Refactoring

Alejandra Garrido
Objetos 2
Facultad de
Informática - UNLP

En la clase pasada…

- Los elementos distintivos de la arquitectura de un sistema no surgen hasta *después* de tener código que funciona
- Construir el sistema perfecto es imposible
- Los errores y el cambio son inevitables
- No se trata sólo de agregar, sino de *adaptar*, *transformar.*
- Hay que aprender del **feedback**

Nos ponemos ágiles

- http://agilemanifesto.org/

- Dos prácticas ágiles esenciales:
  - Refactoring
  - Testing

Refactoring

- "Refactoring Object-Oriented Frameworks".
  - Bill Opdyke, PhD Thesis. Univ. of Illinois at Urbana-Champaign (UIUC). 1992. Director: Ralph Johnson.

  - Framework Choices. Qué cambios ocurren de una iteración a otra?

Surge el refactoring en la OO

- Restructurings in a class hierarchy
  E.g. “Create an abstract superclass”
  - “Creating Abstract Superclasses by Refactoring”.
    Opdyke & Johnson. ACM Conf. Computer Science. 1993

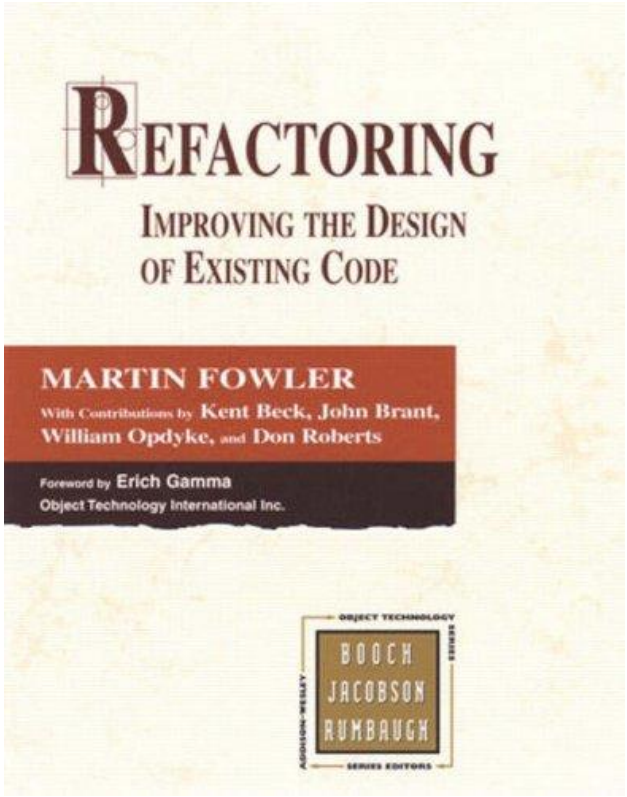
- Restructurings between components
  E.g. “Converting inheritance into aggregation”
  - “Refactoring and Aggregation”.
    Johnson & Opdyke. ISOTAS 1993.

Refactoring

- Refactoring como una transformación que preserva el comportamiento

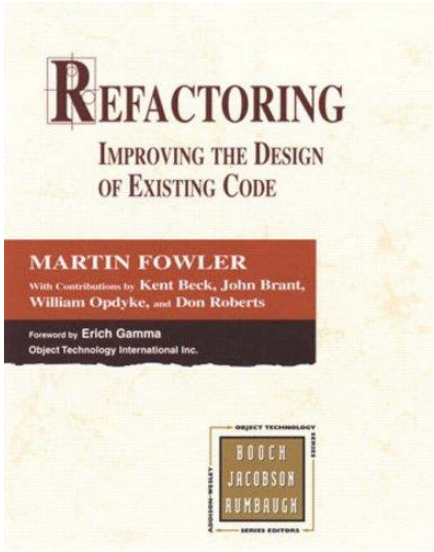
Beautifying code

Refactoring by Martin Fowler

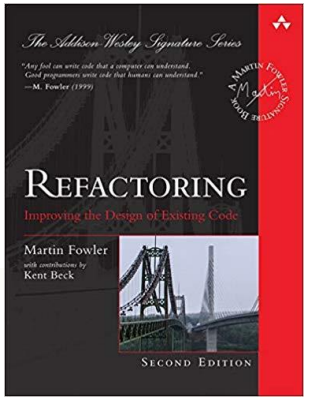


Refactoring

- Es el proceso a través del cual se cambia un sistema de software mejorando la organización, legibilidad, adaptabilidad y mantenibilidad del código luego que ha sido escrito….
  - que **NO altera** el comportamiento externo del sistema,
  - que ***mejora*** su estructura interna



Refactoring by Fowler



- *Refactoring* (sustantivo): cada uno de los cambios catalogados
  - “A change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior”.
  - Con un nombre específico y una receta (mecánica)

- *Refactor* (verbo): el proceso de aplicar refactorings
  - “To restructure software by applying a series of refactorings without changing its observable behavior”

Veamos un ejemplo….

- Imprimir los puntajes de cada set de un jugador en cada partido de tenis de una fecha específica.

Puntajes para los partidos de la fecha 15/5/2019

Partido:

Puntaje del jugador: Rafael Nadal: 6; 5; 7; Puntos del partido: 36

Puntaje del jugador: Roger Federer: 4; 7; 6; Puntos del partido: 34

Partido:

.....

class ClubTenis (1)

```
mostrarPuntajesJugadoresEnFecha: aDate
        | partidosDeLaFecha result |
        result := WriteStream on: String new.
        result nextPutAll: 'Puntajes para los partidos de la fecha' , aDate asString; cr.
        partidosDeLaFecha := coleccionPartidos select: [ :p | p fecha = aDate ].
        partidosDeLaFecha
                do: [ :partido |
                        | j1 j2 totalGames |
                        result
                                nextPutAll: 'Partido:';
                                cr.
                        j1 := partido jugador1.
                        totalGames := 0.
                        result
                                nextPutAll: 'Puntaje del jugador: ';
                                nextPutAll: j1 getNombreJugador;
                                nextPutAll: ': '.
```

class ClubTenis (2)

```
(partido puntosDelJugador: j1)
        do: [ :gamesDelSet |
                result
                        nextPutAll: gamesDelSet asString, ';'.
                totalGames := totalGames + gamesDelSet ].
result nextPutAll: ' Puntos del partido: '.
j1 zona = 'A'
        ifTrue: [ result nextPutAll: (totalGames * 2) asString ].
j1 zona = 'B'
        ifTrue: [ result nextPutAll: totalGames asString ].
j1 zona = 'C'
        ifTrue: [ partido ganador = j1
                        ifTrue: [ result nextPutAll: totalGames asString ]
                        ifFalse: [ result nextPutAll: 0 asString ] ].
```

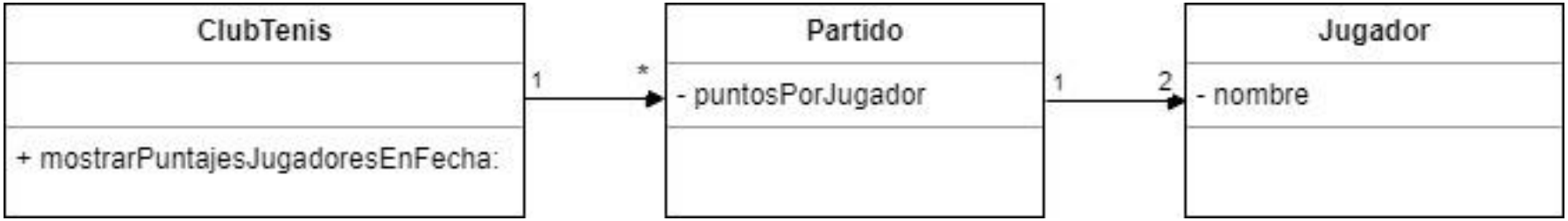
class ClubTenis (3)

```
j2 := partido jugador2.
totalGames := 0.
result
        nextPutAll: 'Puntaje del jugador: ';
        nextPutAll: j2 getNombreJugador;
        cr.
(partido puntosDelJugador: j2)
        do: [ :gamesDelSet |
                result
                        nextPutAll: gamesDelSet asString, ';'.
                totalGames := totalGames + gamesDelSet ].
result nextPutAll: '. Puntos del partido: '.
j2 zona = 'A'
        ifTrue: [ result nextPutAll: (totalGames * 2) asString ].
j2 zona = 'B'
        ifTrue: [ result nextPutAll: totalGames asString ].
j2 zona = 'C'
        ifTrue: [ partido ganador = j2
                        ifTrue: [ result nextPutAll: totalGames asString ]
                        ifFalse: [ result nextPutAll: 0 asString ] ] ].
^result contents
```

```
ClubTenis>>mostrarPuntajesJugadoresEnFecha: aDate
	| partidosDeLaFecha result |
	result := WriteStream on: String new.
	result
		nextPutAll: 'Puntajes para los partidos de la fecha' , aDate asString;		cr.
	partidosDeLaFecha := coleccionPartidos select: [ :p | p fecha = aDate ].
	partidosDeLaFecha
		do: [ :partido | | j1 j2 totalGames |
			result
				nextPutAll: 'Partido:'; cr.
			j1 := partido jugador1.
			totalGames := 0.
			result
				nextPutAll: 'Puntaje del jugador: ';
				nextPutAll: j1 getNombreJugador;
				nextPutAll: ': '.
			(partido puntosDelJugador: j1)
				do: [ :gamesDelSet |
					result
						nextPutAll: gamesDelSet asString, ';'.
					totalGames := totalGames + gamesDelSet ].
			result nextPutAll: ' Puntos del partido: '.
			j1 zona = 'A'
				ifTrue: [ result nextPutAll: (totalGames * 2) asString ].
			j1 zona = 'B'
				ifTrue: [ result nextPutAll: totalGames asString ].
			j1 zona = 'C'
				ifTrue: [ partido ganador = j1
					ifTrue: [ result nextPutAll: totalGames asString ]
					ifFalse: [ result nextPutAll: 0 asString ] ].
			j2 := partido jugador2.
			totalGames := 0.
			result
				nextPutAll: 'Puntaje del jugador: ';
				nextPutAll: j2 getNombreJugador;
				cr.
			(partido puntosDelJugador: j2)
				do: [ :gamesDelSet |
					result
						nextPutAll: gamesDelSet asString;
						nextPut: ';'.
					totalGames := totalGames + gamesDelSet ].
			result nextPutAll: '. Puntos del partido: '.
			j2 zona = 'A'
				ifTrue: [ result nextPutAll: (totalGames * 2) asString ].
			j2 zona = 'B'
				ifTrue: [ result nextPutAll: totalGames asString ].
			j2 zona = 'C'
				ifTrue: [ partido ganador = j2
					ifTrue: [ result nextPutAll: totalGames asString ]
					ifFalse: [ result nextPutAll: 0 asString ] ] ].

	^ result contents
```

Diagrama inicial



Cambios pedidos …

- Cambiará la manera de calcular los puntos
- Pueden cambiar las zonas

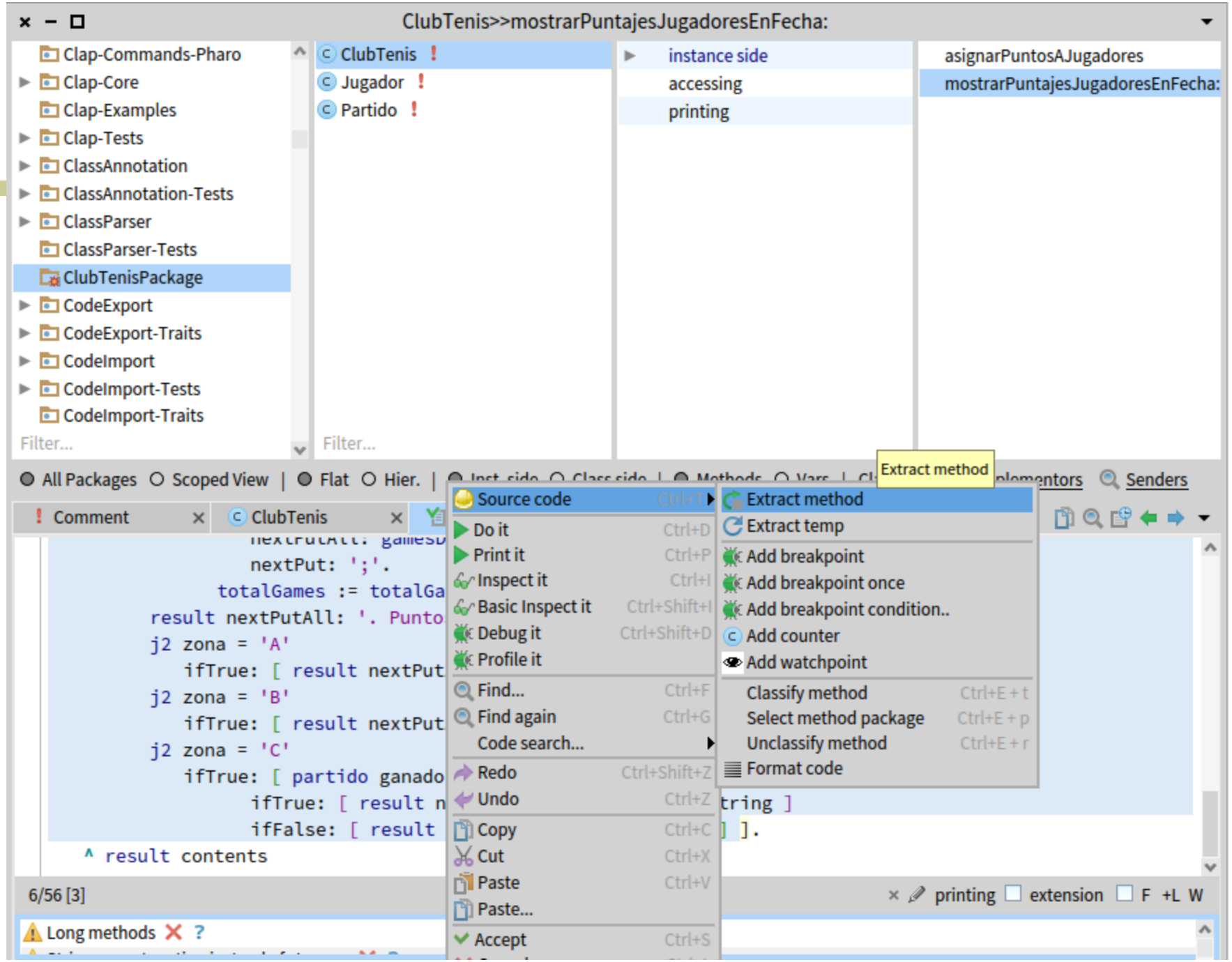
Ejemplo del club de tenis

- Por dónde empezamos?
- Cuáles son los problemas que tiene el código?

MÉTODO LARGO!

Refactoring Extract method

```
mostrarPuntajesJugadoresEnFecha: aDate
        | partidosDeLaFecha result |
        result := WriteStream on: String new.
        result nextPutAll: 'Puntajes para los partidos de la fecha' , aDate asString; cr.
        partidosDeLaFecha := coleccionPartidos select: [ :p | p fecha = aDate ].
        partidosDeLaFecha
                do: [ :partido |
                        self mostrarPartido: partido en: result].
        ^result contents
```


ClubTenis>>mostrarPuntajesJugadoresEnFecha:
Clap-Commands-Pharo
Clap-Core
Clap-Examples
Clap-Tests
ClassAnnotation
ClassAnnotation-Tests
ClassParser
ClassParser-Tests
ClubTenisPackage
CodeExport
CodeExport-Traits
CodeImport
CodeImport-Tests
CodeImport-Traits
Filter...
ClubTenis
Jugador
Partido
Filter...
instance side
accessing
printing
asignarPuntosAJugadores
mostrarPuntajesJugadoresEnFecha:
All Packages
Scoped View
Flat
Hier.
Senders
Comment
ClubTenis
nextPut: ';'.
totalGames := totalGa
result nextPutAll: '. Punto
j2 zona = 'A'
ifTrue: [result nextPut
j2 zona = 'B'
ifTrue: [result nextPut
j2 zona = 'C'
ifTrue: [partido ganado
ifTrue: [result n
ifFalse: [result
^ result contents
tring]
]].
Source code
Do it
Print it
Inspect it
Basic Inspect it
Debug it
Profile it
Find...
Find again
Code search...
Redo
Undo
Copy
Cut
Paste
Paste...
Accept
Extract method
Extract temp
Add breakpoint
Add breakpoint once
Add breakpoint condition..
Add counter
Add watchpoint
Classify method
Select method package
Unclassify method
Format code
Extract method
6/56 [3]
printing
extension
Long methods

Extract Method a mano

- Mecánica:
  - Crear un nuevo método cuyo nombre explique su propósito
  - Copiar el código a extraer al nuevo método
  - Revisar las variables locales del original
  - Si alguna se usa sólo en el código extraído, mover su declaración
  - Revisar si alguna variable local es modificada por el código extraido. Si es solo una, tratar como query y asignar. Si hay más de una no se puede extraer.
  - Pasar como parámetro las variables que el método nuevo lee.
  - Compilar
  - Reemplazar código en método original por llamada
  - Compilar

A tener en cuenta…

- Testear siempre después de hacer un cambio
  - Sí se cometió un error es más fácil corregirlo
- Definir buenos nombres

En la clase ClubTenis…

- A quien pertenece realmente el codigo de

- **mostrarPartido: partido en: result ?**

ClubTenis>>mostrarPartido: partido en: result

```
| j1 j2 totalGames |
        result nextPutAll: 'Partido:'; cr.
        j1 := partido jugador1.
        totalGames := 0.
        result
                nextPutAll: 'Puntaje del jugador: ';
                nextPutAll: j1 getNombreJugador;
                nextPutAll: ': '.
        (partido puntosDelJugador: j1)
                do: [ :gamesDelSet |
                        result nextPutAll: gamesDelSet asString, ';'.
                        totalGames := totalGames + gamesDelSet ].
        result nextPutAll: ' Puntos del partido: '.
        j1 zona = 'A'
                ifTrue: [ result nextPutAll: (totalGames * 2) asString ].
        j1 zona = 'B'
                ifTrue: [ result nextPutAll: totalGames asString ].
        j1 zona = 'C'
                ifTrue: [ partido ganador = j1
                                ifTrue: [ result nextPutAll: totalGames asString ]
                                ifFalse: [ result nextPutAll: 0 asString ] ].
```

…

RESPONSABILIDAD MAL ASIGNADA

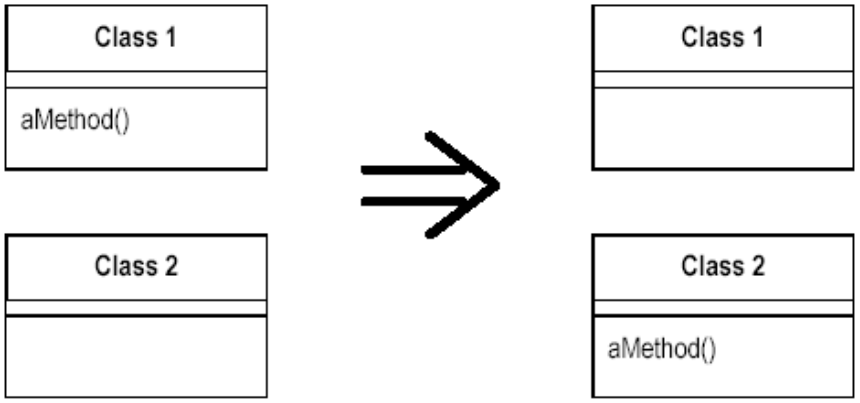
Refactoring Move method

```
ClubTenis>>mostrarPuntajesJugadoresEnFecha: aDate
        | partidosDeLaFecha result |
        result := WriteStream on: String new.
        result
                nextPutAll: 'Puntajes para los partidos de la fecha' , aDate asString;
                cr.
        partidosDeLaFecha := coleccionPartidos select: [ :p | p fecha = aDate ].
        partidosDeLaFecha
                do: [ :partido | partido mostrarEn: result].
^ result contents

Partido>>mostrarEn: result
        | j1 j2 totalGames |
        result nextPutAll: 'Partido:'; cr.
        j1 := self jugador1.
        totalGames := 0.
        result …
```

Move Method

- Motivación:
  - Un método esta usando o usará muchos servicios que están definidos en una clase diferente a la suya (Feature envy)
- Solucion:
  - Mover el método a la clase donde están los servicios que usa.
  - Convertir el método original en un simple delegación o eliminarlo



En la clase Partido…

El código de

- **mostrarEn: result**
  - Nombre inadecuado del parámetro
    - Rename Variable!
  - Sigue siendo bastante largo, porque tiene código duplicado
    - más Extract Method!

Rename Variable: Vale la pena?

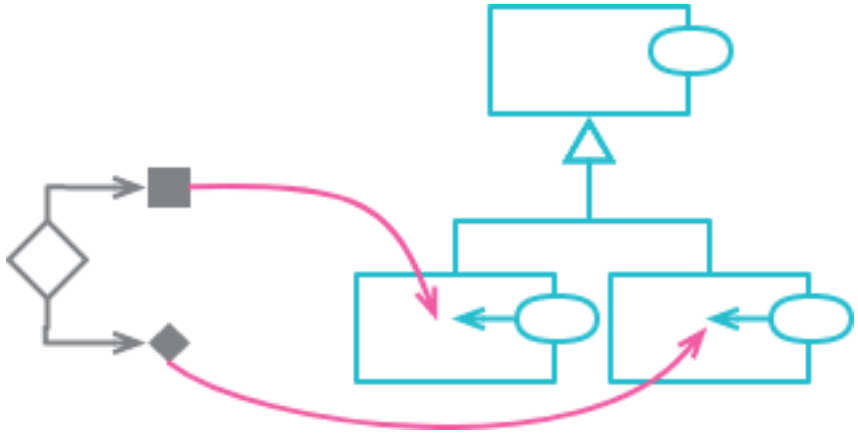
- Todo buen código debería comunicar con claridad lo que hace
- Nombres de variables adecuados aumentan la claridad
- Sólo los buenos programadores escriben código legible por otras personas

```
Partido>>mostrarEn: unStream
        unStream nextPutAll: 'Partido:'; cr.
        self mostrarPuntosDe: self jugador1 en: unStream.
        self mostrarPuntosDe: self jugador2 en: unStream.

Partido>>mostrarPuntosDe: unJugador en: unStream
        | totalGames |
        totalGames := 0.
        unStream
                nextPutAll: 'Puntaje del jugador: ';
                nextPutAll: unJugador getNombreJugador;
                nextPutAll: ': '.
        (self puntosDelJugador: unJugador)
                do: [ :gamesDelSet |
                        unStream nextPutAll: gamesDelSet asString, ';'.
                        totalGames := totalGames + gamesDelSet ].
        unStream nextPutAll: ' Puntos del partido: '.
        unJugador zona = 'A'
                ifTrue: [ unStream nextPutAll: (totalGames * 2) asString ].
        unJugador zona = 'B'
                ifTrue: [ unStream nextPutAll: totalGames asString ].
        unJugador zona = 'C'
                ifTrue: [ self ganador = unJugador
                        ifTrue: [ unStream nextPutAll: totalGames asString ]
                        ifFalse: [ unStream nextPutAll: 0 asString ] ].
```

Seguimos teniendo el switch

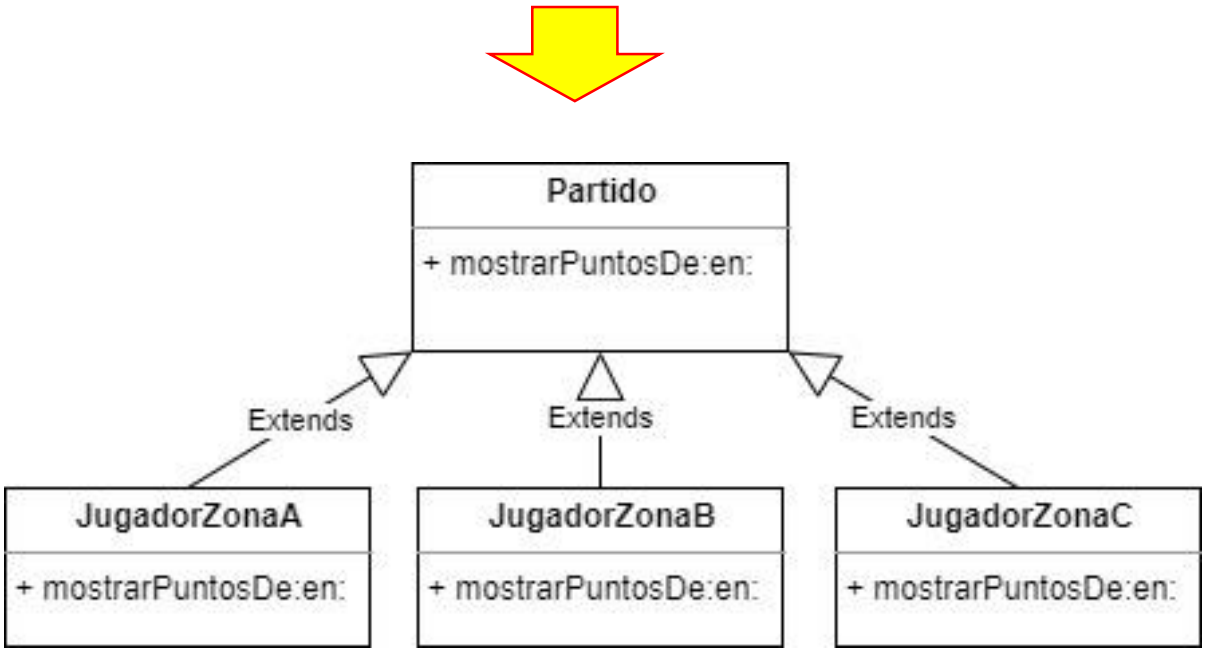
- ¿Cómo eliminar el switch?
- ➔ Replace Conditional with Polymorphism



Partido>>mostrarPuntosDe: unJugador en: unStream

…

```
unJugador zona = 'A'
        ifTrue: [ unStream nextPutAll: (totalGames * 2) asString ].
unJugador zona = 'B'
        ifTrue: [ unStream nextPutAll: totalGames asString ].
unJugador zona = 'C'
        ifTrue: [ self ganador = unJugador
                ifTrue: [ unStream nextPutAll: totalGames asString ]
                ifFalse: [ unStream nextPutAll: 0 asString ] ].
```

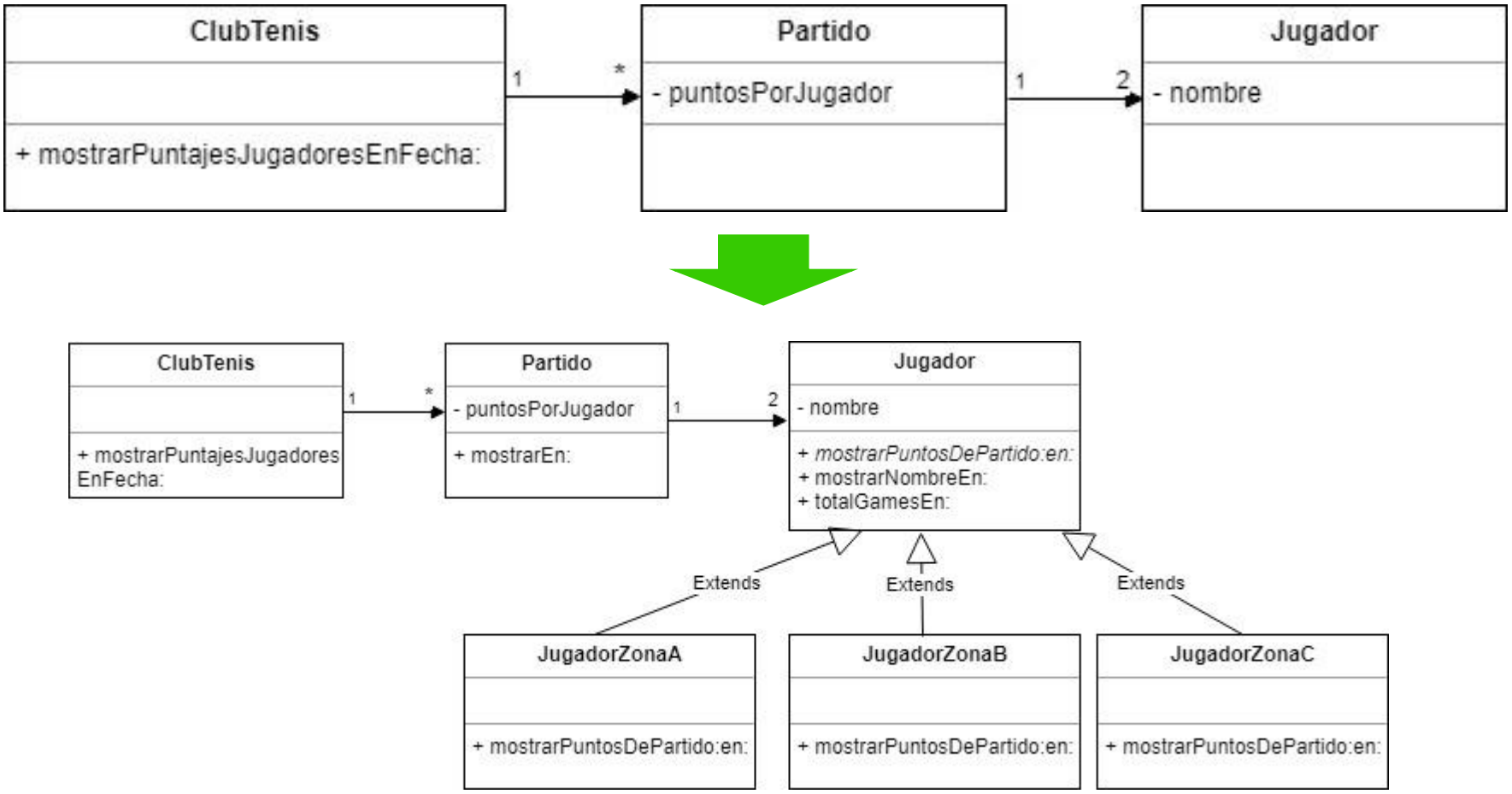


- ¿Tiene sentido hacer subclases de Partido? ¿Corresponde a Partido este cálculo?

No corresponde a Partido

Paso 5) Aplico **Move Method**
Partido>>mostrarPuntosDe:en: a
Jugador>>mostrarPuntosDePartido:en:

Paso 6) Aplico **Replace Conditional with Polymorphism** en
Jugador>>mostrarPuntosDePartido:en:


ClubTenis
+ mostrarPuntajesJugadoresEnFecha:
1
*
Partido
- puntosPorJugador
1
2
Jugador
- nombre
ClubTenis
+ mostrarPuntajesJugadores EnFecha:
1
*
Partido
- puntosPorJugador
+ mostrarEn:
1
2
Jugador
- nombre
+ mostrarPuntosDePartido:en:
+ mostrarNombreEn:
+ totalGamesEn:
Extends
Extends
Extends
JugadorZonaA
+ mostrarPuntosDePartido:en:
JugadorZonaB
+ mostrarPuntosDePartido:en:
JugadorZonaC
+ mostrarPuntosDePartido:en:

En JugadorZonaA…

```
>>mostrarPuntosDePartido: unPartido en: unStream
        | totalGames |
        self mostrarNombreEn: unStream.
        totalGames := 0.
        (unPartido puntosDelJugador: self)
                do: [ :gamesDelSet |
                        unStream nextPutAll: gamesDelSet asString, ';'.
                        totalGames := totalGames + gamesDelSet ].
        unSream nextPutAll: ' Puntos del partido: '.
        unStream nextPutAll: (totalGames * 2) asString.

>>mostrarNombreEn: unStream
        unStream
                nextPutAll: 'Puntaje del jugador: ';
                nextPutAll: self nombre;
                nextPutAll: ': '.
```

Método repetido en c/ subclase

- `>>mostrarNombreEn: unStream`
- Cómo eliminamos esta duplicación?
- → Pull Up Method
  - Si los métodos en subclases son iguales → subir directamente
  - Si los métodos en subclases no son iguales → parametrizar primero

Redundancia de variables temporales

```
>>mostrarPuntosDePartido: unPartido en: unStream
        | totalGames |
        self mostrarNombreEn: unStream.
        totalGames := 0.
        (unPartido puntosDelJugador: self)
                do: [ :gamesDelSet |
                        unStream nextPutAll: gamesDelSet asString, ';'.
                        totalGames := totalGames + gamesDelSet ].
        unSream nextPutAll: ' Puntos del partido: '.
        unStream nextPutAll: (totalGames * 2) asString.
```

Replace Temp with Query

- Motivación: usar este refactoring:
  - Para evitar métodos largos. Las temporales, al ser locales, fomentan métodos largos
  - Para poder usar una expresión desde otros métodos
  - Antes de un Extract Method, para evitar parámetros innecesarios
- Solución:
  - Extraer la expresión en un método
  - Remplazar TODAS las referencias a la var. temporal por la expresión
  - El nuevo método luego puede ser usado en otros métodos

Después de Replace Temp with Query

```
>>mostrarPuntosDePartido: unPartido en: unStream
        self mostrarNombreEn: unStream.
        (unPartido puntosDelJugador: self)
                do: [ :gamesDelSet |
                        unStream nextPutAll: gamesDelSet asString, ';'].
        unSream nextPutAll: ' Puntos del partido: '.
        unStream nextPutAll: (self totalGamesEn: unPartido * 2) asString.
```

Sobre la performance

- La mejor manera de optimizar un programa, primero es escribir un programa bien factorizado y luego optimizarlo, previo profiling …