Programación Orientada a Objetos en Java

Algoritmos y Estructuras de Datos

Cualidades del software

Fundamentales:

Más o menos importantes, dependiendo del **contexto de uso**:

El **diseño** consiste en organizar el programa de tal manera que cumpla con las cualidades requeridas, en algún contexto de uso.

Cualidades del software

Fundamentales:

- Correcto con respecto a una especificación.

Más o menos importantes, dependiendo del **contexto de uso**:

El **diseño** consiste en organizar el programa de tal manera que cumpla con las cualidades requeridas, en algún contexto de uso.

Cualidades del software

Fundamentales:

- Correcto con respecto a una especificación.

Más o menos importantes, dependiendo del **contexto de uso**:

- Eficiente (tiempo, memoria, consumo de energía, ...).

El **diseño** consiste en organizar el programa de tal manera que cumpla con las cualidades requeridas, en algún contexto de uso.

Cualidades del software

Fundamentales:

- Correcto con respecto a una especificación.

Más o menos importantes, dependiendo del **contexto de uso**:

- Eficiente (tiempo, memoria, consumo de energía, ...).
- Reutilizable.

El **diseño** consiste en organizar el programa de tal manera que cumpla con las cualidades requeridas, en algún contexto de uso.

Cualidades del software

Fundamentales:

- Correcto con respecto a una especificación.

Más o menos importantes, dependiendo del **contexto de uso**:

- Eficiente (tiempo, memoria, consumo de energía, ...).
- Reutilizable.
- Extensible / modificable.

El **diseño** consiste en organizar el programa de tal manera que cumpla con las cualidades requeridas, en algún contexto de uso.

Cualidades del software

Fundamentales:

- Correcto con respecto a una especificación.

Más o menos importantes, dependiendo del **contexto de uso**:

- Eficiente (tiempo, memoria, consumo de energía, ...).
- Reutilizable.
- Extensible / modificable.
- Usable.

El **diseño** consiste en organizar el programa de tal manera que cumpla con las cualidades requeridas, en algún contexto de uso.

Cualidades del software

Fundamentales:

- Correcto con respecto a una especificación.

Más o menos importantes, dependiendo del **contexto de uso**:

- Eficiente (tiempo, memoria, consumo de energía, ...).
- Reutilizable.
- Extensible / modificable.
- Usable.
- Legible.

El **diseño** consiste en organizar el programa de tal manera que cumpla con las cualidades requeridas, en algún contexto de uso.

Cualidades del software

Fundamentales:

- Correcto con respecto a una especificación.

Más o menos importantes, dependiendo del **contexto de uso**:

- Eficiente (tiempo, memoria, consumo de energía, ...).
- Reutilizable.
- Extensible / modificable.
- Usable.
- Legible.
- Predecible.

El **diseño** consiste en organizar el programa de tal manera que cumpla con las cualidades requeridas, en algún contexto de uso.

Cualidades del software

Fundamentales:

- Correcto con respecto a una especificación.

Más o menos importantes, dependiendo del **contexto de uso**:

- Eficiente (tiempo, memoria, consumo de energía, ...).
- Reutilizable.
- Extensible / modificable.
- Usable.
- Legible.
- Predecible.
- ...

El **diseño** consiste en organizar el programa de tal manera que cumpla con las cualidades requeridas, en algún contexto de uso.

Mmm, conocemos TADS..

Según la intro...

Mmm, conocemos TADS..

Según la intro...

- TAD quiere decir Tipo Abstracto de Datos

Mmm, conocemos TADS..

Según la intro...

- TAD quiere decir Tipo Abstracto de Datos
- ¿Qué es un Tipo Abstracto de Datos?

Mmm, conocemos TADS..

Según la intro...

- TAD quiere decir Tipo Abstracto de Datos
- ¿Qué es un Tipo Abstracto de Datos?
- Es un tipo de datos porque define un conjunto de valores y las operaciones que se pueden realizar sobre ellos

Mmm, conocemos TADS..

Según la intro...

- TAD quiere decir Tipo Abstracto de Datos
- ¿Qué es un Tipo Abstracto de Datos?
- Es un tipo de datos porque define un conjunto de valores y las operaciones que se pueden realizar sobre ellos
- Es abstracto ya que para utilizarlos, no se necesita conocer los detalles de la representación interna ni cómo están implementadas sus operaciones.

Mmm, conocemos TADS..

Según la intro...

- TAD quiere decir Tipo Abstracto de Datos
- ¿Qué es un Tipo Abstracto de Datos?
- Es un tipo de datos porque define un conjunto de valores y las operaciones que se pueden realizar sobre ellos
- Es abstracto ya que para utilizarlos, no se necesita conocer los detalles de la representación interna ni cómo están implementadas sus operaciones.
- Describe el "qué" y no el "cómo"

Mmm, conocemos TADS..

Según la intro...

- TAD quiere decir Tipo Abstracto de Datos
- ¿Qué es un Tipo Abstracto de Datos?
- Es un tipo de datos porque define un conjunto de valores y las operaciones que se pueden realizar sobre ellos
- Es abstracto ya que para utilizarlos, no se necesita conocer los detalles de la representación interna ni cómo están implementadas sus operaciones.
- Describe el "qué" y no el "cómo"
- Como usuarios nos interesa la "interfaz", o sea el contrato.

Mmm, conocemos TADS..

Según la intro...

- TAD quiere decir Tipo Abstracto de Datos
- ¿Qué es un Tipo Abstracto de Datos?
- Es un tipo de datos porque define un conjunto de valores y las operaciones que se pueden realizar sobre ellos
- Es abstracto ya que para utilizarlos, no se necesita conocer los detalles de la representación interna ni cómo están implementadas sus operaciones.
- Describe el "qué" y no el "cómo"
- Como usuarios nos interesa la "interfaz", o sea el contrato.
- Son una forma de modularizar a nivel de los datos

Mmm, conocemos TADS..

Según la intro...

- TAD quiere decir Tipo Abstracto de Datos
- ¿Qué es un Tipo Abstracto de Datos?
- Es un tipo de datos porque define un conjunto de valores y las operaciones que se pueden realizar sobre ellos
- Es abstracto ya que para utilizarlos, no se necesita conocer los detalles de la representación interna ni cómo están implementadas sus operaciones.
- Describe el "qué" y no el "cómo"
- Como usuarios nos interesa la "interfaz", o sea el contrato.
- Son una forma de modularizar a nivel de los datos
- ¿Y para qué sirven?

Mmm, conocemos TADS..

Según la intro...

- TAD quiere decir Tipo Abstracto de Datos
- ¿Qué es un Tipo Abstracto de Datos?
- Es un tipo de datos porque define un conjunto de valores y las operaciones que se pueden realizar sobre ellos
- Es abstracto ya que para utilizarlos, no se necesita conocer los detalles de la representación interna ni cómo están implementadas sus operaciones.
- Describe el "qué" y no el "cómo"
- Como usuarios nos interesa la "interfaz", o sea el contrato.
- Son una forma de modularizar a nivel de los datos
- ¿Y para qué sirven?
- ¡Modelar la realidad!

Mmm, conocemos TADS..

Según la intro...

- TAD quiere decir Tipo Abstracto de Datos
- ¿Qué es un Tipo Abstracto de Datos?
- Es un tipo de datos porque define un conjunto de valores y las operaciones que se pueden realizar sobre ellos
- Es abstracto ya que para utilizarlos, no se necesita conocer los detalles de la representación interna ni cómo están implementadas sus operaciones.
- Describe el "qué" y no el "cómo"
- Como usuarios nos interesa la "interfaz", o sea el contrato.
- Son una forma de modularizar a nivel de los datos
- ¿Y para qué sirven?
- ¡Modelar la realidad!
- Luego vamos a ver como se especifican

Mmm, conocemos TADS..

Según la intro...

- TAD quiere decir Tipo Abstracto de Datos
- ¿Qué es un Tipo Abstracto de Datos?
- Es un tipo de datos porque define un conjunto de valores y las operaciones que se pueden realizar sobre ellos
- Es abstracto ya que para utilizarlos, no se necesita conocer los detalles de la representación interna ni cómo están implementadas sus operaciones.
- Describe el "qué" y no el "cómo"
- Como usuarios nos interesa la "interfaz", o sea el contrato.
- Son una forma de modularizar a nivel de los datos
- ¿Y para qué sirven?
- ¡Modelar la realidad!
- Luego vamos a ver como se especifican
- En esta clase

Mmm, conocemos TADS..

Según la intro...

- TAD quiere decir Tipo Abstracto de Datos
- ¿Qué es un Tipo Abstracto de Datos?
- Es un tipo de datos porque define un conjunto de valores y las operaciones que se pueden realizar sobre ellos
- Es abstracto ya que para utilizarlos, no se necesita conocer los detalles de la representación interna ni cómo están implementadas sus operaciones.
- Describe el "qué" y no el "cómo"
- Como usuarios nos interesa la "interfaz", o sea el contrato.
- Son una forma de modularizar a nivel de los datos
- ¿Y para qué sirven?
- ¡Modelar la realidad!
- Luego vamos a ver como se especifican
- En esta clase

Mmm, conocemos TADS..

Según la intro...

- TAD quiere decir Tipo Abstracto de Datos
- ¿Qué es un Tipo Abstracto de Datos?
- Es un tipo de datos porque define un conjunto de valores y las operaciones que se pueden realizar sobre ellos
- Es abstracto ya que para utilizarlos, no se necesita conocer los detalles de la representación interna ni cómo están implementadas sus operaciones.
- Describe el "qué" y no el "cómo"
- Como usuarios nos interesa la "interfaz", o sea el contrato.
- Son una forma de modularizar a nivel de los datos
- ¿Y para qué sirven?
- ¡Modelar la realidad!
- Luego vamos a ver como se especifican
- En esta clase → Implementar

Mmm, conocemos TADS..

Según la intro...

- TAD quiere decir Tipo Abstracto de Datos
- ¿Qué es un Tipo Abstracto de Datos?
- Es un tipo de datos porque define un conjunto de valores y las operaciones que se pueden realizar sobre ellos
- Es abstracto ya que para utilizarlos, no se necesita conocer los detalles de la representación interna ni cómo están implementadas sus operaciones.
- Describe el "qué" y no el "cómo"
- Como usuarios nos interesa la "interfaz", o sea el contrato.
- Son una forma de modularizar a nivel de los datos
- ¿Y para qué sirven?
- ¡Modelar la realidad!
- Luego vamos a ver como se especifican
- En esta clase → Implementar ¿Cómo lo hago?

Java....

Vamos a utilizar a nuestro amigo **Java**, aunque los conceptos aplican a la mayoría de lenguajes modernos.

¿Qué utilizaremos?

¿Qué utilizaremos?

- Vamos a utilizar una pequeña parte de Programación Orientada a Objetos

¿Qué utilizaremos?

- Vamos a utilizar una pequeña parte de Programación Orientada a Objetos
- Utilizaremos clases de "Java"

¿Qué utilizaremos?

- Vamos a utilizar una pequeña parte de Programación Orientada a Objetos
- Utilizaremos clases de "Java"
- Un poco más profundo que lo que vienen utilizando

¿Qué utilizaremos?

- Vamos a utilizar una pequeña parte de Programación Orientada a Objetos
- Utilizaremos clases de "Java"
- Un poco más profundo que lo que vienen utilizando

¿Qué utilizaremos?

- Vamos a utilizar una pequeña parte de Programación Orientada a Objetos
- Utilizaremos clases de "Java"
- Un poco más profundo que lo que vienen utilizando

Según wikipedia

La programación orientada a objetos (POO, en español); es un paradigma de programación que parte del concepto de **objetos** como base, los cuales contienen información en forma de campos; a veces también referidos como **atributos** o **propiedades** y código en forma de **métodos**.

Los objetos son capaces de interactuar y modificar los valores contenidos en sus campos o atributos (estado) a través de sus métodos (comportamiento).

Algunas características clave de la programación orientada a objetos son herencia, cohesión, abstracción, polimorfismo, acoplamiento y encapsulamiento.

Conceptos importantes

- **Abstracción**: Énfasis en el "¿qué hace?" más que en el "¿cómo lo hace?".

Conceptos importantes

- **Abstracción**: Énfasis en el "¿qué hace?" más que en el "¿cómo lo hace?".
- **Encapsulamiento**: Ocultar el funcionamiento interno (que podría cambiar), centrándonos en lo que necesita el usuario.

Conceptos importantes

- **Abstracción**: Énfasis en el "¿qué hace?" más que en el "¿cómo lo hace?".
- **Encapsulamiento**: Ocultar el funcionamiento interno (que podría cambiar), centrándonos en lo que necesita el usuario.
- ⚠Usamos una partecita y algunos conceptos....

Conceptos importantes

- **Abstracción**: Énfasis en el "¿qué hace?" más que en el "¿cómo lo hace?".
- **Encapsulamiento**: Ocultar el funcionamiento interno (que podría cambiar), centrándonos en lo que necesita el usuario.
- ⚠Usamos una partecita y algunos conceptos....
- Más en ingeniería de software

Clases

Vamos a utilizar clases !

Definiendo clases

Vamos a definir una clase en varias etapas:

1. Vamos a declarar los *métodos públicos*. O sea sus "operaciones"

Definiendo clases

Vamos a definir una clase en varias etapas:

1. Vamos a declarar los *métodos públicos*. O sea sus "operaciones"
2. Vamos a declarar los *atributos privados* de la clase. Su "estado".

Definiendo clases

Vamos a definir una clase en varias etapas:

1. Vamos a declarar los *métodos públicos*. O sea sus "operaciones"
2. Vamos a declarar los *atributos privados* de la clase. Su "estado".
3. Vamos a implementar los métodos públicos.

Definiendo clases

Vamos a definir una clase en varias etapas:

1. Vamos a declarar los *métodos públicos*. O sea sus "operaciones"
2. Vamos a declarar los *atributos privados* de la clase. Su "estado".
3. Vamos a implementar los métodos públicos.
4. En particular, el *constructor* de la clase (o los constructores).

Definiendo clases

Vamos a definir una clase en varias etapas:

1. Vamos a declarar los *métodos públicos*. O sea sus "operaciones"
2. Vamos a declarar los *atributos privados* de la clase. Su "estado".
3. Vamos a implementar los métodos públicos.
4. En particular, el *constructor* de la clase (o los constructores).

Definiendo clases

Vamos a definir una clase en varias etapas:

1. Vamos a declarar los *métodos públicos*. O sea sus "operaciones"
2. Vamos a declarar los *atributos privados* de la clase. Su "estado".
3. Vamos a implementar los métodos públicos.
4. En particular, el *constructor* de la clase (o los constructores).

- Se puede programar con clases pero no modularmente.

Ejemplo

Modelar el sistema `Cuanto Te Debo - CTD)`, que nos permite compartir gastos entre dos personas y luego obtener cuánto le corresponde compensarle a la otra persona.

Necesitamos:

- Conocer los gastos de cada participante

Ejemplo

Modelar el sistema `Cuanto Te Debo - CTD)`, que nos permite compartir gastos entre dos personas y luego obtener cuánto le corresponde compensarle a la otra persona.

Necesitamos:

- Conocer los gastos de cada participante
- Saber, hasta el momento, cuánto debe cada participante al otro.

Ejemplo

Modelar el sistema `Cuanto Te Debo - CTD)`, que nos permite compartir gastos entre dos personas y luego obtener cuánto le corresponde compensarle a la otra persona.

Necesitamos:

- Conocer los gastos de cada participante
- Saber, hasta el momento, cuánto debe cada participante al otro.
- Saber qué participante debe compensar.

```java
public class CTD {
  private int[] gastosPersona1;
  private int[] gastosPersona2;

    public CTD() {
        //{...}
    }

  public void anotarGasto(int persona, int gasto) {
        //{...}
    }

  private int[] agregarGasto(int[] gasto_persona, int gasto) {
        //{...}
    }

  public int gastos(int persona) {
        //{...}
  }

  public int quienSeTieneQuePoner() {
        //{...}
  }
}
```

Si tuviéramos esta clase en Java, ¿sabríamos cómo usarla?

- ¿Qué gastos tiene una persona al iniciar?

Si tuviéramos esta clase en Java, ¿sabríamos cómo usarla?

- ¿Qué gastos tiene una persona al iniciar?
- ¿Qué devuelve quienSeTieneQuePoner?

Si tuviéramos esta clase en Java, ¿sabríamos cómo usarla?

- ¿Qué gastos tiene una persona al iniciar?
- ¿Qué devuelve quienSeTieneQuePoner?
- Hace falta una especificación !

Clases e instancias

```
CTD c1 = new CTD();

c1.anotarGasto(0, 5);
c1.anotarGasto(1, 3);
c1.anotarGasto(0, 1);
c1.anotarGasto(0, 3);

c1.gastos(0); // 9
c1.gastos(1); // 3
```

```
CTD c2 = new CTD();

c2.anotarGasto(0, 8);
c2.anotarGasto(1, 2);
c2.anotarGasto(1, 3);
c2.anotarGasto(0, 7);

c2.gastos(0); // 15
c2.gastos(1); // 5
```

Clases e instancias

```
CTD c1 = new CTD();

c1.anotarGasto(0, 5);
c1.anotarGasto(1, 3);
c1.anotarGasto(0, 1);
c1.anotarGasto(0, 3);

c1.gastos(0); // 9
c1.gastos(1); // 3
```

```
CTD c2 = new CTD();

c2.anotarGasto(0, 8);
c2.anotarGasto(1, 2);
c2.anotarGasto(1, 3);
c2.anotarGasto(0, 7);

c2.gastos(0); // 15
c2.gastos(1); // 5
```

`class CTD` es la clase.

`c1` y `c2` son **objetos** distintos, pero ambos son **instancias** de la clase CTD.

Para que el comportamiento de la clase pueda llevarse a cabo, hay que implementarla.

La **implementación** está dada por:

- La **representación interna** : un conjunto de `atributos` (variables) que determina el estado interno de la instancia.

Para que el comportamiento de la clase pueda llevarse a cabo, hay que implementarla.

La **implementación** está dada por:

- La **representación interna** : un conjunto de `atributos` (variables) que determina el estado interno de la instancia.
- Un conjunto de **algoritmos** que implementan cada una de las operaciones de la interfaz (métodos), consultando y modificando las variables de la representación interna.

Declaración de atributos privados

```
class CTD {
  /* ... */
  private int[] gastosPersona1;
  private int[] gastosPersona2;

};
```

Estados internos

```
c1.anotarGasto(0, 5);
c1.anotarGasto(1, 3);
c1.anotarGasto(0, 1);
c1.anotarGasto(0, 3);

// Estado interno de c1
c1.gastosPersona1 == [5, 1, 3]
c1.gastosPersona2 == [3]
```

```
c2.anotarGasto(0, 8);
c2.anotarGasto(1, 2);
c2.anotarGasto(1, 3);
c2.anotarGasto(0, 7);

// Estado interno de c2
c1.gastosPersona1 == [8, 7]
c1.gastosPersona2 == [2, 3]
```

Estados internos

```
c1.anotarGasto(0, 5);
c1.anotarGasto(1, 3);
c1.anotarGasto(0, 1);
c1.anotarGasto(0, 3);

// Estado interno de c1
c1.gastosPersona1 == [5, 1, 3]
c1.gastosPersona2 == [3]
```

```
c2.anotarGasto(0, 8);
c2.anotarGasto(1, 2);
c2.anotarGasto(1, 3);
c2.anotarGasto(0, 7);

// Estado interno de c2
c1.gastosPersona1 == [8, 7]
c1.gastosPersona2 == [2, 3]
```

El estado interno **no es accesible** desde afuera del objeto (encapsulamiento).

⚠Un objeto siempre debe ser usado mediante su interfaz.

Comportamiento

¿Cómo definimos comportamiento para las instancias?

Comportamiento

¿Cómo definimos comportamiento para las instancias?
A través de **métodos**:

```
public void anotarGasto(int persona, int gasto) {
  if (persona == 0) {
    gastosPersona1 = agregarGasto(gastosPersona1, gasto);
  } else {
    gastosPersona2 = agregarGasto(gastosPersona2, gasto);
  }
}
```

Comportamiento

Los métodos pueden utilizar métodos privados. Es decir “internos” que sólo serán accesibles desde métodos de la clase

Comportamiento

Los métodos pueden utilizar métodos privados. Es decir “internos” que sólo serán accesibles desde métodos de la clase Pensemos un segundo cómo implementarían agregar gasto

Comportamiento

Los métodos pueden utilizar métodos privados. Es decir “internos” que sólo serán accesibles desde métodos de la clase Pensemos un segundo cómo implementarían agregar gasto Ojo ⚠los arrays de java no son redimensionables !

Comportamiento

Los métodos pueden utilizar métodos privados. Es decir "internos" que sólo serán accesibles desde métodos de la clase Pensemos un segundo cómo implementarían agregar gasto Ojo ⚠los arrays de java no son redimensionables !

```
private int[] agregarGasto(int[] gasto_persona, int gasto) {
  int[] gasto_persona_nuevo = new int[gasto_persona.length + 1];
  for (int i = 0; i < gasto_persona.length; i++) {
    gasto_persona_nuevo[i] = gasto_persona[i];
  }
  gasto_persona_nuevo[gasto_persona_nuevo.length - 1] = gasto;
  return gasto_persona_nuevo;
}
```

Comportamiento

Los métodos pueden utilizar métodos privados. Es decir "internos" que sólo serán accesibles desde métodos de la clase Pensemos un segundo cómo implementarían agregar gasto Ojo ⚠los arrays de java no son redimensionables !

```
private int[] agregarGasto(int[] gasto_persona, int gasto) {
  int[] gasto_persona_nuevo = new int[gasto_persona.length + 1];
  for (int i = 0; i < gasto_persona.length; i++) {
    gasto_persona_nuevo[i] = gasto_persona[i];
  }
  gasto_persona_nuevo[gasto_persona_nuevo.length - 1] = gasto;
  return gasto_persona_nuevo;
}
```

En la clase que viene van a ver una mejor manera de manejar estas secuencias

Ejemplo

```
int main() {
    CTD c1 = new CTD();
    CTD c2 = new CTD();
                        // <---
    c1.anotarGasto(0, 3);
    c1.anotarGasto(1, 1);
    c2.anotarGasto(1, 5);
}
```

Contexto

c1.gastosPersona1	[]
c1.gastosPersona2	[]
c2.gastosPersona1	[]
c2.gastosPersona2	[]

Ejemplo

```
int main() {
    CTD c1 = new CTD();
    CTD c2 = new CTD();
    c1.anotarGasto(0, 3);
                        // <---
    c1.anotarGasto(1, 1);
    c2.anotarGasto(1, 5);

}
```

Contexto

c1.gastosPersona1	[3]
c1.gastosPersona2	[]
c2.gastosPersona1	[]
c2.gastosPersona2	[]

Ejemplo

```
int main() {
    CTD c1 = new CTD();
    CTD c2 = new CTD();
    c1.anotarGasto(0, 3);
    c1.anotarGasto(1, 1);
                        // <---
    c2.anotarGasto(1, 5);

}
```

Contexto

c1.gastosPersona1	[3]
c1.gastosPersona2	[1]
c2.gastosPersona1	[]
c2.gastosPersona2	[]

Ejemplo

```
int main() {
    CTD c1 = new CTD();
    CTD c2 = new CTD();
    c1.anotarGasto(0, 3);
    c1.anotarGasto(1, 1);
    c2.anotarGasto(1, 5);
                        // <---

}
```

Contexto

c1.gastosPersona1	[3]
c1.gastosPersona2	[1]
c2.gastosPersona1	[]
c2.gastosPersona2	[5]

El resto de los ingredientes

La interfaz de CPD tiene métodos para ver el gasto de los jugadores:

```
public int gastos(int persona) {
  int total = 0;
  if (persona == 0) {
    for (int gasto : gastosPersona1) {
      total += gasto;
    }
  } else {
    for (int gasto : gastosPersona2) {
      total += gasto;
    }
  }
  return total;
}
```

El resto de los ingredientes

Pero los miembros privados de una clase no son accessibles desde afuera:

```
void ejemplo() {
  CTD c = new CTD();
  System.out.println(c.gastosPersona1);
    // error: The field c.gastosPersona1 is not visible.
}
```

El resto de los ingredientes

```
void ejemplo() {
  CTD c = new CTD() ;
  System.out.println(c.gastos(1));
}
```

Constructor

- Veamos nuevamente agregarGasto.

```
private int[] agregarGasto(int[] gasto_persona, int gasto) {
  int[] gasto_persona_nuevo = new int[gasto_persona.length + 1];
  for (int i = 0; i < gasto_persona.length; i++) {
    gasto_persona_nuevo[i] = gasto_persona[i];
  }
  gasto_persona_nuevo[gasto_persona_nuevo.length - 1] = gasto;
  return gasto_persona_nuevo;
}
```

Constructor

- Veamos nuevamente agregarGasto.

```
private int[] agregarGasto(int[] gasto_persona, int gasto) {
  int[] gasto_persona_nuevo = new int[gasto_persona.length + 1];
  for (int i = 0; i < gasto_persona.length; i++) {
    gasto_persona_nuevo[i] = gasto_persona[i];
  }
  gasto_persona_nuevo[gasto_persona_nuevo.length - 1] = gasto;
  return gasto_persona_nuevo;
}
```

- ¿Qué contiene gasto_persona al principio?

Constructor

- Veamos nuevamente agregarGasto.

```
private int[] agregarGasto(int[] gasto_persona, int gasto) {
  int[] gasto_persona_nuevo = new int[gasto_persona.length + 1];
  for (int i = 0; i < gasto_persona.length; i++) {
    gasto_persona_nuevo[i] = gasto_persona[i];
  }
  gasto_persona_nuevo[gasto_persona_nuevo.length - 1] = gasto;
  return gasto_persona_nuevo;
}
```

- ¿Qué contiene `gasto_persona` al principio?
- Los constructores son funciones especiales para inicializar una nueva instancia de una clase.

Constructor

- Veamos nuevamente agregarGasto.

```
private int[] agregarGasto(int[] gasto_persona, int gasto) {
  int[] gasto_persona_nuevo = new int[gasto_persona.length + 1];
  for (int i = 0; i < gasto_persona.length; i++) {
    gasto_persona_nuevo[i] = gasto_persona[i];
  }
  gasto_persona_nuevo[gasto_persona_nuevo.length - 1] = gasto;
  return gasto_persona_nuevo;
}
```

- ¿Qué contiene `gasto_persona` al principio?
- Los constructores son funciones especiales para inicializar una nueva instancia de una clase.
- Se escriben con el nombre de la clase.

Constructor

- Veamos nuevamente agregarGasto.

```
private int[] agregarGasto(int[] gasto_persona, int gasto) {
  int[] gasto_persona_nuevo = new int[gasto_persona.length + 1];
  for (int i = 0; i < gasto_persona.length; i++) {
    gasto_persona_nuevo[i] = gasto_persona[i];
  }
  gasto_persona_nuevo[gasto_persona_nuevo.length - 1] = gasto;
  return gasto_persona_nuevo;
}
```

- ¿Qué contiene `gasto_persona` al principio?
- Los constructores son funciones especiales para inicializar una nueva instancia de una clase.
- Se escriben con el nombre de la clase.
- No tienen tipo de retorno (está implícito, en realidad, la clase es el "tipo").

Constructor

- Veamos nuevamente agregarGasto.

```
private int[] agregarGasto(int[] gasto_persona, int gasto) {
  int[] gasto_persona_nuevo = new int[gasto_persona.length + 1];
  for (int i = 0; i < gasto_persona.length; i++) {
    gasto_persona_nuevo[i] = gasto_persona[i];
  }
  gasto_persona_nuevo[gasto_persona_nuevo.length - 1] = gasto;
  return gasto_persona_nuevo;
}
```

- ¿Qué contiene `gasto_persona` al principio?
- Los constructores son funciones especiales para inicializar una nueva instancia de una clase.
- Se escriben con el nombre de la clase.
- No tienen tipo de retorno (está implícito, en realidad, la clase es el "tipo").

Constructor

- Veamos nuevamente agregarGasto.

```
private int[] agregarGasto(int[] gasto_persona, int gasto) {
  int[] gasto_persona_nuevo = new int[gasto_persona.length + 1];
  for (int i = 0; i < gasto_persona.length; i++) {
    gasto_persona_nuevo[i] = gasto_persona[i];
  }
  gasto_persona_nuevo[gasto_persona_nuevo.length - 1] = gasto;
  return gasto_persona_nuevo;
}
```

- ¿Qué contiene `gasto_persona` al principio?
- Los constructores son funciones especiales para inicializar una nueva instancia de una clase.
- Se escriben con el nombre de la clase.
- No tienen tipo de retorno (está implícito, en realidad, la clase es el "tipo").

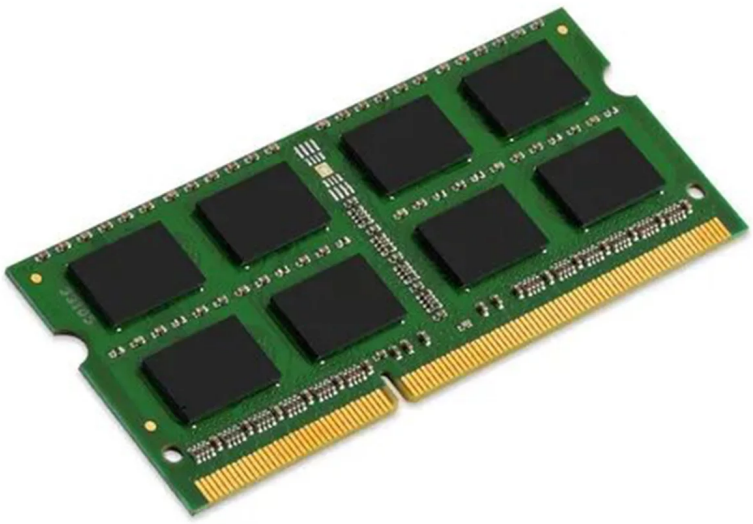
```
public CTD() {
  gastosPersona1 = new int[0];
  gastosPersona2 = new int[0];
}
```

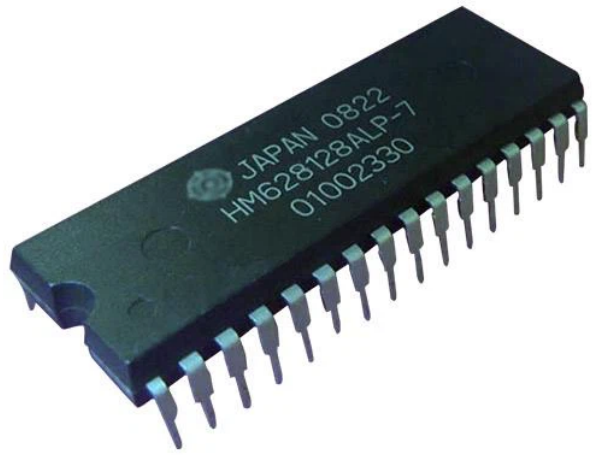
Constructor por copia

A veces queremos tener una copia del objeto en cuestión. Eso lo vamos a ver hoy un poquito más adelante.

Memoria

Esto es una memoria:





Memoria

Y esto para qué me sirve?

Memoria

Y esto para qué me sirve?

- Muchos de los *bugs* que suele haber al programar vienen de un mal manejo de la memoria.

Memoria

Y esto para qué me sirve?

- Muchos de los *bugs* que suele haber al programar vienen de un mal manejo de la memoria.
- En los próximos talleres van a tener problemas cada vez más díficiles que requieren entender bien estos conceptos.

Memoria

Y esto para qué me sirve?

- Muchos de los *bugs* que suele haber al programar vienen de un mal manejo de la memoria.
- En los próximos talleres van a tener problemas cada vez más díficiles que requieren entender bien estos conceptos.
- Veremos lo mínimo necesario para la vida de un programador.

Memoria

Y esto para qué me sirve?

- Muchos de los *bugs* que suele haber al programar vienen de un mal manejo de la memoria.
- En los próximos talleres van a tener problemas cada vez más díficiles que requieren entender bien estos conceptos.
- Veremos lo mínimo necesario para la vida de un programador.
- Más en: Sistemas Digitales, Arquitectura y Organización de Computadoras, Sistemas Operativos.

Memoria

Todo el contenido de las variables ocupa "espacio" en la "memoria". Los sistemas operativos suelen separar la memoria en dos grandes regiones:

Contexto local ⇒ en la pila (*stack*)

Las variables locales viven únicamente dentro del scope local. Aquí se guardan tipos primitivos y referencias a arreglos y objetos.

Memoria

Todo el contenido de las variables ocupa "espacio" en la "memoria". Los sistemas operativos suelen separar la memoria en dos grandes regiones:

Contexto local ⇒ en la pila (*stack*)

Las variables locales viven únicamente dentro del scope local. Aquí se guardan tipos primitivos y referencias a arreglos y objetos.

Dinámica (manual) ⇒ en el *heap*

En el heap se almacenan los objetos. La memoria de un objeto se libera cuando se vuelve inalcanzable.

Ejemplo de código

```
void ejemplo() {
    int x = 8;
    int d = doble(x);
}
int doble(int x) {
    int res = 2 * x;
    return res;
}
```

Seguimiento de código

```
void ejemplo() {
    int x = 8;
    // <---
    int d = doble(x);
}
int doble(int x) {
    int res = 2 * x;
    return res;
}
```

Stack	
x (ejemplo)	8

Seguimiento de código

```
void ejemplo() {
    int x = 8;
    int d = doble(x);// <---
}
int doble(int x) {
    // <---
    int res = 2 * x;
    return res;
}
```

Stack	
x (doble)	8
x (ejemplo)	8

Seguimiento de código

```
void ejemplo() {
    int x = 8;
    int d = doble(x);// <---
}
int doble(int x) {
    // <---
    int res = 2 * x;
    return res;
}
```

Stack	
x (doble)	8
x (ejemplo)	8

¿Qué pasaría si el código de `doble` modificara la variable `x`?

Seguimiento de código

```
void ejemplo() {
    int x = 8;
    int d = doble(x);// <---
}
int doble(int x) {
    int res = 2 * x;
    // <---
    return res;
}
```

Stack	
res (doble)	16
x (doble)	8
x (ejemplo)	8

Seguimiento de código

```
void ejemplo() {
    int x = 8;
    int d = doble(x);
    // <---
}
int doble(int x) {
    int res = 2 * x;
    return res;
}
```

Stack	
d (ejemplo)	16
x (ejemplo)	8

Seguimiento de código

```
void ejemplo() {
    int x = 8;
    int d = doble(x);
    // <---
}
int doble(int x) {
    int res = 2 * x;
    return res;
}
```

Stack	
d (ejemplo)	16
x (ejemplo)	8

¿Qué pasa con la memoria al llamar una función recursiva?

Un ejemplo distinto

```
void ejemplo() {
    int[] xs = new int[]{15, 8, 42};
    duplicar(xs);
}
void duplicar(int[] secu) {
    int i = 0;
    while (i < secu.length) {
        secu[i] = 2 * secu[i];
        i++;
    }
}
```

Un ejemplo distinto

```
void ejemplo() {
    int[] xs = new int[]{15, 8, 42};
    duplicar(xs);
}
void duplicar(int[] secu) {
    int i = 0;
    while (i < secu.length) {
        secu[i] = 2 * secu[i];
        i++;
    }
}
```

¿Dónde se guardará el arreglo? ¿Qué hace el `new`?

Seguimiento

```
void ejemplo() {
    int[] xs = new int[]{15, 8, 42};
    // <---
    duplicar(xs);
}
void duplicar(int[] secu) {
    int i = 0;
    while (i < secu.length) {
        secu[i] = 2 * secu[i];
        i++;
    }
}
```

Stack	
xs (ejemplo)	1104

Heap	
Posición	Valor
...	
1104	[15, 8, 42]
...	

Seguimiento

```
void ejemplo() {
    int[] xs = new int[]{15, 8, 42};
    duplicar(xs);// <---
}
void duplicar(int[] secu) {
    int i = 0;
    // <---
    while (i < secu.length) {
        secu[i] = 2 * secu[i];
        i++;
    }
}
```

Stack	
i (duplicar)	0
secu (duplicar)	1104
xs (ejemplo)	1104

Heap	
Posición	Valor
...	
1104	[15, 8, 42]
...	

Seguimiento

```
void ejemplo() {
    int[] xs = new int[]{15, 8, 42};
    duplicar(xs);// <---
}
void duplicar(int[] secu) {
    int i = 0;
    while (i < secu.length) {
        secu[i] = 2 * secu[i];
        i++;
        // <---
    }
}
```

Stack	
i (duplicar)	1
secu (duplicar)	1104
xs (ejemplo)	1104

Heap	
Posición	Valor
...	
1104	[30, 8, 42]
...	

Seguimiento

```
void ejemplo() {
    int[] xs = new int[]{15, 8, 42};
    duplicar(xs);// <---
}
void duplicar(int[] secu) {
    int i = 0;
    while (i < secu.length) {
        secu[i] = 2 * secu[i];
        i++;
        // <---
    }
}
```

Stack	
i (duplicar)	2
secu (duplicar)	1104
xs (ejemplo)	1104

Heap	
Posición	Valor
...	
1104	[30, 16, 42]
...	

Seguimiento

```
void ejemplo() {
    int[] xs = new int[]{15, 8, 42};
    duplicar(xs);// <---
}
void duplicar(int[] secu) {
    int i = 0;
    while (i < secu.length) {
        secu[i] = 2 * secu[i];
        i++;
        // <---
    }
}
```

Stack	
i (duplicar)	3
secu (duplicar)	1104
xs (ejemplo)	1104

Heap	
Posición	Valor
...	
1104	[30, 16, 84]
...	

Más punteros

```
class Estudiante {
    int edad;
    int[] notas;
    Estudiante (int e, int[] n) {
        edad = e;
        notas = n;
    }
}
void ejemplo() {
    Estudiante[] estudiantes = new Estudiante[2];
    estudiantes[0] = new Estudiante(20, new int[]{8, 7, 9});
    estudiantes[1] = new Estudiante(19, new int[]{6, 8, 4});
}
```

Más punteros

```
class Estudiante {
    int edad;
    int[] notas;
    Estudiante (int e, int[] n) {
        edad = e;
        notas = n;
    }
}
void ejemplo() {
    Estudiante[] estudiantes = new Estudiante[2];
    estudiantes[0] = new Estudiante(20, new int[]{8, 7, 9});
    estudiantes[1] = new Estudiante(19, new int[]{6, 8, 4});
}
```

¿Cómo queda organizada la memoria?

Seguimiento

```
class Estudiante {
    int edad;
    int[] notas;
    ...
}
void ejemplo() {
    Estudiante[] estudiantes = new Estudiante[2];
    // <---
    estudiantes[0] = new Estudiante(20, new int[]{8, 7, 9});
    estudiantes[1] = new Estudiante(19, new int[]{6, 8, 4});
}
```

Stack	
estudiantes	3054

Heap	
3054	[0, 0]

Seguimiento

```
class Estudiante {
    int edad;
    int[] notas;
    ...
}
void ejemplo() {
    Estudiante[] estudiantes = new Estudiante[2];
    estudiantes[0] = new Estudiante(20, new int[]{8, 7, 9});
    // <---
    estudiantes[1] = new Estudiante(19, new int[]{6, 8, 4});
}
```

Stack	
estudiantes	3054

Heap	
3054	[9416, 0]
...	
3107	[8, 7, 9]
...	
9416	Estudiante(edad: 20, notas: 3107)

Seguimiento

```
class Estudiante {
    int edad;
    int[] notas;
    ...
}
void ejemplo() {
    Estudiante[] estudiantes = new Estudiante[2];
    estudiantes[0] = new Estudiante(20, new int[]{8, 7, 9});
    estudiantes[1] = new Estudiante(19, new int[]{6, 8, 4});
    // <---
}
```

Stack	
estudiantes	3054

Heap	
3054	[9416, 5331]
...	
3107	[8, 7, 9]
...	
5331	Estudiante(edad: 19, notas: 6883)
...	
6883	[6, 8, 4]
...	
9416	Estudiante(edad: 20, notas: 3107)

Aliasing

¿Qué recuerdan de IP?

Aliasing

¿Qué recuerdan de IP?

- Cuando dos variables referencian al mismo valor, decimos que hay *aliasing* entre ellas.

Aliasing

¿Qué recuerdan de IP?

- Cuando dos variables referencian al mismo valor, decimos que hay *aliasing* entre ellas.
- En el caso de objetos/arreglos, lo podemos verificar con == (¿por qué?).

Aliasing

¿Qué recuerdan de IP?

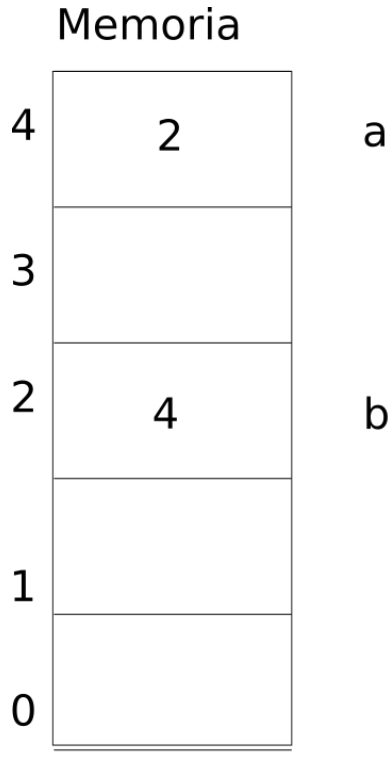
- Cuando dos variables referencian al mismo valor, decimos que hay *aliasing* entre ellas.
- En el caso de objetos/arreglos, lo podemos verificar con == (¿por qué?).
- El aliasing es un problema cuando se trata de objetos mutables (modificables).

Aliasing

¿Qué recuerdan de IP?

- Cuando dos variables referencian al mismo valor, decimos que hay *aliasing* entre ellas.
- En el caso de objetos/arreglos, lo podemos verificar con == (¿por qué?).
- El aliasing es un problema cuando se trata de objetos mutables (modificables).
- Si exponemos referencias a los atributos de nuestra clase, nos exponemos a que el usuario de la misma nos modifique sin que nos demos cuenta (ojo en el taller).

La memoria son solo números



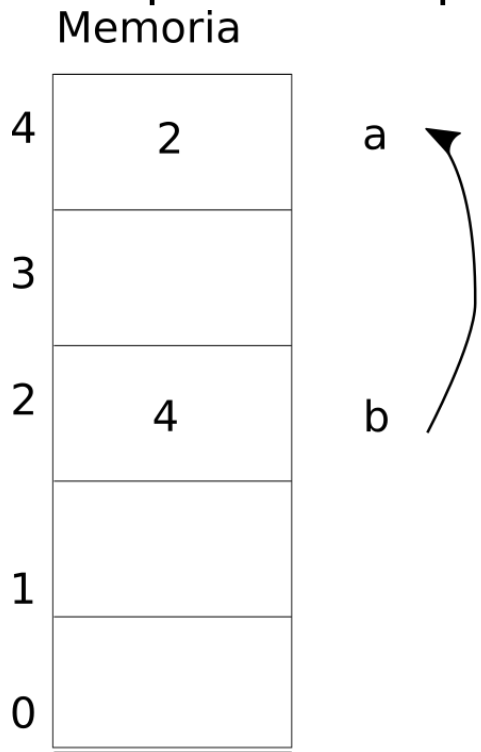
¿Cómo lo interpreto?

La memoria son solo números

Java podría interpretarlo así:

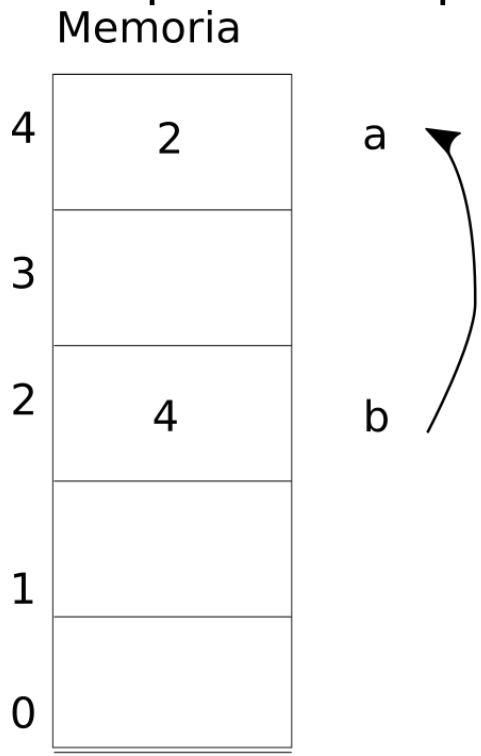
La memoria son solo números

Java podría interpretarlo así:



La memoria son solo números

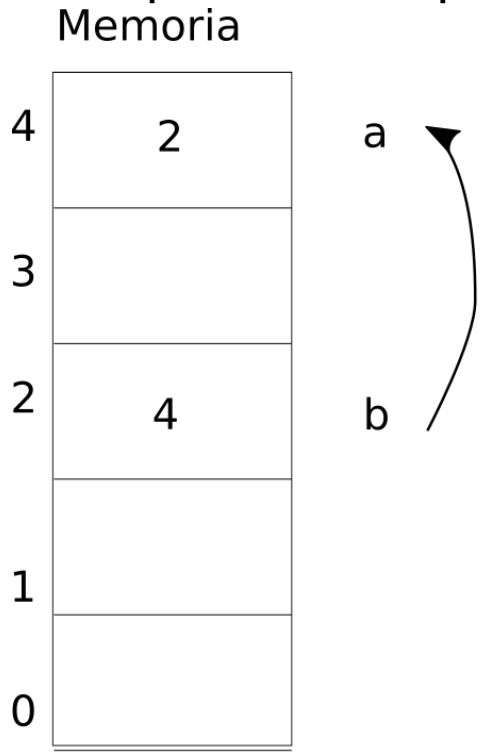
Java podría interpretarlo así:



¿Cómo sabe si es un número o una dirección de memoria?

La memoria son solo números

Java podría interpretarlo así:



¿Cómo sabe si es un número o una dirección de memoria?
¡Conociendo los tipos!

Recuerden:

- Si el tipo es primitivo, `a == b` dice si “a” y “b” son *iguales*.

Recuerden:

- Si el tipo es primitivo, `a == b` dice si "a" y "b" son *iguales*.
- Si el tipo es un objeto/arreglo, `a == b` dice si "a" y "b" son alias de un mismo objeto.

Recuerden:

- Si el tipo es primitivo, `a == b` dice si "a" y "b" son *iguales*.
- Si el tipo es un objeto/arreglo, `a == b` dice si "a" y "b" son alias de un mismo objeto.
- Si el tipo es un objeto, `a.equals(b)` dice si "a" y "b" son *iguales*.

Recuerden:

- Si el tipo es primitivo, `a == b` dice si "a" y "b" son *iguales*.
- Si el tipo es un objeto/arreglo, `a == b` dice si "a" y "b" son alias de un mismo objeto.
- Si el tipo es un objeto, `a.equals(b)` dice si "a" y "b" son *iguales*.

Recuerden:

- Si el tipo es primitivo, `a == b` dice si "a" y "b" son *iguales*.
- Si el tipo es un objeto/arreglo, `a == b` dice si "a" y "b" son alias de un mismo objeto.
- Si el tipo es un objeto, `a.equals(b)` dice si "a" y "b" son *iguales*.

¡Muy importante declarar el `equals` cuando escribimos clases nuevas!

Método `equals`

Volviendo a la primera parte, deberíamos entonces definir equals....

Método equals

Volviendo a la primera parte, deberíamos entonces definir equals....

```
@Override
public boolean equals(Object otro) {
    // Algunos chequeos burocraticos...
    boolean otroEsNull = (otro == null);

    boolean claseDistinta = otro.getClass() != this.getClass();

    if (otroEsNull || claseDistinta) {
        return false;
    }

    //  casting -> cambiar el tipo
    CTD otroCTD = (CTD) otro;

    return gastosPersona1 == otroCTD.gastosPersona1
      && gastosPersona2 == otroCTD.gastosPersona2;
}
```

Método equals

Volviendo a la primera parte, deberíamos entonces definir equals....

```
@Override
public boolean equals(Object otro) {
    // Algunos chequeos burocraticos...
    boolean otroEsNull = (otro == null);

    boolean claseDistinta = otro.getClass() != this.getClass();

    if (otroEsNull || claseDistinta) {
        return false;
    }

    //  casting -> cambiar el tipo
    CTD otroCTD = (CTD) otro;

    return gastosPersona1 == otroCTD.gastosPersona1
      && gastosPersona2 == otroCTD.gastosPersona2;
}
```

¿Notan algo mal?

Método `equals`

Volviendo a la primera parte, deberíamos entonces definir equals....

Método equals

Volviendo a la primera parte, deberíamos entonces definir equals....

¿Pero como comparamos los arrays?

```
public boolean equals(Object otro) {

    // Algunos chequeos burocraticos...
  boolean otroEsNull = (otro == null);

  boolean claseDistinta = otro.getClass() != this.getClass();

  if (otroEsNull || claseDistinta) {
    return false;
  }

  //  casting -> cambiar el tipo
  CTD otroCTD = (CTD) otro;



  // comparar item a ítem
  return arraysIguales(gastosPersona1, otroCTD.gastosPersona1)
    && arraysIguales(gastosPersona2, otroCTD.gastosPersona2);
}
```

Método equals

Volviendo a la primera parte, deberíamos entonces definir equals....

¿Pero como comparamos los arrays?

```
public boolean equals(Object otro) {

    // Algunos chequeos burocraticos...
  boolean otroEsNull = (otro == null);

  boolean claseDistinta = otro.getClass() != this.getClass();

  if (otroEsNull || claseDistinta) {
    return false;
  }

  //  casting -> cambiar el tipo
  CTD otroCTD = (CTD) otro;



  // comparar item a ítem
  return arraysIguales(gastosPersona1, otroCTD.gastosPersona1)
    && arraysIguales(gastosPersona2, otroCTD.gastosPersona2);
}
```

Método equals

```
private boolean arraysIguales(int[] array1, int[] array2) {
  // comparar length
  if (array1.length != array2.length) {
    return false;
  }
  for (int i = 0; i < array1.length; i++) {
    if (array1[i] != array2[i]) return false;
  }
  return true;
}
```

Método equals

```
private boolean arraysIguales(int[] array1, int[] array2) {
  // comparar length
  if (array1.length != array2.length) {
    return false;
  }
  for (int i = 0; i < array1.length; i++) {
    if (array1[i] != array2[i]) return false;
  }
  return true;
}
```

Constructor por copia

Volviendo a la primera parte, deberíamos entonces definir como copiar,
`c1 = c2` que hace ?

Constructor por copia

Volviendo a la primera parte, deberíamos entonces definir como copiar,
`c1 = c2` que hace ? Luego, ¿como copiaríamos?

Constructor por copia

Volviendo a la primera parte, deberíamos entonces definir como copiar,
`c1 = c2` que hace ? Luego, ¿como copiaríamos?

```
    // Copia el CTD, copiando en cascada
public CTD(CTD otro) {
  // Copia el CTD, copiando en cascada
  gastosPersona1 = otro.gastosPersona1.clone();
  gastosPersona2 = otro.gastosPersona2.clone();
}
```

Recursos

- Visualizador de memoria en Java (con ejemplos)

Recursos

- Visualizador de memoria en Java (con ejemplos)
- Otro visualizador, con varios lenguajes