GETTERS Y SETTERS

Los getters y setters se utilizan ampliamente en Java. Parece muy simple, pero no todos los programadores pueden comprender e implementar este método correctamente.

Getter y setter

En Java, los getters y setters son dos métodos tradicionales para recuperar y actualizar valores de variables.

El siguiente código es un ejemplo de una clase simple con variables privadas y dos métodos getter / setter:

```
public class SimpleGetterAndSetter {

    private int number;

    public int getNumber() {

        return this.number;

    }

    public void setNumber(int num) {

        this.number = num;

    }

}
```

Esta clase declara una variable privada Number. Dado que Number es privado, el código fuera de esta clase no puede acceder directamente a las variables, como se muestra a continuación:

```
SimpleGetterAndSetter obj = new SimpleGetterAndSetter();

obj.number = 10;    // compile error, since number is private

int num = obj.number; // same as above
```

En su lugar, el código externo debe llamar a getter, getNumber () y planner, setNumber () para leer o actualizar variables, por ejemplo:

```
SimpleGetterAndSetter obj = new SimpleGetterAndSetter();

obj.setNumber(10);  // OK

int num = obj.getNumber();  // fine
```

Por lo tanto, un establecedor es un método para actualizar el valor de una variable. Un captador es un método para leer el valor de una variable. Los getters y setters también se denominan descriptores de acceso y mutantes en Java.

Necesidad

Mediante el uso de getters y setters, los programadores pueden controlar cómo acceder y actualizar sus variables importantes de una manera adecuada, como cambiar el valor de una variable dentro de un rango específico. Considere el siguiente código para el método de establecimiento:

```
public void setNumber(int num) {

    if (num < 10 || num > 100) {

        throw new IllegalArgumentException();

    }
```

```
    this.number = num;

}
```

Esto asegura que el valor del número siempre se establezca entre 10 y 100. Suponiendo que el número de variable se puede actualizar directamente, la persona que llama puede establecer cualquier valor para él:

```
obj.number = 3;
```

Esto viola la restricción de que el valor de esta variable varía de 10 a 100. Por supuesto, no queremos que esto suceda. Por lo tanto, oculte el número de variable como privado y luego use el setter para guardarlo.

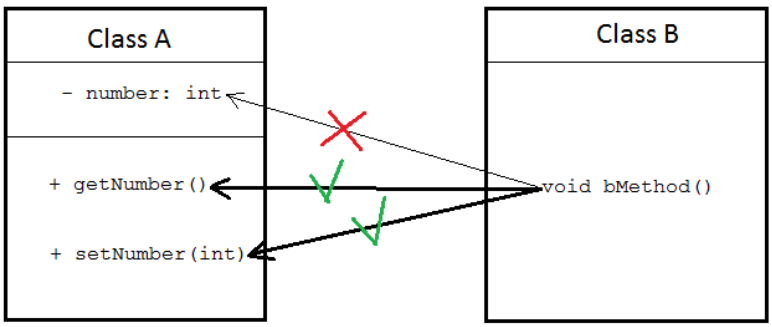
Por otro lado, el método getter es la única forma que tiene el mundo exterior de leer el valor de una variable:

```
public int getNumber() {

    return this.number;

}
```

La siguiente figura ilustra la situación:



Hasta ahora, los métodos setter y getter protegen el valor de las variables de cambios inesperados en el mundo exterior (código de llamada).

Cuando una variable es un modificador privado y solo se puede acceder a través de getter y setter, es encapsulación. La encapsulación es uno de los principios básicos de la programación orientada a objetos (OOP), por lo que implementar getter y setter es uno de los métodos para hacer cumplir la encapsulación en código de programa.

Algunos marcos, como Hibernation, Spring y Pillars, pueden verificar la información a través de getters y setters o inyectar su código de utilidad. Por lo tanto, al integrar código con dichos marcos, es necesario proporcionar getters y setters.

Convenciones de nomenclatura de Getter y Setter

El esquema de nomenclatura de setter y getter debe seguir las convenciones de nomenclatura de Javabean como getXxx () y setXxx (), donde es el nombre de la variable Xxx. Por ejemplo, con los siguientes nombres de variable:

```
private String name;
```

El setter y getter apropiados serían:

```
public void setName(String name) { }

public String getName() { }
```

Si la variable es booleana, entonces el nombre del captador puede ser isXXX () o getXXX (), pero se prefiere el primero. P.ej:

```
private boolean single;

public String isSingle() { }
```

Errores comunes al implementar getter y setter

La gente suele cometer errores y los desarrolladores no son una excepción. Esta sección describe los errores y las soluciones más comunes al implementar setters y getters en Java.

Error 1: tiene establecedores y getters, pero la variable se declara en un ámbito menos restrictivo.

Considere el siguiente fragmento de código:

```
public String firstName;

public void setFirstName(String fname) {

    this.firstName = fname;

}

public String getFirstName() {

    return this.firstName;

}
```

La variable firstName se declara pública, por lo que se puede acceder mediante el punto (.). El funcionamiento directo inutiliza la incubadora y el aspirador. La solución a esta situación es utilizar modificadores de acceso más restringidos, como protected y private:

```
private String firstName;
```

Error 2: asignar referencias de objeto directamente en el setter

Considere el siguiente método de establecimiento:

```
private int[] scores;
```

```
public void setScores(int[] scr) {

    this.scores = scr;

}
```

El siguiente código demuestra este problema:

```
int[] myScores = {5, 5, 4, 3, 2, 4};

setScores(myScores);

displayScores();

myScores[1] = 1;

displayScores();
```

Una matriz de enteros, myScores, inicializa la matriz con 6 valores (línea 1) y pasa la matriz al método setScores () (línea 2). El método displayScores () solo necesita imprimir todas las puntuaciones en la matriz:

```
public void displayScores() {

    for (int i = 0; i < this.scores.length; i++) {

        System.out.print(this.scores[i] + " ");

    }

    System.out.println();

}
```

La línea 3 producirá el siguiente resultado:

```
5 5 4 3 2 4
```

Estos son todos los arreglos myScores. Ahora, en la fila 4, podemos modificar el valor de 2. La matriz myScores en el elemento Nd es la siguiente:

```
myScores[1] = 1;
```

¿Qué pasa si llamamos a este método? displayScores () está en la línea 5 de nuevo? Entonces, producirá el siguiente resultado:

```
5 1 4 3 2 4
```

Sabe que el valor Nd elemento de 2 se cambia de 5 a 1, que es el resultado de la asignación en la fila 4. ¿Porque es esto importante? Esto significa que los datos se pueden modificar fuera del alcance del método de establecimiento, lo que anula el propósito de encapsulación del establecedor. ¿Por qué pasó esto? Veamos nuevamente el método setScores ():

```
public void setScores(int[] scr) {

    this.scores = scr;

}
```

La puntuación de la variable miembro se asigna directamente a la variable de parámetro del método scr. Esto significa que ambas variables apuntan al mismo objeto en la memoria: el objeto de matriz myScores. Por lo tanto, la variable puntajes o myScores se crea realmente en el mismo objeto.

La solución a esta situación es pasar de la matriz scr a la matriz de puntuaciones, una por una. La versión modificada de la incubadora es la siguiente:

```
public void setScores(int[] scr) {

    this.scores = new int[scr.length];

    System.arraycopy(scr, 0, this.scores, 0, scr.length);

}
```

¿Que importa? La variable miembro de puntuaciones ya no se refiere a la variable scr. En cambio, las puntuaciones de la matriz se inicializan en una nueva matriz con un tamaño igual al tamaño de la matriz. scr ... Luego, copiamos todos los elementos de la matriz. Para examinar las puntuaciones de la matriz, utilice el método System.arraycopy ().

Ejecute el siguiente ejemplo nuevamente y nos dará el siguiente resultado:

```
5 5 4 3 2 4

5 5 4 3 2 4
```

Ahora, dos llamadas a displayScores () producen la misma salida. Esto significa que las puntuaciones de la matriz son independientes y diferentes de la matriz. El scr se pasa al planificador, entonces tenemos la tarea:

```
myScores[1] = 1;
```

Esto no afecta a la matriz. puntuaciones.

Por lo tanto, la regla general es: si pasa una referencia de objeto al método setter, no copie la referencia directamente en la variable interna. En su lugar, debería encontrar alguna forma de copiar el valor del objeto pasado en el objeto interno, al igual que usamos el método System.arraycopy ().