Methods and Encapsulation

Scope de variables

El Scope o alcance de las variables especifíca su vida o visibilidad.

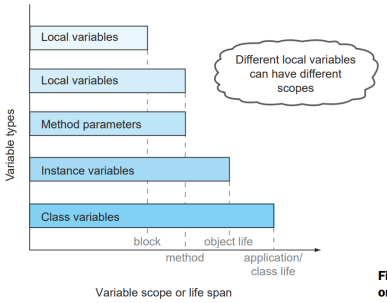


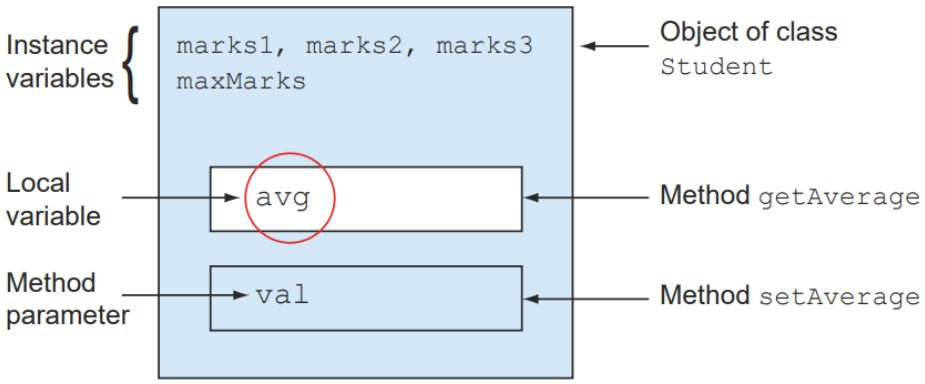
Figure 3.8 Comparing the scope, or life span, of all four variables

Variables locales

Las variables locales se encuentran definidas dentro de un método.

Pueden estar definidas dentro de estructuras como `if else` , loops o `switch` .

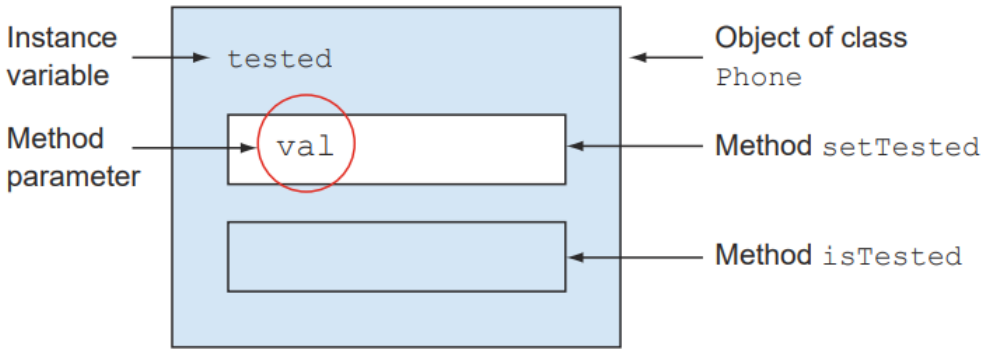
Estas variables son las que tienen el menor alcance.



En el caso de variables ocales no se permite su uso antes de su inicialización → No se inicializan automáticamente.

Parámetros de métodos

Corresponden a las variables que son aceptadas en la firma del método. Estas son accesibles únicamente dentro del método que la define.

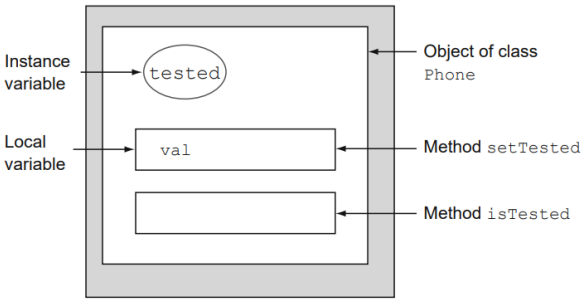


Variables de instancia

Las variables de instancia, o atributos de objetos tienen un ciclo de vida relacionado a la vida del objeto.

Estas variables se declaran dentro de la clase y fuera de los métodos.

Son accesibles para toda la instancia y clase.



Variables de clase

Estas variables se caracterizan por emplear `static` .

No dependen de la creación de objetos.

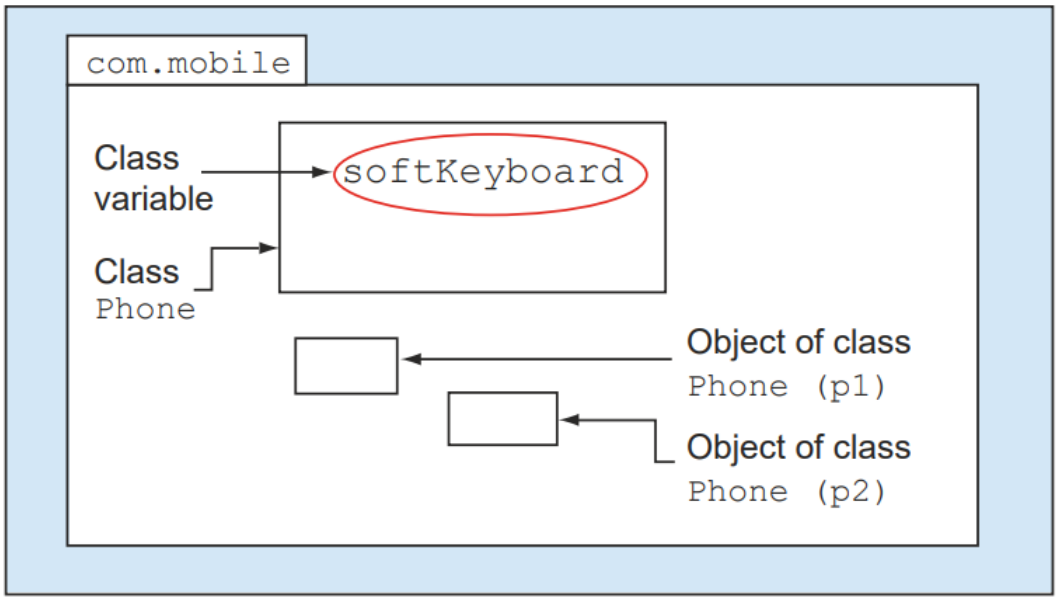
Estas variables pertenecen a la clase, no a un objeto individual.

Una variable de clase es compartida por todos los objetos de esa clase, no se tiene una copia por objeto.

Para acceder a una variables de clase no se necesita un objeto para invocarla, puede ser accedida utilizando el nombre de la clase.

Pueden ser modificadas a través de los objetos, recordar que no se tienen copias, por lo que se modificaría la original.

Se puede acceder a las variables de clase a través de una variable de referencia que no apunte a un objeto.



Variables con el mismo nombre en diferente Scope

Algunas reglas para prevenir conflictos son necesarias:

- No se puede definir una variable de clase y una de instancia con el mismo nombre. → No compila.
- Variables locales y de método tampoco pueden compartir nombre. → No compila.

Una clase puede definir variables locales con el mismo nombre que las variables de instancia, aquí ocurre el shadowing.

Twist in the Tale 3.1

The class Phone defines a local variable and an instance variable, phoneNumber, with the same name. Examine the definition of the method setNumber. Execute the class on your system and select the correct output of the class TestPhone from the given options:

```
class Phone {
 String phoneNumber = "123456789";
 void setNumber () {
     String phoneNumber;
     phoneNumber = "987654321";
 }
}
class TestPhone {
 public static void main(String[] args) {
 Phone p1 = new Phone();
 p1.setNumber();
 System.out.println (p1.phoneNumber);
 }
}
```

a 123456789
b 987654321

c No output
d The class Phone will not compile.

Ciclo de vida de los objetos.

La vida de un objeto comienza cuando se crea, y finaliza hasta que el Scope termina o no es referenciado por una variable.

Nacimiento de un objeto

Un objeto nace cuando se utiliza el operador `new` .

Es importante diferenciar entre declarar una variable e inicializarla.

Si se crea un objeto sin referenciarlo a una variable no podrá ser usado.

Objeto accesible

Cuando el objeto es creado puede ser utilizado mediante su variable de referencia. Esto hace que permanezca en memoria hasta que el Scope finalice o su variable de referencia deje de apuntarlo.

Cuando un objeto deja de ser accesible es considerad por el recolector de basura.

Objeto inaccesible

Un objeto se vuelve inaccesible si se termina el Scope, o su referencia se pierde.

Recolector de basura

El colector tienen una prioridad baja en los hilos de ejecución.

Únicamente se puede determina la elegibilidad de los objetos, no se puede determinar o mandar directamente al colector, es lo controla la JVM.

El método `System.gc()` hace una petición al recolector, pero esta llamada no garantiza que se realice el trabajo.

En el caso de islas aisladas, el recolector las puede detectar.

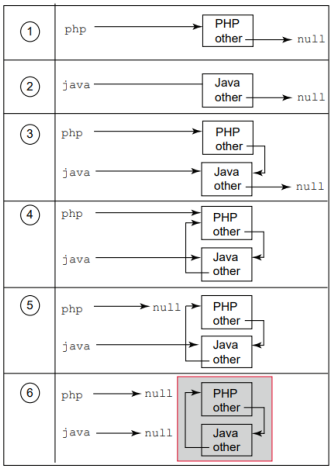


Figure 3.12 A group of instances with no external references forms an island of isolation, which is eligible for garbage collection.

Crear métodos con argumentos y valores de retorno

Un método es un grupo de sentencias identificadas con un nombre.

Los métodos definen los comportamientos de un objeto.

Tipo de retorno

El tipo de retorno hace referencia al tipo del valor que el método va a regresar.

Si el método no regresa nada → `void`

Los métodos pueden regresar primitivos o cualquier clase o interface.

Si no hay retorno no se puede asignar el resultado del método a una variables → No compila

Si hay retorno no se esta obligado a recuperarlo en una variable.

Parámetros de método

Teóricamente no existe límite en el número de parámetros que se pueden definir en un método.

- Parámetro → Variables que aparecen en la firma del método
- Argumento → Valor que se pasa al método para utilizarlo.

Varargs

La ellipsis `...` que sigue a un dato indica que se pueden recibir `n` argumentos o un arreglo.

únicamente se pueden tener un *variable argument* en la lista y tiene que ser la última variable.

Retorno

`return` es utilizado para salir de un método con o sin valor.

Para métodos `void` se puede utilizar `return` sin valor u omitirlo.

En caso de que haya código después de un `return` se marcara error de compilación por código inalcanzable.

Sobrecarga

Reglas para definir métodos sobrecargados

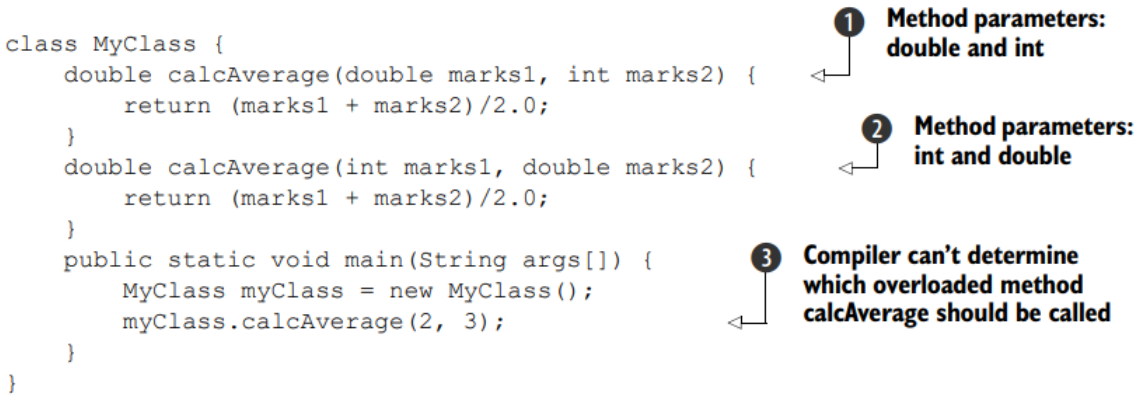
- Tienen el mismo nombre
- Tiene diferente lista de parámetros
- Puede o no definir un diferente tipo de retorno
- Puede o no definir un diferente nivel de acceso
- No puede sobrecargarse cambiando únicamente el tipo de retorno o sus modificadores de acceso o ambos.

Lista de argumentos

Los métodos sobrecargados aceptan una diferente lista de argumentos.

- Los cambios en el número de parámetros son aceptados
- Cambios en el tipo de parámetros son aceptados.
- Cambio en la posición de parámetros es aceptado (basado en tipo).
- No se permite el overload remplazando varargs por arreglos. → No compila

Caso especial dubious



Tipo de return

No se puede considerar sobrecarga si únicamente difiere el tipo de retorno, porque el tipo de retorno no es parte de la firma. → Error de compilación

Nivel de acceso

No se puede sobrecargar métodos si únicamente difieren en su nivel de acceso. → No compila

Constructores de clase

Los constructores son métodos que permiten crear y retornar un objeto de la clase en la que están definidos

Los constructores tiene el mismo nombre de la case y no especifican valor de retorno.

Un constructor puedo cumplir lo siguiente:

- Llamar al constructor de la superclase, esta puede ser una llamada implicita o explicita a través de `super()`
- Inicializar las variables de instancia con sus valores por defecto.

Constructor definido por el usuario

El autor de la clase puede o no definir un constructor en la clase. Si el autor define un constructor, este es conocido como *constructor definido por el usuario*

Al ser el constructor un método, es posible pasar parámetros.

Se pueden definir constructores con cualquiera de los niveles de acceso.

No lleva retorno, en caso de que se defina un retorno, se interpretará como un método más.

Bloque de inicialización vs Constructores

Un bloque de inicialización se define dentro de la clase. Se ejecuta cada vez que un objeto de la clase es creado.

El bloque de inicialización de ejecuta antes que el constructor.

El orden de ejecución entre varios bloques de inicialización depende de su organización en el código, pero siempre van antes que el constructor.

Constructor default

El constructor default es insertado por el compilador cuando no se encuentra un constructor definido por el usuario.

Este constructor no acepta argumentos y asigna valores por defecto a las variables de instancia.

La accesibilidad del constructor default coincide con la accesibilidad de su clase.

Si se encuentra un constructor definido por el usuario no se añadirá el constructo default.

Sobrecarga de constructores

La sobrecarga de constructores sigue las mismas reglas que la sobrecarga de métodos:

- Se debe tener mismo nombre, diferente lista de argumentos.
- Solo el cambio de visibilidad no permite sobrecarga.
- Los constructores sobrecargados pueden ser definidos con diferentes modificadores de acceso.

Invocando constructores sobrecargados desde otros constructores

Los constructores se invocan utilizando `this()` y esta debe ser la primera línea del código o no compilara.

Únicamente se puede llamar a un único constructor dentro de otro constructor, pues debe ser la primera línea.

No se puede invocar al constructor en otro método de la clase.

Twist in the Tale 3.2

Let's modify the definition of the class Employee that I used in the section on overloaded constructors, as follows:

```
class Employee {
 String name;
 int age;
 Employee() {
     this();
 }
 Employee (String newName, int newAge) {
     name = newName;
     age = newAge;
 }
}
```

Question: What is the output of this modified code, and why?

Error de compilación → A method calling itself is called recursion. Two or more methods calling each other, in a circular manner, is called circular method calling.
Starting in Java version 1.4.1, the Java compiler won't compile code with recursive or circular constructors.

Accediendo a los atributos

¿Qué es un campo de objeto?

Un campo de objeto es otro nombre que recibe un variable de instancia o atributo.

- Un setter o mutador es un método utilizado para asignar valor a la variable.
- Un getter es un método para obtener el valor de la variable.

Leer y escribir atributos

- Utilizando métodos
- Utilizando constructores para escribir valores en las variables
- Accediendo directamente a la variable

Los atributos se inicializan con los valores default. `int -> 0` `Object -> null`

Llamando métodos en objetos

Puedes llamar a los métodos definidos en una clase utilizando una variable de referencia.

Java utiliza la notación de punto `.` para ejecutar un método en una variable de referencia.

Si un método no regresa nada y se usar como argumento hay error de compilación.

Aplicando principios de encapsulación

Necesidades de encapsular

Razone para encapsular en Java:

- Prevenir que un objeto extraño ejecute operaciones peligrosas.
- Ocultar detalles de la implementación, y poder modificarla sin tener impacto en otras aplicaciones.
- Minimizar el acoplamiento

Twist in the Tale 3.3

Let's modify the definition of the class Phone that I previously used to demonstrate the encapsulation principle in this section. Given the following definition of class Phone, which of the options, when replacing the code on lines 1–3, makes it a well-encapsulated class?

```
class Phone {
 public String model;
 double weight; //LINE1
 public void setWeight(double w) {weight = w;} //LINE2
 public double getWeight() {return weight;} //LINE3
}
```

Options:

```
a public double weight;
    private void setWeight(double w) { weight = w; }
    private double getWeight() { return weight; }

b public double weight;
    void setWeight(double w) { weight = w; }
    double getWeight() { return weight; }

c public double weight;
    protected void setWeight(double w) { weight = w; }
    protected double getWeight() { return weight; }

d public double weight;
    public void setWeight(double w) { weight = w; }
    public double getWeight() { return weight; }

e None of the above
```

Pasando objetos y primitivos a métodos.

Pasando primitivos

El valor de un dato primitivo en copiado y pasado al método. En consecuencia, la variable del que fue copiado no cambia.

Para acceder a la variable de instancia es recomendable utilizar el prefijo `this` .

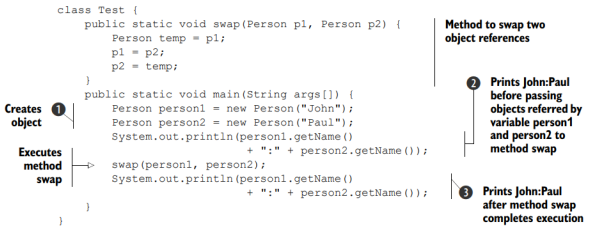
Pasado referencias de objetos

Hay dos casos principales.

- Cuando un método reasigna la referencia pasada a otra variables
- Cuando un método modifica el estado de un objeto pasado

Cuando un método reasigna la referencia del objeto pasado

Cuando se pasa una referencia de objeto a un método, el método puedo asignarla a otra variable y conservar una copia de la referencia.



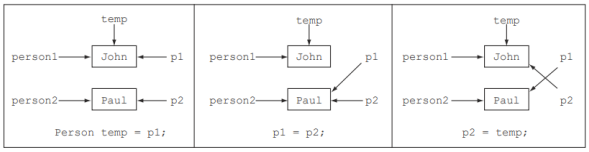
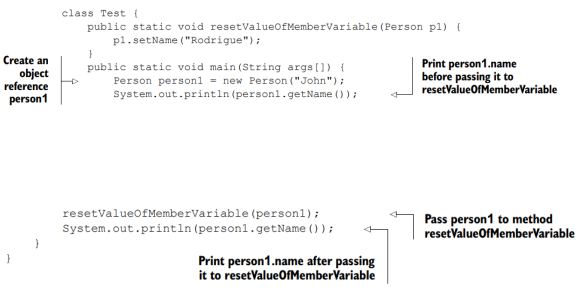


Figure 3.24 The change in the objects referred to by variables during the execution of the method `swap`

Cuando un método modifica el estado de un objeto pasado

What's the output of the following code?



The output of the preceding code is as follows:

```
John
Rodrigue
```