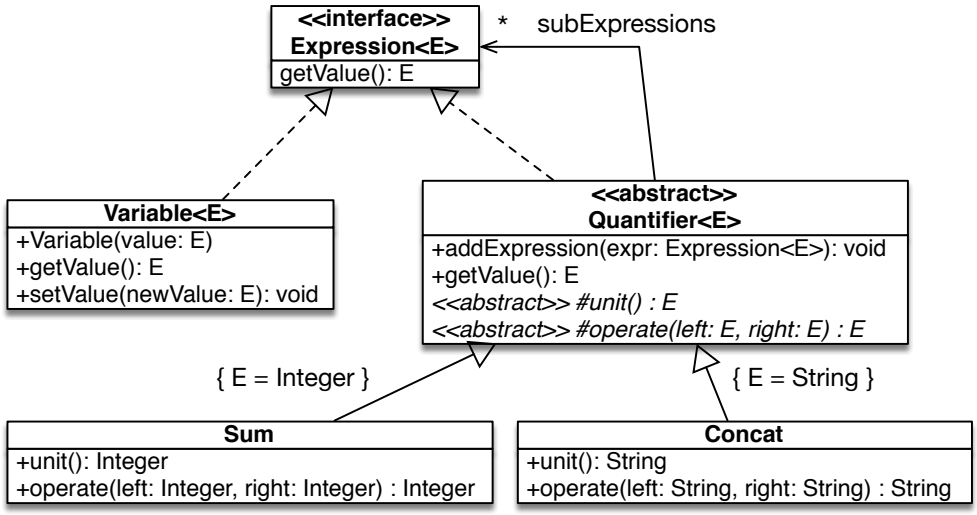

Este diseño permite representar expresiones (que evalúan a un tipo `E` genérico) y que pueden contener tanto variables como una serie de subexpresiones unidas por un cuantificador.

En todos los apartados podéis suponer que las expresiones forman un árbol.

Por ejemplo, las expresiones que podríamos definir, usando constantes en el lugar de las variables para simplificar, se comportarían como:

$$\Sigma\big(4, \Pi(), \Pi(2,3), \Sigma(3,5)\big) = 4 + 1 + (2 * 3) + (3 + 5) = 4 + 1 + 6 + 8 = 19$$

Es decir, un cuantificador viene determinado por el valor que retorna cuando no tiene expresiones (`unit`) y la operación binaria que se esconde detrás de él (`operate`). En términos formales, detrás de un cuantificador, nos encontramos ante una estructura de monoide:

- `unit` será el elemento neutro de `operate`
- `operate` será una operación asociativa.

Por ejemplo, en el caso del cuantificador Sum, `unit` devolverá 0 y `operate` será `left + right`.

A. **(2 puntos)** ¿Qué patrones se han usado en el diseño de las clases y métodos del diagrama? Justifica brevemente la respuesta.

B. **(2 puntos)** Implementad las clases, interfaces y métodos involucrados

   - `Sum`, como se ha indicado, es el cuantificador que suma los enteros que son los valores de las subexpresiones
   - `Concat` es el cuantificador que concatena las cadenas que son los valores de las subexpresiones.

C. **(4 puntos)** Como las variables pueden cambiar de valor, se desea poder **observar** los cambios de valores de las expresiones. Modificad la implementación para conseguirlo, de manera que:

   - solamente se notifique en casos en que realmente se varía el resultado de la evaluación
   - al notificar no se envía información adicional, es decir, se desea que apliquéis la **versión pull** del patrón observador.

   Mostrad el diagrama de clases resultante.

D. (2 puntos) Finalmente, modificad el diseño original (apartado B), de manera que la clase `Quantifier<E>` use el **patrón estrategia** para acomodar las diferentes operaciones. Implementad las nuevas clases y/o interfaces que necesitéis para aplicar el patrón, la nueva clase `Quantifier<E>` y la nueva clase `Sum`.

Por si os es de utilidad, os recuerdo los métodos de la clase `Observable`:

- `public void addObserver(Observer o) { ... }`
- `public void deleteObserver(Observer o) { ... }`
- `public void deleteObservers() { ... }`
- `public int countObservers() { ... }`
- `public void notifyObservers() { ... }`
- `public void notifyObservers(Object arg) { ... }`
- `protected void setChanged() { ... }`
- `protected void clearChanged() { ... }`
- `public boolean hasChanged() { ... }`

Y de la interfaz `Observer`:

- `public void update(Observable o, Object arg)`