EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch–Naturwissenschaftliche Fakultät
Fachbereich Informatik (WSI)
Praktische Informatik
Prof. Dr. Klaus Ostermann

Jonathan Brachthäuser

January 9, 2015

Homework Assignment 9

Software Design & Programming Techniques (WS2014)

Please feel free to ask questions before handing in your solutions. The deadline for this assignment is Jan 14th, 23:59. Hand in the assignments via email to jonathan.brachthaeuser@uni-tuebingen.de. Please include the names of all group members.

1 The "Smart Home" example

Earlier in the lecture we have seen the "Smart Home" example (02_cdp.pdf starting from page 29). The control system is a family of different collaborating types (Location, Room, Floor, etc.). The lecture on inheritance (04_inh.pdf starting from page 87) again showed the example in the context of "variations at the level of multiple objects". Refresh your memory and revisit both sections in the mentioned slides.

1. What are appropriate components for the example?
2. Why can the Java source code sketched in the slides lead to maintainability problems?
3. Draw an UML diagram for a solution to the smart home example, assuming inheritance between components is possible (Similar to the diagram on 04_inh.pdf, slide 80)

2 Scalable Component Abstraction

The paper "Scalable Component Abstractions" that we discussed this week proposed a technique that can be used to solve the problem.

1. What are the three language features the authors identified as crucial for the approach?
2. Given the code on the next page for the base configuration:
   (a) Visually mark and label all occurrences of the three language features in the code example.
   (b) Define the "ShutterControl" and the "LightingControl" components by inheriting from the base component. Locations in the shutter example should store a list of shutters whereas locations in the lighting example should store a list of lights. Remember that the key is to covariantly refine the abstract type members. An example for this looks like

```
trait A { type X <: Foo; trait Foo {} }
trait B extends A { type X <: Foo with Bar; trait Bar {} }
```

   Here the type `X` is "refined" to be also a subtype of `Bar` (in addition to `Foo`).
   It is also recommended to actually try your solution using the Scala compiler, since the typechecker can give you valuable feedback.

```
trait Base {
  type Location
  type Room <: Location

  type Composite <: Location with CompositeApi
  trait CompositeApi { self: Composite =>
    def locations: List[Location]
  }

  type Floor <: Composite with FloorApi
  trait FloorApi { self: Floor =>
    def rooms: List[Room]
    def locations = rooms
  }

  type House <: Composite with HouseApi
  trait HouseApi { self: House =>
    def floors: List[Floor]
    def locations = floors
  }
}
```