Prof. Dr. A. Poetzsch-Heffter
Mathias Weber, M.Sc.

University of Kaiserslautern
Department of Computer Science
Software Technology Group

Advanced Aspects of Object-Oriented Programming (SS 2013)

Practice Sheet 6 (Hints and Comments)

Exercise 1 Immutable Classes in the JDK

- String: The best know immutable class of the JDK. In fact, it is not immutable with respect to the definition of the lecture, because it depends on the global setting for the current locale.
- Number: The final classes implementing the abstract class Number. An object of Byte (Integer, etc) models exactly one value, which is not mutable. For the non-final ones a subtype could break immutability. Other wrapper classes like Character are good candidates for immutable classes as well.
- Classes of the reflection API: (Class, Package, etc) as long as the security manager is not changed.
- ...

A lot of classes are often used as if they are immutable, whereas technically they are not.

Exercise 2 Confined Types

	`ConfinedList`	`ProofTreeNode`	`PTNIterator`
C1	ok	ok	ok
C2	ok	ok	ok
C3	ok	see below	see below
C4	see below	ok	ok
C5	ok	ok	ok
C6	ok	ok	ok
C7	ok	ok	ok
C8	ok	ok	ok
Confined	no	no	no

C4 for `ConfinedList` ConfinedList inherits the method `iterator()`, which creates an object of the inner class `AbstractList.Itr` (in OpenJDK 1.6). This inner class captures the `this`-variable of `AbstractList`, the method `iterator()` can therefore not be anonymous, which breaks rule C4.

C3 for `ProofTreeNode` At line 19 in `ProofContainer` a `ProofTreeNode` is passed to a method that expects `Object`; at line 147 in `ProofTreeNode` widening to `Object` occurs.

C3 for `PTNIterator` In `ProofTreeNodeIt.getProofTreeNodes()` `PTNIterator` gets widened to `Enumeration`.

The `kacheck` tool makes the analysis based on the bytecode of the classes. Since the generic type information is erased during compilation, the list `children` in `ProofTreeNode` is taken to be a `ConfinedList<Object>`. The `add()` method takes a parameter of type `Object` then and as such the call in line 125 performs widening on an object of type `ProofTreeNode`.

The capturing of the **`this`** reference for `ConfinedList` is not detected by the tool.

Exercise 3 Encapsulation on Object Level

a) No, the annotation in not possible due to the implementation of equals. Line 507: `co.data` accesses the data field but the referenced array is part of the representation of the list `co`. Representation objects are only accessible by the owner and by objects of the same representation domain, but the representation domain of the current `this` is different from the domain of the list referenced by `co`.

b) With alias modes it is possible to control the access to objects which belong to other objects. The alias modes work on object level, i.e. an alias mode is always relative to an object. This allows to hide the representation of one object to another one, even if they are created by the same source (i.e. they are of the same class). The confined types only allow to control that all interactions that may happen with object of a confined type are coded inside the package. This guarantees, that all modifications to these objects are under the control of the package developer. In the Confined Types approach, encapsulation is seen from a static point of view, where as the alias modes (and ownership types in general) have a notion of encapsulation based on dynamic properties.