Fakultät Informatik Institut für Software und Multimediatechnik

Lehrstuhl für Softwaretechnologie

Exercise for MOST

Role-based Programming with SCROLL

Professor: Prof. Dr. rer. nat. habil Uwe Aßmann
Lectuer: Sebastian Götz
Author: Max Leuthäuser

1 Role-based Programming

One of the fundamental concepts for separation of concerns in object-oriented systems are *roles*. They extend the classical object model in a natural way. Roles can be related to a *context* as first-class object at runtime. (For instance, the role-based language *ObjectTeams* [1] introduces *Teams*, simple runtime contexts for roles, as first-class language construct.) However, role-based separation of concerns at runtime level will fail at the moment. Many of the suggested role-based programming languages have been abandoned by their developers and do not provide a running compiler; others do not provide a runtime system compatible to one of the major platforms.[1] It is necessary to establish the basic concepts of roles and contexts with an appropriate *light-weight* tooling, available in a major programming platform so that role-based programming becomes available for the masses. Such a light-weight approach is also beneficial to support different shades of the meaning of roles and contexts [2], because it can be adapted easily by an expert programmer or language engineer.

2 Introduction to SCROLL

SCROLL[2] (SCala ROles Language) is essentially a light-weight implementation pattern for roles and context for dynamic and adaptive programming. The library allows for augmenting an object's type at runtime with dynamic role types embedded in reified contexts, so-called *compartments*. They are related to a set of roles. Entering such a context within a running system will activate its compartment and all its related roles (if they are bound to player objects). Hence, a compartment is related to a subset of a system, i.e. all its related roles, which can be switched on and off by its activation or deactivation.

[1]ObjectTeams forms a notable exception in both points.
[2]https://www.github.com/max-leuthaeuser/SCROLL

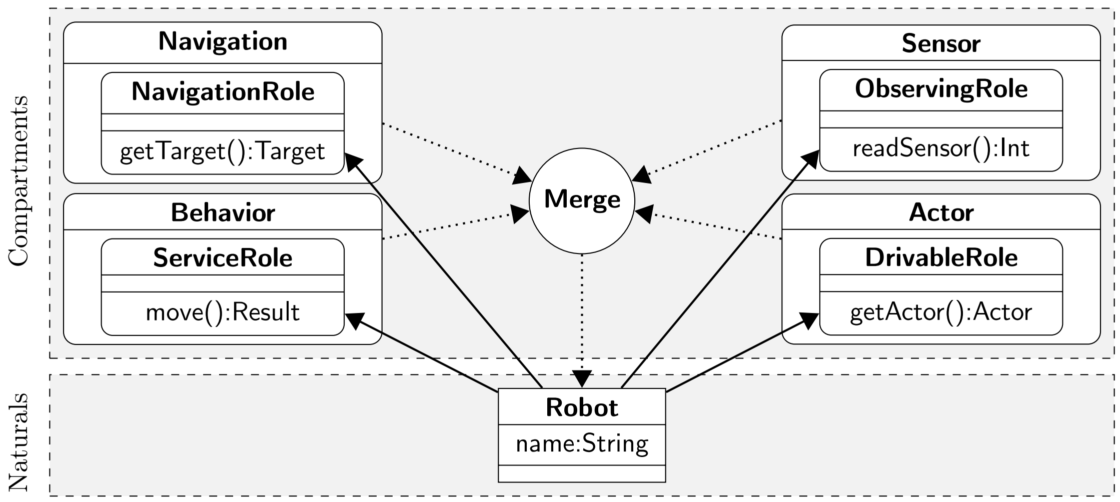

Figure 1: Class `Robot` is constructed (dotted arrows) from different compartments and plays (solid arrows) the contained roles (from [3]).

3 Tasks

1. **Robot**
   This basic task focuses at the dynamic augmenting of player objects with roles i.e. adding behavior. Figure 1 gives the basic structure. The robot gets constructed at runtime by merging the corresponding roles.
   a) Use the task template project:

```
git clone http://git-st.inf.tu-dresden.de/max-leuthaeuser/most-task-scroll-robot.git
```

   b) The **README** file provided there explains how to run and test your solution.
   c) Use the following rules for the actual implementation of the role methods:
      i. `NavigationRole.getTarget():Target` - return a `Target` with name *"kitchen"*.
      ii. `ObservingRole.readSensor():Int` - always return *100*.
      iii. `DrivableRole.getActor():Actor` - return an `Actor` with name *"wheels"*.
      iv. `ServiceRole.move():Result` - return a `Result` containing the robots name, its target, the current sensor value and its actor.
   d) Implement this by adding the missing roles.
   e) Test your solution:

```
sbt test
```

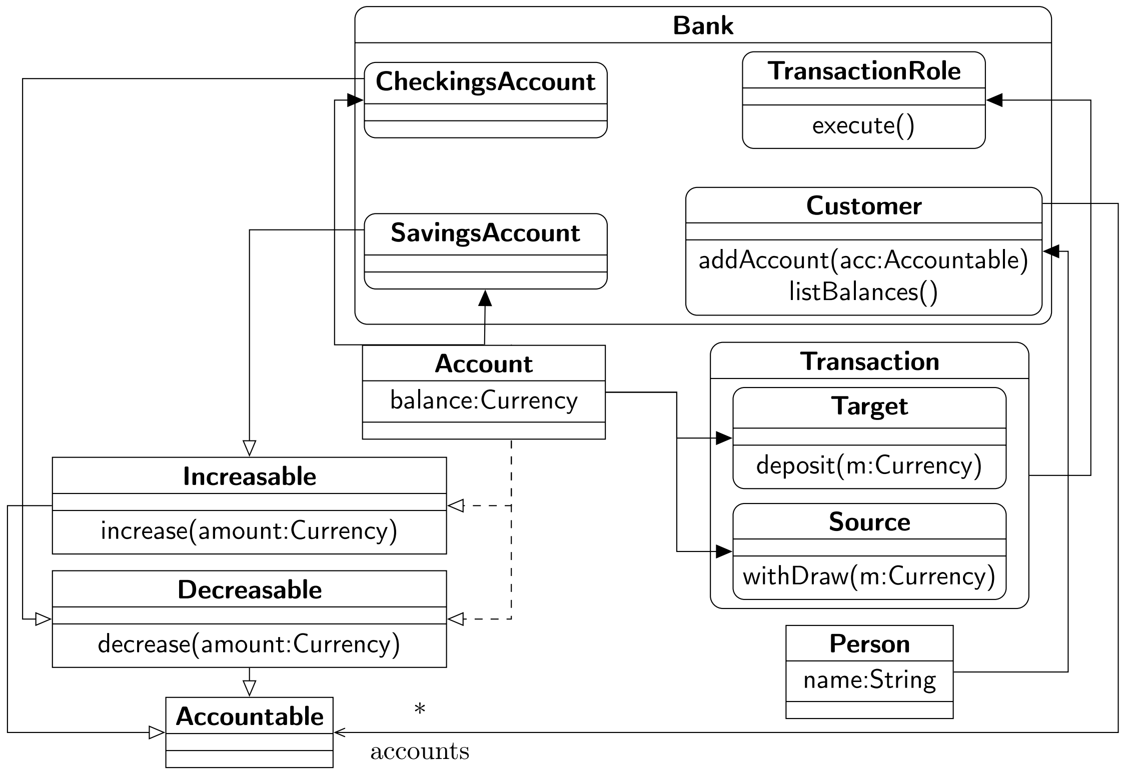

Figure 2: A `Bank` holds `Customers` and their specific `Accounts`. A `Transaction` allows for transferring money between them.

2. **Bank**
   A bank (figure 2) is a financial institution providing banking services to their customers, who are only persons here for the sake of simplicity. Customers own several saving and checking accounts, and perform transactions. Transactions encapsulate the process of transferring money from exactly one source account to one target account. They are initiated by a customer, however, managed and executed by the bank.

   a) Use the task template project:

```
git clone http://git-st.inf.tu-dresden.de/max-leuthaeuser/most-task-scroll-bank.git
```

   b) The **README** file provided there explains how to run and test your solution.

c) Use the following rules for the actual implementation of the role methods:
   i. `Source.withDraw(m:Currency)` - call the players behavior to decrease the amount of money.
   ii. `Target.deposit(m:Currency)` - call the players behavior to increase the amount of money.
   iii. `Transaction.execute()` - query for the corresponding `Source` and `Target` role instances using SCROLLs `one()` query operation[3] and withdraw/deposit the amount of money.
   iv. `CheckingsAccount.decrease(m:Currency)` - call the players behavior to decrease the amount of money.
   v. `SavingsAccount.increase(m:Currency)` - call the players behavior to increase the amount of money. Do not forget to apply a 10% transaction fee!
   vi. `TransactionRole.execute()` - call the players behaviors to execute the actual transaction.

d) Implement this by adding the missing roles and compartments.

e) At this point your solution will still get stuck in an endless loop. Why is that the case? Figure 3 might help. Provide a solution by adding an appropriate dispatch description (see `DispatchQuery`).

f) Test your solution:

```
sbt test
```

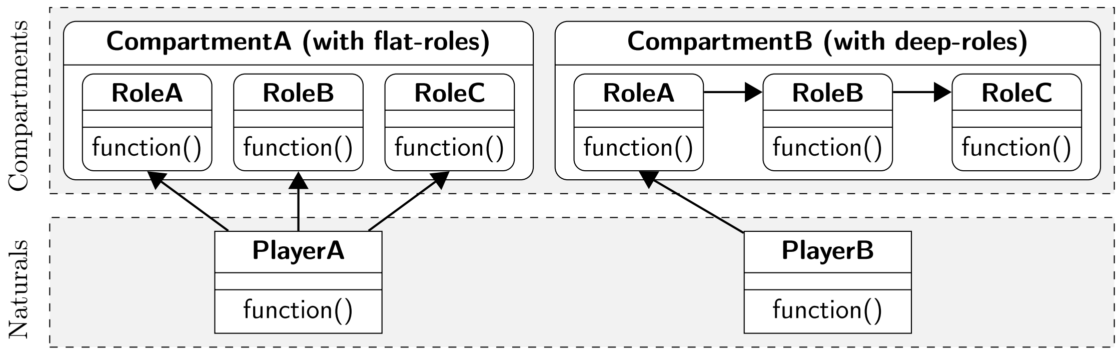

Figure 3: An example for the need of customizable role dispatch. It is ambiguous which role is responsible for answering a call to `function()`. Flat-roles (roles can not play roles themselves, left side) or deep-roles (right side) are semantically the same and are introducing the same ambiguity here (from [3]).

[3]See: http://goo.gl/6hsiLg

4 Important links

- SCROLL can be found here:
  `https://www.github.com/max-leuthaeuser/SCROLL`
- SCROLL API documentation can be found here:
  `http://max-leuthaeuser.github.io/SCROLL`
- The SCROLL Wiki contains more information about SCROLL internals:
  `https://github.com/max-leuthaeuser/SCROLL/wiki`
- You may find the implementation hints for specific role features useful:
  `https://github.com/max-leuthaeuser/SCROLL/wiki/Role-Features-and-their-Implementation-with-SCROLL`

Literatur

[1] S. Herrmann. Programming with roles in ObjectTeams/Java. Technical report, AAAI Fall Symposium, 2005.

[2] T. Kühn, M. Leuthäuser, S. Götz, C. Seidl, and U. Aßmann. A metamodel family for role-based modeling and programming languages. In B. Combemale, D. Pearce, O. Barais, and J. Vinju, editors, *Software Language Engineering*, volume 8706 of *Lecture Notes in Computer Science*, pages 141–160. Springer International Publishing, 2014.

[3] M. Leuthäuser and U. Aßmann. Enabling view-based programming with scroll: Using roles and dynamic dispatch for etablishing view-based programming. In *Proceedings of the 2015 Joint MORSE/VAO Workshop on Model-Driven Robot Software Engineering and View-based Software-Engineering*, MORSE/VAO '15, pages 25–33, New York, NY, USA, 2015. ACM.