LA REFERENCIA THIS

El concepto de *this,* puede resultar en estos momentos un concepto "duro", pero debes esforzarte porque es importante. Para entender bien este boletín, primero debes de entender bien los boletines anteriores.

```
class MiClase{
   int x;
   void miMetodo(int i){
      x=2 +i;
   }
}
```

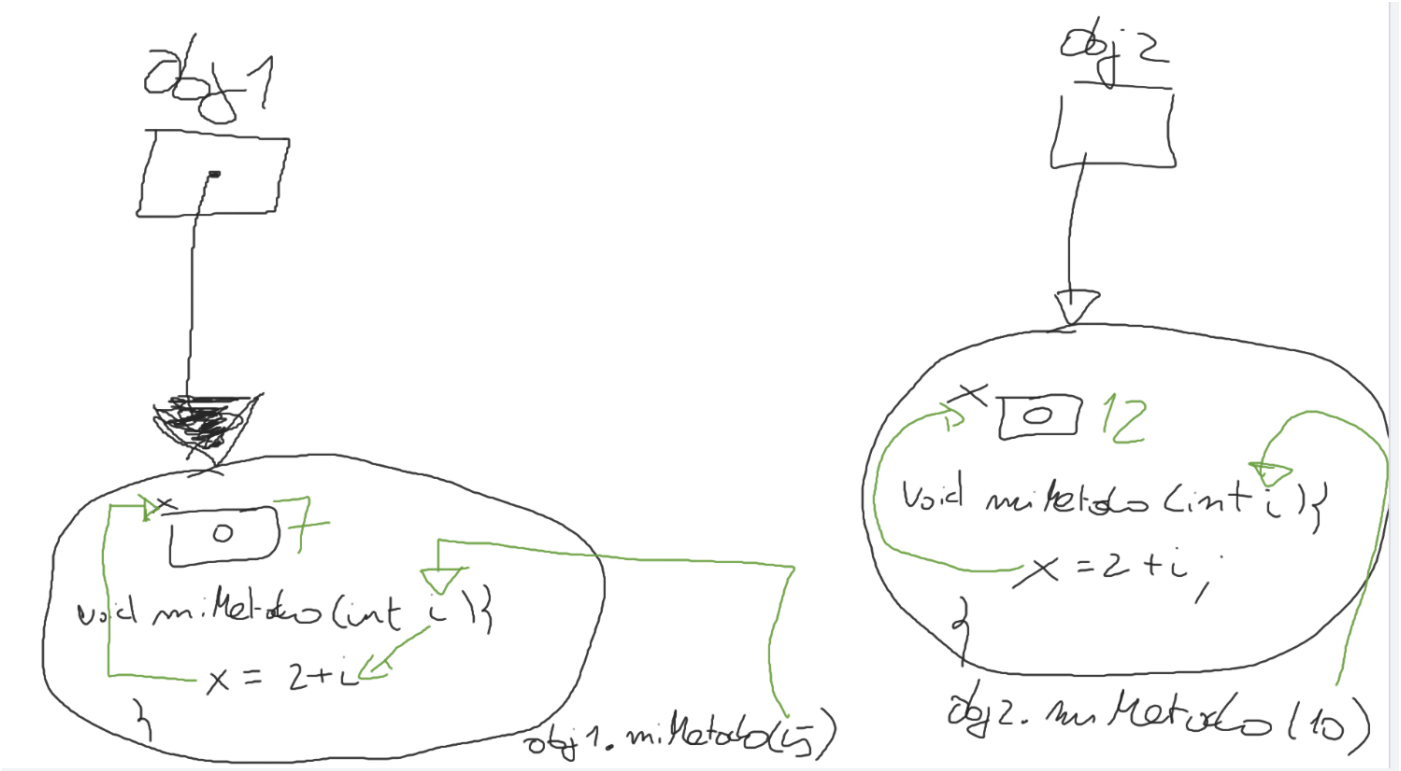
¿a la x de qué objeto nos referimos en el código de miMetodo()?. Ya que la definición de una clase es una suerte de molde o plantilla genérica, en el momento de escribir la clase no se refiere a un objeto concreto. Escribimos el método dentro de la clase para que sea aplicable a todo objeto de tipo MiClase. Luego dicho método "genérico" tendrá una ejecución "concreta" cuando se invoque sobre un objeto en particular y será entonces cuando la x tomará el valor que pertenece a dicho objeto.

Ejemplo:

```
class MiClase{
   int x;
   void miMetodo(int i){
      x=2 +i;
   }
}


public class Unidad2 {
   public static void main(String[] args) {
      MiClase obj1 = new MiClase();
      MiClase obj2 = new MiClase();
      obj1.miMetodo(5);
      obj2.miMetodo(10);
      System.out.println("La x de obj1 vale "+ obj1.x);
      System.out.println("La x de obj2 vale "+ obj2.x);
   }
}
```

- Aunque obj1 y obj2 son referencias, no objetos, permitamos para esta explicación la licencia de llamarlos objetos.
- Cuando se invoca a miMetodo() con obj1.miMetodo(5); El atributo X es el del objeto obj1
- Cuando se invoca a miMetodo() con obj2.miMetodo(10); El atributo X es el del objeto obj2

obj1 y obj2 tienen en común la definición de clase con la que se crean, pero una vez creados, son objetos independientes que llevan "vidas en memoria" totalmente separadas.



La referencia this

this es una palabra reservada para indicar una variable referencia "especial". La variable this no la declaran los programadores si no que la poseen automáticamente todos los objetos, y almacena la dirección del propio objeto. Cuando se crea un objeto con el operador new, este operador reserva memoria para el objeto y devuelve la referencia a dicho objeto. Luego típicamente esa referencia se almacena en una variable. Pero, además, dicha referencia se almacena una variable referencia dentro del propio objeto y a esta variable referencia especial la llamamos this. *this* significa *"este objeto"* es decir, el objeto actual. Parece una perogrullada, pero pronto entenderemos que dentro del propio objeto necesitamos que se referencie a sí mismo, igual que nosotros nos referenciamos a nosotros mismos con la palabra "yo".

```
MiClase{
        int x;
        void miMetodo(int i){
                x=2+i;
        }
}
```

Internamente, podemos "imaginar" que existe de forma automática una variable referencia que se llama this con el siguiente aspecto

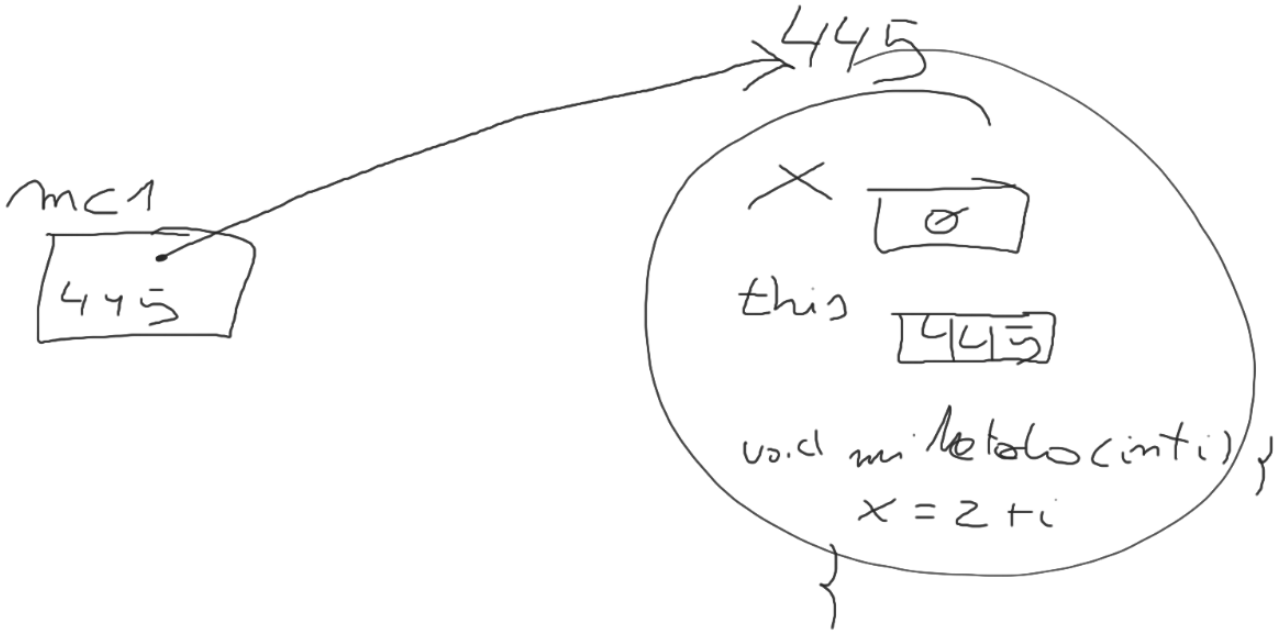
```
MiClase{
        int x;
        MiClase this=referencia de "este objeto";//esto no lo escribe el programador lo incluye java y no se vé
        void miMetodo(int i){
                x=2+i;
        }
}
```

y cuando se crea un objeto con new, en this se almacena la referencia que devuelve el new.

Vamos a dibujar la situación en memoria tras la ejecución de las siguiente sentencia de asignación

MiClase mc1= new MiClase();

Supongamos que new decidió reservar memoria para el objeto en la referencia 445. Observamos que el valor 445 está contenido dentro de mc1 y también dentro de la variable this dentro del propio objeto. El atributo x como no le dimos ningún valor valdrá por el momento el valor 0



Podríamos haber dibujado también dentro del objeto "el constructor por defecto"

¿Cuándo y cómo utilizar this?

La variable *this* se utiliza dentro del código de los métodos y constructores para referenciar a los atributos y métodos del propio objeto. Veremos varios casos en este boletín. El ejemplo anterior utilizando la variable *this* se puede escribir:

```
class MiClase{
   int x;
   void miMetodo(int i){
      this.x=2 +i;
   }
}
```

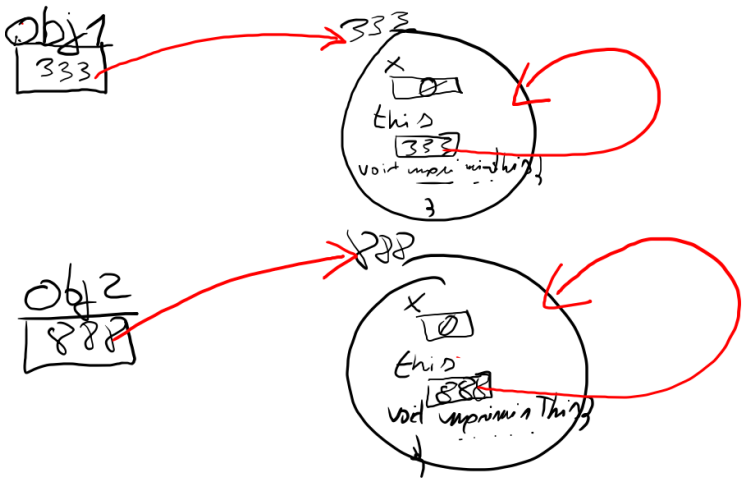
Comprobar que efectivamente el valor de this es individual para cada objeto.

Observa que de ningún modo hay que declarar la variable this, ya lo hace java, el programador simplemente la usa.

```
class MiClase{
   int x;
    void imprimirThis(){
          System.out.println("this vale: " +this);
   }
}

public class Unidad2 {
   public static void main(String[] args) {
      MiClase obj1 = new MiClase();
```

```
        MiClase obj2 = new MiClase();


        System.out.println("La referencia de obj1 es "+ obj1);
            System.out.print("Si llamamos a obj1.imprimirThis()...");obj1.imprimirThis();

            System.out.println("La referencia de obj2 es "+ obj2);
            System.out.print("Si llamamos a obj2.imprimirThis()...");obj2.imprimirThis();


    }
}
```

Suponiendo que el primer new devuelve la referencia 333 y el segundo la 888 una representación de los objetos justo después de su construcción podría ser



Utilizar this cuando coincide el nombre de un miembro con un parámetro o variable local

```
class MiClase{
  int x;
  void miMetodo(int x){
    x=2 +x;   //el parámetro x "oculta" al miembro x. En esta instrucción x se refiere al
              //  parámetro
  }
}
```

En este caso, Si por la razón que fuera no puedo o no quiero cambiar el nombre del parámetro debo de utilizar this

```
class MiClase{
  int x;
  void miMetodo(int x){
    this.x=2 +x;
  }
}
```

variable local(parámetro)

atributo

Observa y entiende las salidas de los siguientes programas

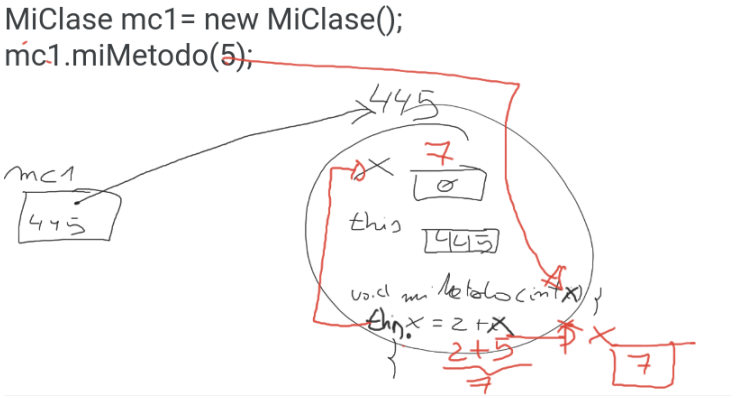
La variable local x oculta al atributo x

```
class MiClase{
    int x;
    void miMetodo(int x){
        x=2 +x;
    }
}

public class Unidad2 {
    public static void main(String[] args) {
        MiClase obj1 = new MiClase();
        MiClase obj2 = new MiClase();
        obj1.miMetodo(5);
        obj2.miMetodo(10);
        System.out.println("La x de obj1 vale "+ obj1.x);
        System.out.println("La x de obj2 vale "+ obj2.x);
    }
}
run:
La x de obj1 vale 0
La x de obj2 vale 0
```

This permite distinguir la x del atributo de la variable local

```
class MiClase{
    int x;
    void miMetodo(int x){
        this.x=2 +x;
    }
}

public class Unidad2 {
    public static void main(String[] args) {
        MiClase obj1 = new MiClase();
        MiClase obj2 = new MiClase();
        obj1.miMetodo(5);
        obj2.miMetodo(10);
        System.out.println("La x de obj1 vale "+ obj1.x);
        System.out.println("La x de obj2 vale "+ obj2.x);
    }
}
run:
La x de obj1 vale 7
La x de obj2 vale 12
```

```
MiClase{
    int x;
    void miMetodo(int x){
            x=2+i;
      }
}
```



Utilizar this para devolver el objeto llamante

En el ejemplo, un objeto círculo se define en este caso por su coordenadas en el espacio y por la longitud de su radio. El método elMayor() de la clase circulo recibe otro círculo por parámetro y devuelve la referencia del círculo mayor. Para simplificar, si es menor o igual se da como mayor el que se pasa por parámetro.

```
class Circulo{
  int coordenadaX;
  int coordenadaY;
  int radio;
  Circulo elMayor(Circulo c){
    if(radio>c.radio){ //o también if(this.radio>c.radio)
      return this;
    }else{
      return c;
    }
  }
}
```

Observa que el código anterior se pudo haber escrito(fíjate en el this en rojo que es la novedad)

```
class Circulo{
  int coordenadaX;
  int coordenadaY;
  int radio;
  Circulo elMayor(Circulo c){
    if(this.radio>c.radio){
      return this;
    }else{
      return c;
    }
  }
}
```

EJERCICIO U2_B5_E1:

Prueba la clase anterior Circulo desde un main() escrito por ti. El main() debe de crear dos círculos, comparar su radio e imprimir las coordenadas del círculo devuelto por elMayor(). Por otro lado, escribe el método elMayor() con el operador condicional sustituyendo al if

EJERCICIO U2_B5_E2:

Añade a Circulo un constructor de forma que funcione con el siguiente main()

```
class Unidad2{
    public static void main(String[] args) {
       Circulo c1=new Circulo(3,3,10);
       Circulo c2=new Circulo(50,45,5);

       Circulo circuloGrande=c1.elMayor(c2);
       System.out.println("el círculo mayor: ");
       System.out.println("\t coordenadaX: "+circuloGrande.coordenadaX);
       System.out.println("\t coordenaday: "+circuloGrande.coordenadaY);
       System.out.println("\t radio: "+circuloGrande.radio);
    }
}
```

EJERCICIO U2_B5_E3:

Reescribe el método elMayor() de forma que en vez de devolver una referencia al mayor de dos círculos, devuelva una copia del círculo mayor, es decir, otro objeto Circulo distinto(un nuevo objeto) pero con los mismos valores que el mayor de los dos iniciales.

```
class Circulo{
  int coordenadaX;
  int coordenadaY;
  int radio;
  Circulo(int x, int y, int r){
     coordenadaX=x;
     coordenadaY=y;
     radio=r;
  }
  Circulo elMayor(Circulo c){
    return this.radio>c.radio?this:c;
  }
}
```

Una vez modificada pruébala con el siguiente main()

```
class Unidad2{
    public static void main(String[] args) {
       Circulo c1=new Circulo(3,3,10);
       Circulo c2=new Circulo(50,45,5);

       Circulo circuloGrande=c1.elMayor(c2);
       System.out.println("el círculo mayor: ");
       System.out.println("\t coordenadaX: "+circuloGrande.coordenadaX);
       System.out.println("\t coordenaday: "+circuloGrande.coordenadaY);
       System.out.println("\t radio: "+circuloGrande.radio);
       System.out.println("demostramos que aunque circuloGrande incialmente es copia de c1 son objetos diferentes");
       System.out.println("cambiamos el radio de circulo grande pero no cambia el de c1");
       circuloGrande.radio=-99;
       System.out.println(circuloGrande.radio+", "+c1.radio);
    }
}
```

EJERCICIO U2_B5_E4:

Anteriormente, hicimos el siguiente ejercicio

```
class Coche {

    String modelo;
    int pasajeros;
    int deposito;
    int kpl;

    public String getModelo() {
        return modelo;
    }

    public void setModelo(String m) {
        modelo =m;
    }

    public int getPasajeros() {
```

```
        return pasajeros;
    }

    public void setPasajeros(int p) {
        pasajeros = p;
    }

    public int getDeposito() {
        return deposito;
    }

    public void setDeposito(int d) {
        deposito = d;
    }

    public int getKpl() {
        return kpl;
    }

    public void setKpl(int k) {
        kpl = k;
    }

    void calcularAutonomia() {
        System.out.println("Autonomía:" + deposito * kpl);
    }

    double gasofaNecesaria(int kilometros) {
        return (double) kilometros / kpl;
    }

}

class Unidad2{
    public static void main(String[] args) {
          Coche citroenC1= new Coche();
          citroenC1.setModelo("Citroen C1 special");
          citroenC1.setPasajeros(4);
          citroenC1.setDeposito(50);
          citroenC1.setKpl(25);

          System.out.println("un citroen C1 permite  " + citroenC1.getPasajeros() + " pasajeros");
          System.out.println("un citroen C1 tiene consumo de " + citroenC1.getKpl() + " kilómetros por litro");
    }
}
```

Modifícalo para que los métodos *set* utilicen *this* de forma que no importe que el nombre del parámetro coincida con el del atributo. Por ejemplo,

```
 public void setModelo(String m) {
        modelo =m;
    }
```

Lo puedo escribir como

```
 public void setModelo(String modelo) {
        this.modelo =modelo;
    }
```

EJERCICIO U2_B5_E5:

En el boletín anterior, vimos este ejemplo

```
class Coche{
  int pasajeros;
  int deposito;
  int kpl;
```

```
  int calcularAutonomia(){
    return  deposito*kpl;
  }

  boolean mayorAutonomia(Coche c){
    return calcularAutonomia()>c.calcularAutonomia();
     }
 }

class Unidad2 {

  public static void main(String[] args) {
      Coche coche1= new Coche();
      coche1.pasajeros=5;
      coche1.deposito=60;
      coche1.kpl=20;

      Coche coche2= new Coche();
      coche2.pasajeros=7;
      coche2.deposito=1000;
      coche2.kpl=30;
      System.out.println("Tiene coche1 más autonomía que coche2? "+ coche1.mayorAutonomia(coche2));
  }
}
```

e indicamos que mayorAutonomia() se entendería mejor si usamos this. Usa this y explica el funcionamiento del método.

EJERCICIO U2_B5_E6:

Recuerda el siguiente ejemplo ya visto. mayorAutonomia() devuelve un coche que internamente crea un objeto Coche y en él copia los datos del coche mayor. No teníamos en el boletín pasado forma de que devolviera directamente el coche propietario del método porque no conocíamos this. Reforma ahora el código de forma que no creamos ningún coche extra, simplemente si coche1 es mayor que coche2 devuelve coche1 y en caso contrario coche2.

OTRO USO DE THIS: una versión de constructor puede llamar a otra versión de constructor

Es un uso menos importante y frecuente que el anterior pero está bien conocerlo.

Recuerda que un constructor puede estar sobrecargado. Desde un constructor puedo aprovecharme del código de otro ya escrito sin estar duplicando código una y otra vez. Observa el siguiente ejemplo:

https://docs.oracle.com/javase/tutorial/java/javaOO/thiskey.html

```
public class Rectangle {
    private int x, y;
    private int width, height;
```

```
    public Rectangle() {
        this(0, 0, 1, 1);
    }
    public Rectangle(int width, int height) {
        this(0, 0, width, height);
    }
    public Rectangle(int x, int y, int width, int height) {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
    }
    ...
}
```

If present, the invocation of another constructor must be the first line in the constructor.

EJERCICIO U2_B5_E7:

Crea una clase Cuenta con los siguientes atributos privados

```
String numeroCuenta;
String titular;
double saldo;
```

La clase cuenta debe de ser manejada por la clase Unidad2 de la siguiente forma:

```
public class Unidad2 {
    public static void main(String[] args) {
        Cuenta c1 = new Cuenta("111-222", "Epi", 50.0);
        System.out.println("el saldo inicial de Epi es: " + c1.getSaldo());

        Cuenta c2 = new Cuenta("999-888", "Blas", 100.0);
        System.out.println("Datos de la cuenta c2: " + c2);

        c1.setSaldo(250.0);
        System.out.println("el nuevo saldo de Epi es: " + c1.getSaldo());

        Cuenta c3 = new Cuenta();
        System.out.println("datos de cuenta creada sin parámetros: " + c3);
    }
}
```

SALIDA obligatoria

```
el saldo inicial de Epi es: 50.0
Datos de la cuenta c2: (999-888, Blas, 100.0)
el nuevo saldo de Epi es: 250.0
datos de cuenta creada sin parÃ¡metros: (sin numero, sin titular, 0.0)
```

Al escribir tu solución ¡utilizar *this* para practicar!

El constructor copia

A menudo, queremos tener el constructor sobrecargado y una de las versiones típicas del constructor va ser una que genera el objeto a partir de los datos de otro objeto de su mismo tipo. A esta versión se le suele llamar *constructor copia*. Las característica principal de esta versión es que se le pasa como parámetro un objeto de su mismo

tipo e inicializa los atributos de *this* con los valores del objeto al que da acceso el parámetro.

```
class Coche {
 String modelo;
 int pasajeros;
 int deposito;
 int kpl;

 Coche(String modelo, int pasajeros, int deposito, int kpl) {
   this.modelo = modelo;
   this.pasajeros = pasajeros;
   this.deposito = deposito;
   this.kpl = kpl;
 }
 //versión constructor que se le suele llamar "constructor copia"
 Coche(Coche c){
   this(c.modelo, c.pasajeros,c.deposito, c.kpl);
 }
 }

class Unidad2 {
 public static void main(String[] args) {
     Coche coche1 = new Coche("Peugeot 308",5,60,20);
     Coche copiaCoche1=new Coche(coche1);

     System.out.println("mismos datos");
     System.out.println("\t"+coche1.modelo+" "+coche1.pasajeros+" "+coche1.deposito+ " "+coche1.kpl);
     System.out.println("\t"+copiaCoche1.modelo+" "+copiaCoche1.pasajeros+" "+copiaCoche1.deposito+ " "+copiaCoche1.kpl);
     System.out.println("diferentes objetos");
     System.out.println("\tcoche1: "+coche1+"     copiaCoche1:"+copiaCoche1);


 }
}
```

Observa que

```
Coche(Coche c){
   this.modelo=c.modelo;
   this.pasajeros= c.pasajeros;
   this.deposito=c.deposito;
   this.kpl= c.kpl;
 }
```

Es totalmente equivalente a

```
 Coche(Coche c){
   this(c.modelo, c.pasajeros,c.deposito, c.kpl);
 }
```

La ventaja de la segunda forma es simplemente que escribo menos.

EJERCICIO U2_B5_E8: Incorpora un constructor copia al siguiente código y utilízalo desde elMayor()

```
class Circulo{
  int coordenadaX;
  int coordenadaY;
  int radio;
  Circulo(int coordenadaX, int coordenadaY, int radio){
      this.coordenadaX=coordenadaX;
      this.coordenadaY=coordenadaY;
      this.radio=radio;
  }
  Circulo elMayor(Circulo c){
    Circulo copiaThis= new Circulo(this.coordenadaX,this.coordenadaY,this.radio);
    Circulo copiaC=new Circulo(c.coordenadaX,c.coordenadaY,c.radio);
    return this.radio>c.radio? copiaThis:copiaC;
  }
}
```

```
class Unidad2{
   public static void main(String[] args) {
      Circulo c1=new Circulo(3,3,10);
      Circulo c2=new Circulo(50,45,5);

      Circulo circuloGrande=c1.elMayor(c2);
      System.out.println("el círculo mayor: ");
      System.out.println("\t coordenadaX: "+circuloGrande.coordenadaX);
      System.out.println("\t coordenaday: "+circuloGrande.coordenadaY);
      System.out.println("\t radio: "+circuloGrande.radio);
      System.out.println("demostramos que aunque circuloGrande incialmente es copia de c1 son objetos diferentes");
      System.out.println("cambiamos el radio de circulo grande pero no cambia el de c1");
      circuloGrande.radio=-99;
      System.out.println(circuloGrande.radio+", "+c1.radio);
   }
}
```