Implementing Package Templates in Java (provisional title)

When modelling e.g. a system of cities and roads or of water pipes and switches, it would be helpful if we could gather in a module the shared aspects of these problems e.g. as the concept of a graph with nodes and edges, so that this module later can be used to form (or could make up the kernel of) an implementation of either cities/roads or switches/pipes. To be really helpful this requires that, when such a module is used (or “instantiated”) in a program, we must be able to add new declarations at any subclass levels of the module classes, and to change names on declarations. Such a system was proposed by Krogdahl in 2001. It is later extended and the mechanism is now called Package Templates (while the textual program modules themselves are simply called templates). The system allows a limited type of multiple inheritance (by merging classes from different templates), and templates may build upon other templates to any depth). The system is not fully implemented, and there exists a number of proposals for extending it.

For a more in depth look at this concept, see *Exploring the use of Package Templates for flexible re-use of Collections of related Classes* by Krogdahl, Møller-Pedersen and Sørensen or *Package Templates: Design, Experimentation and Implementation* by Axelsen.

Scope of the thesis

Initially the main focus of the thesis will be on implementing the basic parts of Package Templates in Java, hereinafter reffered to as PTj. This will be done through making a compiler for this extension of Java, which will either transpile to regular Java code or compile to JVM byte code.

Once the basic parts of PTj have been implemented the master thesis will likely continue in one of the following paths:

A pure implementation of PTj

Here the focus will be on making a more complete implementation of PTj. A full complete implementation of PTj will most likely be larger than the scope of this thesis, so here part of the thesis will focus on which parts of PTj are most essential for a more complete implementation.

Utilize PTj to make an easily extendable compiler

In this path we will use PTj to make an extendable compiler. The focus here is to perform a case study on the usability of PTj in the expression problem as described in *Exploring the Use of Package Templates for Flexible Re-use of Collections of Related Classes*.

Evaluating the usefulness of PTj in existing open source projects

This will also be a case study on PTj and if it can improve on existing real world examples. Here we will evaluate the refactored examples on several metrics such as type-safety, scalability, readability, etc.