“How to train the JDT Dragon” – Object Teams Part

Tutorial at EclipseCon 2012, Reston, 26.03.2012

Prepare:

- Enable
  `Window →Preferences →General →Appearance →Label Decorations →Java Type Indicator`

Warm-up: OTSample-StopWatch

1. Running

- `New → Example → Stop Watch Example`
- `src / ... / Main.java → Run As → Java Application`
- **Find** role class `WatchUI.WatchDisplay`:
  - When are instances of this role created ?
  - When is its method `update()` called?

Exercise: Anti-Demo-Plug-in

2. Adapting Java Naming Conventions

- **Create** an Object Teams Plug-in Project “`AntiDemo`”
  Throughout this wizard all defaults are OK, like:
  - Eclipse version: 3.7
  - ☑ `Generate an activator` …
  - ☑ `This plug-in will make contributions to the UI` …
  - Template: `Hello, World`.
- **Create** and configure a team to adapt classes from `org.eclipse.jdt.core`
  - create dependency to `org.eclipse.jdt.core` (MANIFEST.MF)
  - select tab Extensions, and click Add to create an extension `aspectBinding`. Fill in:
    - basePlugin: `org.eclipse.jdt.core`
    - team: `antidemo.AntiDemoTeam`
      - activation: `ALL_THREADS`

    save `(plugin.xml)`
  - create the team (e.g., click “`class*:`”)
- **Implement** role that creates an Error, i.e., `new Status(IStatus.ERROR,...)` when a java type starts with “Foo”.
- **Hint**: class `org.eclipse.jdt.core.JavaConventions` contains methods for validating all kinds of names wrt Java rules and conventions, see methods `validateXYZ`
- Launch a runtime workbench (`Eclipse Application`)
  - Main tab: ☑ `Enable OT/Equinox`
  - optionally: ▸set workspace location, ▸select Plug-ins
- **Perform**
  - create new Java project
  - create new class called `FooBar` etc.
  - try to cheat: find ways to create such a class, be inventive.
    Hint: to really cheat you can use the “OT/Equinox Monitor” View.

Application: Reachability Analysis

We are going to implement a Plug-in that finds unreachable code.

- All “main” methods are considered reachable.
- All methods called from a reachable method are also reachable.
  - Method calls inside dead code (e.g. “if (false) m();”) should not be considered.
- Analyzing method calls must consider polymorphism /overriding.
- Methods that are only called from unreachable methods are not reachable (including “islands”: cycles of unreachable methods).

Technical requirements:

- We are going to piggy-back on existing analysis inside the compiler (internal code)
- The analysis should run during each full build
- Incremental builds should be unaffected
- It's OK to simply print the result to std out.
- Bonus: create problem markers for all unreachable methods.

Preparations for follow along

The solution will be developed step-by-step on the front screen. However, you're invited to follow us by replaying the history of the following git repository:

git://github.com/aupsy/org.eclipsecon2012.ot.tutorial

1. If not done before, clone the repository, ensure project Reachability is in your workspace
2. Open the History view, e.g.:
   Git Repositories → org.eclipsecon2012.ot.tutorial → Show In... → History
3. Change to the Object Teams perspective and ensure the History view is visible
   a) Ensure Show All Branches and Tags is enabled
4. Start by selecting the initial commit (“Step 1: ...”) and invoke Checkout.
5. For each subsequent step simply check out the corresponding commit
6. See the git howto for inspecting the diffs of any given commit.