación, asociación y dependencia. Las flechas para las seis relaciones son las siguientes:

Entonces llegamos a comprender el contenido específico de la relación de clase.

* 1. ¿Cómo se representa cada relación en un diagrama UML?



* 1. ¿Cuándo es recomendable usar cada tipo de relación?

Asociación: Se utiliza para mostrar una conexión entre clases, indicando que un objeto de una clase está relacionado con un objeto de otra clase.

Agregación: Se usa cuando una clase es parte de otra clase, pero puede existir independientemente de ella. La relación es de tipo "es un componente de".

Composición: Similar a la agregación, pero la clase contenida no puede existir sin la clase contenedora. La relación es de tipo "es un todo de".

Herencia: Indica que una clase hereda atributos y métodos de otra clase. Es una relación "es un tipo de".

Dependencia: Indica que una clase utiliza los servicios de otra clase. La relación es temporal y flexible.

Realización: Se utiliza para indicar que una clase implementa las operaciones definidas por una interfaz o una clase abstracta.

**A CODIFICAR!**

1. Desarrollar un sistema para gestionar una tienda de mascotas que vende diferentes tipos de animales: perros, gatos, aves.
   1. Clases:
      1. Animal: nombre (String), edad (int), tipoAnimal (String), precio (double). Método Alimentar(String nombre). Sobreescribir el método toString() para que devuelva una pequeña presentación del animal.
      2. Perro: raza (String), vacunado (boolean). Hereda de Animal.
      3. Gato: raza (String), esterilizado (boolean). Hereda de Animal.
      4. Ave: especie (String), habla(boolean). Hereda de Animal.
      5. Crear también una clase Pez con los atributos que creas necesarios. Hereda de Animal.

Desarrollá las clases especificadas anteriormente y crea una nueva clase TiendaMascotas para gestionar todas las demás. Esta debe tener los métodos:

* addAnimal(Animal animal): Agrega un animal a la tienda.
* venderAnimal(String nombre): Vende un animal por nombre.
* alimentarAnimal(String nombre): Alimenta a un animal por nombre.
* listarAnimales(): Lista todos los animales de la tienda.

En el main crea una tienda, un animal de cada clase y utilizá todos los métodos definidos.

1. Desarrollar un sistema de gestión para una biblioteca.
   1. Definir una clase Libro con los atributos título (String), autor (String), añoDePublicacion (int), prestado (boolean). Podés crear más atributos si lo considerás necesario.
   2. Crea una interfaz llamada Prestable con los métodos prestar() y devolver().
   3. Libro debe implementar esta interfaz y se deben definir los métodos para cambiar el estado del libro.
   4. Crear las clases Novela, Universitario, Infantil que hereden de Libro. Agregá a cada clase un atributo propio (o más, si te parece necesario).
   5. Definir la clase Biblioteca, que contenga una lista de libros y métodos para agregar un libro, listar los libros disponibles, prestar un libro, devolver un libro.

En el main crea una biblioteca, un libro de cada clase y utilizá todos los métodos definidos.