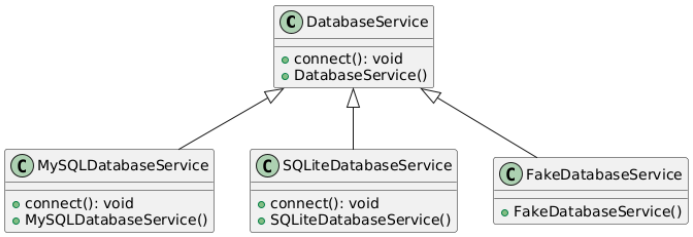
EJERCICIOS PROGRAMACION III
Práctica SEMANA 10

CONTENIDOS: Herencia y Polimorfismo I - Ejercicios básicos de Herencia/Interfaces y MVC con herencia e interfaces.

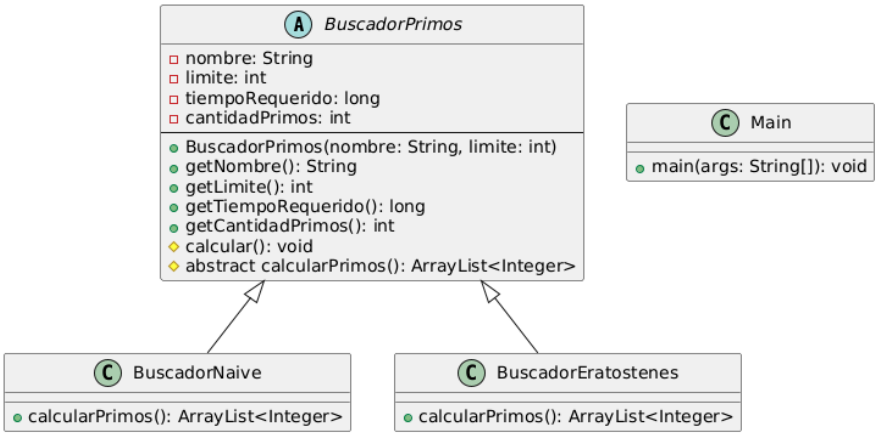
Ejercicios obligatorios:

1. **Revisa detenidamente los proyectos de código proporcionados en Studium antes de continuar con los siguientes ejercicios. En ellos se describen aspectos básicos de herencia y polimorfismo. Revisa también la teoría de la asignatura.**
2. Realiza las siguientes clases java con la relación de herencia mostrada en el siguiente diagrama UML:



El método connect de la clase padre deberá mostrar por pantalla el mensaje "Conectando con la BBDD...". El método connect de cada clase hija deberá sobrescribir el método connect y llamar al método connect de la clase padre y añadir otro mensaje propio "Tipo de BBDD: **X"**. Se deberá crear en el main un ArrayList<DatabaseService> y crear 3 instancias, una de cada subtipo, y añadirlas a dicha lista. Se recorrerá el ArrayList<DatabaseService> y se llamará al método connect de cada una de las instancias en la lista para comprobar qué sucede y el **polimorfismo basado en herencia**.

3. Se debe implementar un programa para buscar números primos hasta un determinado número. Para ello se deben desarrollar las clases representadas en el siguiente diagrama UML:



Explicación del diagrama:

Clase Abstracta `BuscadorPrimos`:

a. Debe contener los siguientes **atributos <u>privados</u>**:
   i. `nombre`: una cadena que identifica al buscador.
   ii. `limite`: número hasta el cual buscar primos.
   iii. `tiempoRequerido`: tiempo en milisegundos que el buscador tarda en realizar la búsqueda.
   iv. `cantidadPrimos`: cantidad de números primos encontrados en el cálculo.

b. Métodos:
   i. Un **constructor** que reciba como parámetros el nombre del buscador y el límite hasta el cual buscar primos.
   ii. Métodos **getter** para acceder a los atributos privados.
   iii. Un **método abstracto** `calcularPrimos()` que devuelve un `ArrayList<Integer>` con los números primos encontrados.
   iv. Un **método final** `calcular()` que:
      1. Mida el tiempo que tarda el buscador en encontrar los números primos (usando la clase `Instant`).
      2. Actualice los atributos `tiempoRequerido` y `cantidadPrimos` con los valores obtenidos.
      3. Llame internamente a `calcularPrimos()` para realizar la búsqueda.

Subclases `BuscadorNaive` y `BuscadorEratostenes`:

c. Deben heredar de `BuscadorPrimos` e implementar el método `calcularPrimos()`.
   i. `BuscadorNaive`: debe usar un enfoque básico para determinar si cada número hasta el límite es primo o no.
   ii. `BuscadorEratostenes`: debe usar el algoritmo de la criba de Eratóstenes para encontrar los números primos.

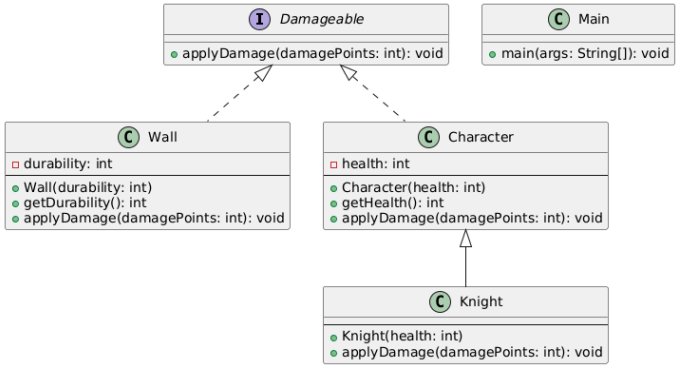
Clase Principal `Main`:

d. En el `main`, se debe:
   i. Crear una lista `ArrayList<BuscadorPrimos>`.
   ii. Añadir instancias de `BuscadorNaive` y `BuscadorEratostenes`, configurándolas con el mismo límite.
   iii. Para cada instancia en la lista, llamar al método `calcular()`.

Mostrar por pantalla un mensaje con el formato. Los valores `X`, `Y`, `Z` y `J` deben extraerse de los atributos de cada buscador.

```
El Buscador X ha tardado Y milisegundos en encontrar Z primos hasta el número J
```

4. Se desea implementar un sistema que gestione objetos que pueden recibir daño, con el objetivo de demostrar cómo las **interfaces funcionan como un contrato en Java**. A continuación el diagrama en UML, Nótese la diferencia entre **herencia** (extends), línea continua e **interfaces** (implements) línea discontinua.

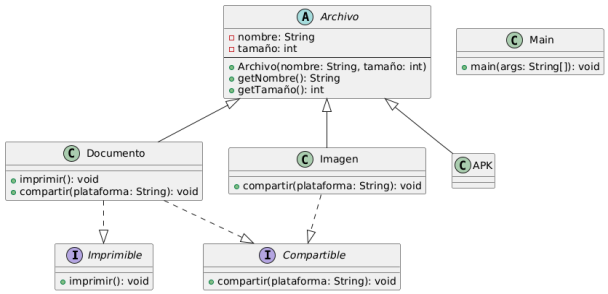


Sigue estas instrucciones:

1. **Interface `Damageable`:**
   - Define un contrato para cualquier clase que desee implementar la funcionalidad de recibir daño.
   - Contiene un único método:
     - `void applyDamage(int damagePoints)`: Aplica daño al objeto.
2. **Clase `Wall`:**
   - Representa una pared que tiene un atributo `durability` (durabilidad).
   - Implementa la interfaz `Damageable`.
   - Al recibir daño, se reduce su durabilidad. Si esta llega a 0 o menos, la pared se considera destruida.

3. **Clase `Character`:**
    - Representa un personaje con un atributo `health` (salud).
    - Implementa la interfaz `Damageable`.
    - Al recibir daño, se reduce su salud. Si esta llega a 0 o menos, el personaje está fuera de combate.
4. **Clase `Knight`:**
    - Hereda de `Character`.
    - Implementa una versión personalizada de `applyDamage`, donde el daño recibido se reduce en un 10%.
5. **Clase principal `Main`:**
    - Crear una lista de objetos `ArrayList<Damageable>` y añadir:
        - Una instancia de `Wall` con 500 puntos de durabilidad.
        - Una instancia de `Character` con 300 puntos de salud.
        - Una instancia de `Knight` con 400 puntos de salud.
    - Recorrer la lista y aplicar 100 puntos de daño a cada objeto, mostrando el estado resultante.

5. Se debe desarrollar un programa en Java con las clases mostradas en el siguiente diagrama UML. Nótese la diferencia entre **herencia** (extends), línea continua e **interfaces** (implements) línea discontinua. También cómo es posible implementar varias interfaces pero no extender de varias clases.



Interfaces:

- Crear la interfaz `Imprimible` con un único método:
    - `imprimir(): void`: Muestra en consola un mensaje indicando que el archivo está siendo enviado a impresión.
- Crear la interfaz `Compartible` con un único método:
    - `compartir(String plataforma): void`: Muestra en consola un mensaje indicando que el archivo está siendo compartido en la plataforma especificada.

Clase Abstracta `Archivo`:

- Atributos:
    - `nombre`: Nombre del archivo.
    - `tamanio`: Tamaño del archivo en KB.
- Métodos:
    - Constructor que reciba el nombre y el tamaño del archivo.
    - Métodos `getNombre()` y `getTamanio()`.

Subclases de `Archivo`:

- `Documento`: Representa un archivo de texto o PDF. Debe implementar las interfaces `Imprimible` y `Compartible`.
- `Imagen`: Representa un archivo de imagen (JPG, PNG). Debe implementar solo la interfaz `Compartible`.
- `APK`: Representa un archivo de apk. No implementa ninguna interfaz.

Clase Principal `Main`:

- Crear una lista `ArrayList<Archivo>` y añadir instancias de `Documento`, `Imagen` y `APK`. Emplear valores arbitrarios.
- Recorrer la lista y:
    - Si el archivo es **imprimible**, llamar al método `imprimir()`.
    - Si el archivo es **compartible**, llamar al método `compartir()` especificando una plataforma como "WhatsApp" o "Email".
    - Mostrar un mensaje diferente si el archivo no es ni imprimible ni compartible.

Para este propósito es posible utilizar instanceof.