Homework 2: Advanced OO Concepts and Language Design

Due: problem 1, October 26, with a meeting to discuss it beforehand before October 16.

The purpose of this homework is to explore language design in the OO paradigm. In addition, I hope this will increase your understanding of the tradeoffs in language design, and increase your critical thinking skills.

Problem 1 is to be done in groups. Follow the course policies, in particular, be sure to have a new leader, write out the certification required, and hand in a list of what worked well and what didn't.

Readings: See the books on reserve for the course, especially: *The Java Language Specification* and *The C++ Programming Language*.

1. (200 points) In this part you will examine the differences between the OO languages Smalltalk, C++, and Java, and put into Java some of the good points of Smalltalk and C++, while retaining its essential character. In particular, you are to:

   - Write a short summary of the goals of C++ and Java, and what differences there are between the goals of the two languages.
   - Write a short summary of the differences between the goals of Smalltalk and those of Java.
   - Write a description of the significant differences between C++ and Java. Focus on significant semantic differences, and not so much on syntactic issues. For each difference, describe why you think the difference arose.
   - Write a description of the main differences between Smalltalk and Java. Again, focus on significant semantic differences, and not so much on syntactic issues. You do not need to explain why these differences arose.
   - What practical problems does Java have as a language (perhaps shared with C++ and Smalltalk) that should be fixed in a redesign? In particular, consider the perspective of using Java to develop component-based software.
   - Describe a set of language design changes to Java that would incorporate into Java the best features of C++ and Smalltalk. For each change to Java, explain the benefits of the change, give a revised syntax, and explain the semantics by precise English descriptions and examples. Be sure to include examples of each change that justify the benefits you are claiming. Discuss any tradeoffs involved

     To limit the size of the changes, don't make changes to Java that aren't necessary, and leave the basic syntax alone. I advise making only minimal changes to the type system, but you can make changes to the type system if you wish. Try to keep Java simple, by taking out features if you put in others. Don't worry about changing minor details of Java, such as the semantics of integers, unless you can justify the importance of such changes.
   - Also describe a set of language design changes that seem, at first, to be reasonable, but which you did *not* make in Java, and briefly explain why you left them out. Include examples.
   - Discuss Java's support for component-based programming, and any improvements that might be made to it.

2. (100 points, extra credit) This problem can be done either individually or in groups.

The Java type system makes many comparisons "by name." For example, a class C only implements an interface I if the declaration of C names I, and a class C_1 is a subtype of C_2 only if the declaration of C_1 says that C_1 extends C_2. How does this make independent development of programs more difficult? What impact does it have on component-based programming in Java? To help explore these issues: (a) create some convincing examples that show how this aspect of the Java type system causes problems, (b) change the type system to get rid of the problems. Does your change still leave Java with a sound type system? Does your change affect the security of Java?