Acceso variable directo

Debe acceder a Common State de la forma más legible posible.

¿Cómo se obtiene y establece el valor de una variable de instancia?

El estado de acceso es un tema muy parecido a la inicialización. Hay dos buenas respuestas. Uno es más legible. Uno es más flexible. También como la inicialización, encontrará adherentes dogmáticos de ambos enfoques.

La forma simple y legible de obtener y establecer los valores de las variables de instancia es usar las variables directamente en todos los métodos que necesitan sus valores. La alternativa requiere que envíe un mensaje cada vez que necesite usar o cambiar el valor de una variable de instancia.

Cuando comencé a programar en Smalltalk, en Tektronix a mediados de los 80, el debate sobre cómo acceder al estado era candente. Había facciones en ambos lados haciendo declaraciones importantes (¿no te encantan los laboratorios de investigación?)

La multitud de Acceso Variable Indirecto se ha ganado el corazón y la mente del público de Smalltalking, sobre todo creo que porque la mayoría de las empresas de formación de alto volumen enseñan acceso indirecto. El problema no es tan simple como "el acceso directo es malo, el acceso indirecto es bueno".

Traté de reabrir el debate en mi columna del Informe Smalltalk hace unos años. Comenzó un pequeño incendio de matorrales que finalmente se apagó. Me preguntaba si estaba dando demasiada importancia a los méritos del acceso directo. Tal vez debería haberme ido lo suficientemente bien solo.

Luego pasé varios meses trabajando para un cliente que insistía en el acceso indirecto. Un programador profesional puede adoptar cualquiera de varios estilos a voluntad, así que yo era un buen soldado y escribía mis getters y setters.

Una vez que terminé con ese cliente, escribí un código para mí. Me sorprendió lo mucho que se leía mejor que en lo que había estado trabajando anteriormente. La única diferencia de estilo fue el acceso directo frente al indirecto.

Lo que noté fue que cada vez que leía:

. . . yo x ...

Hice una pausa por un momento para recordarme a mí mismo que el mensaje "x" solo estaba obteniendo una variable de instancia.
Cuando leo:

. . . X...

Seguí leyendo.

Le conté a Ward mi experiencia. Tenía otra buena explicación. Cuando escribe clases que solo tienen un puñado de métodos, agregar un método de obtención y configuración puede duplicar fácilmente el número de métodos en su clase. El doble de métodos para comprarle flexibilidad que tal vez nunca use.

Por otro lado, es muy frustrante obtener una clase de alguien y ver la oportunidad de subclasificarla rápidamente, solo para descubrir que usaron Direct Variable Access, por lo que no hay forma de realizar los cambios que desea sin cambiar la mayor parte del código en la superclase.

Acceda y establezca variables directamente.

Acceso variable indirecto

Debe acceder a Common State de la manera más flexible posible.

¿Cómo se obtiene y establece el valor de una variable de instancia?

Ahora tengo que mostrar mi verdadera esquizofrenia. Habiéndote convencido en Direct Variable Access de que solo usar variables es lo suficientemente bueno, te voy a pedir que ignores a ese loco bastardo y me escuches decir algo con sentido común aquí.

Cuando utiliza el acceso directo a variables, muchos métodos suponen que el valor de una variable es válido. Para algunos códigos, esta es una suposición razonable. Sin embargo, si desea introducir la inicialización diferida, dejar de almacenar y comenzar a calcular el valor, o si desea cambiar los supuestos en una nueva subclase, se sentirá decepcionado de que el supuesto de validez esté muy extendido.

La solución es utilizar siempre métodos de obtención y métodos de configuración para acceder a las variables. Por lo tanto, en lugar de:

```
Punto >> + aPunto
    ^ x + aPunto x @ (y + aPunto y)
```

tu verias:

```
Punto >> + aPunto
    ^ self x + aPoint x @ (self y @ aPoint y)
```

Si explora las referencias de variable de instancia de una clase que usa Acceso de variable indirecto, solo verá dos referencias a cada variable, el método de obtención y el método de configuración.

Lo que renuncia con el acceso variable indirecto es la simplicidad y la legibilidad. Tienes que definir todos esos métodos de obtención y métodos de configuración. Incluso si su sistema lo hace por usted, tiene muchos más métodos para administrar y documentar. Como se explicó anteriormente, el código que usa Direct Variable Access se lee sin problemas, porque no siempre se está recordando a sí mismo "Oh, sí, eso es solo un método de obtención".

Acceda y cambie el valor de una variable de instancia solo a través de un método de obtención y un método de configuración.

Si necesita codificar la herencia, use el acceso variable indirecto. Las generaciones futuras se lo agradecerán.

Deberá definir un método de obtención y un método de configuración para cada variable. Para las variables que contienen colecciones, considere la posibilidad de implementar métodos de acceso a colecciones y un método de enumeración.

Método de obtención

Está utilizando Inicialización diferida o Acceso variable indirecto.

¿Cómo proporciona acceso a una variable de instancia?

Una vez que haya decidido utilizar el acceso indirecto a variables, se compromete a proporcionar un protocolo basado en mensajes para obtener y establecer valores de variables. Las únicas preguntas reales son cómo lo usa y cómo lo llama.

Aquí está el verdadero secreto de escribir buenos métodos de obtención: ***hazlos privados al principio.*** No puedo enfatizar esto lo suficiente. Estará bien si el mismo objeto invoca e implementa el método Getting. Otra forma de decir esto es que siempre envía mensajes que invocan Getting Methods a "uno mismo".

Algunas personas intentan asegurar esto poniendo el prefijo “my” al nombre del método. Por lo tanto, en lugar de:

```
X
    ^ x
```

tienes:

```
myX
    ^ x
```

Esto asegura que el código como:

```
autolimites origen myX
```

parece estúpido. No siento que esto sea necesario. Prefiero darles a los programadores (incluido yo mismo) el beneficio de la duda. Si algún otro objeto tiene que enviar absolutamente mi método de obtención privado, entonces debería poder hacerlo de la manera más legible posible.

Proporcione un método que devuelva el valor de la variable. Dale el mismo nombre que la variable.

Hay casos en los que publicará la existencia de Getting Methods para su uso en el mundo exterior. Debe tomar una decisión consciente para hacer esto después de considerar todas las alternativas. Es preferible dar más responsabilidad a un objeto que hacer que actúe como una estructura de datos.

Método de configuración

Está utilizando el acceso variable indirecto.

¿Cómo se cambia el valor de una variable de instancia?

Todo lo que dije una vez sobre la obtención de métodos, me gustaría decirlo dos veces sobre la configuración de métodos. Los métodos de configuración deberían ser aún más privados. Una cosa es que otro objeto elimine su estado, y otra muy distinta es golpear en un nuevo estado. Las posibilidades de que el código de los dos objetos se desincronice y se rompa de manera confusa son innumerables.

Revisando la denominación, no creo que sea necesario anteponer "mi" a los nombres de los métodos de configuración. Puede proporcionar un poco más de protección contra el uso no autorizado, pero no creo que la dificultad adicional para leer valga la pena.

Proporcione un método con el mismo nombre que la variable que tome un solo parámetro, el valor que se establecerá.

Incluso si uso el acceso a variable indirecta, si tengo una variable que solo se establece en el momento de la creación de la instancia, no proporcionaría un método de configuración. Usaría el método de parámetro de creación para establecer todos los valores a la vez. Sin embargo, una vez que tenga un método de configuración, debe usarlo para todos los cambios en la variable.

Establezca propiedades booleanas con un método de configuración de propiedades booleanas.

Método de acceso a la colección

Está utilizando el acceso variable indirecto.

¿Cómo proporciona acceso a una variable de instancia que contiene una colección?

La solución más simple es publicar un método Getting para la variable. De esa manera, cualquier cliente que desee puede agregar, eliminar, iterar o usar la colección.

El problema con este enfoque es que abre demasiado la implementación de un objeto al mundo exterior. Si el objeto decide cambiar la implementación, digamos usando un tipo diferente de colección, el código del cliente puede fallar.

El otro problema con solo ofrecer sus colecciones privadas para la visualización pública es que es difícil mantener actualizado el estado relacionado cuando alguien más está cambiando la colección sin notificárselo. Aquí hay un departamento que usa una variable de instancia de almacenamiento en caché para acelerar el acceso a su salario total:

```
Departamento
        superclase: Objeto
        variables de instancia: empleados totalSalary totalSalary

        totalSalary isNil ifTrue: [totalSalary: = self computeTotalSalary]. ^ totalSalario

    computeTotalSalary
        ^ empleados
            inyectar:   0
            en: [: suma: cada uno | suma + cada salario]
    clearTotalSalary
        totalSalario: = nulo
```

¿Qué sucede si el código de cliente elimina a un empleado sin notificar al departamento?

```
. . . a Los empleados del departamento eliminan: un Empleado ...
```

La caché totalSalary nunca se borra. Ahora contiene un número que no coincide con el valor devuelto por computeTotalSalary.

La solución a esto es no dejar que otros objetos tengan sus colecciones. Si usa el acceso a variable indirecta, asegúrese de que los métodos de obtención para las colecciones sean privados. En su lugar, dé a los clientes acceso restringido a las operaciones de la colección a través de los mensajes que implemente. Esto le da la oportunidad de realizar cualquier otro procesamiento que necesite.

La desventaja de este enfoque es que debe implementar todos estos métodos. Dar acceso a una colección requiere un método. Es posible que necesite cuatro o cinco métodos para proporcionar todo el acceso protegido necesario a la colección. Sin embargo, a largo plazo, vale la pena, porque su código se leerá mejor y será más fácil de cambiar.

Implementar métodos que se implementan con Delegación en la colección. Para nombrar los métodos, agregue el nombre de la colección (en forma singular) a los mensajes de la colección.

Si el Departamento quisiera permitir que otros agreguen y eliminen empleados, implementaría los métodos de acceso a la colección:

```
addEmployee: anEmployee
    self clearTotalSalary.
    los empleados agregan: anEmployee
removeEmployee: anEmployee
    self clearTotalSalary.
    los empleados eliminan: anEmployee
```

No se limite a nombrar ciegamente un método de acceso de colección después del mensaje de colección que delega. Vea si puede encontrar una palabra del dominio que tenga más sentido. Por ejemplo, prefiero:

```
emplea: anEmployee
    ^ empleados incluye: un empleado
```

a:

```
incluye Empleado: un Empleado
    ^ empleados incluye: un empleado
```

Implemente un método de enumeración para un acceso general a la colección seguro y eficiente.