ORACLE

Academy

Java Foundations

7-5

Interacción y encapsulación de objetos

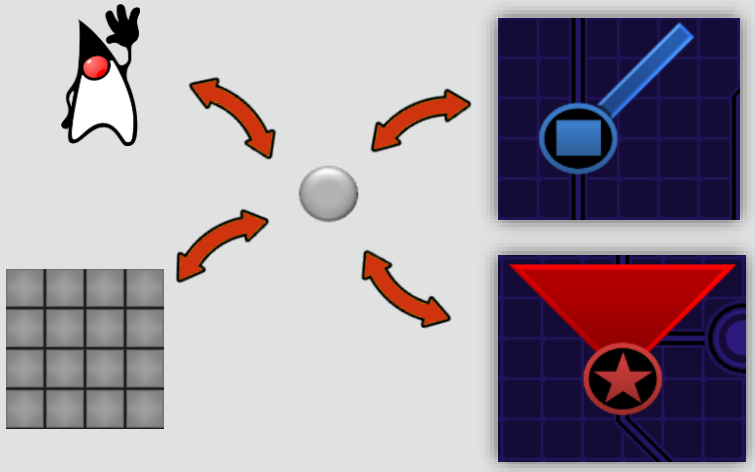
ORACLE
Academy

Objetivos

- En esta lección se abordan los siguientes objetivos:
  - Comprender la interacción de objetos en profundidad
  - Utilizar el modificador private para definir variables de clase
  - Comprender el objetivo de los métodos getter
  - Comprender el objetivo de los métodos setter

Interacción de objetos

- En la sección 2 se presentó la idea de interacción de objetos
  - No se ha prescrito ninguna secuencia sobre cómo deben interactuar los objetos
- En esta lección se explora cómo programar las interacciones



¿Qué es la interacción de objetos?

- Una referencia de objeto es una dirección de memoria
  - Una referencia dirige un objeto a otro
  - Una referencia permite que un objeto interactúe con otro
- Los objetos interactúan mediante...
  - El acceso de los campos de otro objeto
  - La llamada a los métodos de otro objeto
- Si el método main instancia cada objeto...
  - El método main contiene todas las referencias de objeto
  - El método main puede acceder a los campos y métodos de todos los objetos

Programa de ejemplo

- Imagine un programa que modela los objetos Prisoner, Cell y Guard
- El método main puede parecerse a este:

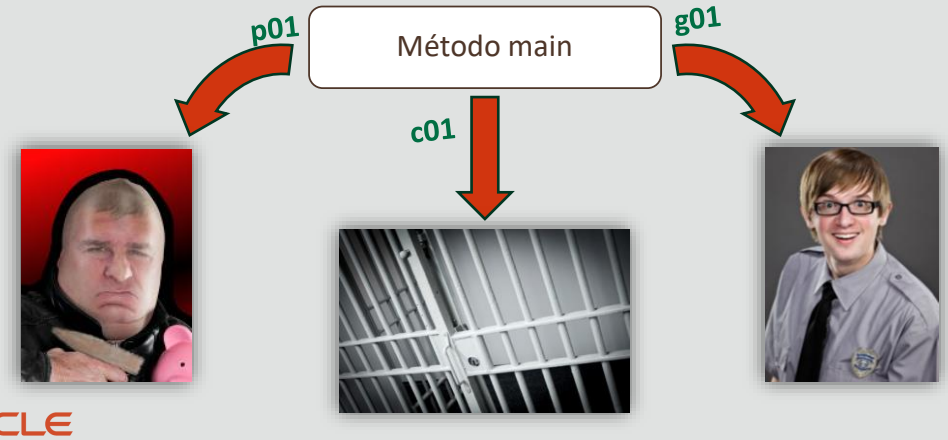
```
public class PrisonTest{
   public static void main(String[] args){
      Prison p01 = new Prisoner();
      Cell   c01 = new Cell();
      Guard  g01 = new Guard();

      p01.name = "Bubba";
      c01.name = "A1";
      g01.name = "Boss Man";
   }//end method main
}//end class PrisonTest
```

Referencias de objetos

Interacciones

Interacciones desde el método main

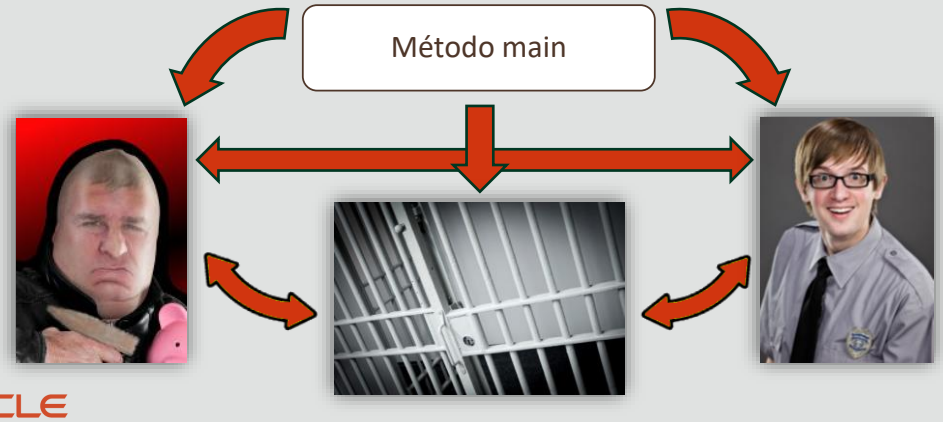
- El método main contiene todas las referencias de objeto
- Por lo tanto, el método main controla todas las interacciones en este sistema



Nota para el instructor: Las imágenes de estas dos diapositivas deben estar en la misma posición.

Interacciones entre objetos

- Sin embargo, en ocasiones deseará un programa donde los objetos interactúen entre sí
- Para ello, los objetos deben conocerse
  - Un objeto debe conocer una referencia al otro objeto



ORACLE Academy

Nota para el instructor: Las imágenes de estas dos diapositivas deben estar en la misma posición.

¿Cómo se conocen los objetos entre sí?

- Las referencias de objeto se deben compartir:
  - Un objeto puede contener otro objeto como un campo
  - El método de un objeto puede aceptar otro objeto como argumento
- Por ejemplo:
  - Una forma de escribir un objeto Prisoner es por su número del objeto Cell
  - Se podría afirmar que un objeto Cell es una propiedad de un objeto Prisoner
  - La clase Prisoner debe contener un campo Cell

Ejercicio 1, parte 1

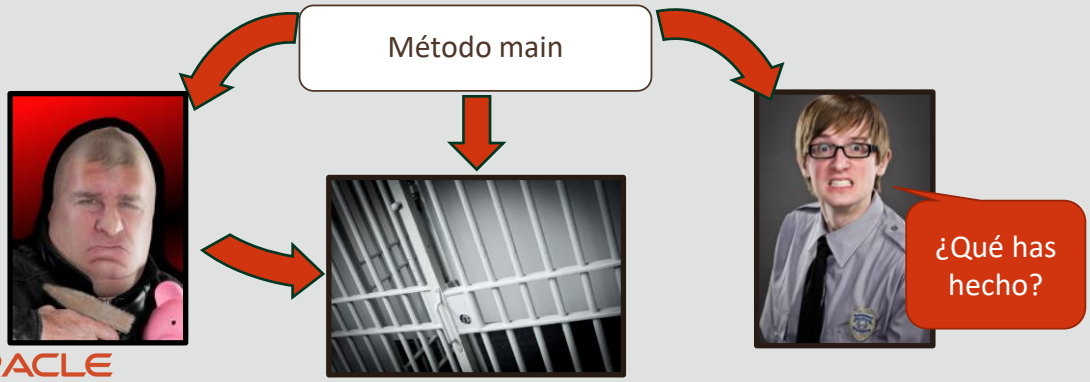
- Continúe con la edición del proyecto `PrisonTest`
  - Se proporciona una versión de este programa en los archivos `PrisonTest_Student_7_5.java` y `Prisoner_Student_7_5.java`
- Cree una clase Cell que incluya lo siguiente:
  - La cadena name con el nombre de la celda
  - Un valor booleano que describa si la puerta está abierta
  - Un constructor de dos argumentos que defina ambos campos
- Modifique la clase Prisoner para que:
  - Incluya un campo Cell
  - Defina el campo Cell según un parámetro de constructor
  - Imprima el valor de name de la celda como parte del método display()

Ejercicio 1, parte 2

- Escriba un método openDoor() para la clase `Prisoner`
  - Acceda y modifique el campo correspondiente en el objeto Cell para que:
    - Si la puerta está cerrada, se abra
    - Si la puerta está abierta, se cierre
  - Imprima si la puerta está abierta o cerrada
- En el método main:
  - Instancie un objeto Cell y Prisoner
  - Llame al método display() del objeto Prisoner una vez
  - Llame al método openDoor() varias veces

¡Glups!

- Los guardias están asustados
- El programa permite a los presos acceder a las puertas de las celdas
- Teniendo en cuenta el plan de venganza de Bubba, no se debería permitir este tipo de interacción



Pensar en las posibles interacciones de los objetos

- Considere qué objetos se deben conocer entre sí
  - Para algunos objetos no resulta útil que se modifiquen los campos de otro
  - Intente minimizar el conocimiento que tienen entre sí...
  - De este modo se evitan los resultados no deseados y se simplifica el código
- Considere en qué dirección pueden producirse las interacciones y qué objetos deben ser propiedades de otros
  - ¿Debe tener un objeto Prisoner una propiedad Cell?
  - ¿Debe tener un objeto Cell una propiedad Prisoner?
  - ¿O no deben conocerse entre sí?

Pensar en cómo distribuir los comportamientos

- Las celdas están diseñadas para abrirse y cerrarse
  - Alguien debe tener acceso para realizar estas interacciones
  - Los prisioneros no deben poder realizar este comportamiento
  - Los guardias sí deben poder realizar este comportamiento
- Decidir la forma de distribuir los comportamientos entre objetos es un reto importante de la programación orientada a objetos
  - Pero no se preocupe. Ya tiene experiencia para hacerlo
  - El objetivo principal de Java Puzzle Ball fue crear situaciones en las que los jugadores tuvieran que pensar detenidamente en la distribución de los comportamientos entre los distintos tipos de objetos

Introducción a la encapsulación

- A veces, los objetos deben conocerse entre sí
- La encapsulación proporciona técnicas para limitar la visibilidad de una clase con otra
- Es posible restringir los campos y métodos que pueden ver otras clases
- Se pueden escribir métodos especiales para decidir cómo se debe acceder a los datos y modificarlos
- El acceso y la visibilidad deben ser lo más limitados posible

Modificadores de acceso

- La palabra clave public es uno de los modificadores de acceso
- Los modificadores de acceso limitan la visibilidad de los campos y los métodos entre las clases

```
public class Cell {
   //Fields
   public String name;
   public boolean isOpen ;

   //Constructor
   public Cell(String name, boolean isOpen){
      this.name = name;
      this.isOpen = isOpen;
   }//end constructor
}//end class Cell
```

Detalles de modificadores de acceso

- public: Visible para cualquier clase
  - Es el menos seguro
  - Normalmente, los métodos son públicos
- Paquete: Visible para el paquete actual
  - No hay ninguna palabra clave para este nivel de acceso
- private: Solo está visible para la clase actual
  - Es el más seguro
  - Normalmente, los campos son privados

Ejercicio 2

- Continúe con la edición del proyecto `PrisonTest`
- Modifique la clase Cell:
  - Cambie sus campos a private
  - Guarde el archivo
- ¿Tiene su IDE alguna queja?
  - ¿Cuáles son las quejas?
  - ¿Dónde se producen?

Efectos de los datos privados

- No se puede acceder a los siguientes campos privados fuera de la clase Cell:
  - isOpen
  - name
- Ni siquiera el método main puede acceder a estos datos
- Es aconsejable que los prisioneros no puedan abrir las puertas de las celdas
- No es recomendable que los prisioneros no conozcan los nombres de sus celdas
  - En el tema siguiente se explica cómo solucionar este problema

Introducción a los métodos getter

- Si un campo es inaccesible, no se puede:
  - Leer
  - Modificar
- Sin embargo, en muchos casos es aconsejable que al menos una clase conozca el valor de los campos de otra clase
  - Al menos un prisionero debe saber el nombre de su celda
  - Para ello, un prisionero debe leer el valor del campo name de un objeto Cell
- Los métodos getter proporcionan una solución

Métodos getter

- Los métodos getter también se denominan accesos
- Los métodos getter son públicos
- Los getters no suelen aceptar argumentos
- Los métodos getter devuelven el valor de una variable concreta
  - La mayoría de las variables privadas necesitan un método getter

```
public class Cell {
   …
   public String getName(){
      return name;
   }//end method getName
   public boolean getIsOpen(){
      return isOpen;
   }//end method getIsOpen
}//end class Cell
```

Introducción a los métodos setter

- En otros casos, es aconsejable que una clase modifique el campo de otra clase
- Sin embargo, esta operación se debe llevar a cabo de forma segura
- Por ejemplo:
  - Un guardia debe poder abrir una puerta, pero no un prisionero
  - Un saldo de cuenta bancaria no debe ser inferior a cero
- Los métodos setter proporcionan una solución

Métodos setter

- También se denominan mutadores
- Los métodos setter suelen ser públicos
- Los métodos setter suelen aceptar argumentos
- Los métodos setter son del tipo void

```
public class Cell {
   …
   public void setName(String name){
      this.name = name;
   }//end method setName
   public void setIsOpen(boolean isOpen){
      this.isOpen = isOpen;
   }//end method setIsOpen
}//end class Cell
```

Diseño de los métodos setter

- Tenga cuidado al escribir los métodos setter como los que se muestran en la diapositiva anterior
  - Los prisioneros volverían a tener acceso a sus puertas
- A veces, se debe reflexionar un poco sobre el diseño de un método setter
  - Una puerta de seguridad puede solicitar un código de seguridad
  - Un software de banca puede comprobar si un importe de retirada dará como resultado un saldo inferior a cero o si el importe de retirada es negativo

Ejercicio 3, parte 1

- Continúe con la edición del proyecto `PrisonTest`
- Modifique la clase Cell de modo que…
  - Haya métodos getter para los campos name e isOpen
  - Exista un campo de código de seguridad de 4 dígitos privado Se inicialice desde el constructor y no tenga ningún método getter
  - Hay un método setter para abrir y cerrar la puerta, y hace lo siguiente:
    - Acepta un código de seguridad como argumento
    - Imprime si el código es incorrecto
    - Si el código es correcto y la puerta está cerrada, se abre
    - Si el código es correcto y la puerta está abierta, la cierra
    - Imprime si la puerta está abierta o cerrada

Ejercicio 3, parte 2

- Modifique la clase `Prisoner` para que...
  - El método display() imprima el nombre de la celda
  - Se elimine el método openDoor()
- Modifique el método main para que...
  - El objeto Cell se instancie correctamente
  - El prisionero ya no intente abrir la puerta de la celda
  - Pruebe la capacidad de una clase cell para abrir y cerrar su puerta
    - Intente proporcionar los códigos de seguridad tanto correctos como incorrectos

Continuación del desarrollo de este software

- Actualmente, el método main prueba la capacidad de la puerta de un objeto Cell de abrirse y cerrarse según el código de seguridad
- La prueba nos permite confirmar que esta función está implantada correctamente
  - Si la función no es correcta, se debe corregir
  - Si la función es correcta, es seguro incluir esta función como parte de otra función
- Un posible paso siguiente será desarrollar una clase Guard con un método para introducir un código de seguridad
  - En última instancia, un guardia, no el método main, sería el responsable de introducir un código de seguridad

Recuerde el modelo espiral de desarrollo.

Rol del método main

- Algunos programas se basan en objetos físicos
- Algunos programas se basan en botones
- En este ejercicio, el método main modela las acciones que activarán el programa
  - Al llamar a `bubba.openDoor()` se modela un prisionero que intenta abrir la puerta de su celda
  - Al llamar a `cellA1.setIsOpen(1234)` se modela una persona que ha introducido un código de seguridad

Ejercicio 4

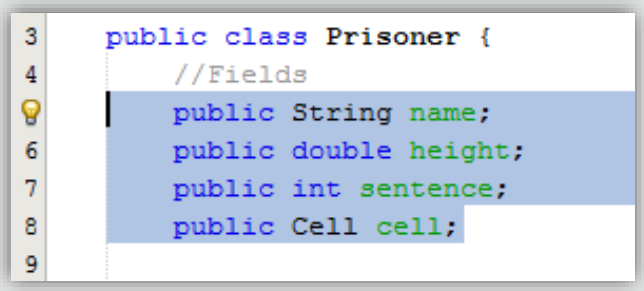
- Continúe con la edición del proyecto `PrisonTest`
- Encapsule la clase `Prisoner`
  - Convierta sus campos en private
  - Proporcione métodos getter y setter para cada campo

El ejercicio no ha sido divertido

- ¿El ejercicio 4 ha sido tedioso y se ha desesperado?
- Algunos programadores prefieren el control del encapsulamiento de algunos campos por sí mismos
- Otros programadores preferirían que su IDE hiciera el trabajo por ellos
  - El IDE puede encapsular campos por usted
  - En las siguientes diapositivas se muestra cómo realizar este proceso en NetBeans
  - Si está utilizando otro IDE, consulte la documentación para obtener información sobre cómo hacerlo

Truco de encapsulación de NetBeans

1. **Resaltar** los campos que desea encapsular

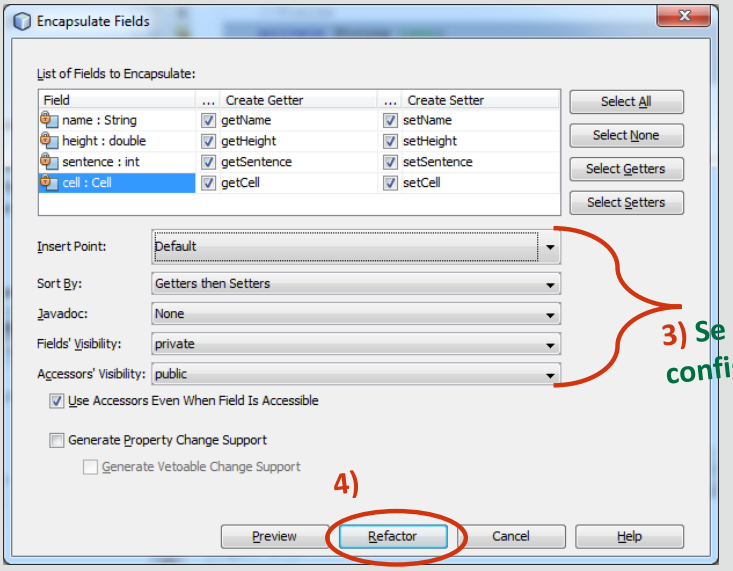


```
3    public class Prisoner {
4        //Fields
         public String name;
6        public double height;
7        public int sentence;
8        public Cell cell;
9
```

2. Hacer clic con el botón derecho y seleccionar **Refactor** >> **Encapsulate Fields**

Truco de encapsulación de NetBeans

3. Ajustar la configuración como desee
4. Hacer clic en **Refactor**



Resumen de la encapsulación

- La encapsulación ofrece técnicas para limitar la visibilidad de una clase
- El acceso y la visibilidad deben ser lo más limitados posible
- La mayoría de los campos deben ser private
- Proporcione métodos getter para devolver el valor de los campos
- Proporcione métodos setter para modificar campos de forma segura

Resumen

- En esta lección, debe haber aprendido lo siguiente:
  - Comprender la interacción de objetos en profundidad
  - Utilizar el modificador private para definir variables de clase
  - Comprender el objetivo de los métodos getter
  - Comprender el objetivo de los métodos setter


ORACLE
Academy