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

Advanced Aspects of Object-Oriented Programming (SS 2015)

Practice Sheet 5

Date of Issue: 19.05.15
Deadline: 26.05.15
(before the lecture as PDF via E-Mail)

Exercise 1   Wildcards I

a) Which of the following assignments is valid Java-Code? If an assignment is not valid, explain why.

```java
import java.util.*;

Vector<LinkedList<String>> v1 = new Vector<LinkedList<String>>();
Vector<List<String>> v2 = v1;
Vector<?> v3 = v1;
Vector<? extends List<?>> v4 = v1;
Vector<? extends List<?>> v5 = v3;
Vector<? extends List<String>> v6 = v1;
Vector<? super LinkedList<?>> v7 = v1;
```

b) Which parameter types can be passed to the method `add` and which types can be expected as return types for method `get` on the variables `l1` to `l4`?

```java
import java.util.*;

List<Number> l1;
List<? super Number> l2;
List<? extends Number> l3;
List<?> l4;
```

Exercise 2   Wildcards II

We want to implement utility functions for use with the Java Collection API. A common task is to convert between arrays and collections.

```java
import java.util.Collection;
import java.util.Iterator;

public class CollectionTools {
  public static <...> void copyFromArray(...[] arr, Collection<...> coll) {
    ...
  }

  public static <...> void copyToArray(Collection<...> coll, ...[] arr) {
    ...
  }
}
```

a) Implement the method `copyFromArray` that takes an array and a collection and adds all elements in the array to the collection. Replace the ellipsis, such that your solution supports as many reasonable scenarios of reuse as possible.

b) Implement the method `copyToArray` that takes a collection and an array and copies as many elements as possible from the collection into the array. Replace the ellipsis, such that your solution supports as many reasonable scenarios of reuse as possible.

Exercise 3   Bounded Type Parameters

Implement the class `MinMaxWrapper` as Subtype of `Set`.

```java
import java.util.*;

public class MinMaxWrapper<...> implements Set<...> {
  private Set<...> theSet;

  public MinMaxWrapper(Set<...> set) {
    this.theSet = set;
  }

  ...

  public ... getMinimum() {
    ...
  }

  public ... getMaximum() {
    ...
  }
}
```

This class should implement a wrapper for sets to compute the minimum and maximum of a set. Replace the ellipsis with reasonable code.

Exercise 4   Extended Iterators

A lot of applications display lists of data to the user, and usually the internal data representation has to be transformed into a human readable format. In such an application you may find code like this:

```java
Collection<E> c = ... // the data managed by the application

Iterator<E> iter.iterator();
while (iter.hasNext()) {
  E entry = iter.next();
  F transformed = transform(entry); // e.g. convert into human readable format
  display (transformed);
}
```

It would be nice to have a more intelligent iterator that can be parameterized with a transform function and automatically emit already transformed objects.

The resulting code could the look like:

```java
Collection<E> c = ... // the data managed by the application

// a more intelligent iterator, generics are intentionally removed
Transformer t = new FancyTransformer(); // the transform function
Iterator i = new TransformingIterator(c.iterator(), t);

while (i.hasNext()) {
  F entry = i.next();
  display (entry);
}
```

a) A TransformingIterator transforms the values of the original iterator by a transformation function before returning them. Write a generic class `TransformingIterator` and the accompanying interface `Transformer`, that decorates an existing Iterator with a transformation function. Because Java does not know higher order functions, we have to represent the transformer functions as objects of type `Transformer`.

Use the following code skeleton and replace the ellipsis, such that your solution supports as much reasonable scenarios of reuse as possible.

```java
public interface Transformer<...> {
  public ... transform(... o);
}

public class TransformingIterator <...> implements Iterator <...> {
  public TransformingIterator(Iterator<...> inputIterator, Transformer<...> t) {
```

```
        ...
    }

    // see the documentation of Iterator for the specification of these methods
    public ... hasNext() ...

    public ... next() ...

    public ... remove() ...
}
```

b) Use your iterators to write a program that manages a list of Doubles and prints all numbers truncated/extended to two positions and prefixed EUR. For example the list 19.248, 7.0, 1.8882, -0.1992 will be presented as `EUR 19.24`, `EUR 7.00`, EUR `1.88`, EUR `-0.19`.

Exercise 5   Aliasing

a) Give two examples for *dynamic* aliasing, one where aliasing is desired and one where aliasing has undesired effects.

b) Describe the relationship between capturing, leaking and aliasing.

c) Give a solution to the signers issue on slide 8 of chapter 4.