

Programming errors to  
avoid

# Playing with HashMap

- Some eclipse tests
  - See lesson1 in codeSamples
    - Demo1 to 4

# equals and hashCode

- Always redefined together
- Mostly redefined for data object
- Eclipse settings :
  - Preferences - java - compiler - errors/warnings
    - Potential programming problems – Class overrides 'equals()' but not 'hashCode' => Error

# Fields used to compute equals and hashCode

- Are used in both methods
- Generally those who defined object state
  - Fields that do not define state
    - Cache, temp working variables, coworkers (eventually)
- Should be immutable (with final reference)
  - Eclipse preferences - save actions - code style - use 'final'

# Lessons to learn

- Difficult to conform to an API
  - For client and provider (you have to read the doc for correct use)
  - Even hashCode and equals are misused
- Hard to foresee impacts of variable changes
- Most bugs come from wrong state
  - Programming instructions do not change unlike data
  - Test from clear state ok but crash in running application
- Mandatory to take care of object state
- Prefer usage of immutable
  - Most simple way to keep state clean
  - Naturally avoid many programming errors
  - Force to define state at construction

# Better programming techniques

- Use immutable objects as much as possible
  - Immutable collection in guava
  - At least set the maximum possible fields 'final' and even immutable
  - <http://www.javapractices.com/topic/TopicAction.do?Id=29>
- 'final' keyword is your friend
  - Mutable setters should be avoid most of the time
  - Object state defined at construction time (see Demo5)
- Every implementation class should derived from an interface
- Never use 'new' operator outside a factory/builder
  - Inject dependencies (coworkers)
  - Create factory or factory methods for every implementation (see Demo6 - 7)
- Avoid (see Demo8)
  - Public static methods
  - Private static fields

# Limitations with immutables

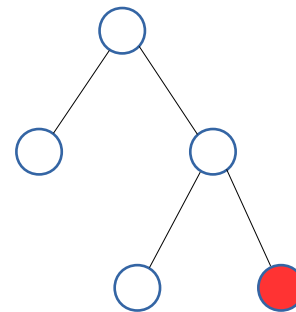
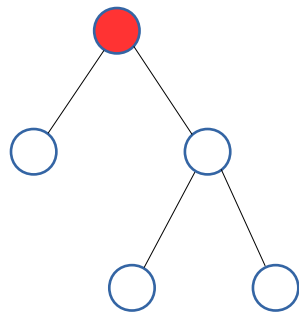
- Performance issues
  - Often change the ‘same’ Type object locally
    - Create a TypeBuffer (like StringBuffer)
  - Small updates in a complex object (like tree or list)
    - Collections from vavr library
  - Lot of objects to change often in multiple place
    - Make them mutable
- But also allow performance improvements

# Immutable Complex Objects

- Performance issues
  - List: Changing only one element force to regenerate the whole list



- Tree: The most inner node force to regenerate the whole tree





# Long term objectives

- Introduce good practices
  - Annoying for now
  - Basis for other improvements
    - True object oriented programming
    - Introduce to functional programming
    - Truly reliable and fully automated tests
    - State of art in clean architecture design

# Reference

- Effective Java - Joshua Bloch
- Out of the Tar Pit - Ben Moseley,  
Peter Marks