Test Driven Development

¿Qué es TDD?

- Técnica de Aprendizaje
  - Iterativa e Incremental
  - Basada en Feedback Inmediato
- Como side-effect:
  - Recuerda todo lo aprendido
  - Y permite asegurarnos de no haber "desaprendido"
- Incluye análisis, diseño, programación y testing

¿Cómo se hace TDD?

1) Escribir un test
   - Debe ser el más sencillo que se nos ocurra
   - Debe fallar al correrlo

¿Cómo se hace TDD?

1) Escribir un test
   - Debe ser el más sencillo que se nos ocurra
   - Debe fallar al correrlo
2) Correr todos los tests
   - Implementar la solución más simple que haga pasar el/los test/s
   - GOTO 2 hasta que "todos los tests" pasen

¿Cómo se hace TDD?

1) Escribir un test
   - Debe ser el más sencillo que se nos ocurra
   - Debe fallar al correrlo
2) Correr todos los tests
   - Implementar la solución más simple que haga pasar el/los test/s
   - GOTO 2 hasta que "todos los tests" pasen
3) Reflexiono - ¿Se puede mejorar el código?
   - Sí -> Refactorizar. GOTO 2
   - No -> GOTO 1

¿Cómo se hace TDD?

1) Escribir un test
   - Debe ser el más sencillo que se nos ocurra
   - Debe fallar al correrlo
2) Correr todos los tests
   - Implementar la solución más simple que haga pasar el/los test/s
   - GOTO 2 hasta que "todos los tests" pasen
3) Reflexiono - ¿Se puede mejorar el código?
   - Sí -> Refactorizar. GOTO 2
   - No -> GOTO 1

¿Cómo se hace TDD?

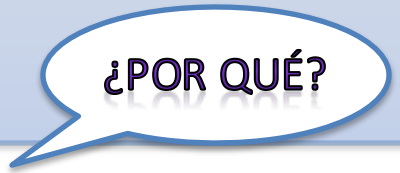
1) Escribir un test
   - Debe ser el [illegible]cillo que se nos ocurra
   - Debe fallar al correrlo

¿POR QUÉ?

2) Correr todos los tests
   - Implementar la solución más simple que haga pasar el/los test/s
   - GOTO 2 hasta que "todos los tests" pasen
3) Reflexiono - ¿Se puede mejorar el código?
   - Sí -> Refactorizar. GOTO 2
   - No -> GOTO 1

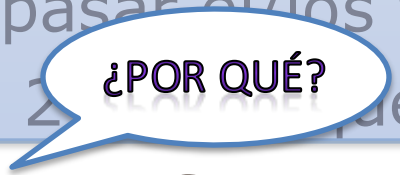
¿Cómo se hace TDD?

1) Escribir un test
   - Debe ser el más sencillo que se nos ocurra
   - Debe fallar al correrlo
2) Correr todos los tests
   - Implementar la solución más simple que haga pasar el/los test/s
   - GOTO 2 hasta que "todos los tests" pasen
3) Reflexiono - ¿Se puede mejorar el código?
   - Sí -> Refactorizar. GOTO 2
   - No -> GOTO 1



¿Cómo se hace TDD?

1) Escribir un test
   - Debe ser el más sencillo que se nos ocurra
   - Debe fallar al correrlo
2) Correr todos los tests
   - Implementar la solución más simple que haga pasar el/los test/s
   - GOTO 2 hasta que "todos los tests" pasen



3) Reflexiono - ¿Se puede mejorar el código?
   - Sí -> Refactorizar. GOTO 2
   - No -> GOTO 1

Framework de Testing: xUnit

- Implementado por Kent Beck
- Framework de caja blanca pero simple
- Los casos de test se implementan en métodos, ej: testSimpleAdd
- Los casos de test se agrupan en clases (suites "naturales"), ej: MoneyTest
- El Test Runner usa reflexión para saber que tests correr
- Primera implementación: SUnit
- Nombre es "misleading"

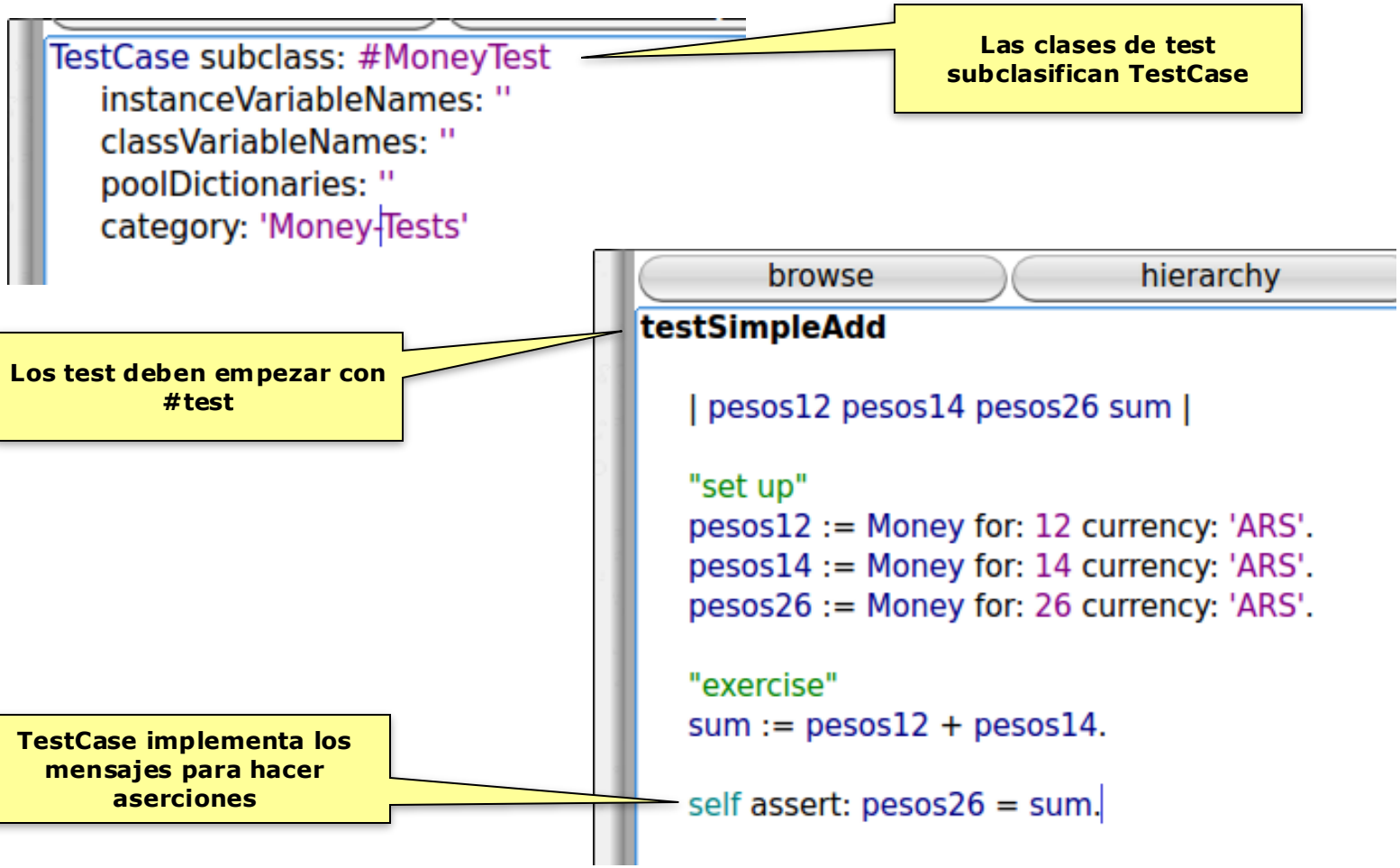
Estructura de los tests

Establece el contexto inicial para la ejecución del test. Pre-condición del test (puede ser reificado en mensaje setUp)

Ejercita la funcionalidad específica que se está testeando. Determina QUÉ se está testeando.

Verifica que los resultados sean los esperados. Post-condición del test

En Smalltalk: SUnit



Runner en el mismo ambiente de desarrollo

Ejemplo

- Modelar un Calendario de días feriados al que se le pueda preguntar si una fecha es feriado o no
- Se pueda indicar qué días son feriados de la siguiente manera:
  - Por medio de un día de la semana, ej. Domingo
  - Por medio de un día de un mes, ej. 25 de Diciembre
  - Por medio de un día particular, ej. 20/4/2012

Conclusiones - Diseño

- Es difícil poner nombres
  - Se debe a que no conocemos el dominio aún!
  - No perder mucho tiempo en eso al principio
  - Poner “Nombres sin Significado” en vez de “Malos Nombres”

Conclusiones - Diseño

- Recordar que es una aprendizaje continuo
  - Con el tiempo tuvimos más conocimiento del dominio
  - Pudimos elegir mejores nombres
  - Refactorizamos y mejoramos el modelo (proceso incremental)

Conclusiones - Diseño

- Implementar la solución más sencilla, acorde al problema presentado
- No caer en “análisis parálisis”

Conclusiones - Técnica

- Escribir el assert primero
- Técnica de testing 1: Assertar por casos negativos, no solo positivos
- Escribir un test por caso!
  (no implica un assert por
  test)

Conclusiones - Técnica

- Implementar la funcionalidad mínima para hacer pasar el test:
  - Nos hace pensar en todos los otros casos a tener en cuenta
  - Nos asegura que vamos a cubrir todos los casos y estamos haciendo un desarrollo incremental

Conclusiones - Impacto

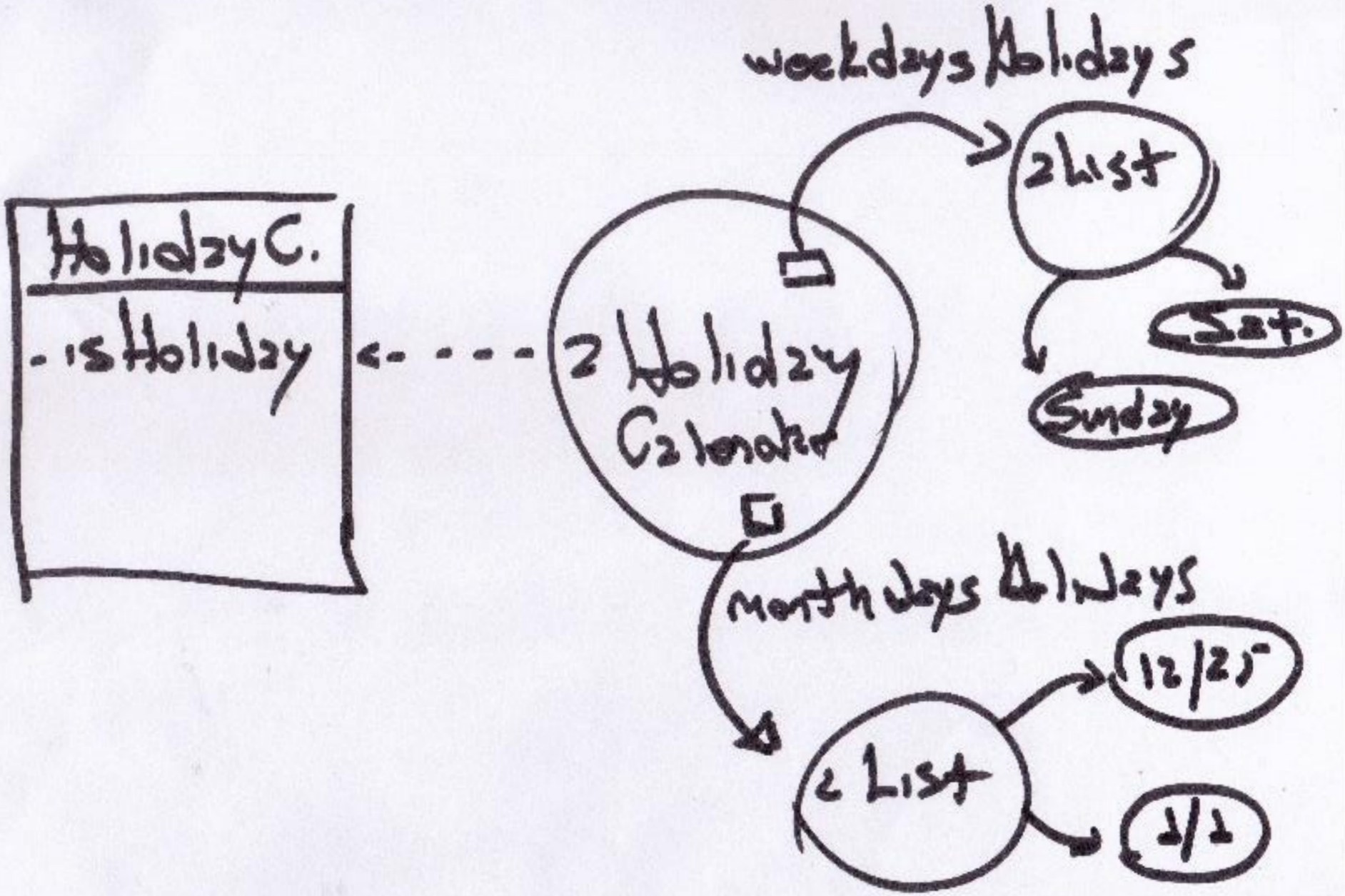
- TDD nos permite tener un modelo inicial rápidamente
- ¡Está funcionando! → Fuerte efecto Psicológico

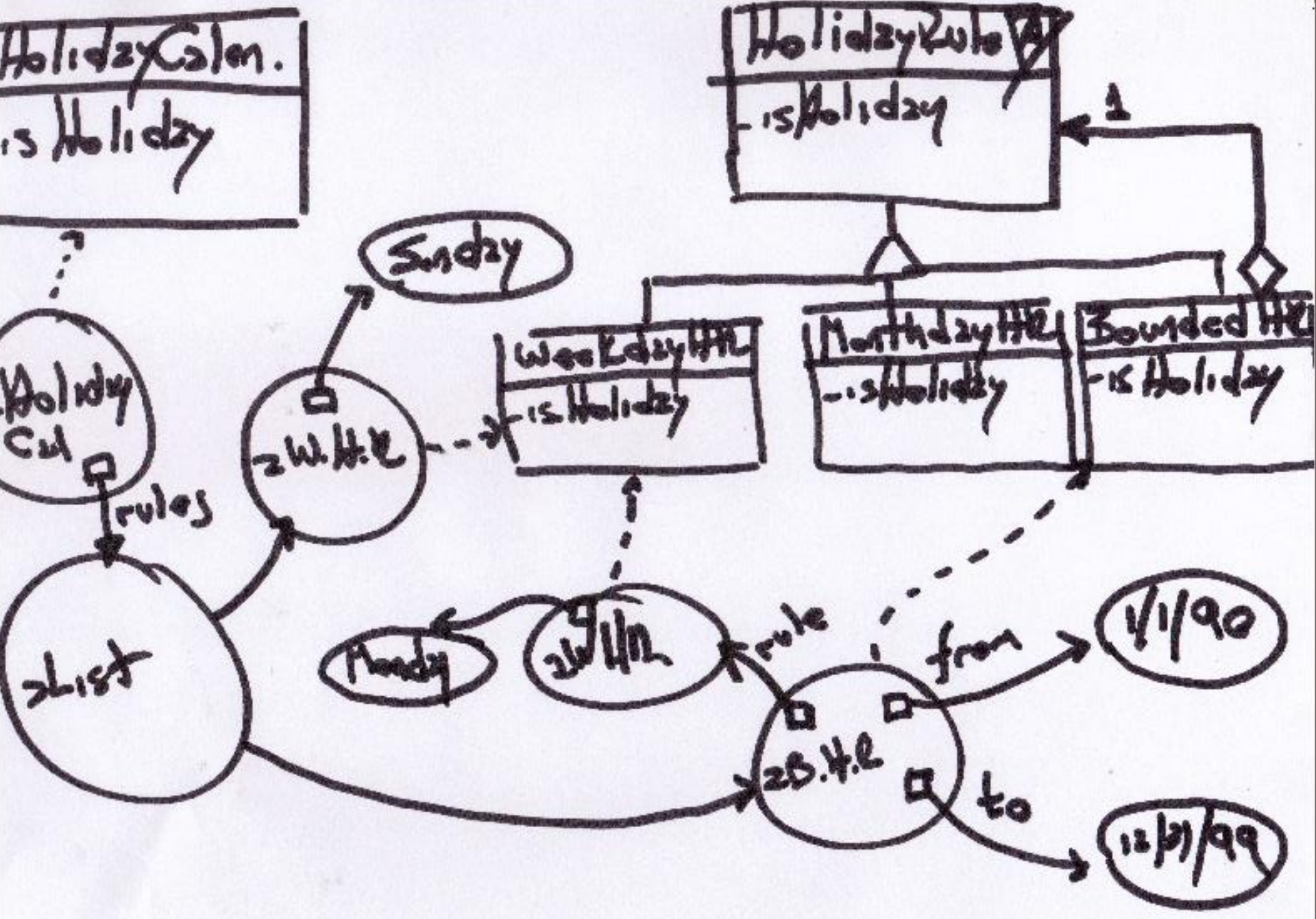
Conclusiones - Impacto

- Obtuvimos feedback inmediato sobre un error!
- Permite asegurarnos que aquello que aprendimos sigue funcionando con lo nuevo que aprendemos

Nuevo Requerimiento!

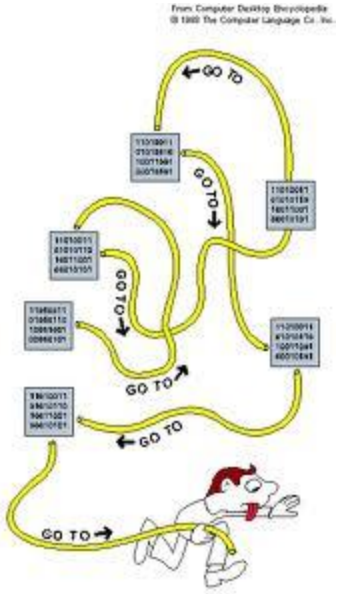
- Los feriados son válidos en un intervalo de tiempo! Ejemplos:
  - A partir del 2/8/2002 el 24/3 es feriado
  - Del 1/1/1990 al 31/12/1999 fueron feriado los Lunes


weekdays Holidays
HolidayC.
-isHoliday
HolidayCalendar
2List
Sat.
Sunday
monthdays Holidays
12/25
2List
1/1


HolidayCalen.
isHoliday
HolidayRule
isHoliday
1
Sunday
WeekdayHR
isHoliday
MonthdayHR
isHoliday
BoundedHR
isHoliday
Holdy Cal
rules
2W.H.R
2list
Monday
2W/M
rule
from
1/1/90
2B.H.R
to
12/31/99

Conclusiones - Diseño

- El código nos habla!
  - La solución anterior funcionaba pero no escalaba
  - Test difíciles de escribir son un indicio de mal diseño!



Conclusiones - Diseño

- TDD no implica buen diseño
- ¡Los buenos diseños son hechos por buenos diseñadores!

Conclusiones - Diseño

- La Nueva Solución implica un "Salto Conceptual" muy grande
  - Es lo que queremos lograr en esta materia

Conclusiones - Impacto

- Pudimos implementar un nuevo requerimiento asegurando calidad en lo existente
- Se hizo un cambio de diseño muy grande, con un impacto muy fuerte…
  - … pero no tuvimos miedo en hacerlo
  - … pudimos asegurar que el cambio no "rompiera nada"