Arbeitsblatt Erzeugen von Objekten 2

Ziele

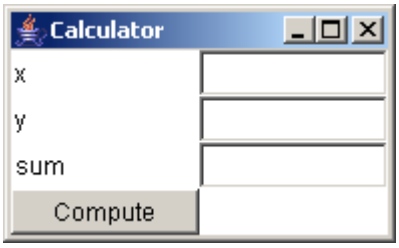
Eine *Factory Method* hilft, das Erzeugen eines Objekts von dessen Benutzung zu entkoppeln. Aber auch dieser Ansatz kann noch zu starr sein, wenn man beispielsweise erst zur Laufzeit des Programms darüber bestimmen möchte, welche Klassen instanziiert werden sollen.

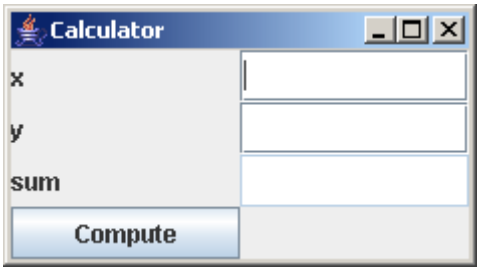
Aus der Kombination einer *Factory Method* mit einer *Strategy* lässt sich eine flexiblere Factory-Implementierung ableiten: Die *Abstract Factory*.

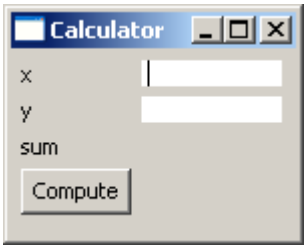
Dieses Design Pattern sollen Sie sich mit den folgenden Aufgaben erarbeiten.

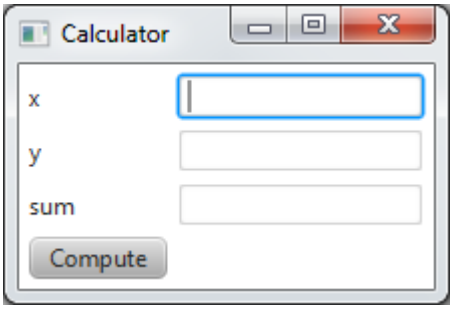
Ausgangslage

Sie finden im Wochen-Projekt zum Thema Factory einen einfachen Rechner (der tatsächlich nur zwei Zahlen addieren kann – aber die eigentliche Anwendung ist für uns nebensächlich und deswegen einfach gehalten). Die Implementierung des Rechners benutzt eine als Interface ausgeprägte Abstraktion `Components`, um damit einen GUI-Dialog zu bauen. Implementierungen dieser Abstraktion stehen basierend auf den GUI Toolkits *AWT*, *Swing*, *SWT* und *FX* zur Verfügung.









Die Klasse `Gui01FactoryMethods` enthält eine lauffähige Implementierung des Rechners, basierend auf einfachen *Factory Methods*. Diese entscheiden jeweils über eine `switch`-Anweisung, welche konkreten *Components* instanziiert werden müssen.

Aufgaben

Studieren Sie die Klasse `Gui01FactoryMethods`. Riechen Sie den Code Smell? Besonders mühsam war es als wir letzthin die FX-Variante als weitere Möglichkeit hinzugefügt haben.

Als Gegenmittel bei dieser Art von Code Smell haben Sie das *Strategy Pattern* kennengelernt. Wenden Sie dieses nun an, um die Erzeugung der *Components* besser zu gestalten.

a) Zeichnen Sie ein vereinfachtes Klassen-Diagramm basierend auf der abgegebenen Implementierung von `Gui01FactoryMethods`. *Vereinfacht* bedeutet dabei, dass das allgemeine Schema erkennbar sein soll, Sie aber nicht alle parallel existierenden Klassen explizit darstellen müssen.

b) Zeichnen Sie ein auf die konkrete Anwendung angepasstes Klassen-Diagramm des *Strategy Patterns*. Überlegen Sie sich dabei, welche Methoden in der Strategie-Schnittstelle für diesen Fall definiert werden müssen.

c) Implementieren Sie Ihren in b) gefundenen Entwurf in einer eigenen Rechner-Klasse und testen Sie ihn mit verschiedenen GUI-Varianten.

d) Es ist schwierig, die Kopplung an die konkreten Components vollständig zu vermeiden, denn irgendwie muss ja die zu verwendende Strategie „gesetzt" werden. Welche Möglichkeiten fallen Ihnen dafür ein? Skizzieren Sie Ihre Ideen und kennzeichnen Sie, wo etwas verändert werden muss, um eine andere Strategie zu verwenden.

e) Welche Klassen müssen Sie anpassen bzw. neu hinzufügen, um eine neue GUI-Technologie zu unterstützen?

f) Angenommen, wir möchten neben Fenstern (`Frame`), Feldern (`Field`), Schaltflächen (`Button`) und Labels (`Label`) neu auch Checkboxen unterstützen: Welche Klassen müssen Sie dazu anpassen bzw. neu hinzufügen?

Bemerkung zu SWT:

SWT (Standard Widget Toolkit) ist ein GUI-Toolkit das im Kontext von Eclipse entwickelt worden ist. Die SWT-Implementierung unserer Komponenten kann erst dann fehlerfrei übersetzt werden, wenn auch das entsprechende `swt.jar` File auf dem Klassenpfad aufgenommen worden ist. Wir haben dazu in das build.gradle-File ein Plugin eingebunden, das diese plattformspezifischen Abhängigkeiten in ihr Projekt laden sollte. Falls dies jedoch nicht funktioniert, können Sie diesen Fehler auch einfach ignorieren oder die Klasse `ComponentsSWT` einfach löschen (zusammen mit den entsprechenden `cases` in den `switch`-Anweisungen in der Klasse `Gui01FactoryMethods`).

Falls Sie nicht Gradle verwenden und SWT dennoch ausprobieren wollen, so müssen Sie das entsprechende `swt.jar` File für ihr Betriebssystem bereitstellen und auf den Klassenpfad aufnehmen. Gehen Sie dazu auf die Seite

http://download.eclipse.org/eclipse/downloads/,

wählen Sie eine Eclipse-Version (z.B. 4.9) und suchen Sie nach SWT und laden Sie entsprechenden Release herunter. Für Version 4.9 stehen folgende Downloads zur Verfügung:

- swt-4.9-win32-win32-x86.zip — Windows (32 bit version)
- swt-4.9-win32-win32-x86_64.zip — Windows (64 bit version)
- swt-4.9-gtk-linux-x86.zip — Linux (32 bit version)
- swt-4.9-gtk-linux-x86_64.zip — Linux (64 bit version)
- swt-4.9-gtk-linux-ppc64le.zip — Linux (64 bit version for Power PC)
- swt-4.9-cocoa-macosx-x86_64.zip — Mac OSX (64 bit version)

Packen Sie das entsprechende zip-File aus und fügen Sie das im zip-File enthaltene `swt.jar` File ihrem Klassenpfad hinzu.

Falls Sie mit Eclipse arbeiten, können Sie auch direkt das richtige zip-File aus dem plugins-Verzeichnis ihrer Eclipse-Installation als externes jar File auf den Klassenpfad aufnehmen, unter Windows (64bit)/Photon z.B. die Datei:

`eclipse\plugins\org.eclipse.swt.win32.win32.x86_64_3.108.0.v20180904-1901`.