



10 Pines

Diseño a la Gorra - Episodio 17
Excepciones y Final



Hernán Wilkinson



hernan.wilkinson@10pines.com



@HernanWilkinson

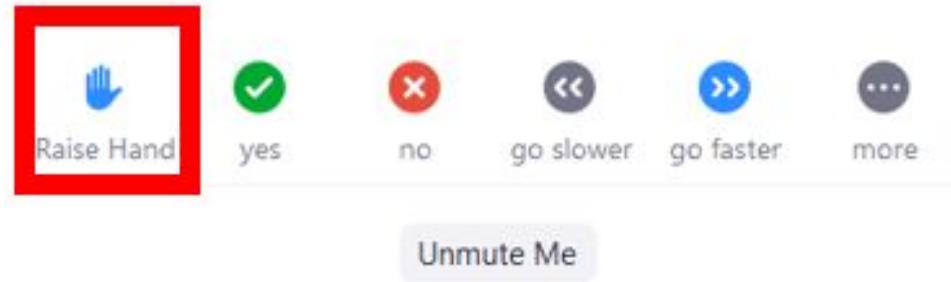


<https://alagorra.10pines.com>





Estar muteados a menos que sea necesario



No voy a poder leer el chat



La comunicación visual es importante. Usarla a discreción





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.

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 **REGISTRARTE ACA.**

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" para terminar 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 😊.

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!

Donaciones



\$250 - Una buena 🍺

Pagar

\$500 - Menos que 🍔 + 🍕

Pagar

\$1.000 - Dos buenos 🍷

Pagar



Donate



¿Querés donar un monto variado o en otra plataforma?

<https://alagorra.10pines.com>

Online

Construcción de Software Robusto con TDD

📅 Empieza el **18/01**

📅 Del 18/01 al 25/01. Días: Días Laborales del Lunes 18 al Lunes 25, de 9 a 13 hrs GMT-3.

💰 **AR\$ 22.500-** (IVA incluido)

Hacer una consulta

Inscribirme ahora **10%** Dto.



Dictado por:
Máximo Prieto

~~AR\$ 25.000-~~

AR\$ 22.500-

(IVA incluido)

~~U\$D 330-~~

U\$D300-

(impuestos incluidos)

🔥 **Early Bird ¡10% menos!**



¡GRACIAS POR TANTO!





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.

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 [REGISTRARTE ACA.](#)

Trataremos muchos temas que irán desde cuestiones filosóficas como *qué* significa Diseñar en nuestra profesión y dónde está expresado ese Diseño, hasta por consejos y heurísticas para diseñar "mejor" para terminar con ejemplos concretos de cómo aplicar esas heurísticas en la *vida real*.

Los webinars son "*language agnostic*", cosa 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 😊.

Todo el código y presentaciones estarán disponibles para que lo puedan usar y consultar en cualquier momento [acá](#).

¡Tráete ganas de aprender y pasarla bien!

Donaciones



\$250 - Una buena 🍺

Pagar

\$500 - Menos que 🍔 + 🍕

Pagar

\$1.000 - Dos buenos 🍷

Pagar



Donate



¿Querés donar un monto variado o en otra plataforma?

<https://alagorra.10pines.com>

¿Por qué?



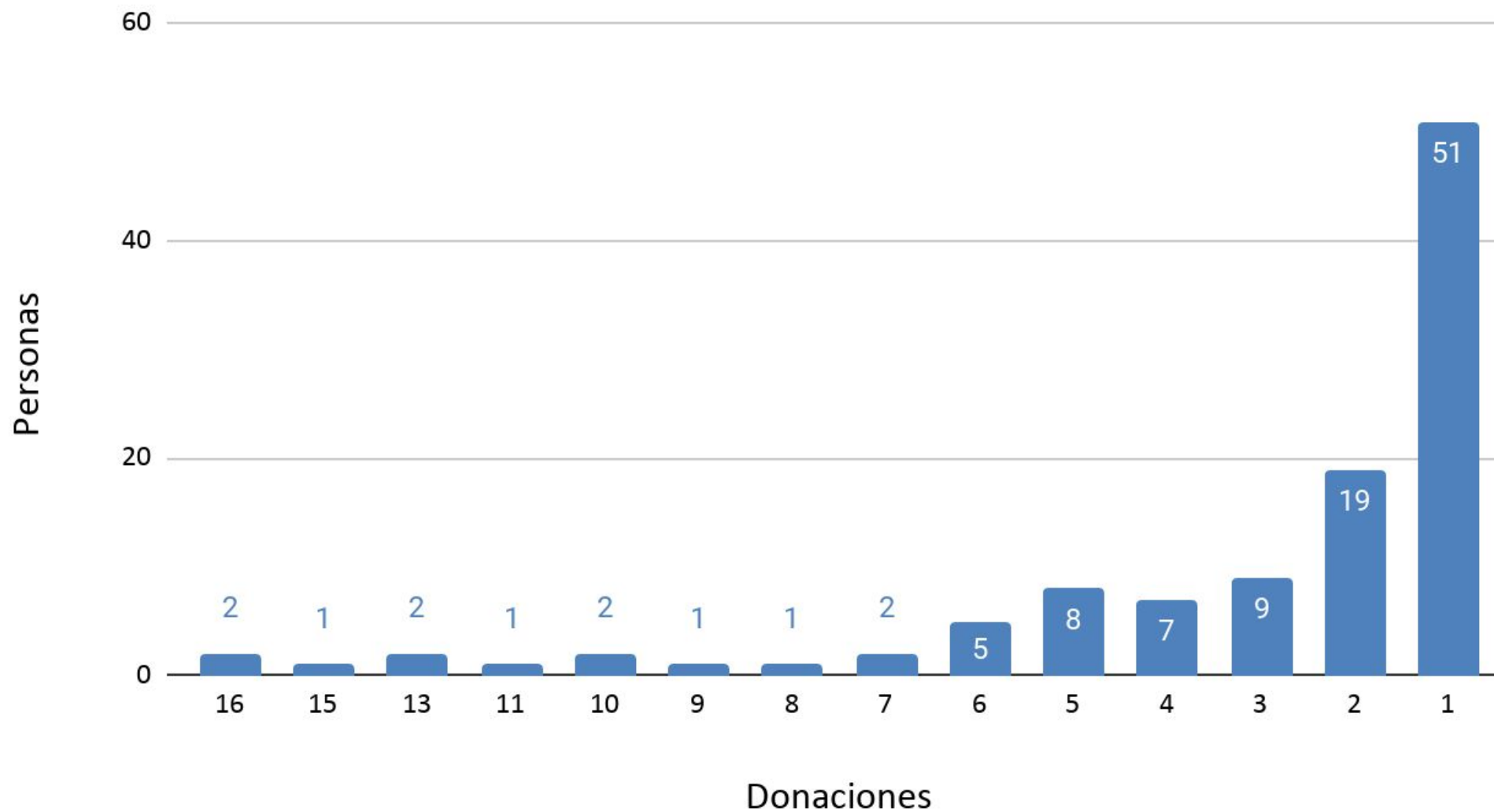
- Perdón por no haberlo anunciado antes
- El “modelo comercial” tiene un bug :-)
 - El público que participa no se renueva
- Ingresos/Egresos = 0.45
 - Tenemos claro que el dinero no es todo
- Fuerte desgaste semanal, es como salir a un escenario con un papel nuevo cada semana



Algunas Estadísticas



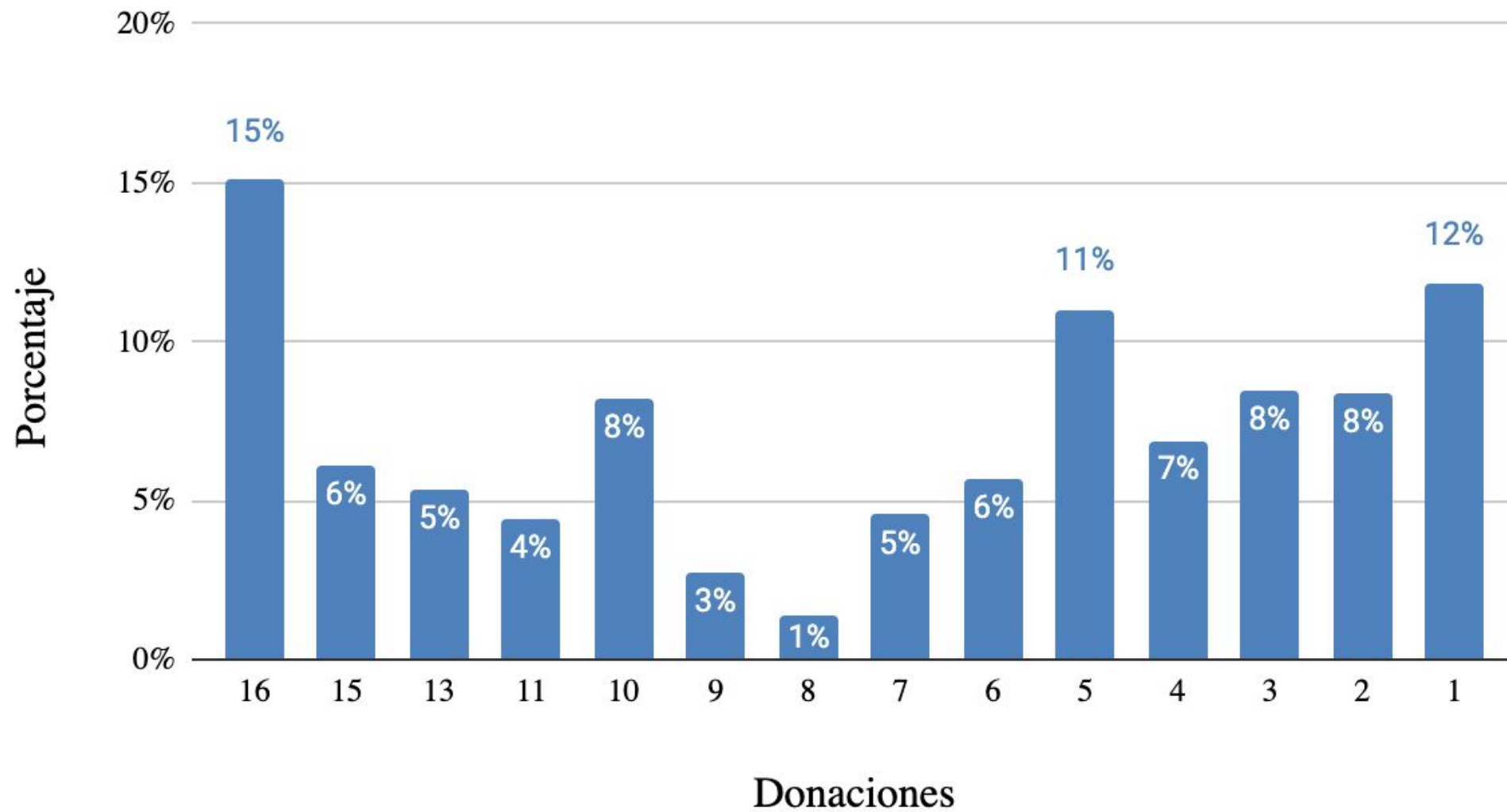
Donaciones x Personas



**Cantidad de
Personas
111**



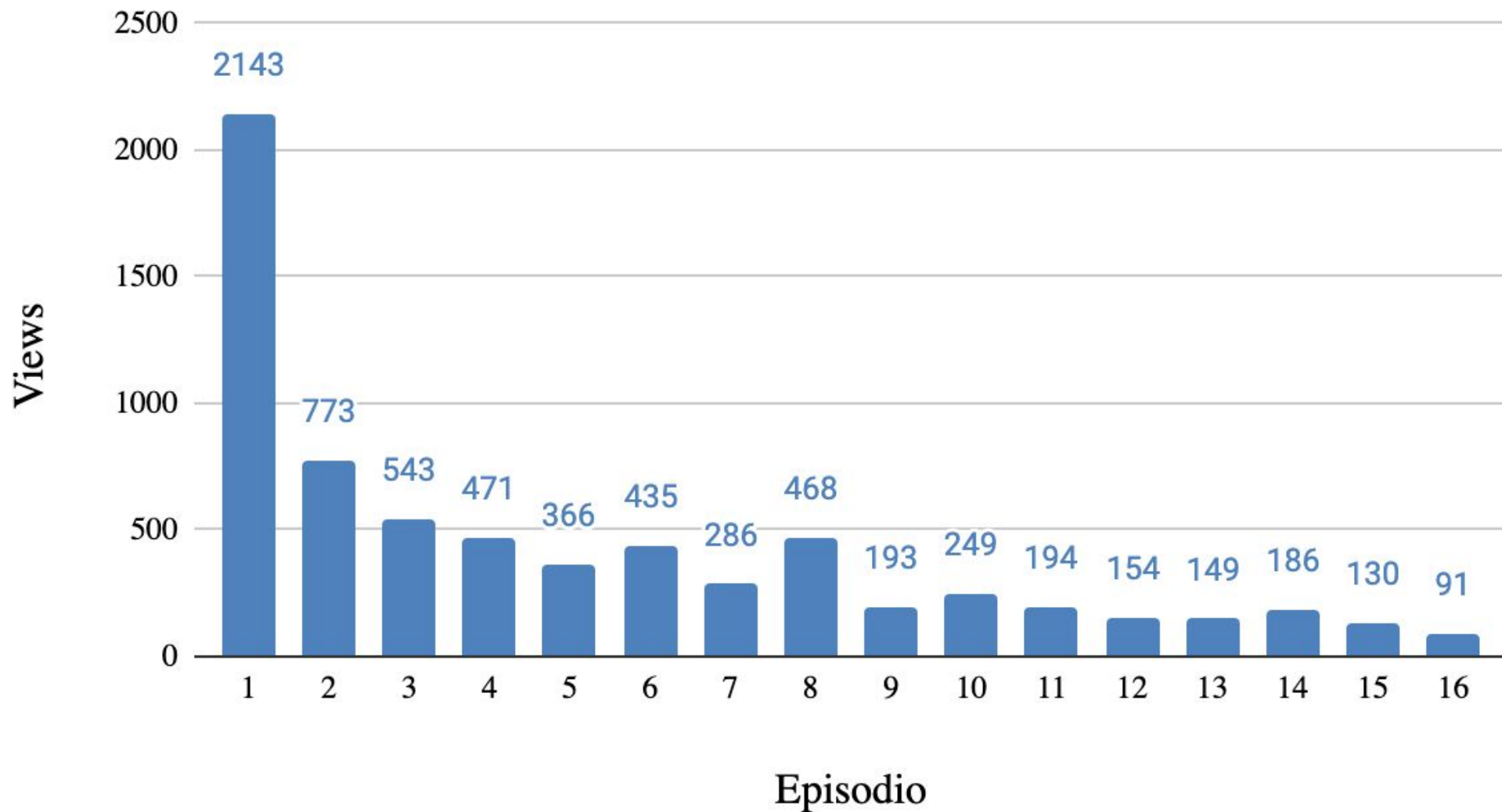
Porcentaje x Donaciones



8 personas: 39%
17 personas: 25%
86 personas: 35%



Views por Episodio



Total Views
6831



todo tiene un final ... pero
siempre aparecen cosas nuevas 🤔
(más al final...)



Ahora si...
¡Excepciones!



¿De dónde vienen?

- Explicación Pragmática
- Explicación Conceptual



Explicación Pragmática

- Sacar código repetido de la técnica de “código de retorno”



Técnica de Código de Retorno

```
public Object m1 () {  
    ...  
    resultado = objeto.m2();  
    if (resultado.isError()) return resultado;  
  
    resultado = objeto.m3();  
    if (resultado.isError()) return resultado;  
  
    resultado = objeto.m4();  
    if (resultado.isError()) return resultado;  
    ...  
}
```



Técnica de Código de Retorno

Qué se quiere
hacer

```
public Object m1 () {  
    ...  
    resultado = objeto.m2();  
    if (resultado.isError()) return resultado;  
    resultado = objeto.m3();  
    if (resultado.isError()) return resultado;  
    resultado = objeto.m4();  
    if (resultado.isError()) return resultado;  
    ...  
}
```



Técnica de Código de Retorno

Se verifica si
hubo error

```
public Object m1 () {  
    ...  
    resultado = objeto.m2();  
    if (resultado.isError()) return resultado;  
    resultado = objeto.m3();  
    if (resultado.isError()) return resultado;  
    resultado = objeto.m4();  
    if (resultado.isError()) return resultado;  
    ...  
}
```



Técnica de Código de Retorno

```
public Object m1 () {
```

```
...
```

```
resultado = objeto.m2();
```

```
if (resultado.isError()) return resultado;
```

```
resultado = objeto.m3();
```

```
if (resultado.isError()) return resultado;
```

```
resultado = objeto.m4();
```

```
if (resultado.isError()) return resultado;
```

```
...
```

```
}
```

Se “handlea”
el error



Problemas

- Código Repetido → Problema de diseño
- Propenso a error
 - Nos podemos olvidar de poner el `if isError...`
- Difícil de leer qué se quiere hacer por estar entremezclado el código de adm. de error
- No está estandarizado (salvo en Go)



Sacar Código Repetido de la Técnica de Código de Retorno

```
public Object m1 () {  
    ...  
    resultado = objeto.m2();  
    if (resultado.isError()) return resultado;  
  
    resultado = objeto.m3();  
    if (resultado.isError()) return resultado;  
  
    resultado = objeto.m4();  
    if (resultado.isError()) return resultado;  
    ...  
}
```



```
public Object m1 () {  
    INTENTAR {  
        objeto.m2();  
        objeto.m3();  
        objeto.m4();  
    } isError() {  
        return resultado;  
    }  
}
```



Excepciones en lenguajes con sintaxis tipo C

```
try {  
    doSomething();  
} catch (RuntimeException e) {  
    handle(e);  
}
```



Excepciones en lenguajes con sintaxis tipo C

```
try {  
    doSomething();  
} catch (RuntimeException e) {  
    handle(e);  
}
```

Condición de Handleo.
Acoplamiento con tipo de
Excepción



Excepciones con Objetos y Mensajes

```
[ self doSomething ]  
  on: Error  
  do: [ :anError | self handle: anError ]
```



Excepciones con Objetos y Mensajes

```
try {  
    doSomething();  
} catch (RuntimeException e) {  
    handle(e);  
}
```

```
[ self doSomething ]  
on: Error  
do: [ :anError | self handle: anError ]
```



Excepciones con Objetos y Mensajes

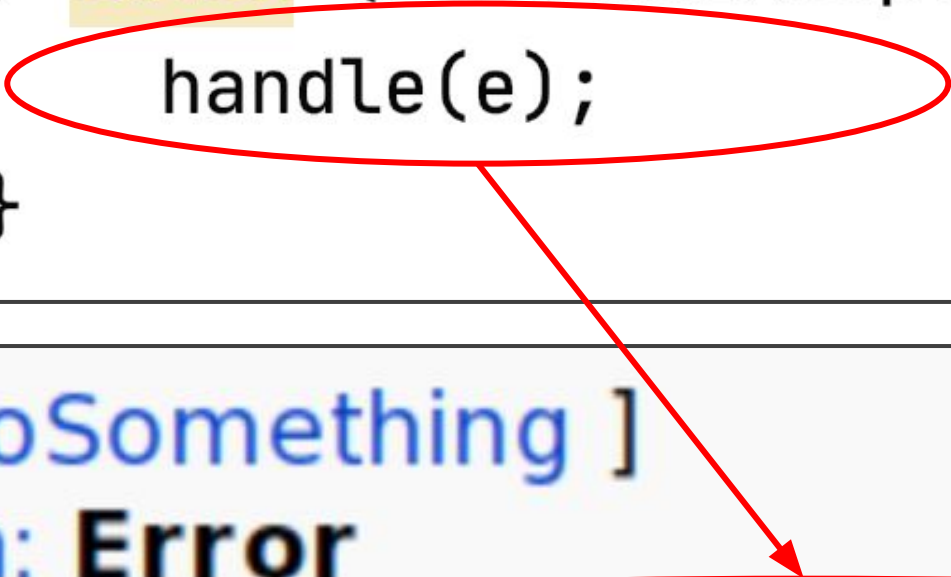
```
try {  
    doSomething();  
} catch (RuntimeException e) {  
    handle(e);  
}
```

```
[ self doSomething ]  
on: Error  
do: [ :anError | self handle: anError ]
```



Excepciones con Objetos y Mensajes

```
try {  
    doSomething();  
} catch (RuntimeException e) {  
    handle(e);  
}
```



```
[ self doSomething ]  
on: Error  
do: [ :anError | self handle: anError ]
```



Explicación Conceptual

- Programación por Contratos - Bertrand Meyer
- Contratos explícitos
- Contratos implícitos
- Romper el Contrato → Excepción



Definición de Contratos en Objetos

- Pre-Condiciones
- Post-Condiciones
- Invariantes



Pre-Condiciones

- Condiciones que se deben mantener para ejecutar un método
- Ej: Monto a extraer de una caja de ahorro debe ser ≥ 0
- ¿Quién debe verificarlas? → más adelante



Post-Condicionales

- Condiciones que se deben mantener después de ejecutar un método
- Ej. en una extracción:
saldo = prev(saldo) - montoAExtraer
- Ej. en un algoritmo de Sorting:
La colección está ordenada
- Cuando se verifican:
 - Soporte explícito del lenguaje (Ej. Eiffel)
 - Tests



Invariantes

- Condiciones que siempre se deben mantener en las instancias de una clase
- Ej: El saldo de una cuenta bancaria siempre debe ser ≥ 0
- Cuándo se verifican:
 - Soporte explícito del lenguaje (Ej. Eiffel)
 - Tests



Uso de Excepciones

- ¿Quién debe verificar que el contrato se cumpla?
- ¿Quién generalmente la informarlas?
- ¿Quién principalmente debe handlearlas?
- ¿Cómo se puede handlearlas?
- ¿Qué excepción informar?



¿Quién debe verificar que el contrato se cumpla?

- Escuela C:
El objeto que envía el mensaje/función llamadora debe asegurar las pre-condiciones
- Escuela Lisp:
El objeto que recibe el mensaje/función llamada debe asegurar la pre-condiciones



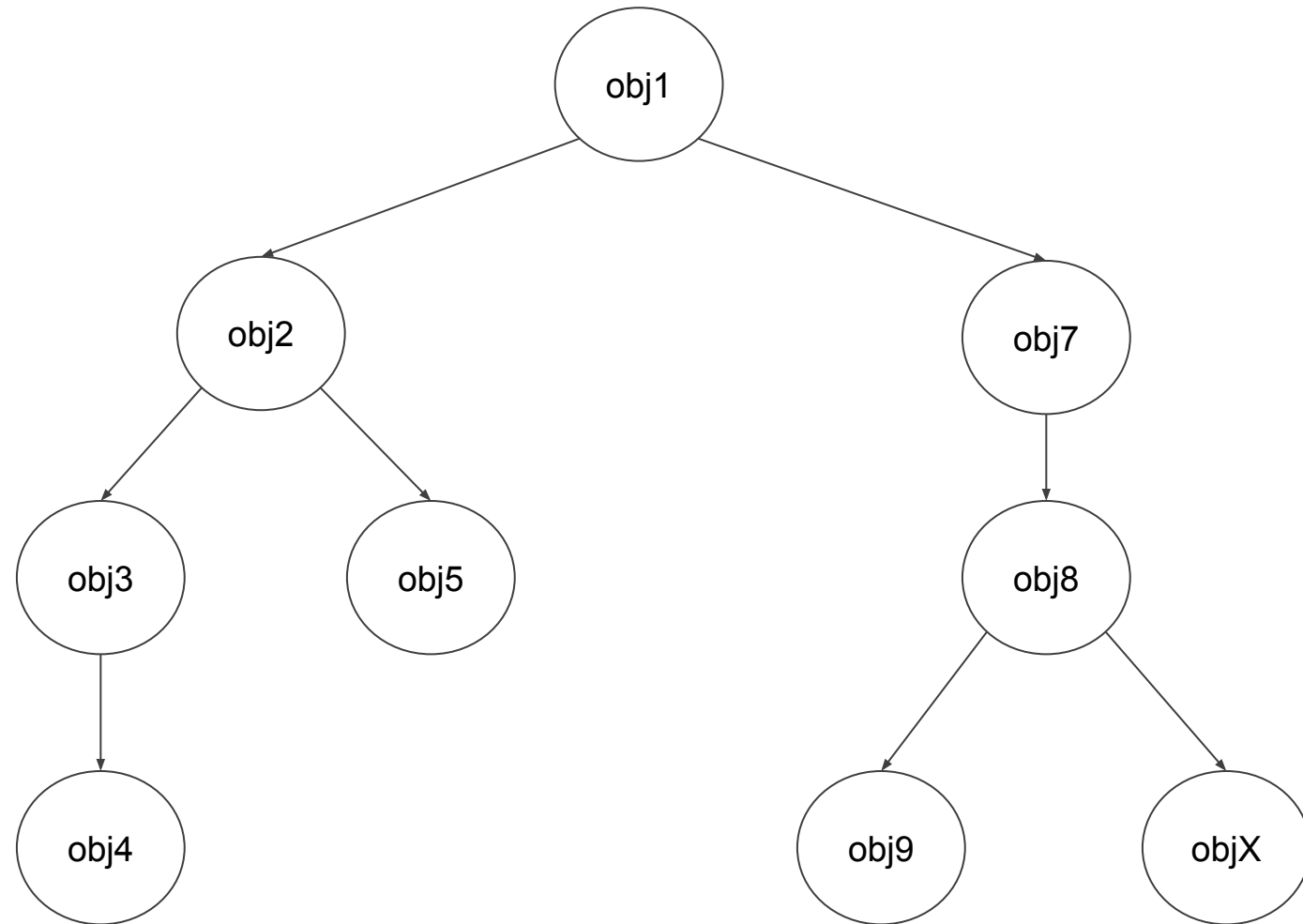
¿Quién debe verificar que el contrato se cumpla?

- Escuela C:
El objeto que envía el mensaje/función llamadora debe asegurar las pre-condiciones
- Escuela Lisp:
El objeto que recibe el mensaje/función llamada debe asegurar la pre-condiciones

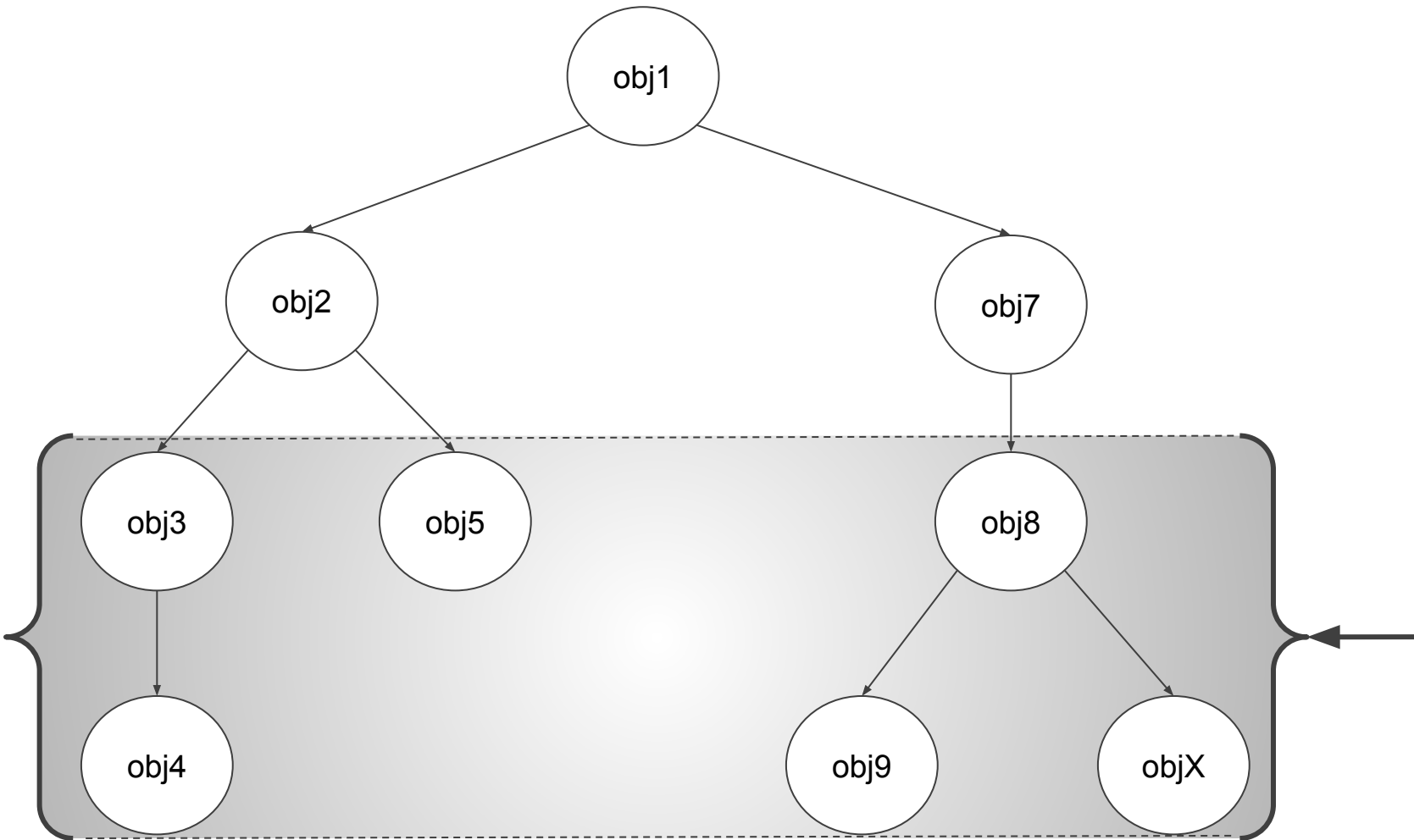


¿Quién generalmente las informa?

Árbol de Ejecución



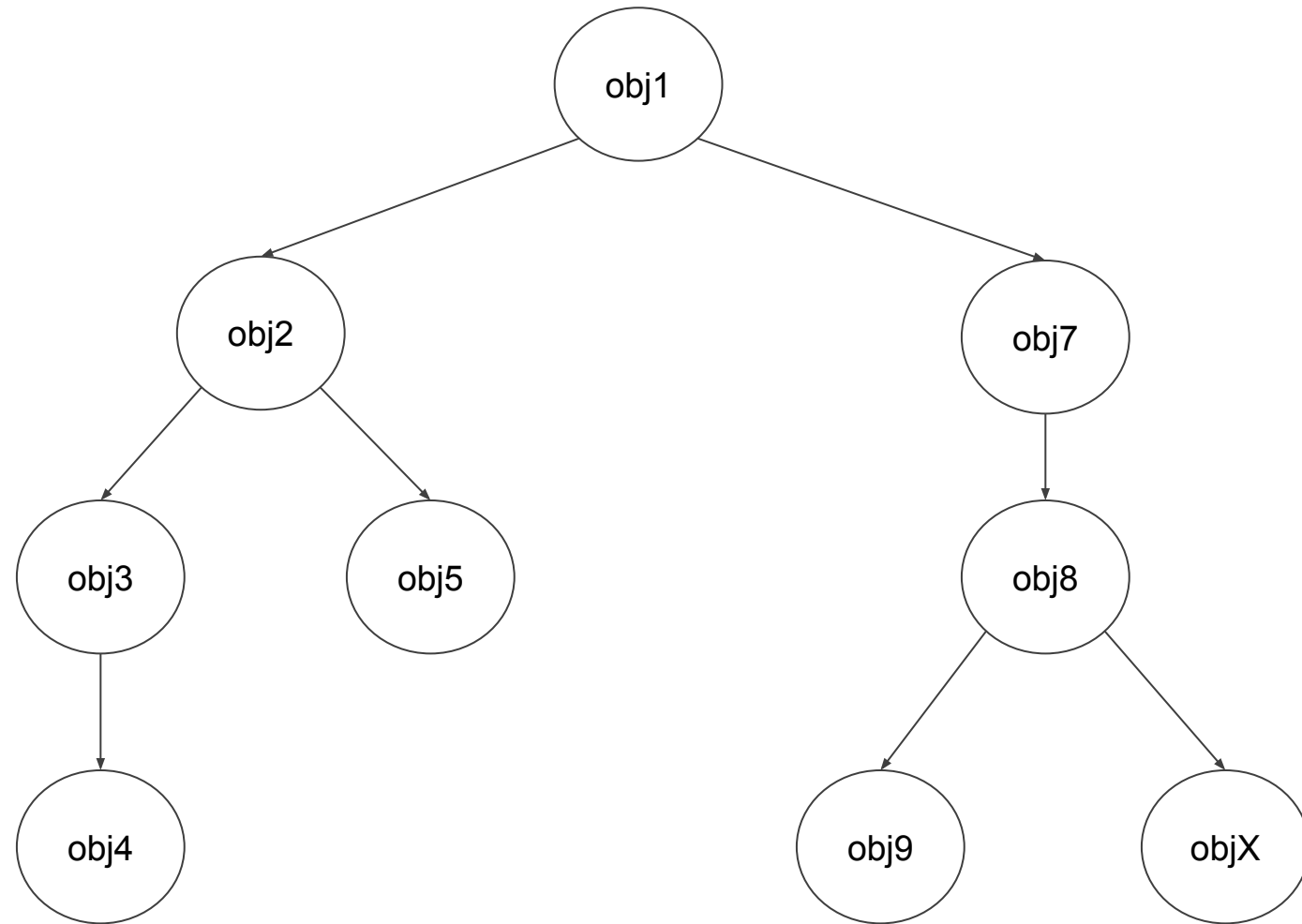
¿Quién generalmente las informa?



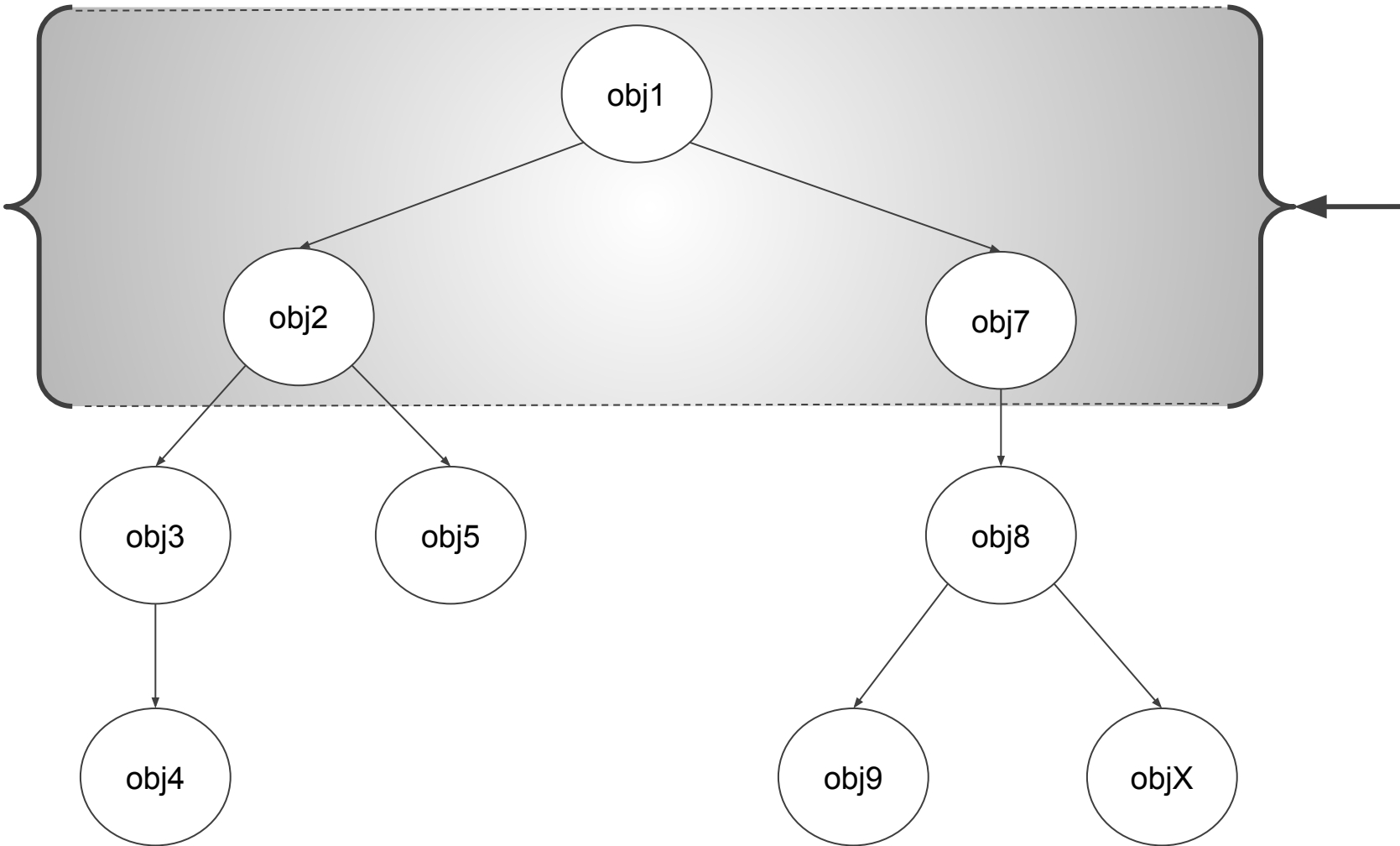
Los objetos que se encuentran más abajo en el árbol de ejecución porque son los que “realmente hacen algo”



¿Quién debe handlearlas?



¿Quién debe handlearlas?



Los objetos que se encuentran más arriba en el árbol de ejecución ya que tienen más contexto de qué se está haciendo y por lo tanto puede decidir mejor qué hacer



Cómo se puede handlearlas

- En implementaciones “cerradas”:
 - Terminar el bloque donde se generó la excepción (bloque del try)
 - Pasar la excepción al siguiente handler
- En implementaciones “abiertas”:
 - Terminar el bloque donde se generó la excepción
 - Pasar la excepción al siguiente handler
 - Reintentar el bloque que generó la excepción
 - Continuar con la siguiente colaboración



Qué excepción informar

- Un tipo de excepción por cada condición
Ej: *IndexOutOfBoundsException*, *InvalidBalance*, *InvalidName*, etc.
- La misma excepción siempre, no importa la condición de error
Ej: Usar siempre *RuntimeException* (Java), *Error* (Smalltalk)
- Un mix



Qué excepción informar

- Lo que determina la necesidad de existir de un tipo de excepción es si ***se la handlea o no***
- Esto se debe al acoplamiento que existe en la condición de handleo (catch/on:)
- Solo crear nuevos tipos de excepciones si se los va a handlear
 - Depende del tipo de software under dev.
- Ampliar la condición de handler
 - ej. CuisSmalltalk



Implementación

- Cerrada, en el runtime del lenguaje:
 - Java, C#, Python, Ruby, etc.
- Abierta, usando el mismo lenguaje:
 - Smalltalk, CommonLisp, Self
- Ventajas de Abierta:
 - Se puede ver la implementación
→ se puede aprender de ella
 - Se puede ampliar las prestaciones
→ Ej. CuisSmalltalk
 - No se pierde el contexto donde fue levantada la excepción
→ Se puede hacer mejor debugging



Bibliografía

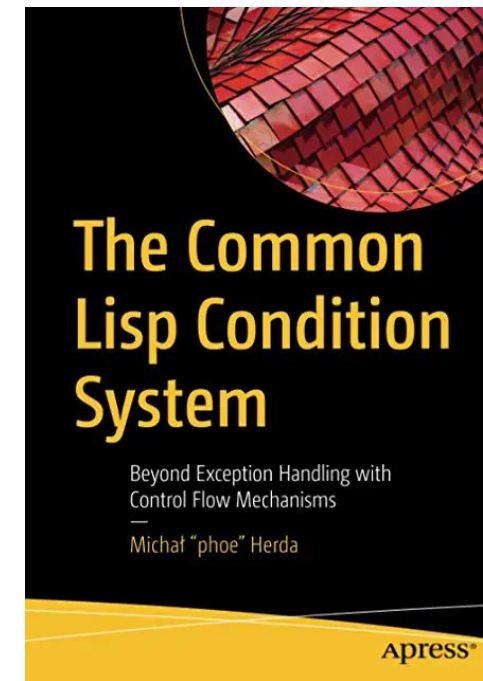
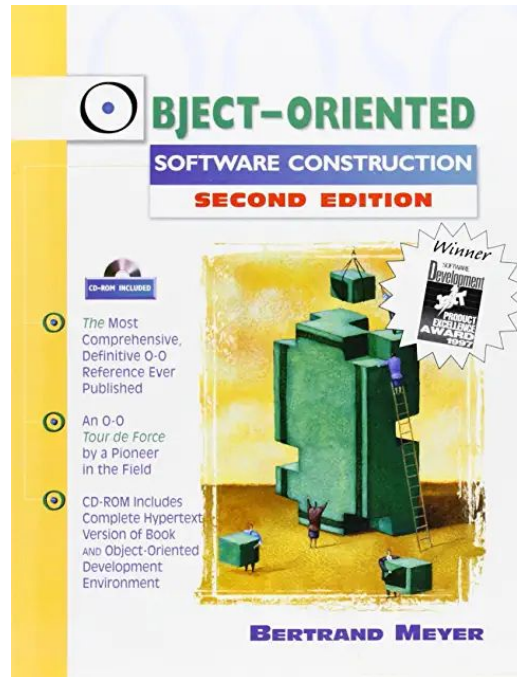
- Aprendiendo Smalltalk en época de Cuarentena, episodio de Excepciones:
<https://youtu.be/7wP-AuinuQk>
- Implementando Excepciones en Ruby:
https://youtube.com/playlist?list=PLMkq_h36PcLA4yY58tQgj5FAXRzMaZAaY



Bibliografía

- **Christophe Dony:**

- A Fully Object-Oriented Exception Handling System: Rationale and Smalltalk Implementation
- Exception Handling and Object-Oriented Programming: towards a synthesis
- Improving exception handling with Object-Oriented Programming



Menti.com → 20 63 00 3



Canal de Slack: #diseño-a-la-gorra
<https://tinyurl.com/slack-disenio-gorra>



Video de Despedida



¡A brindar!



iFin!



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