Prof. Dr. A. Poetzsch-Heffter
Dipl.-Inform. Kathrin Geilmann

University of Kaiserslautern
Department of Computer Science
Software Technology Group

Advanced Aspects of Object-Oriented Programming (SS 2010)

Practice Sheet 7 (Hints and Comments)

Exercise 1 Aliasing

An alias is called dynamic, if the involved variables belong to the stack, e.g. local variables or method parameters.

a) Desired dynamic alias: iterate through an array. The loop variable tmp aliases the array elements one after another.

```
class C {
    Object[] values;

    public void m () {
        for (Integer tmp : values) {
            ...
        }
    }
}
```

Undesired dynamic alias: Store an alias to an object in the local variable tmp; hideAll components and make the aliased one visible again. This breaks the invariant and is therefor an undesired alias.

```
class C {
    // invariant: either all components are visible or all are hidden.
    JComponent[] values;

    public void m () {
        JComponent tmp = values[0];
        hideAll();
        tmp.setVisible(true);
        }
     public void hideAll() {
        for (JCompoment c : values) {
            c.setVisible(false);
        }
    }
}
```

b) Aliasing means, that in a program state more than one reference to an object exists. Capturing occurs if a reference passed as a parameter to a method is stored in a field of the called object. If this is done, the reference remains available even after the method has been finished. I.e. Capturing happens, if a new static alias to a parameter object is established. Leaking means, that a reference to an object is passed outside. This means, that the caller of the leaking method can establish an alias to the leaked object.

c) Instead of returning the reference to the original array, return a copy of the array.

```
public Identity[] getSigners() {
    Identity[] signers = new Identity[this.signers.length];
    System.arraycopy(this.signers, 0, signers, 0, signers.length);
    return signers ;
}
```

Exercise 2 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 local.
- Number: The final classes implementing the interface 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, where as technically they are not.

Exercise 3 Confined Types

	`ConfinedList`	`ProofTreeNode`	`PTNIterator`
C1	ok	ok	ok
C2	ok	ok	ok
C3	ok	not ok, in ProofContainer.19 a ProofTreeNode is passed to a method that expects an Object; in ProofTreeNode.147 widening to Object	ok
C4	ok	ok	ok
C5	ok	not ok. in line 69 call to a unconfined, non anonymous constructor.	ok
C6	ok	ok	ok
C7	ok	ok	ok
C8	ok	ok	ok
Confined	yes	no	yes

Exercise 4 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 `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.