CAL - QUARK

HaL 4, 12. Juni 2009 in Halle

Alf Richter

Eine Praktische Einführung mit
Eclipse, Excel-Reader, XML- und SQL-Builder

iba Consulting Gesellschaft
for intelligent business architecture

Was ist CAL? 1/2

- CAL wurde seit 1999 von Business Objects (Firma 2008 gekauft von SAP) mit folgenden Zielen entwickelt:
  - "Business Logic" als wiederverwendbare und kombinierbare Komponenten (funktional)
  - in Java einbettbare Lösung, welche nahtlos mit anderen Java Objekten und der Applikationslogik verbunden werden kann.
  - Java als Applikationsplattform (GUI, Applikationserver, Datenbanken, ...) erweitert um funktionalen Kern für knackige Probleme.
  - Komfortabler Zugang zur getypten funktionalen Welt für den durchschnittlichen (Java) Entwickler
- Seit 2007 ist CAL Compiler/IDE/Quark Open Source unter BSD Lizenz!

Was ist CAL? 2/2

- lazy, streng getypt mit viel Ähnlichkeit zu Haskell
- volle, dynamische Kontrolle von CAL-Funktionskomposition - und -erzeugung aus Java (ohne CAL)
- Kompiler der effizienten Java Byte Code erzeugt
- Debugging: Wertinspektion ohne erzwungene Evaluation (Laziness bleibt erhalten)
- Multithreaded Kompiler und Laufzeitumgebung
- Kontrolle der Evaluation von Java aus: suspendieren, fortsetzen, Introspektion von Teilergebnissen mit Just-In-Time Evaluation

Warum CAL? 1/2

- läuft auf der **Java Virtual Machine**
  - CAL kann **Java Bibliotheken** benutzen (z.B. Excel r/w, PDF Erstellung, JDBC)
  - **Portierbarkeit**
  - **Einbettbarkeit** in Java Anwendungen, CAL Compiler und Interpreter in Java geschrieben
- gutes Basis Framework **QUARK** mit **sehr guter Dokumentation**
  - Standard Libs: (von Haskell) Prelude, Monad, Pretty Printer, Parsec...
  - Erweiterte Libs: XML - und SQL-Builder (u.a. Oracle, MS-SQL) und einfacher SQL Parser

Warum CAL? 2/2

- **Entwicklungsumgebung**: **Eclipse Plugin** und ice (CLI: Interpreter und Compiler-Tools, vgl. ghci)
- **leichte Verteilung** von CAL Anwendungen als JAR Archive, sonst nichts weiter notwendig.
- Erhöhung der Akzeptanz durch Plattform Unabhängigkeit und Sicherheit von JAVA, selbst in Konservativen Umgebungen (Strenge IT Policies)
- leichte **Erlernbarkeit**, vereinfachter Haskell Syntax

CAL vs. Haskell

- Expression Style (kein f 2 2= 3 ; f 3 _ = 4;)
- keine do Notation
- kein Layout Rule (sondern immer ;)
- keine Operatorüberladung (>>= , >>, wie in Java)
- Java Kommentarezeichen (/* */ und //)
- gute Funktions-, Typ- und Typklassennamen: dup vs. removeDuplicates, Monoid vs. Appendable, null vs. isEmpty
- nicht pur (Funktionen können Seiteneffekt haben), notwendig um mit Java Objekten (mutable) arbeiten zu können => verbergen der Seiteneffekte in privaten Funktionen hinter öffentlichen und puren Funktionen, leicht in CAL durch Kontrolle der Laziness und Ausführungsreihenfolge (seq, deepSeq, eager keyword ..)
- Keine Multiparameter Type Class

CAL <= Haskell

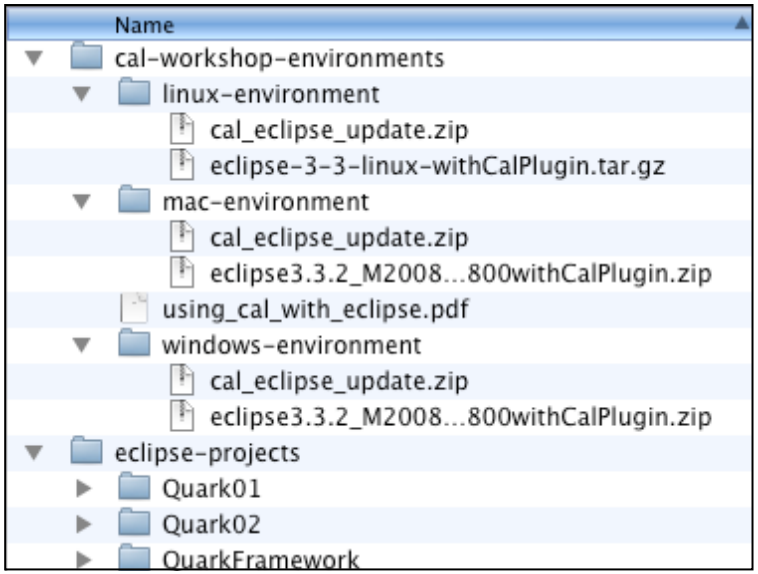
- CAL has the following features from Haskell:
  - algebraic functions with parametric polymorphism and inferred types
  - data declarations for algebraic types, strictness flags for data constructor arguments
  - expression syntax supporting if-then-else, case, let (for both local variable and function definitions) and lambda expressions
  - support for most of Haskell's expression operators
  - special syntax for tuples, strings, characters, numbers and lists
  - single parameter type classes
  - superclasses, derived instances, such as the instance declaration for Eq, List
  - deriving clauses for common classes, default class method definitions
  - higher-kinded type variables, such as with the Functor type class
  - dynamics support via the Typeable type class and Dynamic type
  - user documentation generated from source code (similar to Haddock)
  - foreign function support

Eclipse + CAL 1

- **Voraussetzungen**:
  - Java >= 1.5
  - Eclipse 3.3 oder 3.4
  - aktuelles CAL Plugin
    - 3.3 Update Site: http://resources.businessobjects.com/labs/cal/cal_eclipse_update/site.xml
    - 3.4 Update Site: http://iba-cg.de/update-site/site.xml
  - Open Quark Framework Binary Package

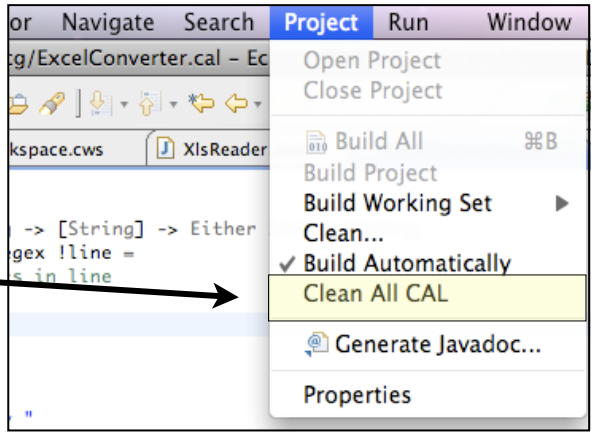
Eclipse + CAL 2

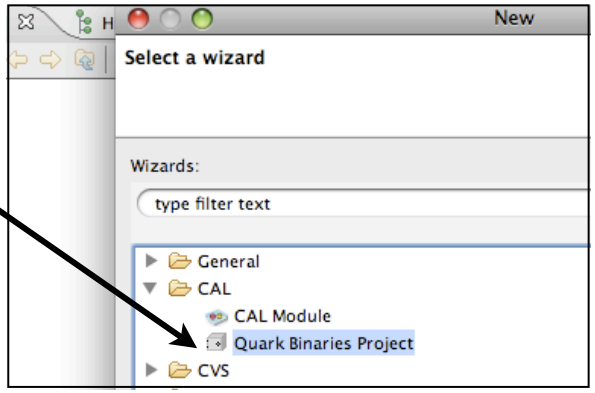
- **Inhalte der CD: (ab 20. Juni als Download)**
  - „cal-workshop-environments“: enthält Eclipse 3.3 + CAL Plugin für
    - Linux (bei evtl. Fehlern: Dateinamen auf Groß/Klein-Schreibung checken)
    - Mac OS X >=10.4,
    - Windows XP/ Vista
    - *Entsprechende Version ins Homeverzeichnis kopieren und entpacken*
  - „eclipse-projects“: (Framework und Beispiele)
    - „Quark“: Binary Package (Quark Framework)
    - „Quark01“: einfaches „Hello World“ als Ausgangspunkt für eigene Projekte
    - „Quark02“: komplexes Beispiel mit Aufruf von Java Bibliothek POI von CAL aus
    - *gesamten Ordner „eclipse-projects“ ins Homeverzeichnis kopieren*

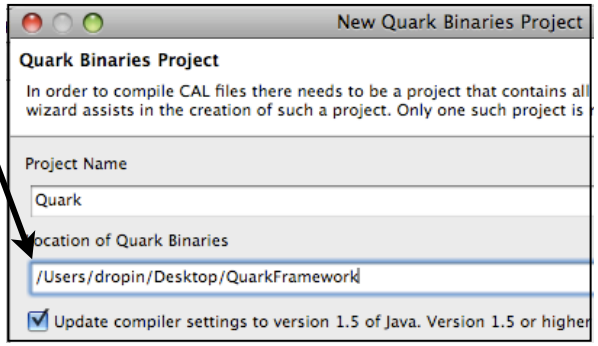


Eclipse + CAL 3

- **Eclipse starten und vorbereiten**
  - Eclipse starten
  - neuen Workspace anlegen: z.B. „hal4-workspace“
  - wenn CAL Plugin korrekt installiert muss im Menü „Project > Clean All CAL“ stehen
  - *Ergebnis: Eclipse ist mit CAL Plugin ausgestattet*
- **Neues Quark Binary in Eclipse anlegen**
  - Menü: „File > New > Other > ... > CAL/Quark Binaries Project“
  - Button: „Next“
  - Eingabe:
    - Project Name: „Quark“
    - Location: im Dateidialog „eclipse-projects/QuarkFramework“ auswählen
  - Button: „Finish“
  - *Ergebnis: Im Package Explorer ist das (Binary) Projekt „Quark“ angelegt*



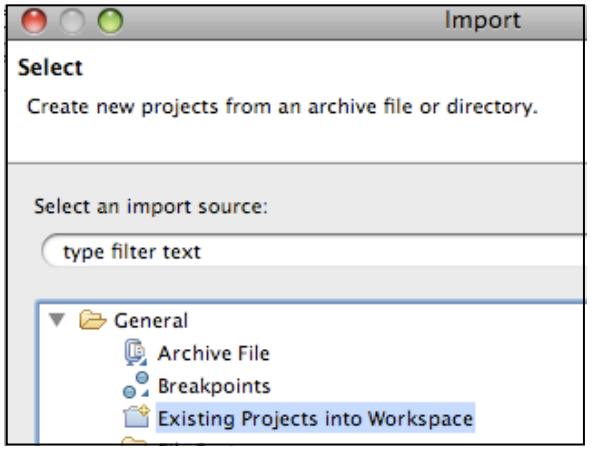


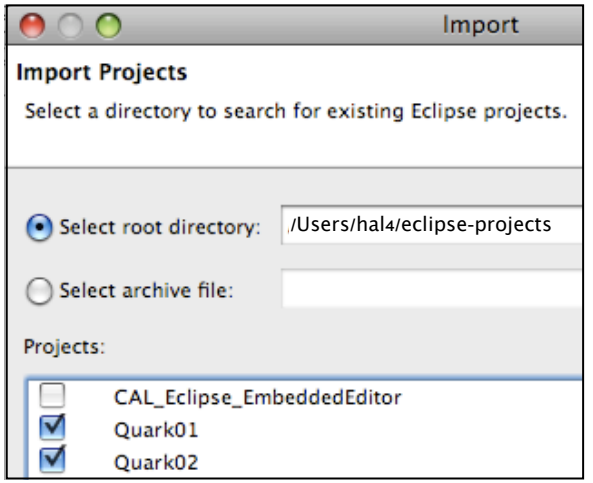


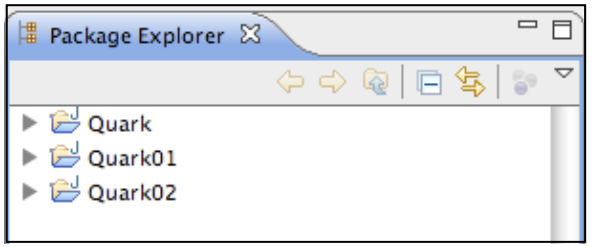
iba Consulting Gesellschaft
for intelligent business architecture

Eclipse + CAL 4

- **Beispiel Projekte importieren**
  - Menü: „File > Import ... > Generell/Existing Projects into Workspace“
  - Button: „Next“
  - Select root directory: Button: „Browse ...“
    - Dateiauswahl-Dialog: „eclipse-projects“ wählen
  - Auswählen:
    - [x] Quark01
    - [x] Quark02
  - Menü: „Project > Clean All CAL“
  - *Ergebnis: Projekte Quark01 und Quark02 in Workspace importiert*



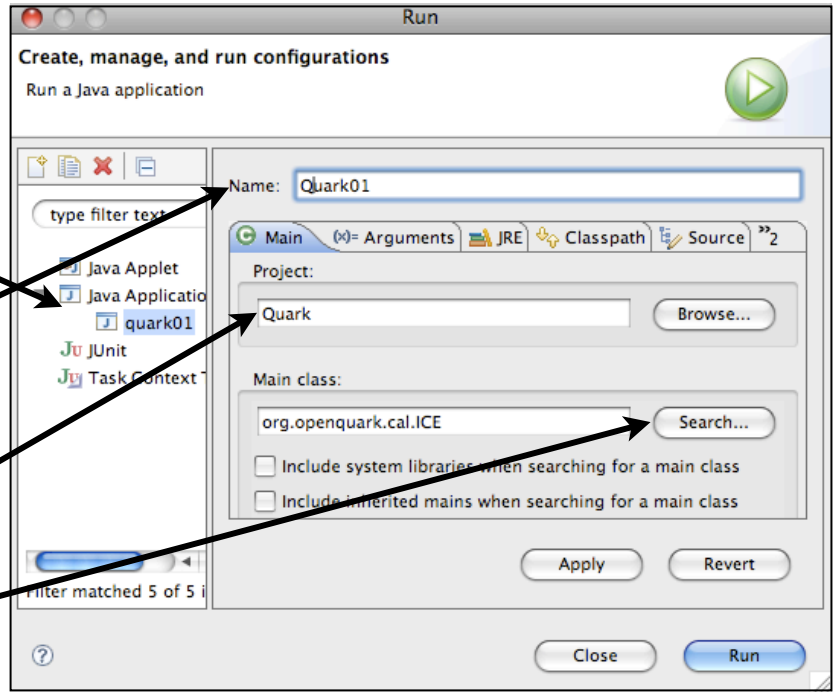




iba CONSULTING GESELLSCHAFT
for intelligent business architecture

Eclipse + CAL 5

- **Quark01 „Run Configuration“ anlegen**
  - Menü: „Run > Open Run Dialog“
  - Doppelt klicken auf „Java Application“ (hierdurch wird eine neue Run Configuration angelegt)
  - Diese Run Configuration „New_configuration“ auswählen
    - Eingabe in Feld Name: „Quark01“
    - Tab Main:
      - Project: Button „Browse ..“ , Framework „Quark“ auswählen
      - Main class: Button „Search...“
        - „ICE - org.openquark.cal“ Auswählen
    - *weiter auf nächster Folie >>>>*

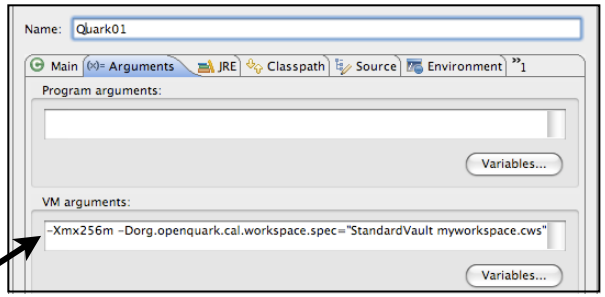


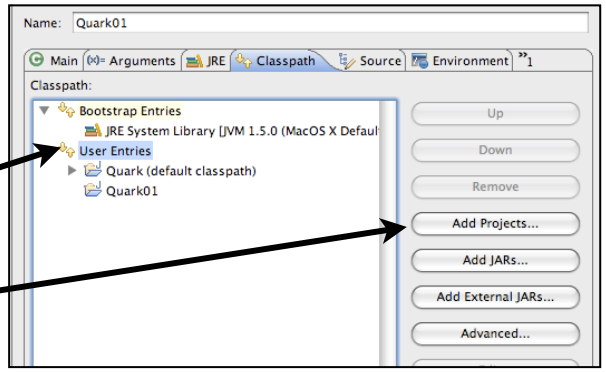
Eclipse + CAL 6

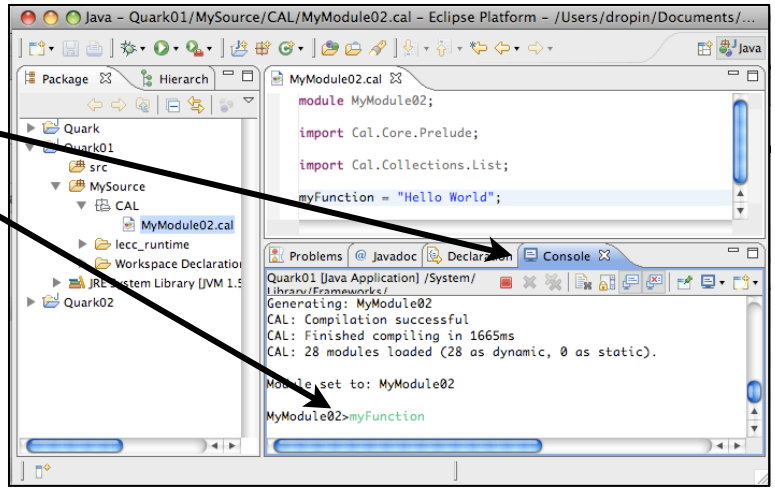
- **Quark01 „Run Configuration: Quark01“ (Fortsetzung)**
  - Tab Arguments:
    - Eingabe in VM Arguments:

```
-Xmx256m -Dorg.openquark.cal.workspace.spec="StandardVault myworkspace.cws"
```

  - Tab Classpath:
    - Auswahl: Classpath > User Entries
    - Button: „Add Projects...“
      - Auswahl „Quark01“
  - unten rechts Button: „Run“
- nun startet eine Console in Eclipse
  - Eingabe in der Console: „`myFunction`“ + RETURN
  - *Ergebnis: Neue Run Configuration für Projekt „Quark01“ angelegt und erste Funktion in Console ausgeführt*
- Dateien des Tutorials: Quark01/MySource/CAL/MyModule02.cal

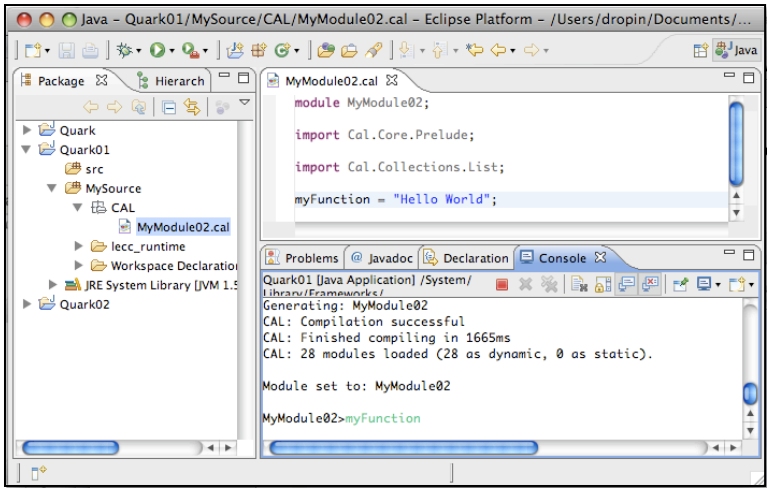






iba CONSULTING GESELLSCHAFT
for intelligent business architecture

Eclipse + CAL 7



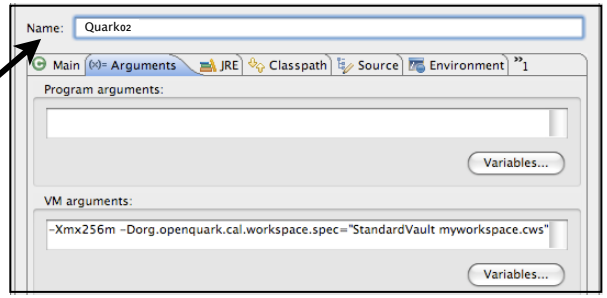
- **Die ICE Console:**
  - in der ICE Console können Funktionen getestet werden
    - wechseln in ein Modul:
      - `:sm MODULENAME`
      - z.B. `:sm MyModule02`
    - Aufruf einer Funktion: `myFunction ++ ", I am CAL!"`
      - Ausgabe: `Hello World, I am CAL!`
    - Ermitteln des Typs einer Funktion: `:t map`
      - Ausgabe: `(a -> b) -> [a] -> [b]`
    - Nach Änderungen an der Quell Datei, speichern und neu laden in ICE Console mittles: `:rc`
    - Laden eines Workspaces mittels: `:ldw hal4.cws`
    - Hilfe aufrufen: `:help`

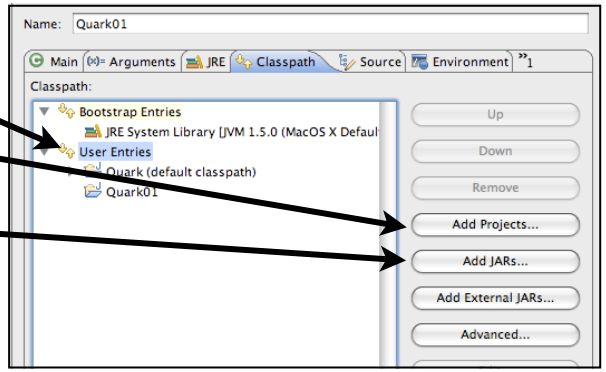
Eclipse + CAL 8

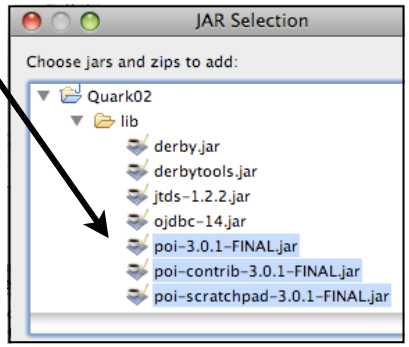
- **Quark02 „Run Configuration: Quark02“**
  - **Schritte analog zu Quark01 bis auf**
    - Name: Quark02
    - Tab Classpath:
      - Auswahl: Classpath > User Entries
      - Button: „Add Projects...“
        - Auswahl „Quark02“
      - Button: „Add Jars...“
        - Auswahl der Jars: „poi-3.0.1-FINAL“, „poi-contrib-3.0.1-FINAL“ und „poi-scratchpad-3.0.1-FINAL“ unter „Quark02/lib“
      - unten rechts Button: „Run“
- nun startet eine Console in Eclipse
  - Wechsel in das Modul mittels Eingabe in der Console:

    `:sm ExcelConverter` RETURN
  - Zum Ausführen der Beispiele Programme in der Console:

    `ex1` bis `ex15` einzeln gefolgt von RETURN eingeben
- Dateien des Tutorials: Quark02/src/main/cal/CAL/Ibacg/ExcelConverter.cal und Quark02/src/main/java/de/ibacg/xls/XlsReader.java







Hinweise zu ICE

- **Eigenarten der ICE Console:**
  - Wie in anderen Konsolen in Eclipse kann leider nicht die letzte Eingabe mittels drücken auf die Cursor-Taste „Pfeil Nach oben“ zurück geholt werden.
  - Um den gleichen Komfort wie in GHCi zu erreichen benutzen wir bei iba ein externes Terminal Programm (Mac: iTerm.app oder Terminal.app, Linux: XTerm, Konsole oder Ähnliches, für Windows: ?) in der Kombination mit rlwrap:
    - *„rlwrap is a readline wrapper, a small utility that uses the GNU readline library to allow the editing of keyboard input for any other command. It maintains a separate input history for each command, and can TAB-expand words using all previously seen words and/or a user-specified file.“*

      von http://freshmeat.net/projects/rlwrap/
    - rlwrap wird bei den meisten Linux Distributionen mit geliefert, unter Mac OS X kann es z.B. mittels port installiert werden

      ```
      sudo port install rlwrap
      ```
    - Zum starten von ICE mit rlwrap gibt es das Skript „Quark02/ice.sh“ und die Konfiguratiosdatei „Quark02/config“ im Eclipse Workspace.

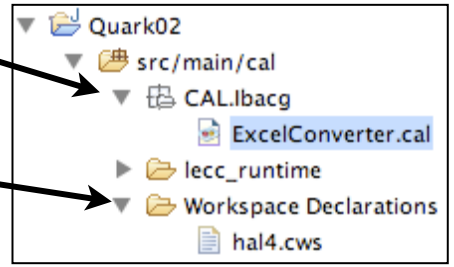
      Der entscheide Aufruf in diesem Skript ist:

      ```
      rlwrap --remember --histsize 10000 ./javacp-workspace.sh $WS
         -Dorg.openquark.cal.machine.lecc.output_directory=$OUTPUT org.openquark.cal.ICE $*
      ```
    - Mit rlwrap im Suchpfad kann auf der Kommandozeile (Bash Syntax) im Verzeichnis „eclipse-projects/Quark02“ das Skript wie folgt aufgerufen werden:

      ```
      export QUARK=../QuarkFramework   # Pfad zum Framework; kann auch in .profile eingetragen werden
      ./ice.sh
      ```

Hinweise zu CAL

- In Eclipse sind CAL Projekte normale Java Projekte mit einer sog. CAL-Nature (mit besonderen Editoren und Buildern) => es lassen sich Java und CAL Quellen in einem Projekt mischen
- CAL Quelltext muss in einer Package, welche mit Namen **CAL.** beginnt liegen (siehe rechts). Eine CAL Package muss in einem Source Folder liegen
- Projekte werden in CAL mittels einer Text Datei, der so genannte **Workspace Declaration** (Endung .cws) definiert
  - Die Workspace Declaration muss in einen Source Folder mit dem exakten Name „Workspace Declaration“ (Groß- und Kleinschreibung beachten!) liegen
  - Diese Declaration zählt alle CAL-Module auf, welche für eine Ausführung geladen werden sollen (z.B. StandardVault MyModule02)
  - In einer Workspace Declaration können andere (im Classpath liegende) Workspace Declartion importiert werden (z.B. import StandardVault cal.platform.cws)
  - Compiler und Interpreter werden mit einer Workspace Declaration gestartet (siehe Run Configuration und ./ice.sh mit ./config)




```
StandardVault MyModule02
import StandardVault cal.platform.cws
```