Dashboard > Training > ... > Object Boot Camp > Bester

Search

ThoughtWorks® Training **Bester**

Welcome Luca Minudel | History | Preferences | Log Out

View | Attachments (1) | Info | Review

Browse Space

Added by Rolf Russell, last edited by David S Wood on Dec 11, 2006 (view change)
Labels: (None) EDIT

Bester

Session Objectives

By the end of this session students should ...

- Be able to define polymorphism
- Be able to write polymorphic code
- Understand that duplication across classes can sometimes be removed using polymorphism
- Be able to create an interface allowing disparate objects to share functionality
- Understand the separation of logic between the objects being compared and object doing the comparison

Session Overview

Activity
Sorting People Role Play
Lab - betterThan()
Lab - Best Quantity
Lab - Best Probability
Demonstration - Generics
Pairing Demo
Bad Pair Play Acting

Session Notes

Sorting People Role Play

Setup
5 people: one *sorter* & 4 *sortees*. Take sortees out of room and tell them that their sort order is month & day of birth. They don't need to tell each other the dates now, they can whisper to each other while being sorted back in the room. Return to room and ask sorter to sort them (don't tell sorter on what criteria). The sorter should figure out to ask a sortee "are you better than" another sortee. Take sortees out of room again and change sort order to alphabetically by last name. Sorter sorts again. Also it will be better if you mention Group 1 sortees as Tigers and Group 2 sortees as Lions. This way it will be easy to explain that Bester can find best among Tigers and Lions without knowing the criteria.

Discussion

- Did the sorting algorithm have to change? No the same algorithm could sort different people. The thing that changed was the people (class).
- We introduce objects when we expect code to change. What might change in this situation? Different objects needing to be sorted by the same algorithm. Or different algorithms sorting different numbers of objects.
- This situation is the essense of polymorphism*: having a sorter use the "are you better" call that can be implemented differently for each object (height vs date of birth vs alphabetically)

*In this case, subtyping polymporphism, as each object used by the bester must be derived from a common type (class or interface) that has and appropriate "are you better" method defined

Lab - betterThan()

Based on what we have discussed, implement betterThan() for Quantity first (we'll do Probability later).

Common mistakes

- Not converting to base for comparison
- Allowing comparison of uncomparable quantities. should throw exception
- Creating an interface
- Not removing duplication of throwing exception between betterThan() and add(). should make bombIfCannotCompareTo() & cannotCompareTo() methods. make sure that cannotCompareTo() is used in equals() as well.

Lab - Best Quantity

Discussion

- Will need Bester or Quantities object.

Common mistakes

- Creating an interface between Bester and Quantity. There is only one implementation for it.
- Finding best in Quantity itself. Working on an array of quantities is not a part of the job of Quantity
- Not defining a job for Bester/Quantities
- Not writing a test for finding best of empty array throwing exception (ArrayIndexOutOfBoundsException)
- Writing a test for null parameter. Null is not a valid parameter, so adding the null check in causes us to delay failure. Fail fast! Adding null checks/returning null all over the place causes code to explode and brittleness of code.

Introduce

- Meaningful names: champion and challenger
- (optionally) Varargs: public static Bestable best(Bestable... bestables)

Lab - Best Probability

Discussion
Students should create the Bestable interface. Discuss the order of refactoring to get to the interface. The following order takes advantage of the IDE to maintain a green bar at all times:
1. extract an interface (Bestable)
2. change the method signature to take a Bestable as parameter.
3. make the other class implement Bestable

Common mistakes

- Casting to anything other than Bestable in Bester
- Not defining a job for Bestable
  - Understands an order between objects
  - Understands the requirements for determining the better of two objects
- (lower priority) Test for the cast in betterThan()

Demonstration - Generics

Show the students how to make Bestable generic in java 1.5 (or C# 2.0)

Instructors may point out that Generics (templates) can be considered an example of Parametric Polymorphism, although language purists might disagree with this classification. See Polymorphism on wikipedia for more info.

Pairing Demo

Sometime during this or the previous exercise it may become apparant that not all students are pairing well. To address this, the instructors can choose to reimplement one of the preceeding steps vocalizing their pairing session.

This can be done with the tester half of the pair wearing a Darth Vader hat and the coder half of the pair wearing a Happy Birthday hat. The students can then wear these hats for the next exercise and the instructors can enforce that the keyboard exchanges hands often.

Bad Pair Play Acting

To solidify good pairing practices, the attached bad pairing situations can be play acted in front of, or by the students. For each one, select one person as the antagonist and another as the protagonist.

Pair Play Acting

Children

Hide Children | View in hierarchy

- Demonstration - Generics (Training)
- Lab - Best Probability (Training)
- Lab - Best Quantity (Training)
- Lab - betterThan() (Training)
- Sorting People Role Play (Training)

0 comments | Add Comment