10 Pines

Diseño a la Gorra - Episodio 01

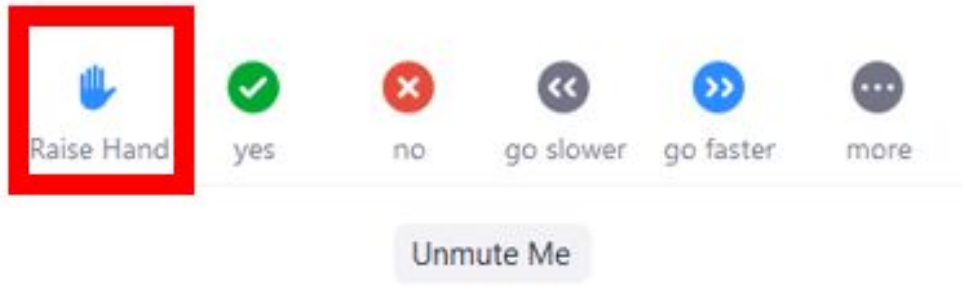
Hernán Wilkinson

hernan.wilkinson@10pines.com

@HernanWilkinson

https://alagorra.10pines.com

Estar muteados a menos que sea necesario

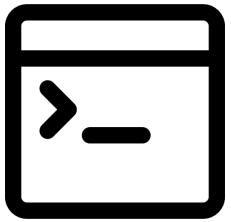


No voy a poder leer el chat

La comunicación visual es importante. Usarla a discreción

¿De qué trata?

Charlas de Diseño de Software con Objetos



Ejemplos prácticos de código

Martes 19 hrs - GMT-3

Todos lo que podamos y tenga sentido

Temas:

- Modelado
- Encapsulamiento
- Antropomorfismo
- Polimorfismo
- Código repetido
- Declaratividad
- Excepciones
- TDD
- Refactorings
- Mantenimiento
- Lenguajes de programación
- Tecnologías y Frameworks
- Historia
- … y mucho más

¿Por qué “a la gorra”?

Diseño ¡a la gorra!

¡Bienvenidos!
Durante esta serie de Webinars exploraremos *qué* significa **Diseñar Software con Objetos** y *cómo* lo podemos hacer cada vez mejor.
Trataremos muchos temas que irán desde cuestiones filosóficas como qué significa Diseñar en nuestra profesión y dónde está expresado ese Diseño, pasando por consejos y heurísticas para diseñar "mejor" y terminado con ejemplos concretos de cómo aplicar esas heurísticas en la *vida real.*
Los webinars son "*language agnostic*", o sea que no dependen de un lenguaje de programación en particular, aunque los ejemplos que usaremos estarán hechos principalmente en **Java**, **JavaScript**, **Ruby**, **Python** y mi querido **Smalltalk** cuando amerite 😉 .

Te esperamos todos los Martes a las 19 Hrs GMT-3 a partir del Martes 11 de Agosto de 2020.
Para poder participar tenes que registrate acá.

Todo el código y presentaciones estarán disponibles para que lo puedan usar y consultar en cualquier momento acá.

¡Trae ganas de aprender y pasarla bien!

¿Por qué a la Gorra?
Al igual que cuando Diseñamos Software está bueno usar una **Metáfora** para entender qué estamos modelando, en este caso usamos una metáfora para explicar cómo *financiaremos*

Donaciones

mercado pago

$100 - Casi una 🍷 Pagar

$250 - Una buena 🍺 Pagar

$500 - Menos que 🍔 +🍟 Pagar

PayPal Donate

¿Querés donar un monto variado o en otra plataforma?

Dictado por:
Hernán Wilkinson

https://alagorra.10pines.com

Online

Construcción de Software Robusto con TDD

Empieza el **24/08**

Del 24/08 al 28/08. Días: De Lunes a Viernes — 9:00 a 13:00 GMT-3.

AR$ **13.500-** (IVA incluído)

Hacer una consulta

Inscribirme ahora 10% Dto.

Dictado por:
Máximo Prieto

~~AR$ 15.000-~~

AR$ **13.500-**

(IVA incluído)

~~U$D 250-~~

U$D225-

(impuestos incluídos)

Early Bird - Hasta el 17/8/2020

http://academia.10pines.com

¿Qué es Diseñar?

Formar un **plan** o **esquema** para su **posterior ejecución**

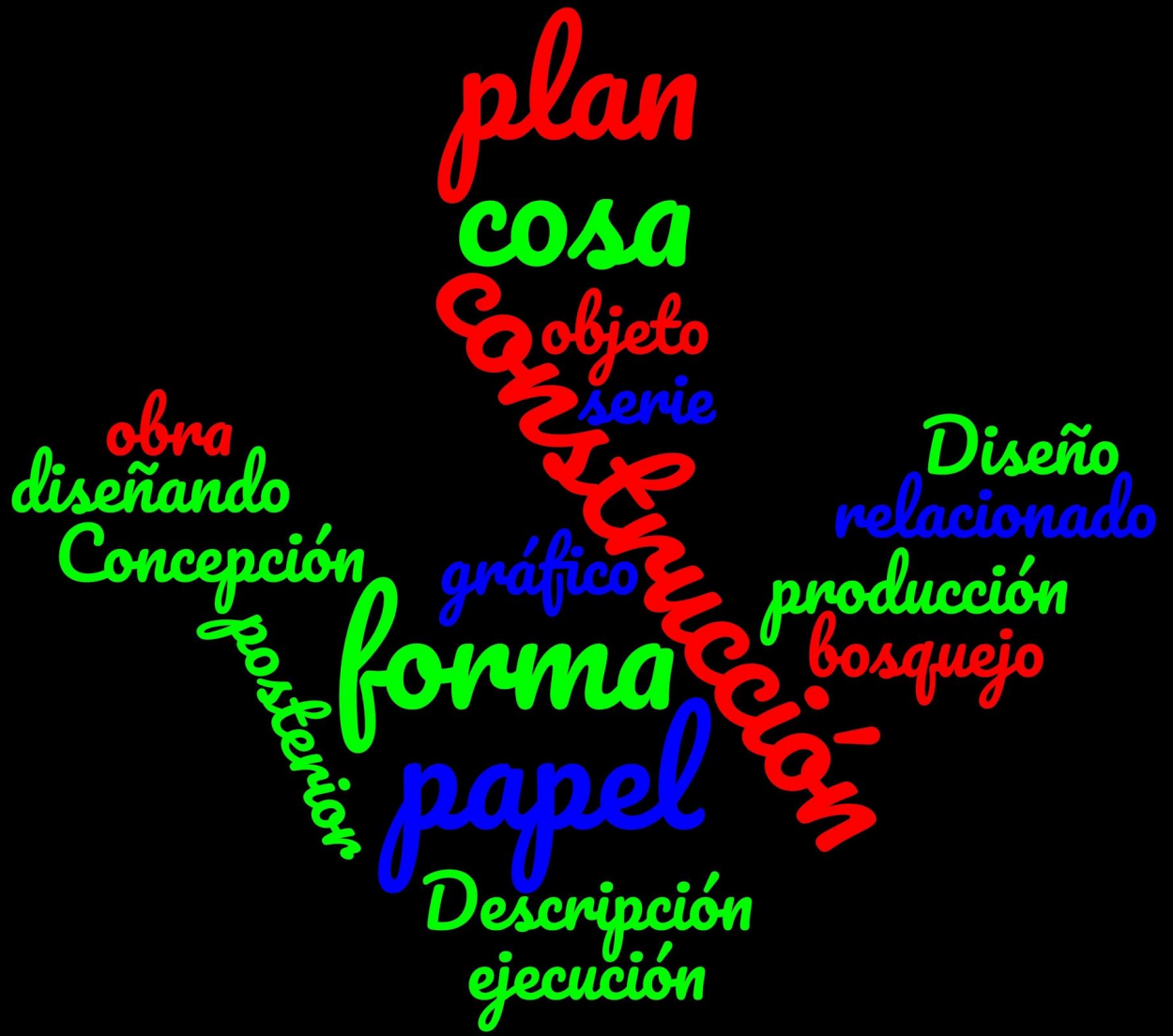
Ilustrar en un **papel** una forma que sirve de **modelo** de una cosa **a construir**

Hacer un **Diseño**

¿Qué es Diseño?

Concepción original de un objeto u obra destinados a la **producción en serie**

Descripción, **gráfico** o **bosquejo en papel** relacionado con la cosa que se está diseñando, **destinado a su construcción**


plan
cosa
objeto
serie
construcción
obra
diseñando
Concepción
gráfico
Diseño
relacionado
producción
bosquejo
forma
posterior
papel
Descripción
ejecución

Diseño es la
definición que usamos para **construir**

¿Dónde está expresado el Diseño de Software?

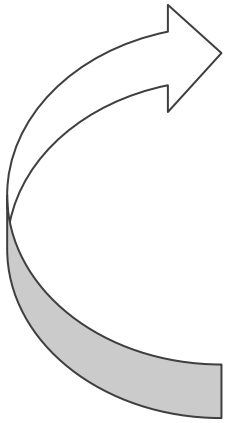
¿Qué es lo que construimos?

¡Software!

¿Qué es lo que usamos para construir Software?
¿A partir de qué “obtenemos” el Software?

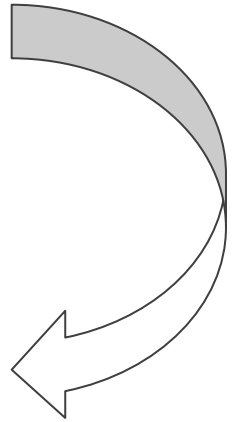
¡Código Fuente!

Código Fuente = Diseño de Software



Programar es Diseñar

Diseñar es Programar

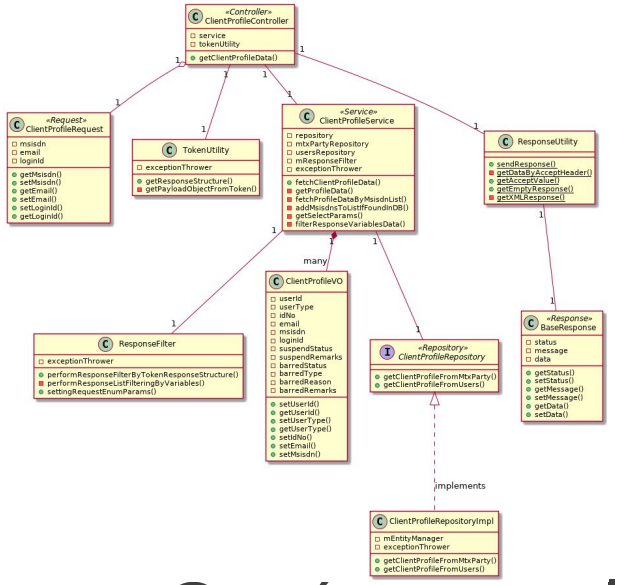


COMPILER

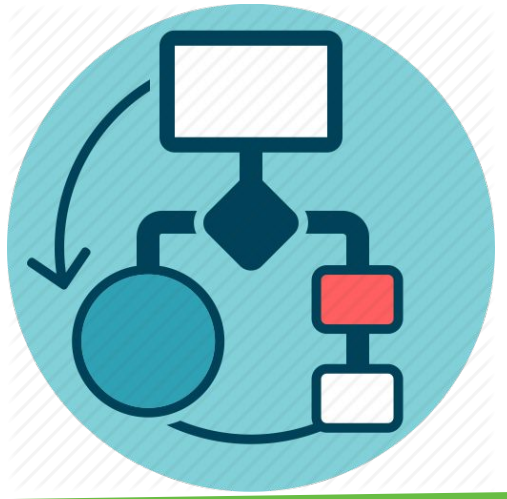
Nuestros “Constructores” son los Compiladores

Testear y **Debuggear** son actividades de **Diseño**

(validación y refinamiento de un diseño)



¿Qué papel juegan los “diagramas de diseño”? (Diagrama de Clases, de secuencia, etc)



Visualizar el Diseño
Pero no son el Diseño

What is Software Design?

by

Jack W. Reeves

Object oriented techniques, and C++ in particular, seem to be taking the software world by storm. Numerous articles and books have appeared describing how to apply the new techniques. In general, the questions of whether O-O techniques are just hype have been replaced by questions of how to get the benefits with the least amount of pain. Object oriented techniques have been around for some time, but this exploding popularity seems a bit unusual. Why the sudden interest? All kinds of explanations have been offered. In truth, there is probably no single reason. Probably, a combination of factors has finally reached critical mass and things are taking off. Nevertheless, it seems that C++ itself is a major factor in this latest phase of the software revolution. Again, there are probably a number of reasons why, but I want to suggest an answer from a slightly different perspective: C++ has become popular because it makes it easier to design software and program at the same time.

https://wiki.c2.com/?WhatIsSoftwareDesign

Algunas Conclusiones Importantes

- El Software nos obliga a cambiar definiciones
  → El Diseño no es únicamente en papel ni gráfico
  → Por eso nos cuesta entender tanto qué hacemos
- Estimar es difícil porque no estimamos **Cómo Construir** sino **Cómo Diseñar**
- Es Software es recursivo

¿Qué es Software?

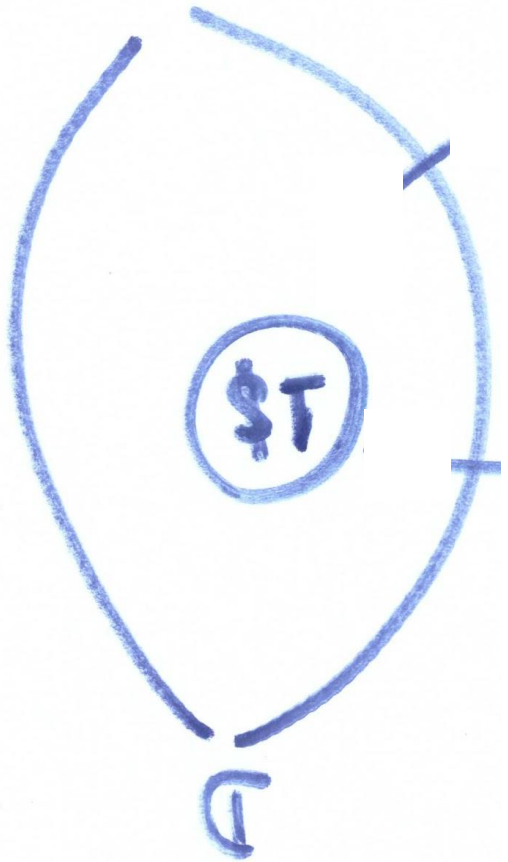
El software es el **diseño de una máquina** en la que se convierte una computadora

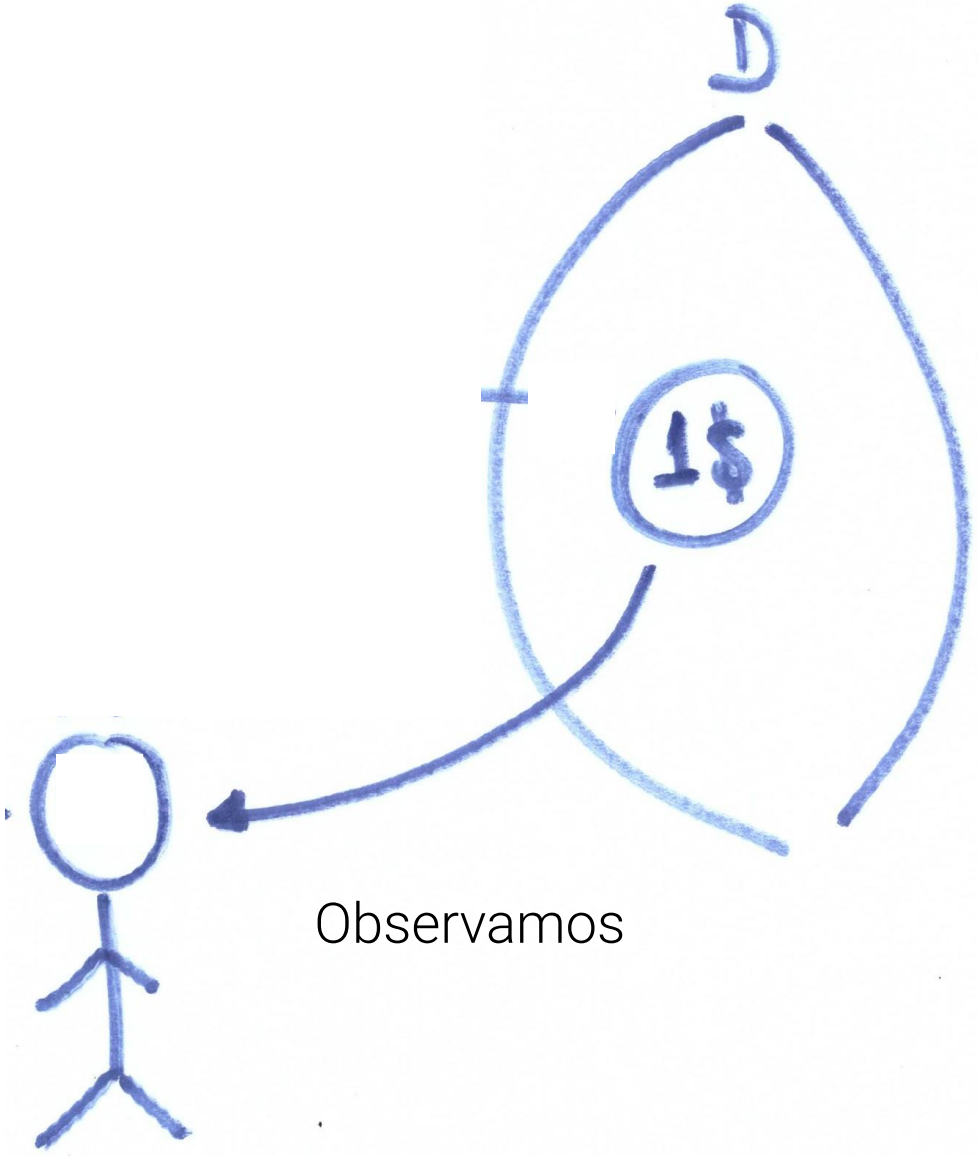
https://wiki.c2.com/?WhatIsSoftware

Modelo Computable
de un **Dominio de Problema**
de la Realidad

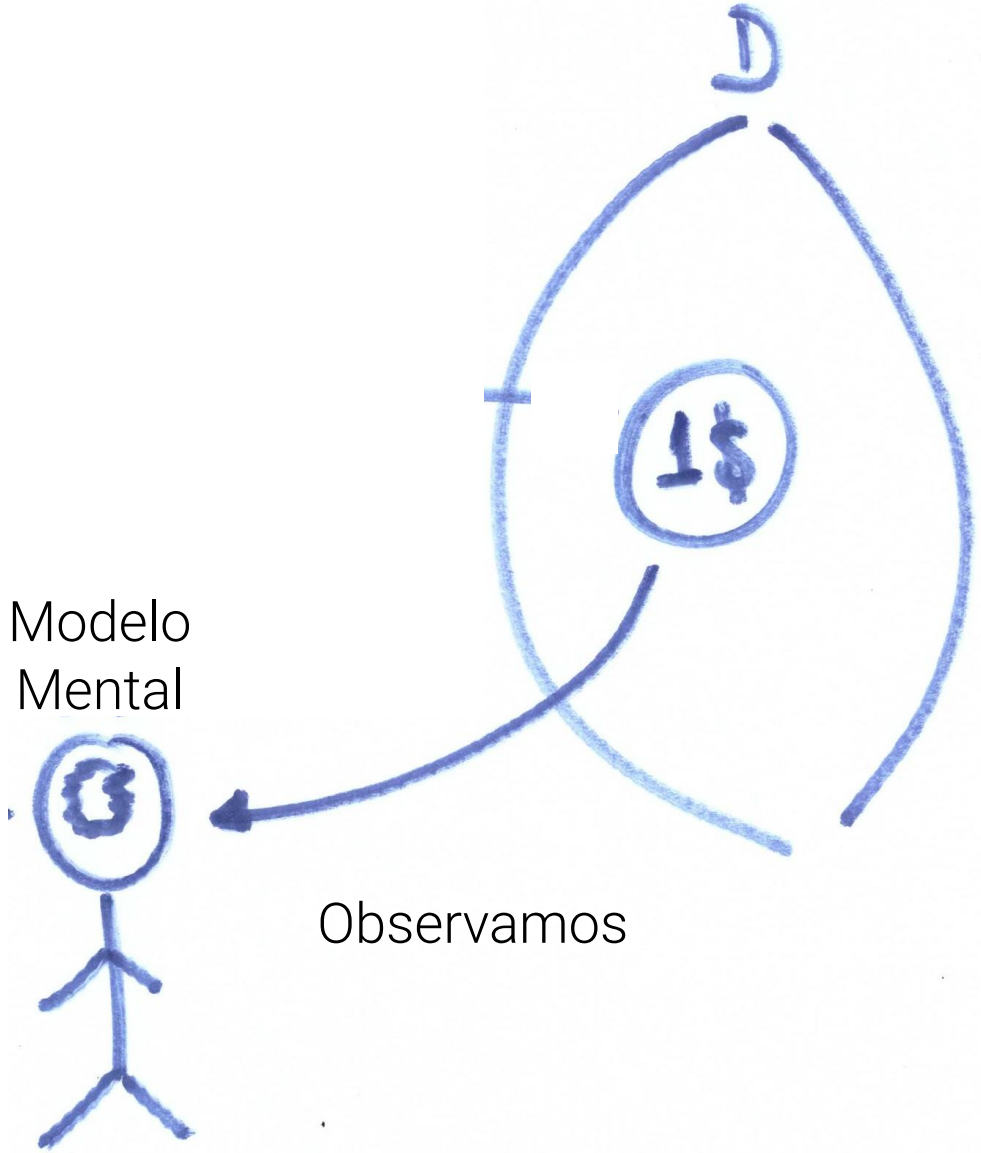
¿Cómo “creamos” ese Modelo?

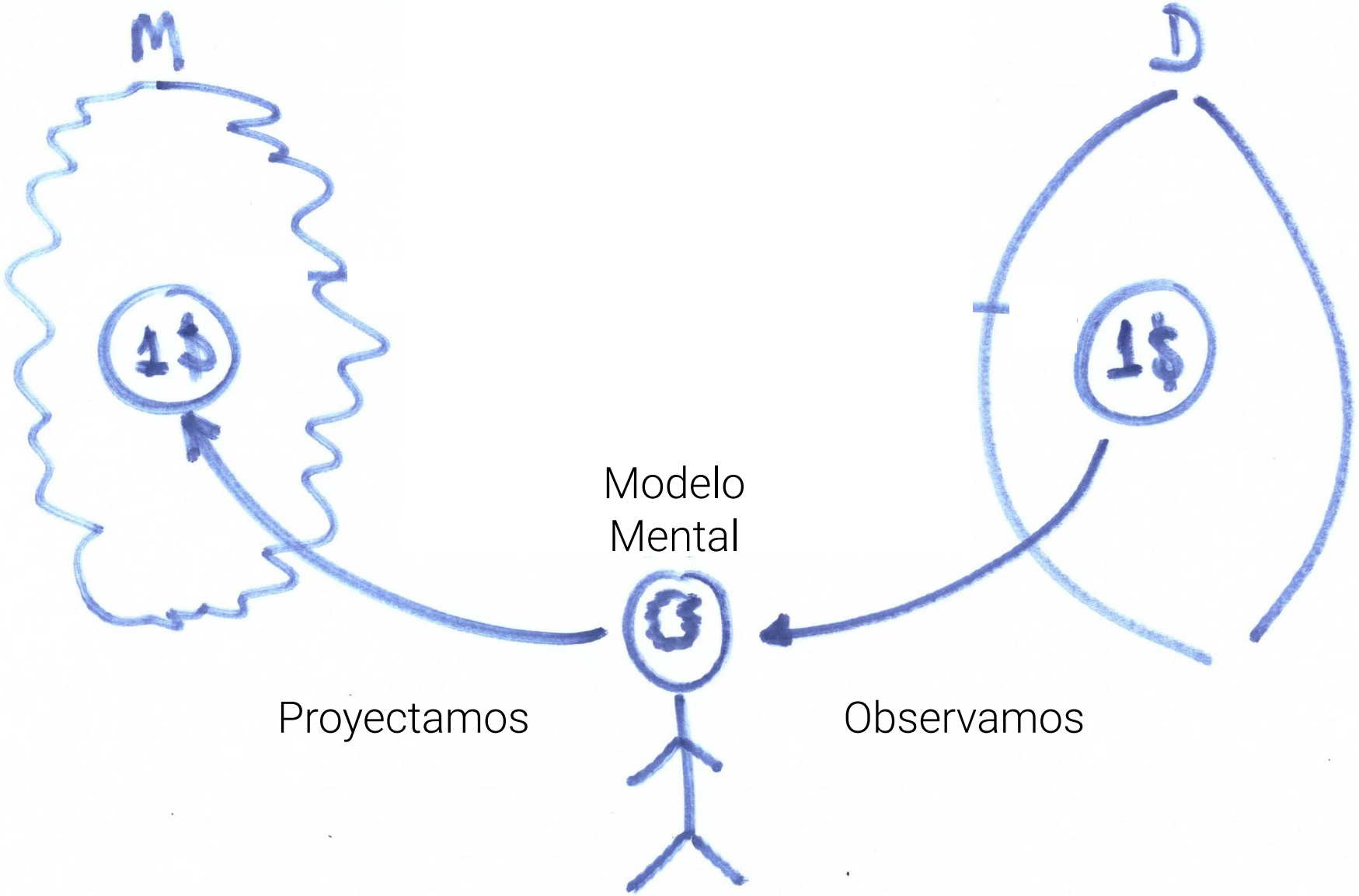
¿Cómo Desarrollamos Software?

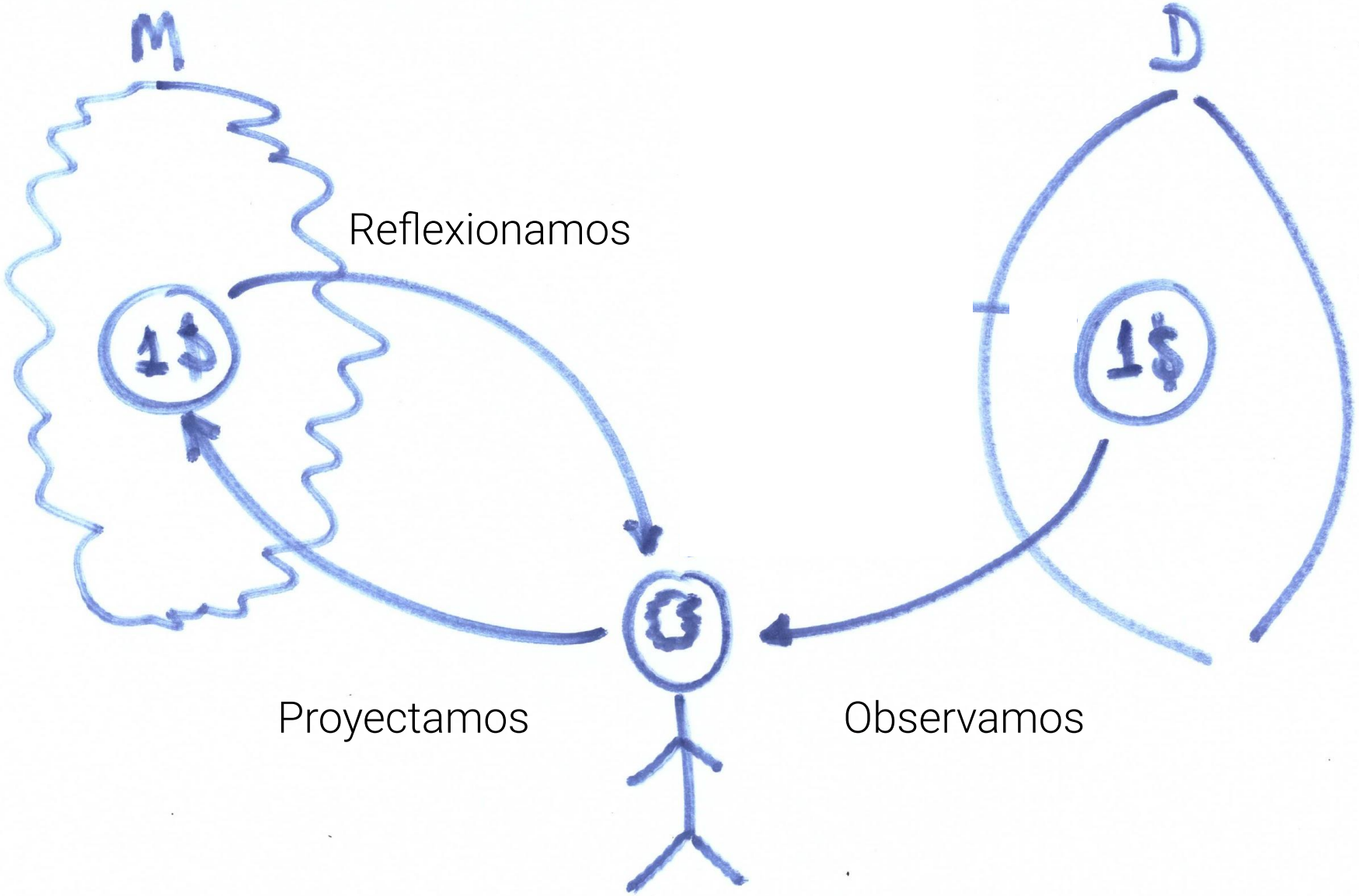

$T

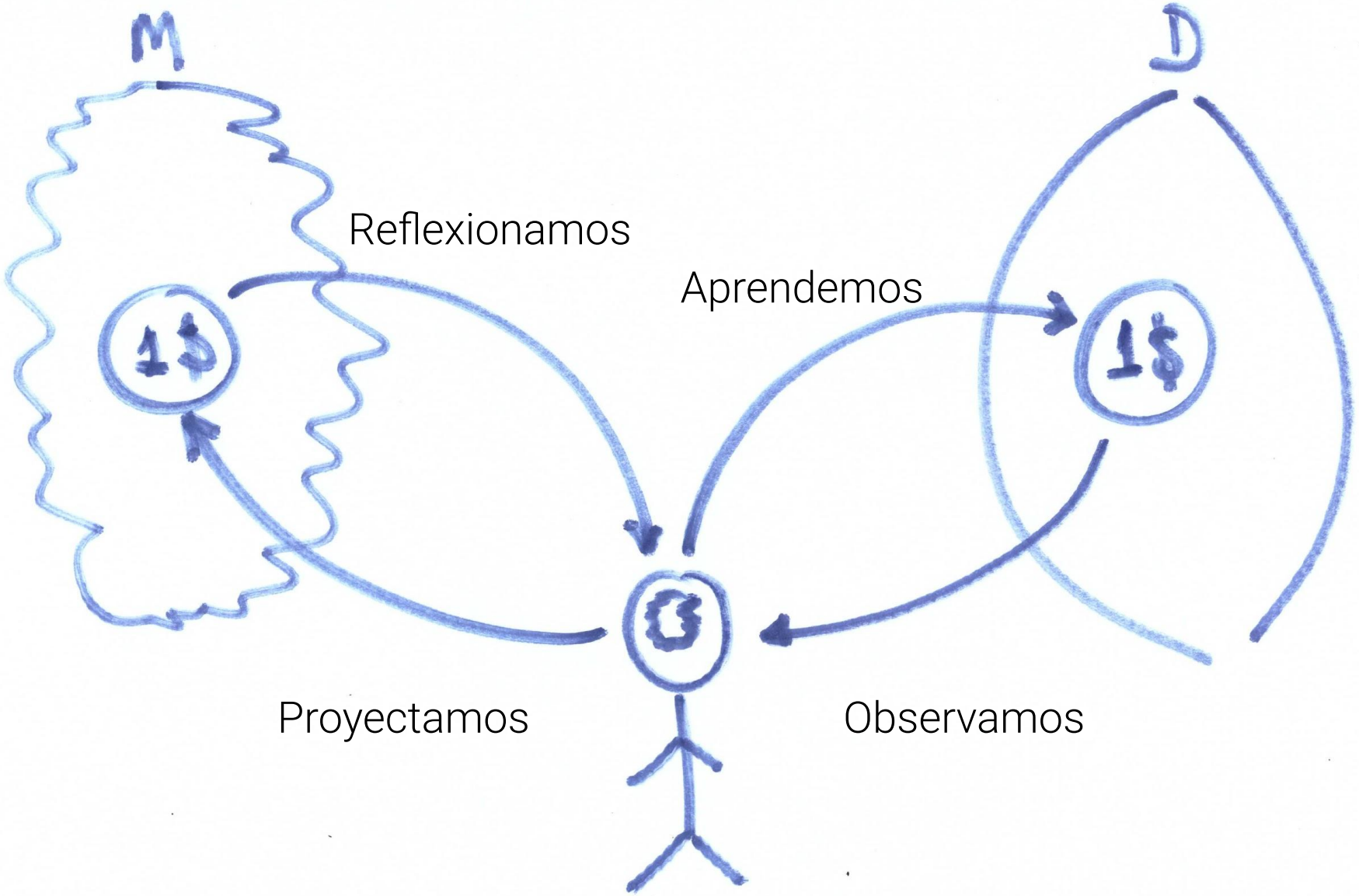


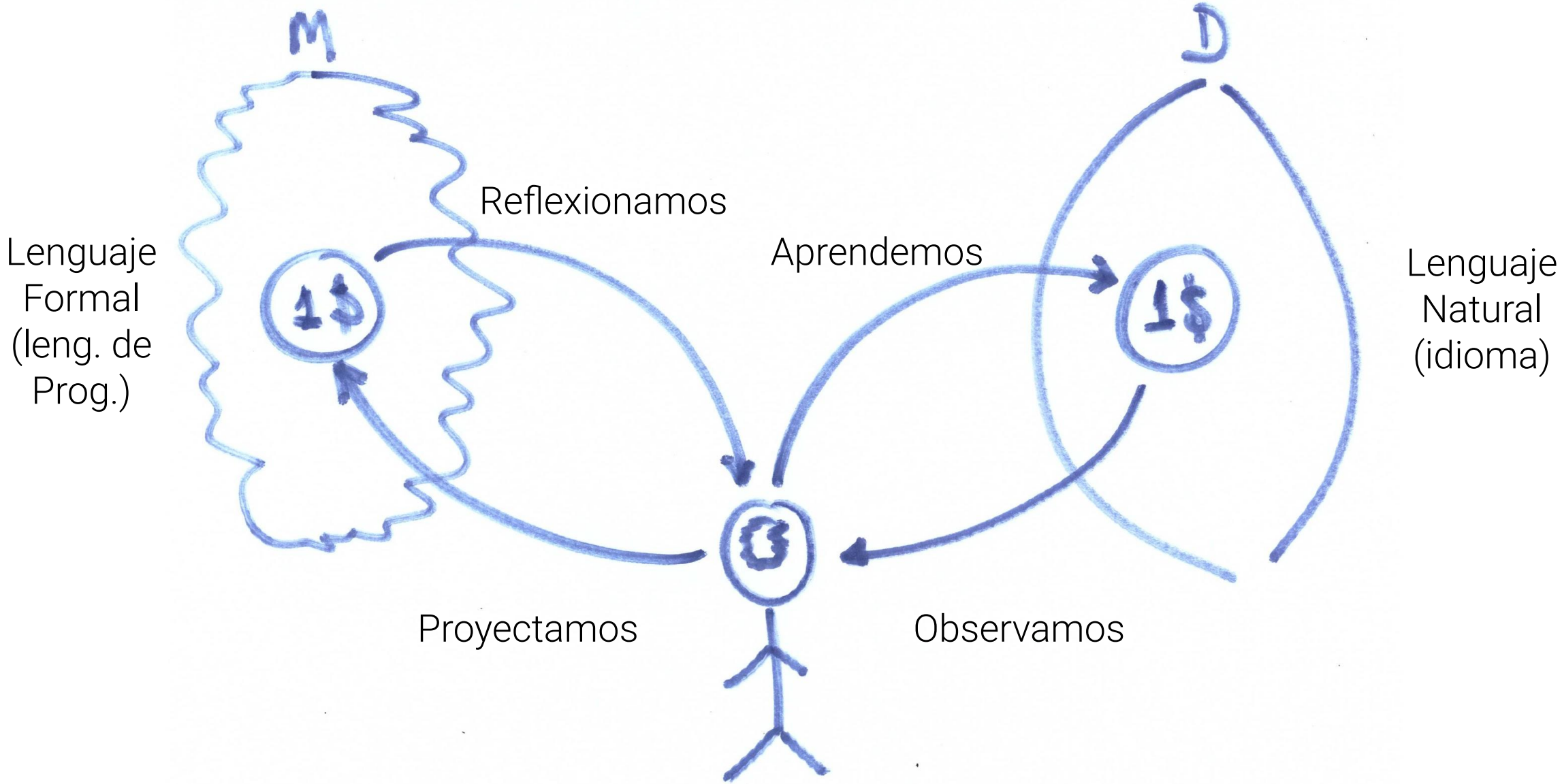
Observamos


Modelo
Mental
Observamos

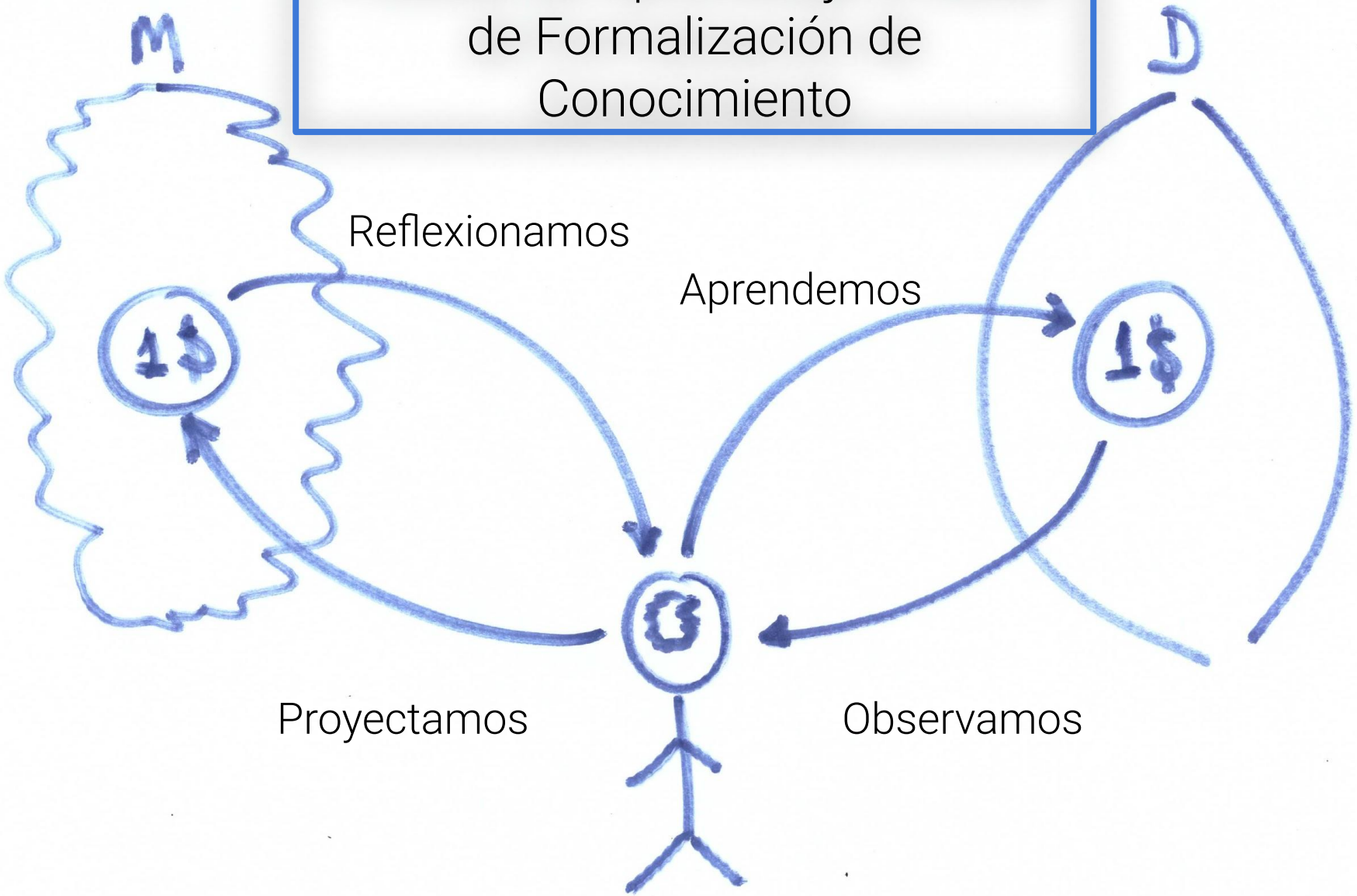

M
D
Modelo
Mental
Proyectamos
Observamos


M
D
Reflexionamos
Proyectamos
Observamos


M
D
Reflexionamos
Aprendemos
Proyectamos
Observamos


M
D
Lenguaje Formal (leng. de Prog.)
Reflexionamos
Aprendemos
Lenguaje Natural (idioma)
Proyectamos
Observamos

Proceso de Aprendizaje a través de Formalización de Conocimiento

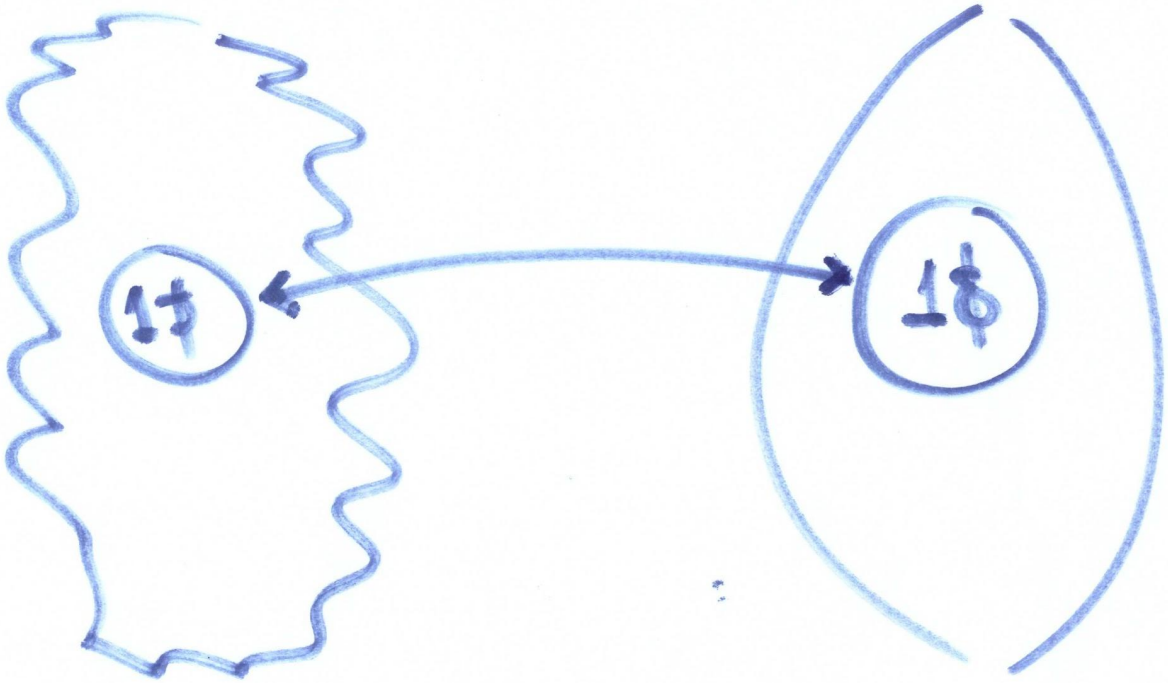


Lenguaje Formal (leng. de Prog.)

Lenguaje Natural (idioma)

¿Qué es un Objeto?

Representación esencial de un **ente** del Dominio de Problema



Esa representación la realiza a través de los **mensajes** que sabe **responder**

Diseñar con Objetos implica respondernos

- ¿Qué debo modelar?

  → ¿Qué entes de la realidad debo representar?

  ¿Los físicos? ¿Los abstractos? ¿Ambos?
- ¿Cómo lo debo modelar?

  → ¿Qué mensajes debe responder el objeto?

  → ¿A quién debe conocer el objeto?

  → ¿Desde cuándo el objeto debe representar al ente?

  → ¿Cómo debe el objeto “enseñar” qué representa?

Característica fundamental: Paso del Tiempo ...

System Browser: Object

Kernel-Objects
Kernel-Classes
Kernel-Magnitudes
Kernel-Numbers
Kernel-Text
Kernel-Chronology
Kernel-Methods

ProtoObject
Object
ActiveModel
Boolean
False
True

instance | ? | class

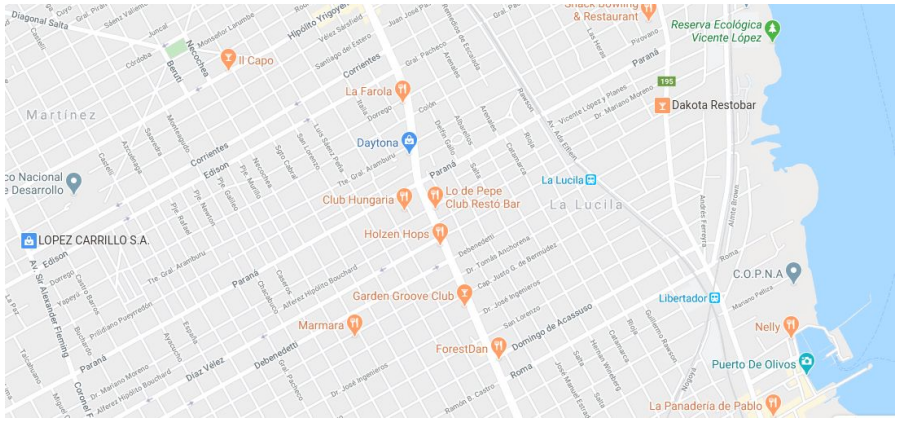
casing
class membership
comparing
converting
copying
events-old protocol
error handling

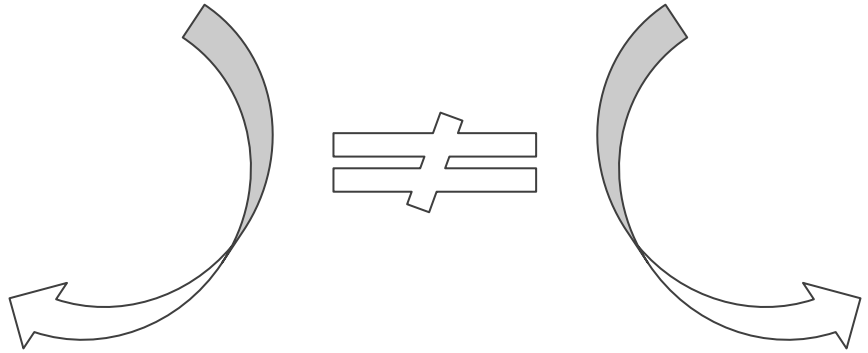
=
~=
hash
literalEqual:

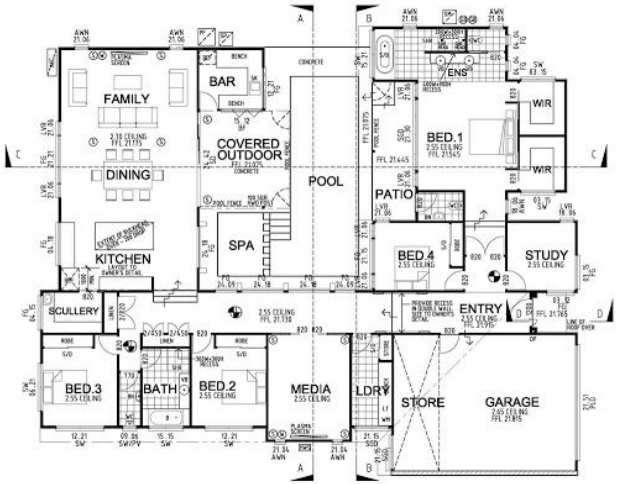
no timeStamp ° comparing ° part of base system (i.e. not in a package) ° in no change set °

browse | senders | implementors | versions | inheritance | hierarchy | inst vars | class vars | show...

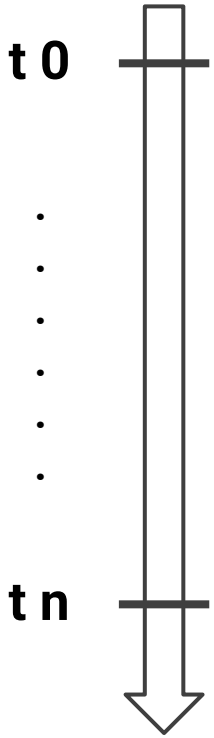
```
= anObject
    "Answer whether the receiver and the argument represent the same
    object. If = is redefined in any subclass, consider also redefining the
    message hash."

    ↑self == anObject
```







Es Dinámico - El tiempo transcurre

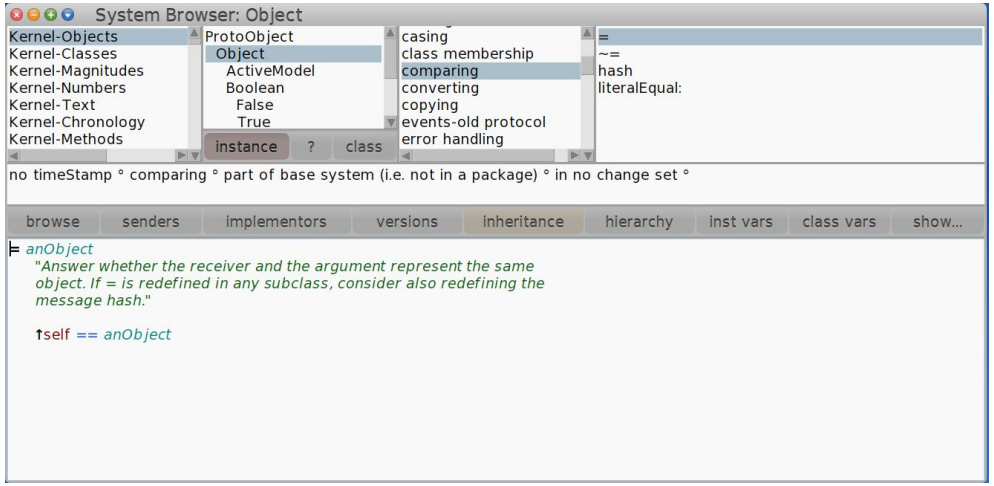



```
printOn: aStream
    "Append to the argument, aStream, a sequence of characters that
    identifies the receiver."

    | title |
    title ← self class name.
    aStream
        nextPutAll: title aOrAnPrefix;
        space;
        nextPutAll: title
```

Paso del Tiempo y Aprendizaje

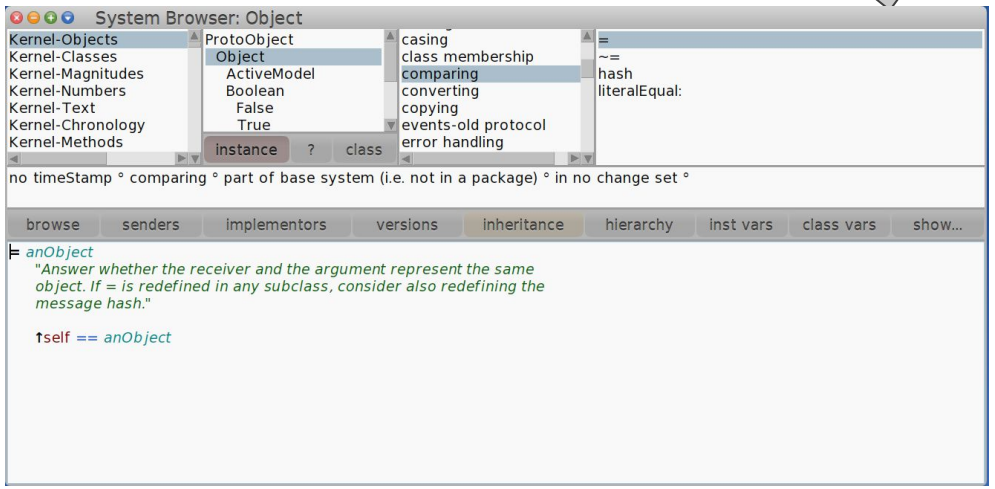
Para Entender

Diseñamos

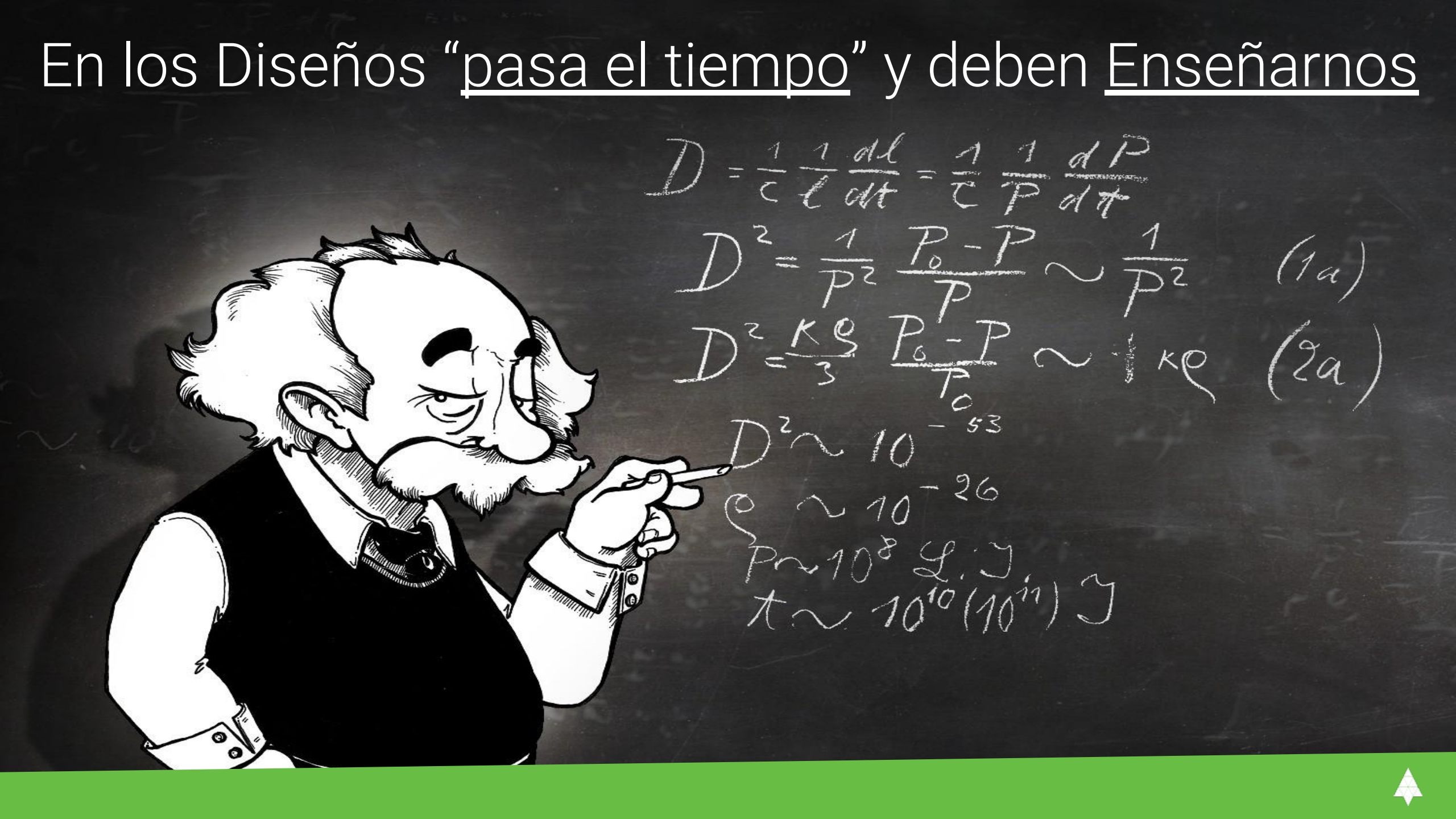


t0 . . . . . tn

Transmisión de Conocimiento



En los Diseños “pasa el tiempo” y deben Enseñarnos



Diseñar con Objetos implica respondernos

- ¿Qué debo modelar?

  → ¿Qué entes de la realidad debo representar?

  ¿Los físicos? ¿Los abstractos? ¿Ambos?

- ¿Cómo lo debo modelar?

  → ¿Qué mensajes debe responder el objeto?

  → ¿A quién debe conocer el objeto?

  → ¿Desde cuándo debe representar al ente?

  → ¿Cómo debe “enseñar” qué representa?

Ejemplo Time

Un Objeto debe **representar** el ente del dominio de problema **desde el momento en que existe**

Un Objeto debe "**enseñar**" de manera explícita **qué necesita para representar** ese ente

Heurística 1: Crear Objetos Completos

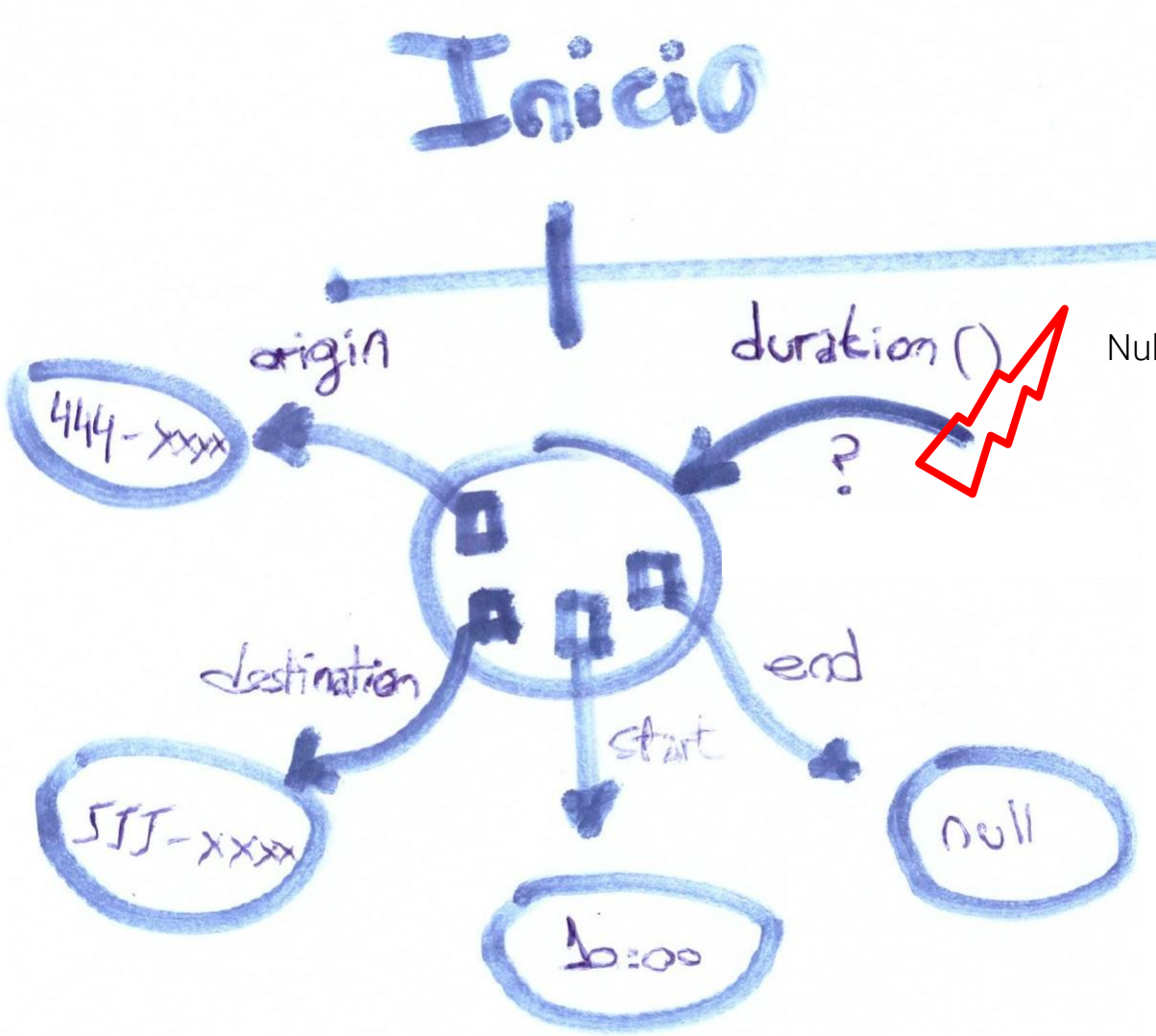
H2: Tener un único constructor/mensaje de construcción de instancia principal.
El resto debe estar implementado
en función de este

H3: ¡No romper el encapsulamiento!

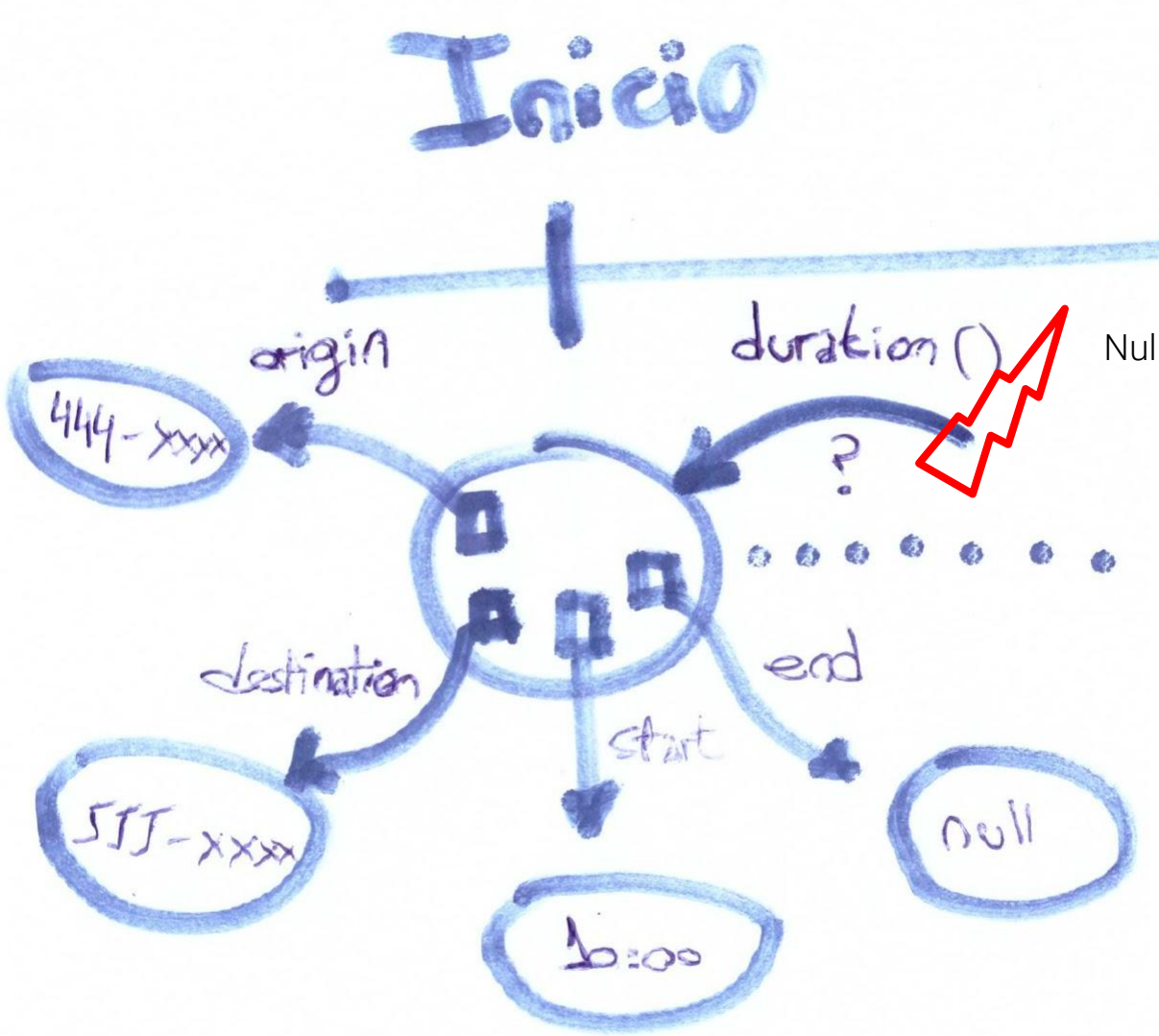
Bye bye clases anémicas 😉

Ejemplo PhoneCall

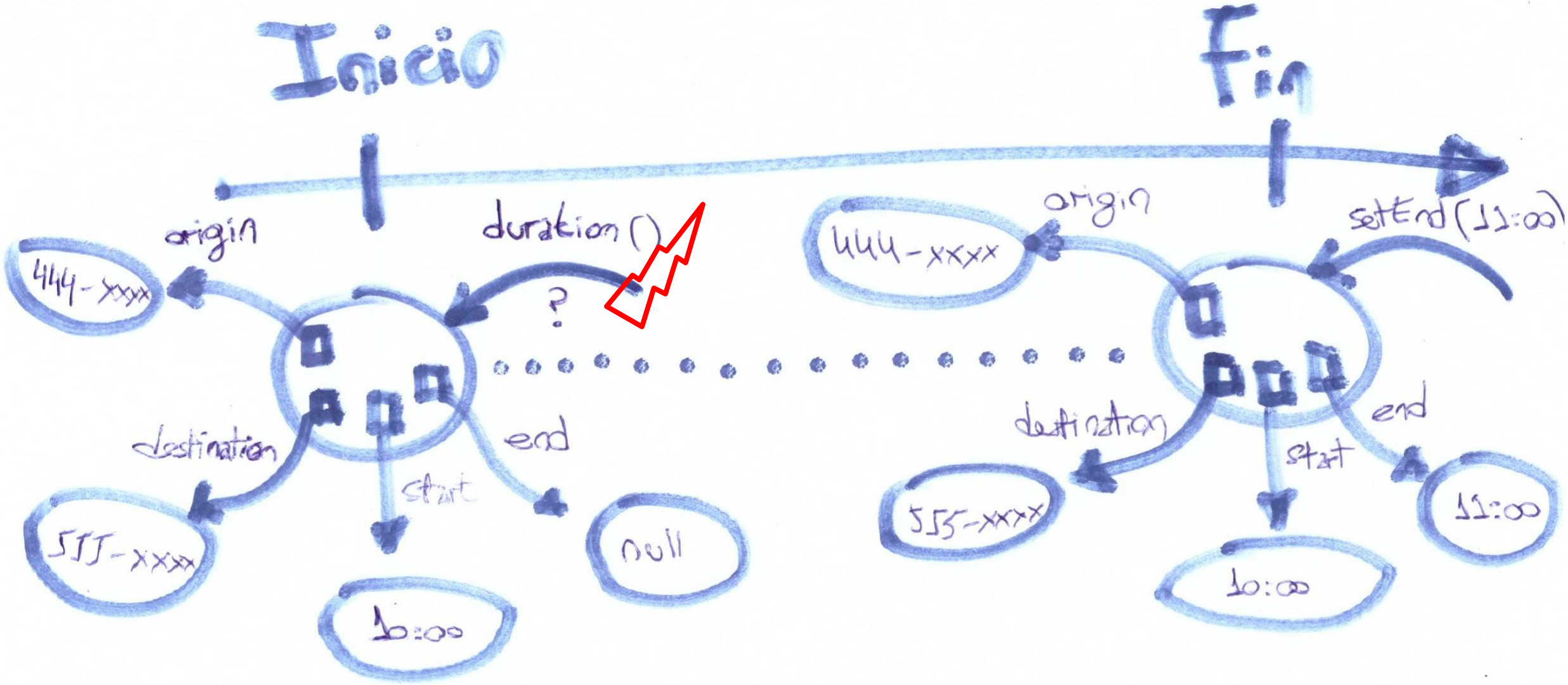

Inicio
Fin
origin
444-xxxx
destination
end
start
null
5JJ-xxxx
1o:oo

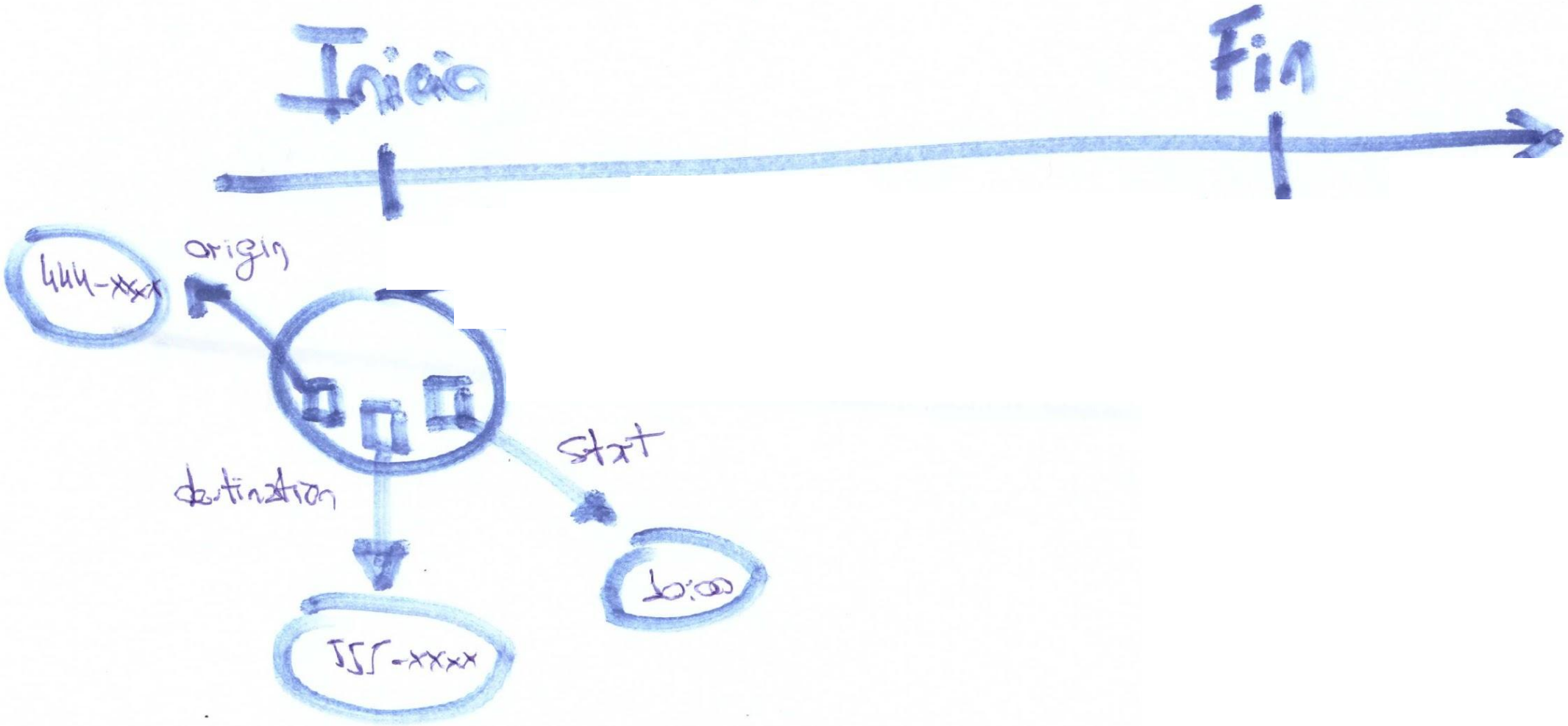


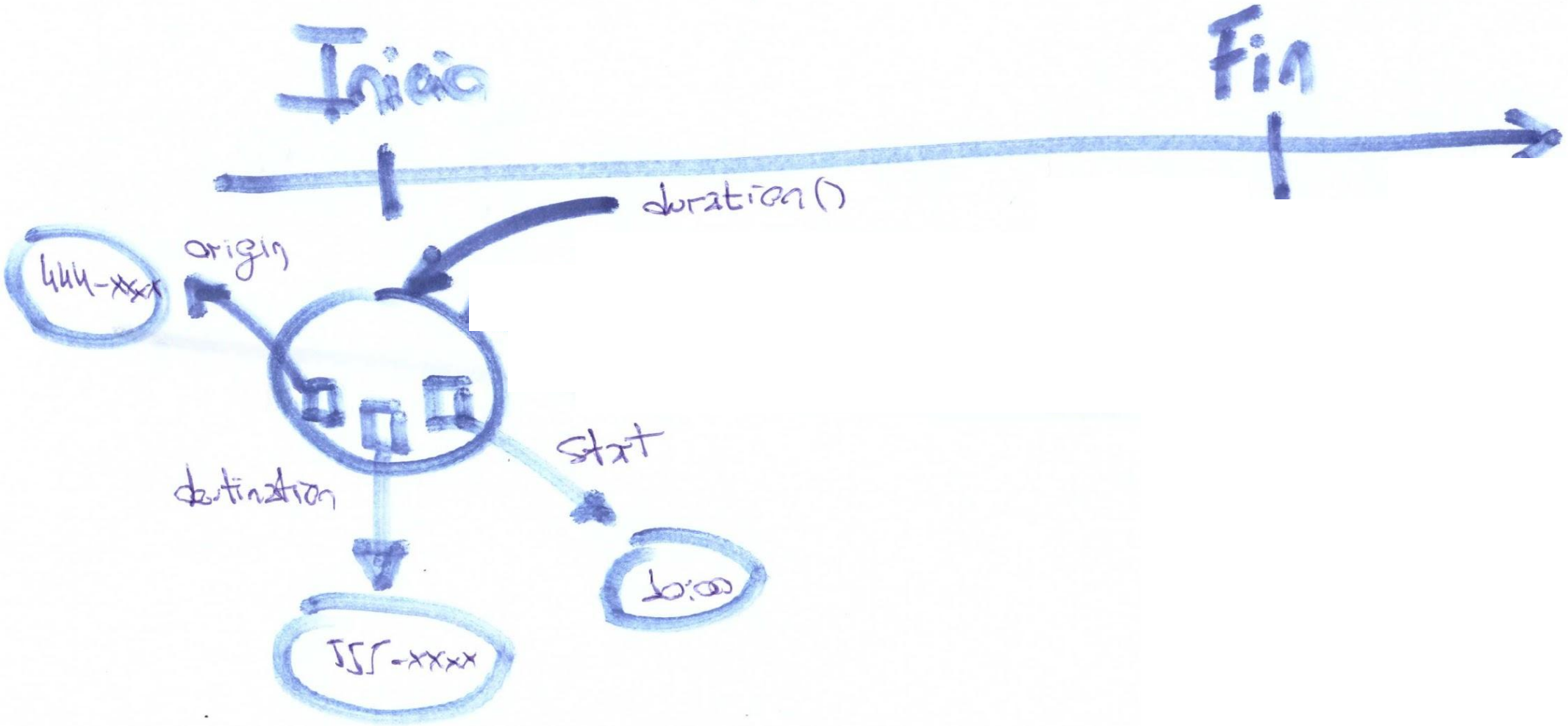
NullPointerException - doesNotUnderstand:

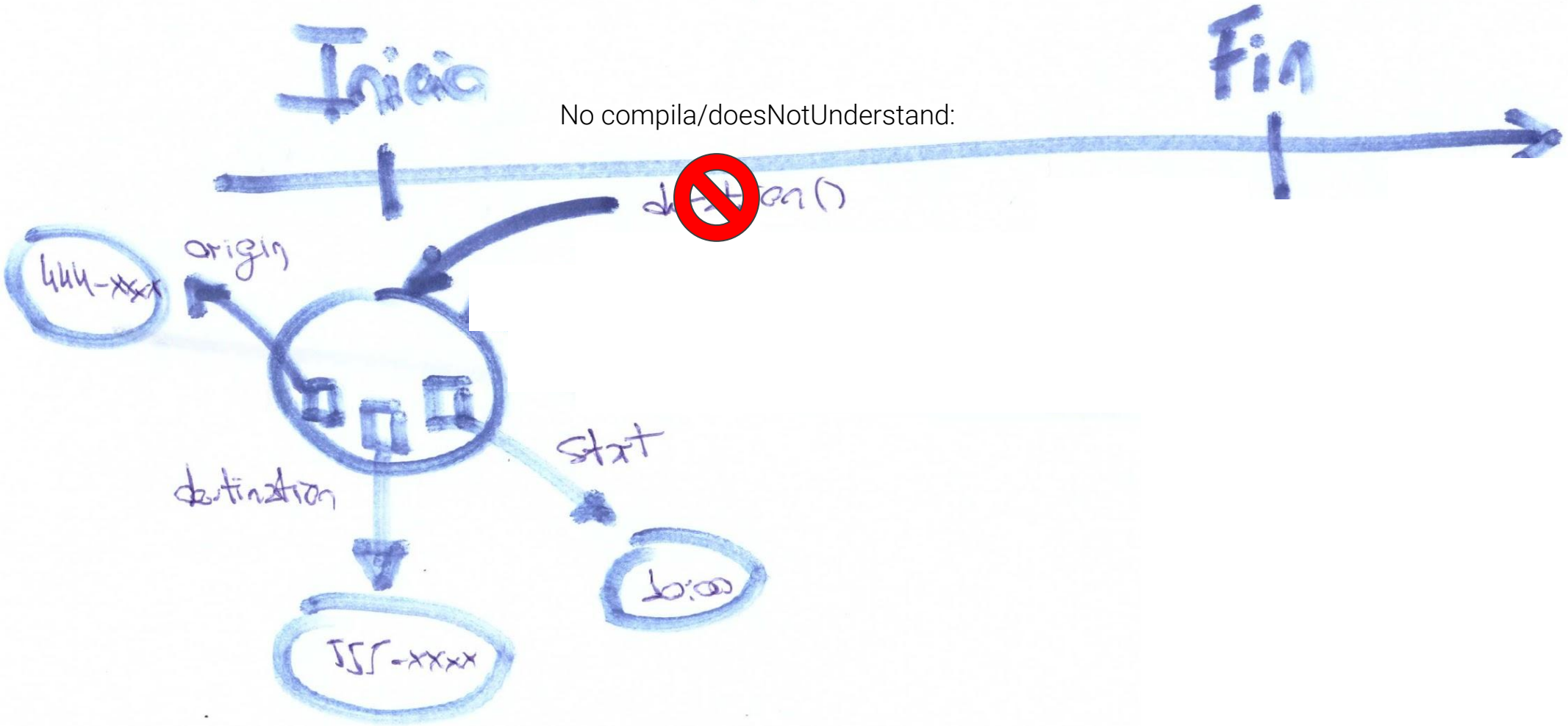


NullPointerException - doesNotUnderstand:

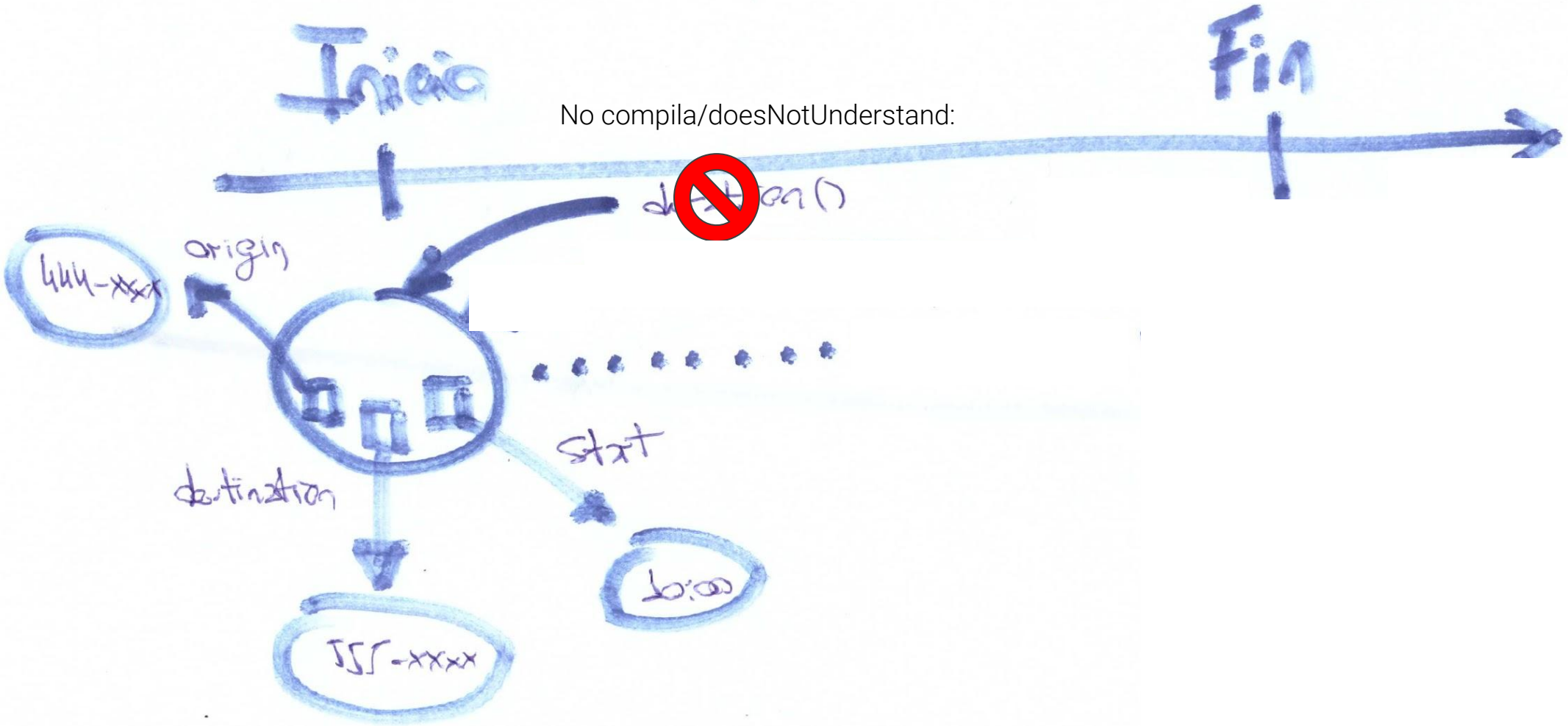

Inicio
Fin
origin
duration()
?
444-xxxx
destination
555-xxxx
end
null
start
10:00
origin
444-xxxx
setEnd(11:00)
destination
555-xxxx
start
10:00
end
11:00


Inicio
Fin
444-xxx
origin
start
destination
555-xxxx
bios

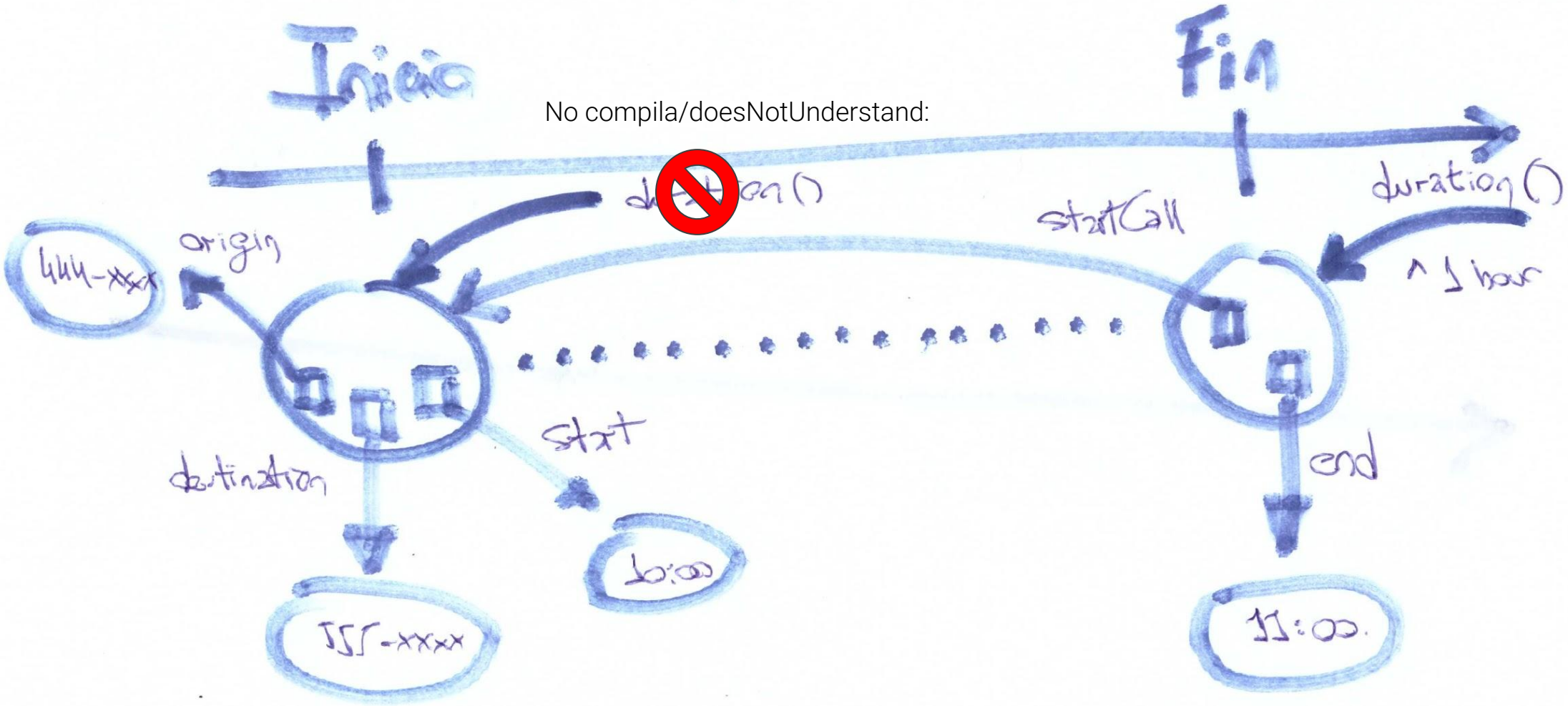

Inicio
Fin
duration()
origin
444-xxx
destination
start
555-xxxx
b:oo



No compila/doesNotUnderstand:



No compila/doesNotUnderstand:


Inicio
Fin
No compila/doesNotUnderstand:
duration()
startCall
duration()
origin
444-xxx
^ 1 hour
start
destination
end
b:00
555-xxxx
11:00.

H4: Si hay variables de instancias sin inicializar, eso indica que conviene separar en dos objetos

Ejemplo InvoiceForm & Invoice

H5: Si la construcción de un objeto es compleja/implica mucho “paso del tiempo”, utilizar un Builder

H6: Si un framework me obliga a tener constructores sin parámetros (ej. Hibernate) eso no significa que lo tengamos que usar

Conclusiones

- Hay que tener en cuenta **el paso del tiempo** en el **dominio** de problema
- Hay que tener en cuenta **el paso del tiempo** en los **diseños**
- Los diseños deben "**enseñar**" lo que representan

- Un Objeto debe **representar** el ente del dominio de problema **desde el momento en que existe**

- Un Objeto debe "**enseñar**" de manera explícita **qué necesita para representar** ese ente

Diseño ¡a la gorra!

¡Bienvenidos!
Durante esta serie de Webinars exploraremos *qué* significa **Diseñar Software con Objetos** y *cómo* lo podemos hacer cada vez mejor.
Trataremos muchos temas que irán desde cuestiones filosóficas como qué significa Diseñar en nuestra profesión y dónde está expresado ese Diseño, pasando por consejos y heurísticas para diseñar "mejor" y terminado con ejemplos concretos de cómo aplicar esas heurísticas en la *vida real.*
Los webinars son "*language agnostic*", o sea que no dependen de un lenguaje de programación en particular, aunque los ejemplos que usaremos estarán hechos principalmente en **Java**, **JavaScript**, **Ruby**, **Python** y mi querido **Smalltalk** cuando amerite 😉 .

Te esperamos todos los Martes a las 19 Hrs GMT-3 a partir del Martes 11 de Agosto de 2020.
Para poder participar tenes que registrate acá.

Todo el código y presentaciones estarán disponibles para que lo puedan usar y consultar en cualquier momento acá.

¡Trae ganas de aprender y pasarla bien!

¿Por qué a la Gorra?
Al igual que cuando Diseñamos Software está bueno usar una **Metáfora** para entender qué estamos modelando, en este caso usamos una metáfora para explicar cómo *financiaremos*

Donaciones

mercado pago

$100 - Casi una 🍷 Pagar

$250 - Una buena 🍺 Pagar

$500 - Menos que 🍔 +🍟 Pagar

PayPal Donate

¿Querés donar un monto variado o en otra plataforma?

Dictado por:
Hernán Wilkinson

https://alagorra.10pines.com

Muchas gracias

10 Pines

Creative Software Development

10pines.com

info@10pines.com

+54 (011) 6091-3125 / 4893-2057

Av. Leandro N. Alem 896 6° - Bs. As. - Argentina

@10pines