



ORACLE

Academy



Java Foundations

7-3

Constructores

ORACLE
Academy



Objetivos

- En esta lección se abordan los siguientes objetivos:
 - Entender los valores por defecto
 - Bloquear el programa con una referencia null
 - Usar el constructor por defecto
 - Escribir un constructor que acepte argumentos
 - Inicializar los campos con un constructor
 - Utilice this como una referencia de objeto



Recordar la clase Prisoner

- Podría parecerse al código siguiente
- Contiene campos y métodos

```
public class Prisoner {  
    //Fields  
    public String name;  
    public double height;  
    public int sentence;  
  
    //Methods  
    public void think(){  
        System.out.println("I'll have my revenge.");  
    } //end method think  
} //end class Prisoner
```

Los campos son variables

- Las variables contienen valores
- Se puede acceder a los valores
- El código puede necesitar acceder a las variables para...
 - Realizar cálculos
 - Comprobar los valores actuales
 - Cambiar un valor
- ¿Qué sucedería si se accede a un campo antes de que se le asigne un valor?

Ejercicio 1

- Continúe con la edición del proyecto `PrisonTest`
 - Se proporciona una versión de este programa en los archivos `PrisonTest_Student_7_3.java` y `Prisoner_Student_7_3.java`
- Investigue qué sucede cuando se accede a los campos antes de que se les asignen valores
 - Instancie un objeto `Prisoner`
 - Pruebe a imprimir el valor de cada campo



Variable: p01
Name: ???
Height: ???
Sentence: ???

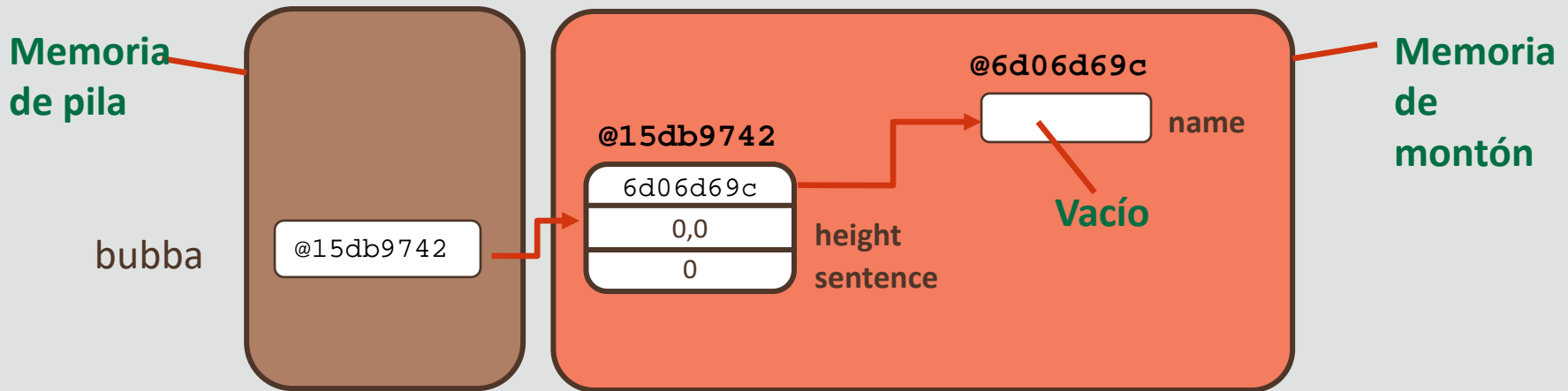
Acceso a campos sin inicializar

- Si los campos no se han inicializado, adquieren un valor por defecto
- Java ofrece los siguientes valores por defecto:

Tipo de Dato	Valor por Defecto
<code>boolean</code>	<code>false</code>
<code>int</code>	<code>0</code>
<code>double</code>	<code>0,0</code>
<code>String</code>	<code>null</code>
Cualquier tipo de objeto	<code>null</code>

Referencias a objetos nulos

- Los objetos pueden tener un valor null
- Un objeto nulo apunta a una ubicación vacía en la memoria
- Si un objeto tiene otro objeto como un campo (por ejemplo, String), su valor por defecto es null



El acceso a los objetos nulos es peligroso

- ¿Qué ocurre si un objeto nulo contiene un campo o método al que es necesario acceder?
 - Esto provoca que el programa se bloquee
 - El error concreto es NullPointerException

```
public static void main(String[] args){  
    String test = null;  
    System.out.println(test.length());  
} //end method main
```

La importancia de inicializar los campos

- Siempre es recomendable reducir las posibilidades de que se bloquee el programa
- Y, en ocasiones, los valores por defecto de Java no son aconsejables
- En los temas restantes de esta lección examinaremos alternativas útiles para inicializar los campos

Definición de los campos de Prisoner

- Actualmente, necesitamos una línea del código para definir cada campo
- Se necesitan cuatro líneas para cada objeto Prisoner

```
public class PrisonTest {  
    public static void main(String[] args){  
        Prisoner p01 = new Prisoner();  
        Prisoner p02 = new Prisoner();  
  
        p01.name = "Bubba";  
        p01.height = 2.08;  
        p01.sentence = 4;  
        p02.name = "Twitch";  
        p02.height = 1.73;  
        p02.sentence = 3;  
    } //end method main  
} //end class PrisonTest
```

Los métodos hacen que el código sea más eficaz

- Si descubre que está repitiendo líneas de código similares...
 - La programación puede volverse tediosa
 - Es posible realizar el mismo trabajo con menos líneas
 - Intente escribir dicho código como parte de un método

```
p01.name = "Bubba";  
p01.height = 2.08;  
p01.sentence = 4;
```

} Primera incidencia

```
p02.name = "Twitch";  
p02.height = 1.73;  
p02.sentence = 3;
```

} Repetición

Ejercicio 2

- Continúe con la edición del proyecto `PrisonTest`
- ¿Se pueden definir los campos de forma más eficaz?
 - Agregue un método `setFields()` a la clase `Prisoner`
 - Este método debe aceptar tres argumentos, que se utilizan para definir los valores de cada campo
 - Sustituir el código del método `main` con llamadas a este método



Variable: p01
Name: Bubba
Height: 6'10"
(2,08m)
Sentence: 4 years



Variable: p02
Name: Twitch
Height: 5'8"
(1,73m)
Sentence: 3 years

Escritura de un método para definir campos

- Su solución puede parecerse a esta:

```
public class Prisoner {  
    public String name;  
    public double height;  
    public int sentence;  
  
    public void setFields(String n, double h, int s){  
        name = n;  
        height = h;  
        sentence = s;  
    } //end method setFields  
} //end class Prisoner
```

Definición de los campos de Prisoner

- Se necesitan dos líneas para cada objeto Prisoner
- Pero es posible realizar el mismo trabajo incluso con menos líneas

```
public class PrisonTest {  
    public static void main(String[] args){  
        Prisoner p01 = new Prisoner();  
        Prisoner p02 = new Prisoner();  
  
        p01.setFields("Bubba", 2.08, 4);  
        p02.setFields("Twitch", 1.73, 3);  
  
    } //end method main  
} //end class PrisonTest
```


Llamada a constructores

- Un constructor es un método especial
- Su objetivo es "construir" un objeto mediante la definición de los valores de campo iniciales
- El constructor de un objeto se llama una vez
 - Esto se produce durante la instanciación
 - Y no se vuelve a llamar
- Hemos estado llamando a constructores todo este tiempo

Llamada de método constructor

```
Prisoner p01 = new Prisoner();
```

El constructor por defecto

- Java proporciona automáticamente un constructor para cada clase
- Nunca se escribe explícitamente en una clase
- A este constructor se le denomina constructor por defecto
- Se considera un constructor sin argumentos

Acepta cero argumentos



```
Prisoner p01 = new Prisoner();
```

Escritura de un método constructor

- Puede sustituir el constructor por defecto por un constructor propio
- Los constructores se escriben como cualquier otro método, con las siguientes excepciones:
 - No tienen tipo de devolución (ni siquiera `void`)
 - Tienen el **mismo** nombre que la clase

```
//Constructor  
public Prisoner(){  
    System.out.println("This is a constructor");  
}//end constructor
```

Ejercicio 3, parte 1

- Continúe con la edición del proyecto `PrisonTest`
- Copie el constructor en la clase `Prisoner`
 - Ejecute el programa
 - Observe cómo se ejecuta el código de este método cuando se instancian los objetos `Prisoner`

```
//Constructor  
public Prisoner(){  
    System.out.println("This is a constructor");  
}//end constructor
```

Ejercicio 3, parte 2

- ¿Cómo puede modificar este constructor para que defina todos los campos de la clase?
 - Utilice su conocimiento de los métodos para buscar una solución
 - Recuerde que los constructores son métodos
 - Elimine el método setFields()
 - Su solución convertiría este método en redundante
- Su IDE se quejará en el método main:
 - ¿Cómo se pueden solucionar estos problemas?
 - Ejecute el programa después de encontrar una solución

Puede que haya advertido lo siguiente...

- Los constructores se pueden escribir para que acepten argumentos que definan los valores de campo iniciales
- Si escribe su propio constructor, el constructor por defecto ya no estará disponible
- El código se convierte en más útil y necesita menos líneas
 - En las diapositivas siguientes se ilustra esta mayor eficacia

```
//Constructor
public Prisoner(String n, double h, int s){
    name = n;
    height = h;
    sentence = s;
} //end constructor
```

Definición de campos sin un constructor

- Se necesitan cuatro líneas para cada objeto Prisoner

```
public class PrisonTest {  
  
    public static void main(String[] args){  
        Prisoner p01 = new Prisoner();  
        Prisoner p02 = new Prisoner();  
  
        p01.name = "Bubba";  
        p01.height = 2.08;  
        p01.sentence = 4;  
  
        p02.name = "Twitch";  
        p02.height = 1.73;  
        p02.sentence = 3;  
    } //end method main  
} //end class PrisonTest
```


Definición de campos con un método

- Se necesitan dos líneas para cada objeto Prisoner

```
public class PrisonTest {  
  
    public static void main(String[] args){  
        Prisoner p01 = new Prisoner();  
        Prisoner p02 = new Prisoner();  
  
        p01.setFields("Bubba", 2.08, 4);  
        p02.setFields("Twitch", 1.73, 3);  
  
    }//end method main  
}//end class PrisonTest
```



Definición de campos con un constructor

- Se necesita una línea para cada objeto Prisoner

```
public class PrisonTest {  
  
    public static void main(String[] args){  
        Prisoner p01 = new Prisoner("Bubba", 2.08, 4);  
        Prisoner p02 = new Prisoner("Twitch", 1.73, 3);  
  
    }//end method main  
}//end class PrisonTest
```

Parámetros de nomenclatura

- Se suelen utilizar nombres de variable de un solo carácter...
 - Si la variable tiene un ámbito muy limitado
 - Si no hay muchas variables de las que se realiza el seguimiento
 - Para fines de prueba
- Pero anteriormente en este curso hemos recomendado asignar nombres descriptivos a las variables
 - Esto ayuda a evitar confusiones
 - Debe seguir esta convención para los campos
 - A algunos desarrolladores les gusta aplicar esta regla a los parámetros de método

Asignación de los mismos nombres a parámetros y campos

- También es una práctica habitual, especialmente con los constructores
 - Resulta más claro a lo que hacen referencia los parámetros
 - Pero se crean complicaciones de ámbito
- En el siguiente código, ¿se imprime el campo o el parámetro name?

```
public class Prisoner {  
    public String name;  
  
    public setName(String name){  
        System.out.println(name);  
    } //end method setName  
} //end class Prisoner
```



¿Qué versión de name se imprime?

- Se imprime el parámetro
 - Tienen prioridad las variables en el ámbito más local
 - Es decir, las variables dentro del ámbito más reciente
- ¿Se puede acceder al campo?
 - Sí. El campo está en el ámbito de sus métodos de clase
 - Pero se necesita más sintaxis para acceder a ellos

```
public class Prisoner {  
    public String name;  
  
    public setName(String name){  
        System.out.println(name);  
    } //end method setName  
} //end class Prisoner
```

Palabra clave this

- this es una referencia al objeto actual
 - Puede tratarlo como cualquier objeto de referencia
 - Lo que significa que puede utilizar el operador de punto (.)
- this.name accede al campo del objeto Prisoner
- this.setName() accede al método del objeto Prisoner

```
public class Prisoner {  
    public String name;  
  
    public setName(String name){  
        System.out.println(name);  
    } //end method setName  
} //end class Prisoner
```

Ejercicio 4

- Modifique el constructor de `Prisoner`
 - Cambie los parámetros de este método para que el nombre de cada parámetro coincida con el nombre de un campo
 - Defina el valor de cada campo mediante la palabra clave `this`

Resumen de los constructores

- Son métodos especiales dentro de una clase
- Tienen el mismo nombre que la clase
- No tienen tipo de devolución (ni siquiera void)
- Solo se llaman una vez durante la instanciación de objeto
- Pueden aceptar argumentos
- Se usan para definir los valores iniciales de los campos
- Si no escribe su propio constructor, Java proporciona un constructor sin argumentos por defecto

Resumen

- En esta lección, debe haber aprendido lo siguiente:
 - Entender los valores por defecto
 - Bloquear el programa con una referencia null
 - Usar el constructor por defecto
 - Escribir un constructor que acepte argumentos
 - Inicializar los campos con un constructor
 - Utilizar this como una referencia de objeto





ORACLE

Academy

