Vorgehensweise und Entscheidungsfindung

Erweitert hinzugekommen sind zunächst nur Annotationen, der Rest war soweit bereits umgesetzt.

Wir haben nochmal angepasst, dass wenn ein Objekt keine weiteren Felder oder Methoden hat, in die wir hineinzoomen können oder das Objekt NULL ist, wird kein neuer Node im Baum dafür erzeugt. Dies bringt uns eine verbesserte Übersicht.

Um unsere Implementierung zu testen, haben wir noch ein Testinterface IDummyClass und eine Test-Annotation erstellt, MyAnnotation.

Damit das zu inspizierende Objekt „in einem Thread läuft“, haben wir den ObjectHolder entworfen. Im Konstruktor des ExplorerTree erzeugen wir nun also nicht mehr einfach nur das zu inspizierende Objekt, sondern zusätzlich einen ObjectHolder, dieser bekommt dann das Objekt übergeben. Weiterhin setzen wir dem ObjectHolder noch einen neuen objectChangedListener. Ändert sich nun das Objekt, kann der ObjectHolder die objectChanged Methode aufrufen und es baut sich der ExplorerTree neu auf. Wir haben leider noch keinen anderen Weg gefunden, den Tree zu aktualisieren, so ist der Baum nach dem rebuild allerdings nicht mehr ausgeklappt, sondern wieder minimal.

In unserem ObjectHolder prüfen wir in der run-Methode, ob es sich um ein DummyClass-Objekt handelt, nur dann maniupulieren wir das Objekt. setAccessible(true) brauchten wir nicht, da wir ein set/get-Methoden verwenden. Diese Lösung dient dem Aufgabenteil 4, wie der Objektbrowser „mitbekommt“, wenn sich ein Objekt ändert.

Man könnte jetzt noch beigehen und das ganze irgendwie automatisieren, dass der ObjectHolder prüft, ob es primitive Typen gibt, diese änderbar sind (ansonsten setAccessible(true)) und diese für alle übergebenen Objekte anwenden kann. Für die Beantwortung der Aufgabe schien uns allerdings unser Beispiel ausreichend ☺

Martin Slowikowski Jan-Tristan Rudat

Matrikelnummer: 1999 166 Matrikelnummer: 2007 852