Implementation Plan

Test-driven approach
- Write test cases for
  - one AST node at a time
  - one type check at a time
  - one possible type at a time (start with atomic types)
- Set up a regression testing script
  - capture your compiler output on test case to a temp file
  - compare output to a handwritten output for test case
- Implement
  - one AST node at a time
  - one type check at a time
  - one possible type at a time (start with atomic types)

Advantages
- turn in your program at any point to get partial credit
- separate two most difficult pieces: understanding MiniJava typing and implementing the typecheck with the provided data structures

Total points so far (32% of the final grade)

Plan for Today

PA5 status
- read emails going to mailing list
  - specification for lines and positions was fixed to match example output
  - isSameOrExtends implementation was buggy
- consider working with a partner
- implement and test one semantic error at a time

Error Recovery during
- lexical analysis
- syntax analysis
- semantic analysis

Implementing type checking for MiniJava (Slide 3)

Syntax
AST production AST node

Errors
- [LINENUM,POSNUM] Class CLASSNAME does not exist
- [LINENUM,POSNUM] Receiver of method call must be a class type
- [LINENUM,POSNUM] Method METHODNAME does not exist in class type CLASSNAME
- [LINENUM,POSNUM] Method METHODNAME requires exactly NUM arguments
- [LINENUM,POSNUM] Invalid argument type for method METHODNAME
Error Handling Goals

Provide program with a list of as many errors as possible

Provide USEFUL error messages
  – appropriate line and position information
  – guidance for fixing the error

Avoid infinite loops or recursion

Find “all” errors in program before translation begins

Error handling in the SableCC lexer

Input

3_id454 @

Lexer only

> java MainLexer t < temp.java
minijava.node.TTIntegerLiteral 3
[1,2] Unknown token: _

Parser only

minijava.parser.ParserException: [1,1] expecting: 'class'
at minijava.parser.Parser.parse(PARSER.java:599)
at SemanticAnalysis.main(SemanticAnalysis.java:64)

Predictive parser for midterm review example

```java
void S() { switch (tok) {
  case ID:
      case EOF:// the 2 characters in the FIRST(Stmlist EOF)
    Stmlist(); eat(EOF); break;
    default: print "error"; error($); break;
  }
}
void Stmlist() { switch (tok) {
  case ID: // FIRST(Stm Stmlist ) = { ID }
    Stm(); Stmlist(); break;
  case EOF: // FOLLOW(Stmlist) = { EOF }
    break;
    default:print "error"; error(Stmlist); break;
  }
}
void Stm() { switch (tok) {
  case ID: 
    eat(id,Stm); eat(assign,Stm); eat(float,Stm);
    break;
    default:print "error"; error(Stm); break;
  }
```