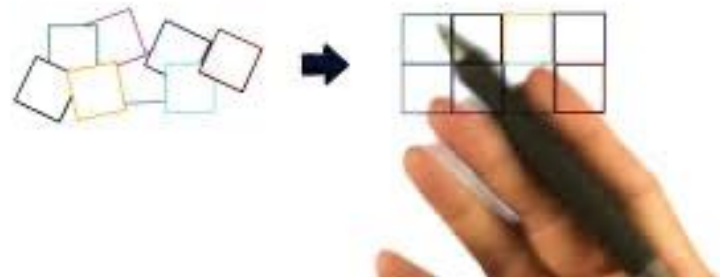


# Refactoring tools



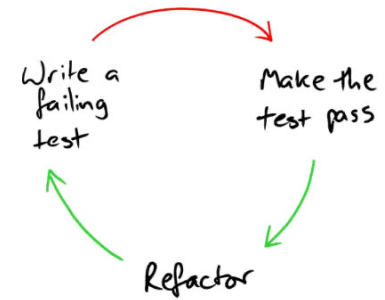
REFACTORING



Alejandra Garrido  
Objetos 2  
Facultad de  
Informática - UNLP

# [ Cuando aplicar refactoring ]

- En el contexto de TDD
- Cuando se descubre código con mal olor, aprovechando la oportunidad
  - dejarlo al menos un poco mejor, dependiendo del tiempo que lleve y de lo que esté haciendo
- Cuando no puedo entender el código
  - aprovechar el momento en que lo logro entender
- Cuando encuentro una mejor manera de codificar algo



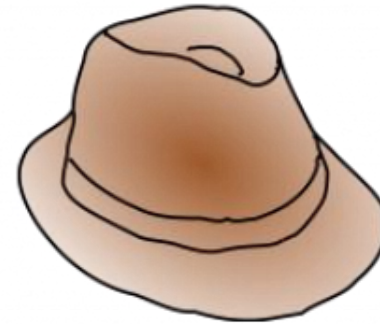
Not Page

# [The 2 hats



## Adding Function

Se exploran ideas, se corrigen bugs



## Refactoring

Solo puedo refactorizar  
con tests en verde

Puedo cambiar de sombrero frecuentemente  
Pero solo puedo usar **1 sombrero por vez**

# [Automatización del refactoring]

- Refactorizar a mano es demasiado costoso: lleva tiempo y puede introducir errores
- Herramientas de refactoring
- Características de las herramientas:
  - potentes para realizar refactorings útiles
  - restrictivas para preservar comportamiento del programa (uso de *precondiciones*)
  - interactivas, de manera que el chequeo de precondiciones no debe ser extenso

# El refactoring automático nace con Smalltalk

- Primera herramienta de refactoring: Refactoring Browser (RB) (en UIUC by John Brant & Don Roberts del grupo de Ralph Johnson)
- Practicamente todos los lenguajes tienen herramienta de refactoring hoy en día, y copian la misma arquitectura / técnica del RB
- Más adelante: herramienta Code Critic que detecta code smells
- Smalltalk 1ro en:
  - XUnit
  - Refactoring



OOPSLA'98 Code Fest

# [Las herramientas...

- Solo chequean lo que sea posible desde el árbol de sintaxis y la tabla de símbolos
- Pueden ser demasiado **conservativas** (no realizan un refactoring si no pueden asegurar preservación de comportamiento) o **asumir** buenas técnicas de programación

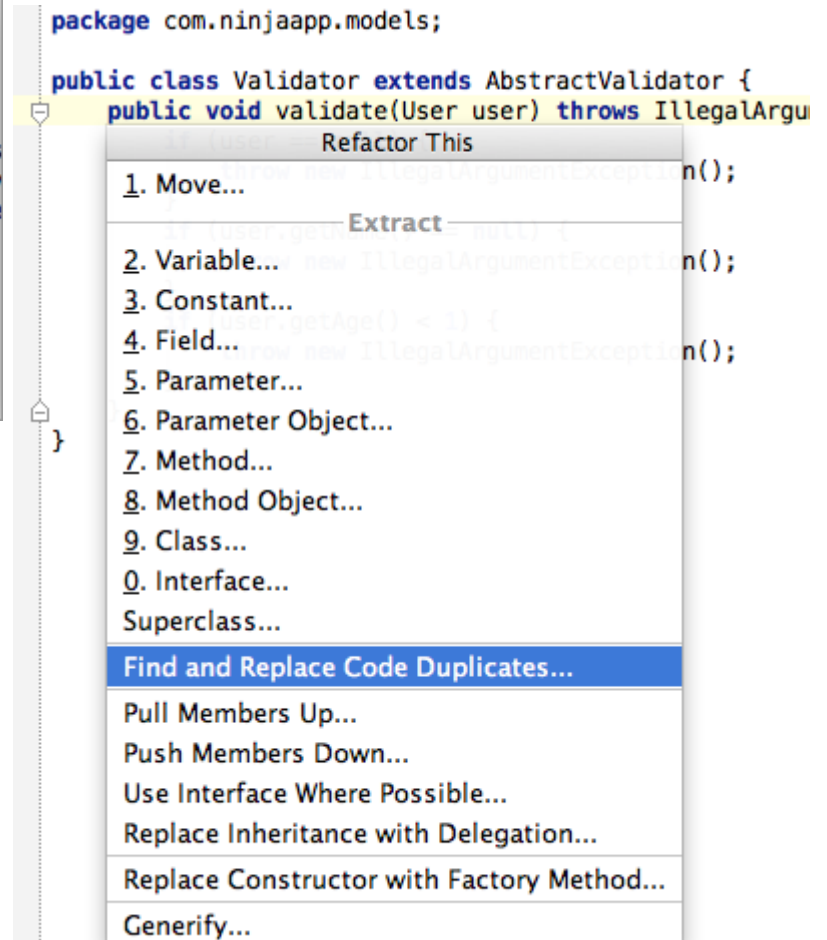
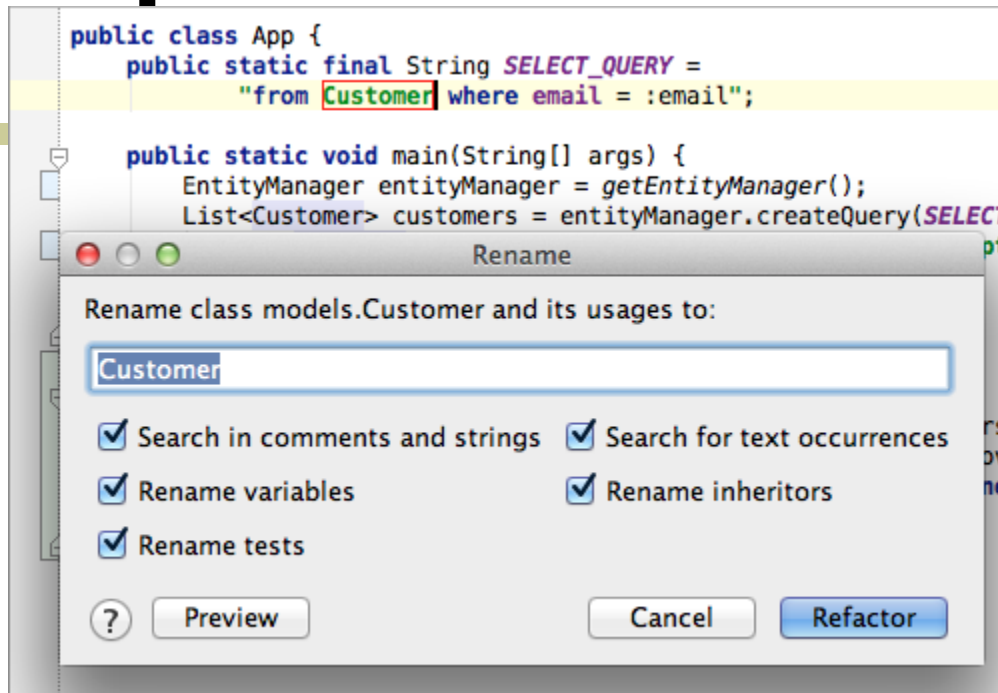
# Smalltalk RB

The screenshot displays the Smalltalk RB IDE interface. The main window is divided into several panes:

- Class Browser (Left):** Shows a hierarchy of classes under the package `Pkg1|^Pkg2|Pk.*Core$`. The class `ClubDeTenis` is selected.
- Code Editor (Center):** Displays the source code for the `mostrarPuntajesJugadoresEnFecha:` method. The code is as follows:

```
mostrarPuntajesJugadoresEnFecha: aDate
| partidosDeLaFecha result |
result := WriteStream on: String new.
result
  nextPutAll: 'Puntajes para los partido
  cr.
  partidosDeLaFecha := coleccionPartidos se
  partidosDeLaFecha
    do: [ :partido |
      | j1 j2 totalGames |
      result
        nextPutAll: 'Partido: ']
```
- History Navigator (Top Right):** Shows a list of methods: `-- all --`, `initialize`, and `printing`.
- Context Menu (Overlaid):** A menu is open over the code editor, listing various actions such as `Create cascade`, `Extract method`, `Extract method to component`, `Extract to temporary`, `Inline method`, `Inline method from component`, `Inline temporary`, `Move variable definition`, `Rename temporary/parameter`, `Split cascade`, `Temporary to instvar`, `Undo`, and `Redo`.
- Format Menu (Bottom Right):** A sub-menu is open showing options like `Format`, `Source code refactoring`, `Suggestions...`, `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`, and `Cancel`.
- Status Bar (Bottom):** Displays the current line and column (7/54 [23] -- 7/54 [74]) and a list of warnings: `Long methods`, `String concatenation instead of streams`, and `Use cascaded nextPutAll's instead of # in #nextPutAll:`.

# IntelliJ IDEA





# Sonar qube

