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 2011)

Practice Sheet 6

Date of Issue: 17.05.11
Deadline: 24.05.11
(until 10 a.m. as PDF via E-Mail)

Exercise 1 Questions

a) Add one or more questions to the document at `http://bit.ly/lGbs2v`.

Exercise 2 Type Erasure

```
public class ComparableTest<A,B> implements Comparable<A>,
                                            Comparable<B> {
  public int compareTo(A a) { return -1; }
  public int compareTo(B b) { return 1; }

  public static void main(String ... args) {
    ComparableTest<String,Integer> C=new ComparableTest<String,Integer>();
  }
}
```

a) Transform the given source code using the type erasure algorithm utilized by Java 5.

b) Explain why proper compilation of the source code is impossible.

Exercise 3 Generics

a) Java does not support the creation of generic arrays, i.e. `new List<E>[], new List<String>[], new E[]`, where E is a declared type variable, are illegal expressions. In order to find out the reasons for it, assume that the statement `List<String>[] stringLists = new List<String>[1];` is legal.

Write a code snippet that leads to a ClassCastException at a cast, that has been inserted by the type erasure. Your code should be legal Java except for the given statement. *Hint: Take advantage of the covariance of arrays and try to assign a list of a different type to* `stringLists[0]`.

b) Implement a generic static method `flatten` having the following signature

```
public static <...> List<...> flatten (List<...> l)
```

The method takes a list of lists of anything and flattens it by one level. Replace each ellipsis such that your implementation provides maximum re-usability without violating type correctness.

Exercise 4 Extended Iterators

A lot of applications display a list of data to the user and allow him to apply filters to this list. In such an application you may find code like this:

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

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

It would be nice have a more intelligent iterator that automatically skips the entries not matching the predicate, and one that returns the already transformed objects.

The resulting code could the look like:

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

Iterator<E> iter = c.iterator(); // the standard iterator

// a more intelligent iterator, generics are intentionally removed
Predicate p = new FancyPredicate(); // the predicate function
Transformer t = new FancyTransformer(); // the transform function
Iterator i = new TransformingIterator(new FilteringIterator(iter, p), t);

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

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.

```
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) Write a generic class FilteringIterator and the accompanying interface Predicate, that can be used as an iterator, but automatically skips the values of the underlying iterator, which do not match the predicate.

```
public interface Predicate<...> {
  public boolean evaluate(... o);
}

public class FilteringIterator <...> implements Iterator <...> {
  public FilteringIterator(Iterator<...> inputIterator, Predicate<...> p) {
    ...
  }

  ...
}
```

c) Use your iterators to write a program that manages a list of Doubles and prints all positive numbers truncated to two positions and prefixed EUR. For example the list 19.248, 7.0, -9.0, 1.8882, -0.1992 will be presented as EUR 19.24, EUR 7.00, EUR 1.88.