Scholarship Skills, Winter 2006
Exercise 10 -- Shorten Paragraphs

Due February 16, 2006

The goal is to remove 20% of the text (approximately 60 words) from the following paragraphs while preserving as much of the meaning as possible. You can perform "grammatical repair." Give your answer by marking up this page directly (or a copy). [Note that we are not implying there is anything wrong with the paragraphs as is, nor do we expect the "trimmed" paragraphs to be wonderful prose.]
From: Z. Budimlic and K. Kennedy, Optimizing Java: Theory and Practice, *Concurrency: Practice and Experience* 9(6), June 1997, pages 445-463.

4. Relaxed Model

A more effective approach to optimizing Java would be possible if we could relax some rules of the current Java compilation model. In particular, given the standard Java object-oriented coding style—frequent method calls, many short methods—interprocedural optimization techniques such as aggressive method inlining could lead to substantive performance gains.

Unfortunately, interprocedural analysis as described in the literature for conventional languages is not permissible with the existing Java compilation and execution model, except with **final** classes and methods. The reason is that the compiler must assume late binding of the method calls inside the same class as well as to the methods from other objects. This is a consequence of not having the whole program in hand at compilation time.

However, if we change the model, some interprocedural optimizations become possible, *Object inlining*, described in the next section, looks at all the classes that are available to the compiler and, whenever it can prove that a method invocation must be static, inlines whole objects. *Code duplication* relies on the run-time mechanism to distinguish between the optimized and non-optimized classes and to decide which one to use for instantiation and which one for inheritance.

4.1. Object Inlining

One of the simplest and the most effective interprocedural optimizations is inlining. It has two major positive effects on the compiled code: elimination of subroutine call overhead and exposure of the method body to further optimization in the context of the original invocation. The potential negative effect is explosion of code size and the corresponding increases in compilation times. Intuitively, inlining would be most effective for code that has many subroutine calls and short subroutines. Thus, object oriented languages and programs written in object oriented style would profit the most from this optimization. Many of the current C++ compilers implement extensive inlining and achieve significant performance improvements as a result.