Objetos

Programación I - UNGS

¿Qué es un objeto?

- **Objeto:** Es una entidad con estado y comportamiento propios.

¿Qué es un objeto?

- **Objeto:** Es una entidad con estado y comportamiento propios.
- El estado está asociado a los datos, el comportamiento está asociado a los métodos.

¿Qué es un objeto?

- **Objeto:** Es una entidad con estado y comportamiento propios.
- El estado está asociado a los datos, el comportamiento está asociado a los métodos.
- Los Strings son objetos:

¿Qué es un objeto?

- **Objeto:** Es una entidad con estado y comportamiento propios.
- El estado está asociado a los datos, el comportamiento está asociado a los métodos.
- Los Strings son objetos:
  1. Datos: Una secuencia de caracteres

¿Qué es un objeto?

- **Objeto:** Es una entidad con estado y comportamiento propios.
- El estado está asociado a los datos, el comportamiento está asociado a los métodos.
- Los Strings son objetos:
  1. Datos: Una secuencia de caracteres
  2. Métodos: charAt(), length(), indexOf(), toUpperCase(), etc.

¿Qué es un objeto?

- **Objeto:** Es una entidad con estado y comportamiento propios.
- El estado está asociado a los datos, el comportamiento está asociado a los métodos.
- Los Strings son objetos:
  1. Datos: Una secuencia de caracteres
  2. Métodos: charAt(), length(), indexOf(), toUpperCase(), etc.
- ¿Qué otro tipo de objetos se puede tener en Java?

Point

- Otro tipo de objetos que podemos encontrar en Java son los objetos Point

Point

- Otro tipo de objetos que podemos encontrar en Java son los objetos Point
- Representan un punto en la pantalla y se describen con un par de coordenadas enteras (x, y).

Point

- Otro tipo de objetos que podemos encontrar en Java son los objetos Point
- Representan un punto en la pantalla y se describen con un par de coordenadas enteras (x, y).
- Pertenecen al paquete $\mathrm{java.awt}$, con lo cual debemos importarlo con la siguiente línea al principio del archivo:

```
1 import java.awt.*;
```

Point

- Veamos cómo se crea uno:

```
1 Point p;
2 p = new Point(3, 4);
```

Point

- Veamos cómo se crea uno:

```
1 Point p;
2 p = new Point(3, 4);
```

- Esto se puede leer de la siguiente manera:

Point

- Veamos cómo se crea uno:

```
1 Point p;
2 p = new Point(3, 4);
```

- Esto se puede leer de la siguiente manera:
  - Crea una variable p del tipo Point

Point

- Veamos cómo se crea uno:

```
1 Point p;
2 p = new Point(3, 4);
```

- Esto se puede leer de la siguiente manera:
  - Crea una variable p del tipo Point
  - Crea un **nuevo objeto** del tipo Point (que tendrá los valores 3 y 4 en sus coordenadas)

Point

- Veamos cómo se crea uno:

```
1 Point p;
2 p = new Point(3, 4);
```

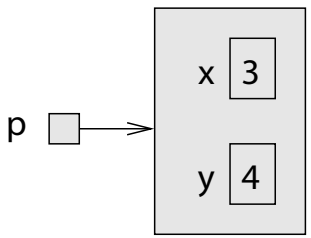
- Esto se puede leer de la siguiente manera:
  - Crea una variable p del tipo Point
  - Crea un **nuevo objeto** del tipo Point (que tendrá los valores 3 y 4 en sus coordenadas)
  - Asigna en p una **referencia** al nuevo objeto recién creado.

Point

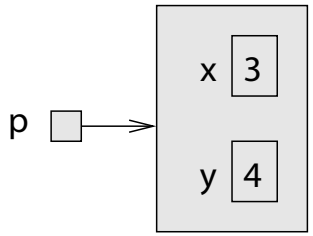
- Veamos cómo se crea uno:

```
1 Point p;
2 p = new Point(3, 4);
```

- Esto se puede leer de la siguiente manera:
  - Crea una variable p del tipo Point
  - Crea un **nuevo objeto** del tipo Point (que tendrá los valores 3 y 4 en sus coordenadas)
  - Asigna en p una **referencia** al nuevo objeto recién creado.
- Gráficamente:

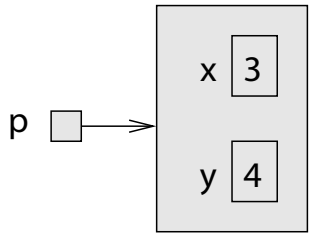


Instancias



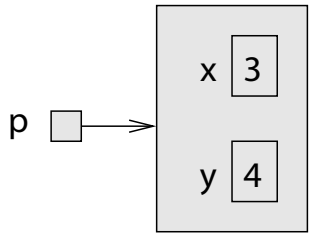
- El nuevo objeto creado es una instancia de la clase Point

Instancias



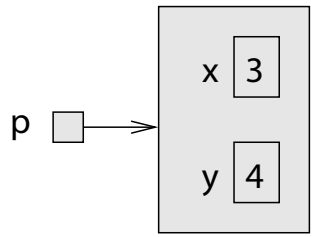
- El nuevo objeto creado es una instancia de la clase Point
- Dentro del nuevo objeto, los datos se almacenan en variables de instancia

Instancias



- El nuevo objeto creado es una instancia de la clase Point
- Dentro del nuevo objeto, los datos se almacenan en variables de instancia
- Cada vez que creamos una instancia de una clase, se crean las variables asociadas a dicha instancia

Instancias



- El nuevo objeto creado es una instancia de la clase Point
- Dentro del nuevo objeto, los datos se almacenan en variables de instancia
- Cada vez que creamos una instancia de una clase, se crean las variables asociadas a dicha instancia
- Para acceder a ellas tenemos que utilizar la sintaxis <nombre del objeto>.<nombre de la variable>:

```
1        System.out.println("El valor de x es " + p.x);
```

Variables de instancia

- Los objetos son como cajones que contienen compartimentos. Los compartimentos son las variables de instancia.

Variables de instancia

- Los objetos son como cajones que contienen compartimentos. Los compartimentos son las variables de instancia.
- Las variables de instancia funcionan como cualquier otra variable en Java. Podemos asignarlas y leerlas:

```
1        p.x = p.x + 10; // incrementa en 10 el valor de x
```

Variables de instancia

- Los objetos son como cajones que contienen compartimentos. Los compartimentos son las variables de instancia.
- Las variables de instancia funcionan como cualquier otra variable en Java. Podemos asignarlas y leerlas:

```
1        p.x = p.x + 10; // incrementa en 10 el valor de x
```

- Como cualquier otra variable, puede formar parte de cualquier expresión:

```
1        System.out.println("El punto está en (" + p.x + "," + p.y + ")");
2        int distancia = Math.sqrt(p.x * p.x + p.y * p.y);
```

Memoria dinámica

- Los objetos se alojan en una parte de la RAM reservada al proceso denominada memoria dinámica.

Memoria dinámica

- Los objetos se alojan en una parte de la RAM reservada al proceso denominada memoria dinámica.
- Las variables se alojan en memoria estática.

Memoria dinámica

- Los objetos se alojan en una parte de la RAM reservada al proceso denominada memoria dinámica.
- Las variables se alojan en memoria estática.
  1. Para los tipos básicos, las variables contienen el **valor del tipo**.

Memoria dinámica

- Los objetos se alojan en una parte de la RAM reservada al proceso denominada memoria dinámica.
- Las variables se alojan en memoria estática.
  1. Para los tipos básicos, las variables contienen el **valor del tipo**.
  2. Para los objetos, las variables contienen **referencias a memoria dinámica**.

Memoria dinámica

- Los objetos se alojan en una parte de la RAM reservada al proceso denominada memoria dinámica.
- Las variables se alojan en memoria estática.
  1. Para los tipos básicos, las variables contienen el **valor del tipo**.
  2. Para los objetos, las variables contienen **referencias a memoria dinámica**.
- El ciclo de vida de las variables es distinto de acuerdo al tipo de memoria.

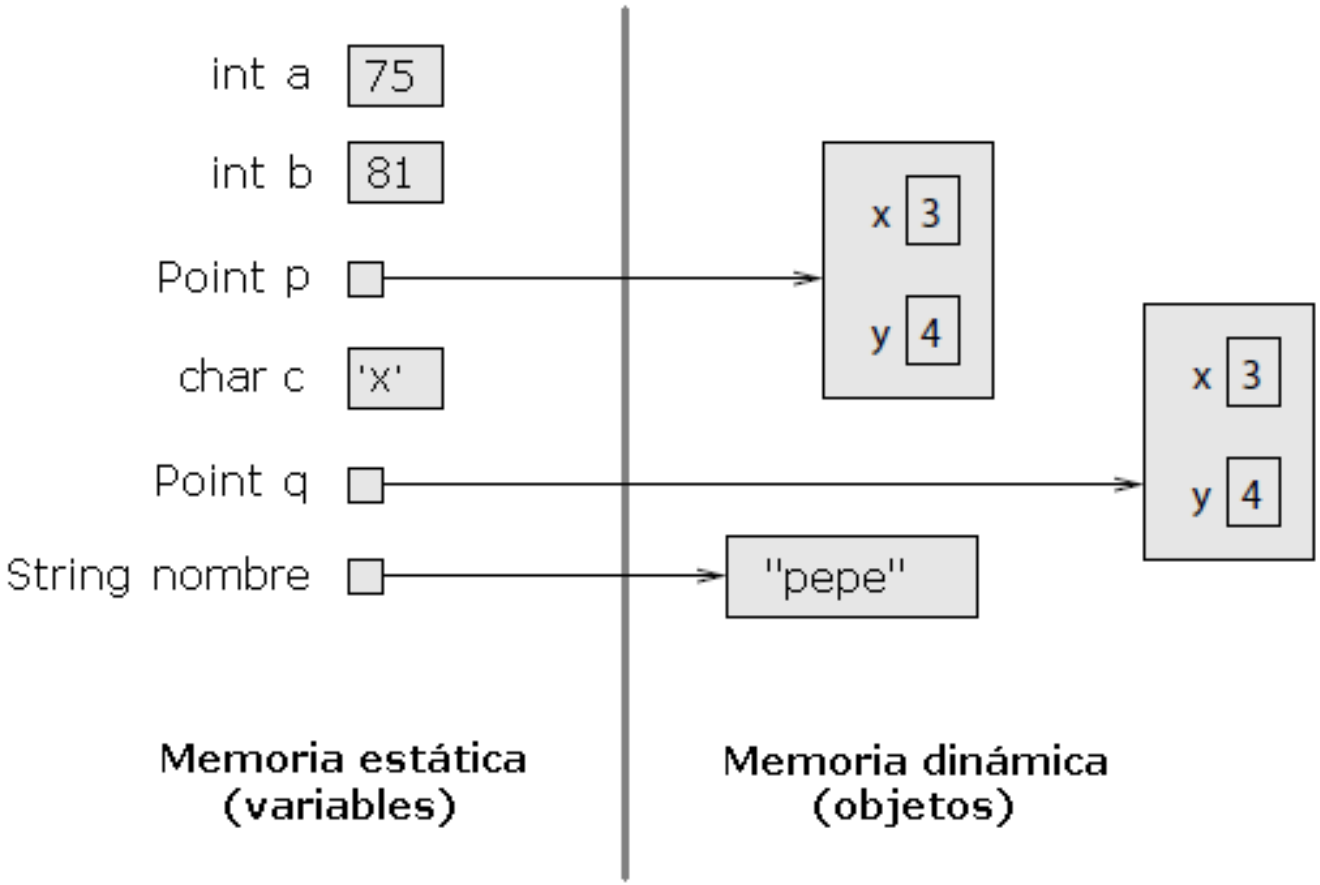
Memoria dinámica

- Los objetos se alojan en una parte de la RAM reservada al proceso denominada memoria dinámica.
- Las variables se alojan en memoria estática.
  1. Para los tipos básicos, las variables contienen el **valor del tipo**.
  2. Para los objetos, las variables contienen **referencias a memoria dinámica**.
- El ciclo de vida de las variables es distinto de acuerdo al tipo de memoria.
  1. Las variables (en memoria estática) desaparecen de la memoria cuando se cierra el bloque en el cual fueron declaradas.

Memoria dinámica

- Los objetos se alojan en una parte de la RAM reservada al proceso denominada memoria dinámica.
- Las variables se alojan en memoria estática.
  1. Para los tipos básicos, las variables contienen el **valor del tipo**.
  2. Para los objetos, las variables contienen **referencias a memoria dinámica**.
- El ciclo de vida de las variables es distinto de acuerdo al tipo de memoria.
  1. Las variables (en memoria estática) desaparecen de la memoria cuando se cierra el bloque en el cual fueron declaradas.
  2. Los objetos se mantienen en memoria dinámica mientras haya alguna variable que los referencie.

Memoria dinámica



Rectangle

- Veamos un objeto de otro tipo:

```
1    Rectangle caja = new Rectangle(0, 0, 100, 200);
```

Rectangle

- Veamos un objeto de otro tipo:

```
1    Rectangle caja = new Rectangle(0, 0, 100, 200);
```

- Al igual que lo que ocurre con Point:

Rectangle

- Veamos un objeto de otro tipo:

```
1    Rectangle caja = new Rectangle(0, 0, 100, 200);
```

- Al igual que lo que ocurre con Point:
  1. Creamos una variable del tipo Rectangle

Rectangle

- Veamos un objeto de otro tipo:

```
1    Rectangle caja = new Rectangle(0, 0, 100, 200);
```

- Al igual que lo que ocurre con Point:
  1. Creamos una variable del tipo Rectangle
  2. Creamos una **instancia** de Rectangle usando la sentencia **new**

Rectangle

- Veamos un objeto de otro tipo:

```
1    Rectangle caja = new Rectangle(0, 0, 100, 200);
```

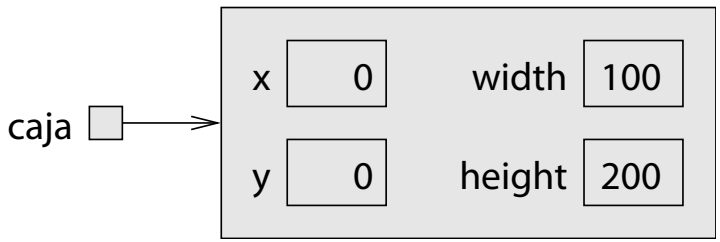
- Al igual que lo que ocurre con Point:
  1. Creamos una variable del tipo Rectangle
  2. Creamos una **instancia** de Rectangle usando la sentencia **new**
  3. Se guarda una referencia al objeto creado en la variable caja

Rectangle

- Veamos un objeto de otro tipo:

```
1   Rectangle caja = new Rectangle(0, 0, 100, 200);
```

- Al igual que lo que ocurre con Point:
  1. Creamos una variable del tipo Rectangle
  2. Creamos una **instancia** de Rectangle usando la sentencia **new**
  3. Se guarda una referencia al objeto creado en la variable caja
- Si lo diagramamos tenemos esto:

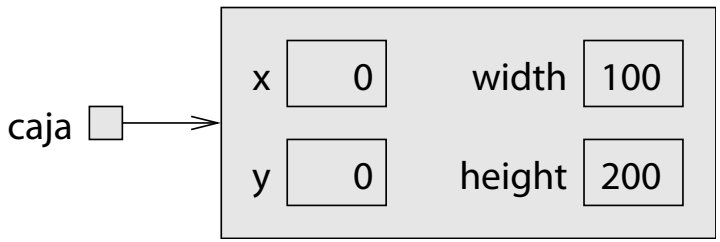


Rectangle

- Veamos un objeto de otro tipo:

```
1    Rectangle caja = new Rectangle(0, 0, 100, 200);
```

- Al igual que lo que ocurre con Point:
  1. Creamos una variable del tipo Rectangle
  2. Creamos una **instancia** de Rectangle usando la sentencia **new**
  3. Se guarda una referencia al objeto creado en la variable caja
- Si lo diagramamos tenemos esto:



- ¿Cómo sabe qué parámetro corresponde a cada variable de instancia?

Constructores

- Cuando llamamos al comando new, el sistema llama inmediatamente a un método especial llamado **constructor**.

Constructores

- Cuando llamamos al comando new, el sistema llama inmediatamente a un método especial llamado **constructor**.
- El constructor no es más que un método especial que se ejecuta cuando se crea una instancia de un objeto

Constructores

- Cuando llamamos al comando new, el sistema llama inmediatamente a un método especial llamado **constructor**.
- El constructor no es más que un método especial que se ejecuta cuando se crea una instancia de un objeto
- La particularidad es que puede recibir parámetros que son los que le pasamos a la sentencia new luego del tipo del objeto.

Constructores

- Cuando llamamos al comando new, el sistema llama inmediatamente a un método especial llamado **constructor**.
- El constructor no es más que un método especial que se ejecuta cuando se crea una instancia de un objeto
- La particularidad es que puede recibir parámetros que son los que le pasamos a la sentencia new luego del tipo del objeto.
- Vamos a ver cómo se escriben la clase que viene cuando veamos cómo crear nuevos tipos.

Manipulando objetos

- A diferencia de los Strings, al común de los objetos podemos modificarlos:

```
1        caja.x = caja.x + 50;
2        caja.y = caja.y + 100;
```

Manipulando objetos

- Podemos programar un método que generalice este comportamiento:

```
1   static void mover(Rectangle caja, int dx, int dy) {
2       caja.x = caja.x + dx;
3       caja.y = caja.y + dy;
4   }
```

Manipulando objetos

- Podemos programar un método que generalice este comportamiento:

```
1   static void mover(Rectangle caja, int dx, int dy) {
2     caja.x = caja.x + dx;
3     caja.y = caja.y + dy;
4   }
```

- Y para llamarlo:

```
1         Rectangle caja = new Rectangle(0, 0, 100, 200);
2         mover(caja, 50, 100);
3         System.out.println(caja);
```

que imprime un objeto caja, con coordenadas $x = 50, y = 100$

Manipulando objetos

- Cuando pasamos un objeto por parámetro, estamos pasando una **referencia** al objeto.

Manipulando objetos

- Cuando pasamos un objeto por parámetro, estamos pasando una **referencia** al objeto.
- El llamado a la función anterior es equivalente a usar el método caja.translate(50,100)

Manipulando objetos

- Cuando pasamos un objeto por parámetro, estamos pasando una **referencia** al objeto.
- El llamado a la función anterior es equivalente a usar el método caja.translate(50,100)
- translate es lo que se conoce como un **método de instancia** en oposición a los **métodos de clase** que conocemos hasta ahora.

Manipulando objetos

- Cuando pasamos un objeto por parámetro, estamos pasando una **referencia** al objeto.
- El llamado a la función anterior es equivalente a usar el método caja.translate(50,100)
- translate es lo que se conoce como un **método de instancia** en oposición a los **métodos de clase** que conocemos hasta ahora.
- Cuando llamamos a un método de instancia, el objeto sobre el cual se llama el método, es un parámetro *implícito* de la función.

Manipulando objetos

- Cuando pasamos un objeto por parámetro, estamos pasando una **referencia** al objeto.
- El llamado a la función anterior es equivalente a usar el método caja.translate(50,100)
- translate es lo que se conoce como un **método de instancia** en oposición a los **métodos de clase** que conocemos hasta ahora.
- Cuando llamamos a un método de instancia, el objeto sobre el cual se llama el método, es un parámetro *implícito* de la función.
- Vamos a profundizar más este tema en la clase que viene.

Aliasing

- Consideremos este código:

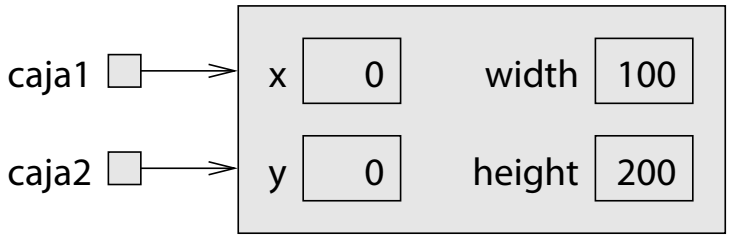
```
1    Rectangle caja1 = new Rectangle(0, 0, 100, 200);
2    Rectangle caja2 = caja1;
```

Aliasing

- Consideremos este código:

```
1    Rectangle caja1 = new Rectangle(0, 0, 100, 200);
2    Rectangle caja2 = caja1;
```

- Esto genera que ambas variables **referencien al mismo objeto**



Aliasing

- Ahora, supongamos que invocamos al método grow que ensancha y alarga el rectángulo en cada sentido, con las dimensiones especificadas.

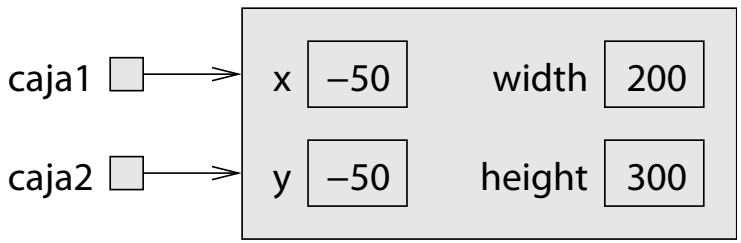
```
1        System.out.println(caja2.width);
2        caja1.grow(50,50);
3        System.out.println(caja2.width);
```

Aliasing

- Ahora, supongamos que invocamos al método grow que ensancha y alarga el rectángulo en cada sentido, con las dimensiones especificadas.

```
1        System.out.println(caja2.width);
2        caja1.grow(50,50);
3        System.out.println(caja2.width);
```

- Si bien llamamos a un método que modificaba a $\mathrm{caja1}$, se modificó también $\mathrm{caja2}$.

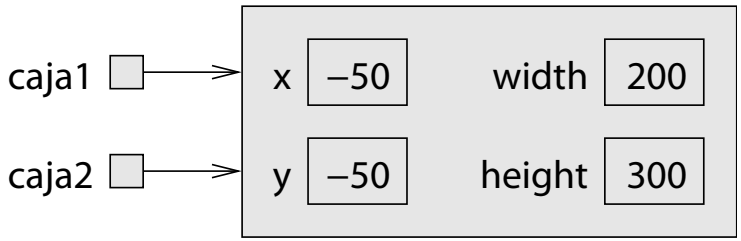


Aliasing

- Ahora, supongamos que invocamos al método grow que ensancha y alarga el rectángulo en cada sentido, con las dimensiones especificadas.

```
1        System.out.println(caja2.width);
2        caja1.grow(50,50);
3        System.out.println(caja2.width);
```

- Si bien llamamos a un método que modificaba a $\mathrm{caja1}$, se modificó también $\mathrm{caja2}$.



- Esto se debe a que las dos variables referencian al mismo objeto. A esto se lo conoce como **aliasing**.

Comparando objetos

- Consideremos este código:

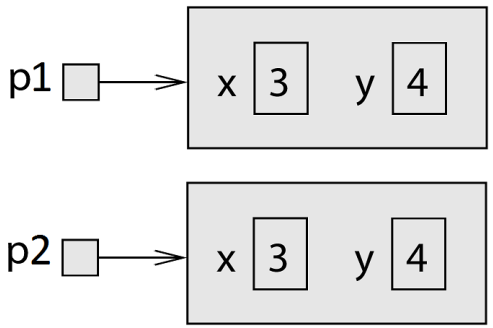
```
1        Point p1 = new Point(3, 4);
2        Point p2 = new Point(3, 4);
```

Comparando objetos

- Consideremos este código:

```
1    Point p1 = new Point(3, 4);
2    Point p2 = new Point(3, 4);
```

- Si lo diagramamos tenemos algo así:

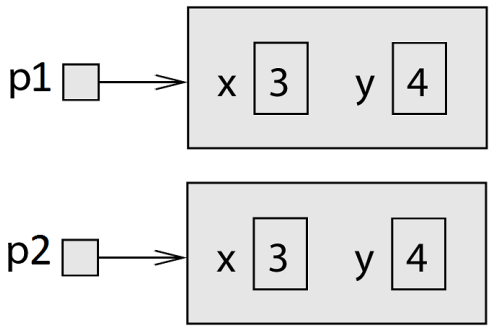


Comparando objetos

- Consideremos este código:

```
1    Point p1 = new Point(3, 4);
2    Point p2 = new Point(3, 4);
```

- Si lo diagramamos tenemos algo así:



- ¿Cuál sería el resultado de la comparación $\mathrm{p1} == \mathrm{p2}$?

Comparando objetos

- Consideremos este otro código:

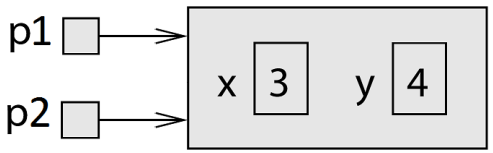
```
1        Point p1 = new Point(3, 4);
2        Point p2 = p1;
```

Comparando objetos

- Consideremos este otro código:

```
1    Point p1 = new Point(3, 4);
2    Point p2 = p1;
```

- Es decir, tenemos el siguiente diagrama:

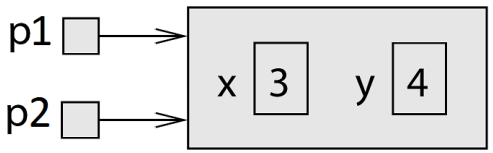


Comparando objetos

- Consideremos este otro código:

```
1    Point p1 = new Point(3, 4);
2    Point p2 = p1;
```

- Es decir, tenemos el siguiente diagrama:



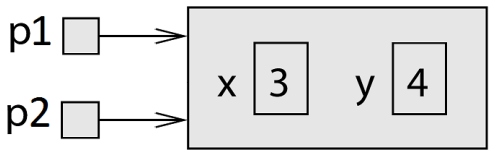
- ¿Cuál sería el resultado de $\mathrm{p1} == \mathrm{p2}$ ahora?

Comparando objetos

- Consideremos este otro código:

```
1        Point p1 = new Point(3, 4);
2        Point p2 = p1;
```

- Es decir, tenemos el siguiente diagrama:



- ¿Cuál sería el resultado de $\mathrm{p1} == \mathrm{p2}$ ahora?
- Al igual que con los $\mathrm{Strings}$, la forma de comparar objetos es a través del método **equals()**.

El método **equals()**

- Cada clase de Java cuenta con una implementación de este método que compara ("inteligentemente") dos objetos de esa clase.

El método **equals()**

- Cada clase de Java cuenta con una implementación de este método que compara ("inteligentemente") dos objetos de esa clase.
- La clase que viene vamos a ver qué significa "inteligentemente", lo importante ahora es que cada clase **debe definir** su propia *igualdad entre objetos*.

El método **equals()**

- Cada clase de Java cuenta con una implementación de este método que compara ("inteligentemente") dos objetos de esa clase.
- La clase que viene vamos a ver qué significa "inteligentemente", lo importante ahora es que cada clase **debe definir** su propia *igualdad entre objetos*.
- Cuando hagamos nuestras propias clases, tendremos que definir nuestro propio método para que se puedan comparar dos objetos de nuestra clase.

El método **equals()**

- Cada clase de Java cuenta con una implementación de este método que compara ("inteligentemente") dos objetos de esa clase.
- La clase que viene vamos a ver qué significa "inteligentemente", lo importante ahora es que cada clase **debe definir** su propia *igualdad entre objetos*.
- Cuando hagamos nuestras propias clases, tendremos que definir nuestro propio método para que se puedan comparar dos objetos de nuestra clase.
- La forma de comparar correctamente los dos objetos del ejemplo anterior es escribiendo $\mathrm{p1.equals(p2)}$ (o bien $\mathrm{p2.equals(p1)}$).

El valor null

- Cuando declaramos una variable del tipo de algún objeto y no le damos ningún valor inicial, éste, por defecto, es **null**. Las siguientes declaraciones son equivalentes:

```
1        Point p1;
2        Point p2 = null;
```

El valor null

- Cuando declaramos una variable del tipo de algún objeto y no le damos ningún valor inicial, éste, por defecto, es **null**. Las siguientes declaraciones son equivalentes:

```
1    Point p1;
2    Point p2 = null;
```

- Si tratamos de acceder a un miembro o llamar a un método de un objeto null, vamos a tener un error: NullPointerException

Garbage Collector

- El valor null nos sirve también para borrar referencias, por ejemplo

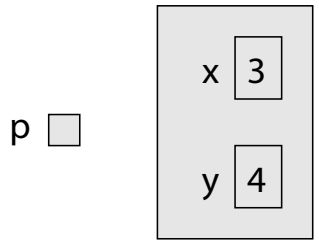
```
1    Point p = new Point(3, 4);
2    p = null;
```

Garbage Collector

- El valor null nos sirve también para borrar referencias, por ejemplo

```
1    Point p = new Point(3, 4);
2    p = null;
```

- En este punto hay un objeto en memoria que no lo referencia nadie, porque p ahora no apunta a nada.



Garbage Collector

- Los objetos que dejan de ser referenciados por alguna variable, son liberados por el **Garbage Collector**

Garbage Collector

- Los objetos que dejan de ser referenciados por alguna variable, son liberados por el **Garbage Collector**
- Entonces, el **new** reserva memoria para un nuevo objeto. Y esa memoria se sigue usando, hasta que el Garbage Collector la recupere.

Garbage Collector

- Los objetos que dejan de ser referenciados por alguna variable, son liberados por el **Garbage Collector**
- Entonces, el **new** reserva memoria para un nuevo objeto. Y esa memoria se sigue usando, hasta que el Garbage Collector la recupere.
- Esto no ocurre así para las variables de los tipos nativos (int, double, boolean). Cuando el contexto en el cual se creó la variable (función, ciclo, etc) se termina, la variable se libera.

Funciones que devuelven objetos

- Es importante destacar que un objeto que creamos dentro de una función, lo podemos devolver:

```
1   static Point calcularCentro(Rectangle caja) {
2       int x = caja.x + caja.width/2;
3       int y = caja.y + caja.height/2;
4       Point p = new Point(x, y);
5       return p;
6   }
```

En el libro...

Lo que vimos en esta clase lo pueden encontrar en el **Capítulo 8** del libro.