TP 5: Clases Abstractas vs. Interfaces

Programación Orientada a Objetos II
Comisión 2
2º Cuatrimestre de 2024

Nicolás Salvanescki

1 Actividad de lectura: clases abstractas e interfaces

1. Las ventajas polimórficas que presentan las interfaces son:

    1. En java no se permite la herencia múltiple, sin embargo permite la implementación de múltiples interfaces en una clase. Esto significa que se puede adquirir la firma (signatura) de varias interfaces para luego definirles comportamiento en la clase.
    2. Permite un mayor desacoplamiento, al desprenderse la implementación específica de la clase. Al tener que responder a los métodos de la interfaz, se puede abstraer de la implementación.
    3. Al permitir compartir un contrato (o protocolo) en común, las interfaces facilitan el intercambio entre las clases y sin relación jerárquica. Esto permite mayor flexibilidad en la arquitectura. Un humano y un perro son muy distintos entre si pero comparten, por ejemplo, la interfaz de ser Moviles (caminar, correr, detenerse, etc).

    Los puntos anteriores mejoran el polimorfismo en el código, ya que estas clases que comparten una interfaz, pueden responder a los mensajes de la interfaz sin importar la implementación interna o extra que tengan.

2. Porque en algún momento el único aspecto que compartan esas clases serán los mensajes que saben responder polimórficamente, pero el resto de estado y comportamiento que NO tengan en común va a estorbar e impedir una correcta herencia de clases (al heredar métodos o atributos que no se van a utilizar en la subclase). En estos casos, las interfaces proveen de una solución más granular en la cual dicha interfaz es único que tienen en común.

3. Que permiten definir comportamiento y/o estado en común a todas las subclases. Las interfaces, en cambio, no permiten definir estado (ya que podría presentar problemas heredar múltiples estados que colisionen) ni comportamiento, solo protocolo.

4. No se puede, solo define un protocolo, no tiene implementado nada así que no tendría sentido instanciar una interfaz.

5. Porque podría romper con el polimorfismo en alguna parte. La interfaz está definida para que las clases que la implementen sepan responder a los mensajes que define, si modificaramos alguna firma de la interfaz, alguna parte del código podría romperse al no estar entendiendo el mensaje.

6. En smalltalk todos son objetos, y todos los objetos saben responder a un mensaje con solo tener definido el método. Es decir que, a diferencia de C o Java, no hay una estructura rígida o estática en los tipos, es un lenguaje puro de objetos.

2 Interfaces, Colecciones y otras yerbas

Para reemplazar XXX, YYY, WWW y ZZZ por las clases Colection, List, ArrayList y LinkedList, debería fijarme simplemente cuales saben responder los mensajes que envían los métodos a dichas colecciones.

En el método **getFirstFrom**, el mensaje **get** que se envía a la colección pasada por parámetro, irónicamente no puede entenderlo la interfaz Collection. Sin embargo, List y las clases que la implementan (ArrayList y LinkedList) pueden entenderlo, por lo que XXX podría ser cualquiera de ellas. Para ser lo más general y polimórfico posible, XXX debería ser List.

En **addLast**, el mensaje **add** puede entenderlo Collection y cualquier clase o interfaz que la implemente. Por lo tanto, cualquiera de las clases anteriormente mencionadas podría reemplazar al tipo YYY. Lo correcto sería que YYY sea reemplazada por la interfaz Collection.

En **getSubCollection**, el mensaje **subList**, al igual que **get**, puede ser entendido por todas aquellas clases e interfaces que implementen List. Para variar, ArrayList podría perfectamente reemplazar al tipo ZZZ. Sin embargo, el tipo de retorno de la función es Collection, habría que hacer, además, un upcasting del tipo List al tipo Collection.

Por último, en **containsElement**, el mensaje **contains** puede ser entendido por todas las clases que implementen Collection, por lo que LinkedList podría reemplazar WWW.

3 Interfaces, Colecciones y otras yerbas

4. No fue necesario porque ambas entienden el mensaje **getName**. El mecanismo de abstracción que permite esto es el polimorfismo.

5. No se puede asegurar con herencia. Sin embargo, al utilizar interfaces podría crear una llamada **Nameable** que solo tenga la firma getName y, al permitir multi implementación Java, podría hacer que Persona y Mascota implementen dicha interfaz sin importar la herencia, jerarquía o que otras interfaces más implementen.