USO DE SUPER EN EL CONSTRUCTOR DE UNA SUBCLASE

El siguiente texto está literalmente extraído del famoso libro *piensa en java* y explica: *Cuando se crea un objeto de la clase derivada, éste contiene en su interior un subobjeto de la clase base. Este subobjeto es idéntico a lo que tendríamos si hubiéramos creado directamente un objeto de la clase base. Lo que sucede, simplemente, es que, visto desde el exterior, el subobjeto de la clase base está envuelto por el objeto de la clase derivada. Por supuesto, es esencial que el subobjeto de la clase base se inicialice correctamente, y sólo hay una forma de garantizar esto: realizar la inicialización en el constructor invocando al constructor de la clase base, que es quien tiene todos los conocimientos y todos los privilegios para llevar a cabo adecuadamente la inicialización de la clase base.*

Desde un punto de vista práctico nos quedamos con lo siguiente: la inicialización de un objeto de una subclase comprende dos pasos.

1. La invocación al constructor de la superclase (llamada a super() en la primera línea del constructor)
2. Ejecución del resto de instrucciones propias del constructor de la subclase.

Y a esto hay que añadir al respecto del paso 1:

Si no se incluye una llamada a super() dentro del constructor de la clase base, el compilador java incluye automáticamente una llamada a super(). Este mecanismo sólo funciona cuando en la clase base hay una versión de constructor sin parámetros o bien no hay definido constructor y se usa el constructor por defecto. Observa que es similar a los casos cuando invocamos un constructor tras el operador *new*.

Ejemplo: La invocación al constructor de la superclase se realiza con super() y debe ser por tanto la primera instrucción del constructor de la subclase

```
class Base{
   int intBase=1;
}
class Derivada extends Base{
   int intDerivada;
   Derivada(){
      super();
      intDerivada=2;
   }
}
public class App {
   public static void main(String[] args) {
      Derivada d =new Derivada();
      System.out.println("atributo clase base: "+ d.intBase);
      System.out.println("atributo clase derivada: "+ d.intDerivada);
   }
}
```

Ejercicio: comprueba en el ejemplo anterior el error de compilación si super() no es la primera instrucción

Ejemplo: No incluimos en constructor de *Derivada* llamada a super() pero la clase *Base* no tiene definido constructor. En este caso java, internamente, incluye un super() que invoca al constructor por defecto de la clase base.

```
class Base{
   int intBase=1;
}
class Derivada extends Base{
   int intDerivada;
   Derivada(){
      intDerivada=2;
   }
}
public class App {
   public static void main(String[] args) {
       Derivada d =new Derivada();
       System.out.println("atributo clase base: "+ d.intBase);
       System.out.println("atributo clase derivada: "+ d.intDerivada);
   }
}
```

Ejemplo: No incluimos en constructor de *Derivada* llamada a super() pero la clase base tiene definido constructor sin parámetros. En este caso java incluye un super() que invoca al constructor sin parámetros. Aprovechamos para meter un println() en constructores y observar el orden de inicialización(primero parte *Base*, luego parte *Derivada*).

```
class Base{
   int intBase;
   Base(){
       System.out.println("construyendo objeto base");
       intBase=1;
   }
}
class Derivada extends Base{
   int intDerivada;
   Derivada(){
      System.out.println("construyendo objeto derivada");
      intDerivada=2;
   }
}
public class App {
   public static void main(String[] args) {
       Derivada d =new Derivada();
       System.out.println("atributo clase base: "+ d.intBase);
       System.out.println("atributo clase derivada: "+ d.intDerivada);
   }
}
```

Ejemplo: Reincidiendo en el caso anterior, al super() implícito de java no le molesta que el constructor esté sobrecargado(varias versiones), el simplemente utiliza la versión con la que encaja que es la versión sin parámetros.

```
class Base{
```

```
    int intBase;
    Base(){
        System.out.println("construyendo objeto base");
        intBase=1;
    }
    Base(int x){
        intBase=x;
    }
}
class Derivada extends Base{
    int intDerivada;
    Derivada(){
       System.out.println("construyendo objeto derivada");
       intDerivada=2;
    }
}
class App {
    public static void main(String[] args) {
        Derivada d =new Derivada();
        System.out.println("atributo clase base: "+ d.intBase);
        System.out.println("atributo clase derivada: "+ d.intDerivada);
    }
}
```

Ejemplo: El siguiente código da error porque el super() implícito no encaja

```
class Base{
    int intBase;
    Base(int x){
        intBase=x;
    }
}
class Derivada extends Base{
    int intDerivada;
    Derivada(){
       System.out.println("construyendo objeto derivada");
       intDerivada=2;
    }
}
public class App {
    public static void main(String[] args) {
        Derivada d =new Derivada();
        System.out.println("atributo clase base: "+ d.intBase);
        System.out.println("atributo clase derivada: "+ d.intDerivada);
    }
}
```

Ejemplo: para arreglar el código anterior sin tocar la clase Base nos vemos obligados a usar super()

```
class Base{
    int intBase;
    Base(int x){
        intBase=x;
    }
}
class Derivada extends Base{
    int intDerivada;
    Derivada(){
```

```
        super(1);
       System.out.println("construyendo objeto derivada");
       intDerivada=2;
    }
}
public class App {
    public static void main(String[] args) {
        Derivada d =new Derivada();
        System.out.println("atributo clase base: "+ d.intBase);
        System.out.println("atributo clase derivada: "+ d.intDerivada);
    }
}
```

Conclusión: ocurren los mismos casos idénticos a cuando creo un objeto "normal" (sin ser subobjeto de clase base)

Ejemplo: ¡hay que distinguir entre herencia y composición!

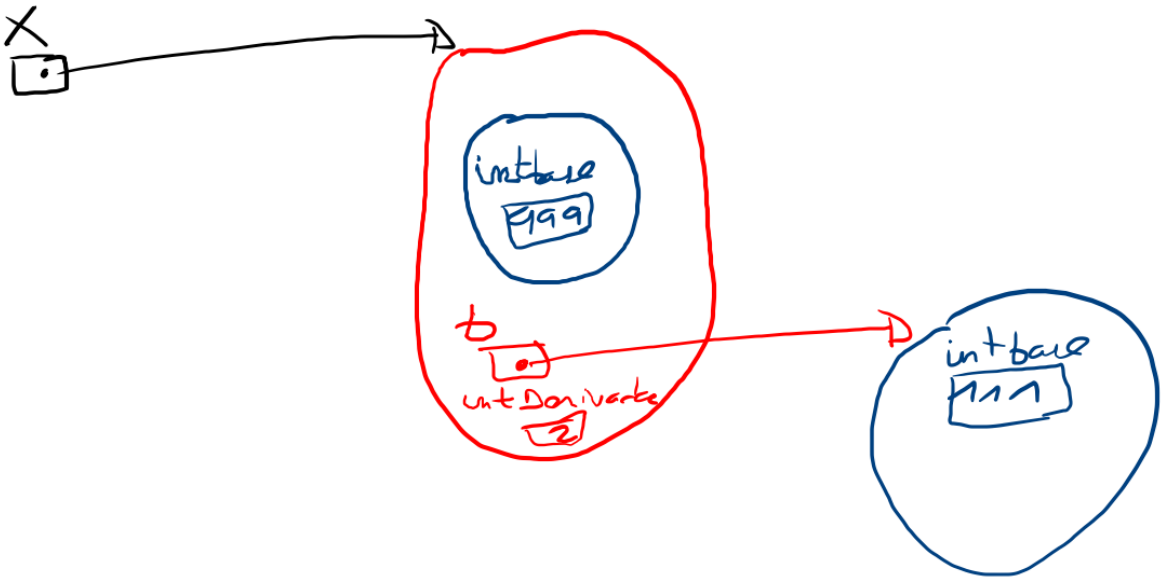
```
class Base{
    int intBase;
    Base(int x){
        intBase=x;
        System.out.println("construyendo objeto base");
    }
}

class Derivada extends Base{
    Base b;
    int intDerivada;
    Derivada(){
        super(999);
        System.out.println("construyendo objeto derivada");
        intDerivada=2;
        b=new Base(111);

    }
}
class App{
    public static void main(String[] args) {
        Derivada x= new Derivada();
    }
}
```

En el código anterior observamos que Derivada() invoca a su vez a dos contructores:

- Con super() se invoca la construcción de un objeto de tipo Base, pero ojo, este objeto será Padre del objeto this de Derivada() (relee el texto de *"piensa en java"*)
- Con new Base() se invoca también a la construcción de un objeto de tipo Base, pero en este caso, es un objeto de la clase Base con relación de composición con el objeto Derivada NO HAY RELACIÓN DE HERENCIA.



OTRO USO DE SUPER

La palabra reservada super aparece en dos contextos:

1. Para invocar al constructor de la clase base, como ya vimos:

   super() (o con parámetros)

2. Como variable para invocar otras partes del padre que no son constructor:

   super.miembro

Hay paralelismo con el uso de this

this() para invocar a un constructor del propio objeto this

This.miembro para referirse a método o atributo

Si this era una variable referencia "especial" que se usa para referirnos al propio objeto que se está ejecutando, super se usa para referenciar a los miembros del objeto padre.

Por tanto esto

```
class Base{
    int intBase;
    }
class Derivada extends Base{
    int intDerivada;
    Derivada(){
       intDerivada=intBase+1;
    }
}
```

es equivalente a

```
class Base{

   int intBase;
   }
class Derivada extends Base{
   int intDerivada;
   Derivada(){
     intDerivada=super.intBase+1;
   }
}
```

En el caso anterior. El uso de la referencia *super* es superfluo, pero al igual que pasa con *this*, hay situaciones en las que su uso es necesario, por ejemplo, cuando hay miembros de la clase Base y Derivada de idéntico nombre, en cuyo caso, los miembros de la clase derivada "ocultan" a los de la base. En este caso es necesario usar super para acceder a los miembros ocultos del padre.

Ejemplo:

```
class Base{
   int x=1;
   int y=9;
   }
class Derivada extends Base{
   int x=2;
   Derivada(){
      System.out.println("this.y: "+ this.y);
      System.out.println("this.x: "+ this.x);
      System.out.println("x: "+ x);
      System.out.println("super.x: "+ super.x);
   }
}
public class App {
   public static void main(String[] args) {
       Derivada d =new Derivada();
   }
}
```

Ejemplo: cuando no hay ocultación de un atributo y, super.y y this.y son equivalentes

```
class Base{
   int y=1;
   }
class Derivada extends Base{
   int x=2;
   Derivada(){
      System.out.println("this.x: "+ x);
      System.out.println("x: "+ x);
      System.out.println("super.y: "+ super.y);
      System.out.println("this.y: "+ this.y);
      System.out.println("y: "+ y);
   }
```

```
}
public class App {
    public static void main(String[] args) {
        Derivada d =new Derivada();
    }
}
```

Ejercicio U5_B3C_E1: resuelve el ejercicio U5_B3a_E3 con super(el de figuras)

Ejercicio U5_B3C_E2: resuelve el ejercicio U5_B3a_E4 con super(el de perros y gatos)

Ejercicio U5_B3C_E3:
Somos un poco "tozudos" y nos empeñamos en tener un BigInteger igual que el del API pero que tenga una versión de constructor sin parámetros. Escribe la clase MiBigInteger para conseguir dicho efecto.