Advanced Programming (I00032)
Introduction to iTasks

Assignment 4

For this week it is not necessary to hand in code. The purpose of the assignment is to get the iTask system installed, and do some preliminary experiments to enhance your understanding of the system. The next assignment will actually ask you to implement a program in iTask.

1 Installing the iTask Library

Instructions to install the iTask system and to get the allExamples project running.

1. Download the snapshot of the Clean compiler, Clean IDE, and libraries from `http://wiki.clean.cs.ru.nl/ITasks` (*Snapshot for AFP2012 Course*).

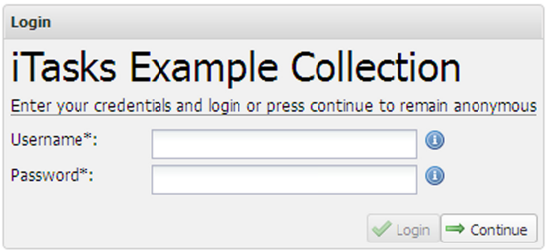
2. Unzip it to a directory of your liking. This should result in a directory with content directories *Config*, *Examples*, *Help*, *iTasks-SDK*, *Libraries*, *Temp*, *Tools*, the Clean IDE *CleanIDE.exe*, and the license conditions *CleanLicenseConditions.txt*.

3. Start the Clean IDE. Open the project *iTasks-SDK/Examples/BasicAPIExamples.prj*, bring it up-to-date, and launch it. This should create a console window with text:

   ```
   *** BasicAPIExamples HTTP server ***

   Running at http://localhost/
   ```

4. Start your favorite web-browser (note that there might be issues with IE9), and direct it to `http://localhost/`. This should open the following page:

   Press *Continue* to log in anonymously. A task gets available to edit new user accounts.

2 Making your first form with iTasks

Make your own iTask application. To prevent path problems it is most convenient to place your project in a new subfolder of `iTasks-SDK/Examples`. Your application should create a form to enter, alter, and view student data. Make sure your example contains a record, some basic types and an algebraic data type. A complete implementation looks like:

```
import iTasks

Start :: *World  → *World
Start world = startEngine
                ( manageWorklist
                  [ workflow "new student" "enter student credentials" studentTask ]
                ) world

studentTask :: Task Student
studentTask = enterInformation "Enter student credentials" []

:: Student  = // use your own definition

:: Person   = { firstName     :: String
              , surName       :: String
              , dateOfBirth   :: Date
              , gender        :: Gender
              }

:: Gender   = Male
            | Female

derive class iTask Student, Person, Gender
```

3 Working with views

Alter the program of part 2 in such a way that the empty list parameter on line 10 is filled with an `EnterOption` viewing function. Define a view that allows the user to edit only parts of your `Student` type. Experiment with the `Display` type, for instance by altering the type of the `gender` field into `Display Gender`.

4 A list of students

Alter the result type of `studentTask` into `Task [Student]` in part 2. Recompile and investigate what the effect is of this change.

5 Transforming task results

Experiment with the combinator `@?` to transform the list of students into a task that has a *Stable* result as soon as atleast 5 students have been entered.