```
 1 package test;
 2
 3 import static org.junit.Assert.fail;
 4
 5 import java.io.StringReader;
 6
 7 import lexer.Lexer;
 8
 9 import org.junit.Test;
10
11 import parser.Parser;
12 import ast.CompilerError;
13 import ast.List;
14 import ast.Module;
15 import ast.Program;
16
17 public class SemanticAnalyserTests {
18     private void runtest(String... srcs) {
19         runtest(true, srcs);
20     }
21
22     private void runtest(boolean succeed, String... srcs) {
23         Parser parser = new Parser();
24         try {
25             List<Module> modules = new List<Module>();
26             for(String src : srcs)
27                 modules.add((Module)parser.parse(new Lexer(new
   StringReader(src))));
28             Program prog = new Program(modules);
29             prog.namecheck();
30             if(!prog.hasErrors()) prog.typecheck();
31             if(succeed) {
32                 if(prog.hasErrors()) {
33                     StringBuffer errors = new StringBuffer();
34                     for(CompilerError err : prog.getErrors())
35                         errors.append("\n" + err);
36                     fail("Program was expected to pass semantic checks
   , but the following " +
37                          "errors were reported:" + errors);
38                 }
39             } else if(!prog.hasErrors()) {
40                 fail("Program was expected to fail semantic checks,
   but passed.");
41             }
42         } catch (Throwable e) {
43             if(e instanceof AssertionError) {
44                 throw (AssertionError)e;
45             } else {
46                 e.printStackTrace();
47                 if(succeed)
```

```
48                      fail(e.getMessage());
49              }
50          }
51      }
52
53      @Test
54      public void testModuleNameClash() {
55          runtest(false,
56                  "module M { }",
57                  "module M { }");
58      }
59
60      @Test
61      public void testModuleNameNoClash() {
62          runtest("module M { }",
63                  "module N { }");
64      }
65
66      @Test
67      public void testUnresolvedImport() {
68          runtest(false,
69                  "module M { import N; }");
70      }
71
72      @Test
73      public void testResolvedImport() {
74          runtest("module M { import N; }",
75                  "module N { }");
76      }
77
78      @Test
79      public void testSelfImport() {
80          runtest(false, "module M { import M; }");
81      }
82
83      @Test
84      public void testFunctionNameClash() {
85          runtest(false,
86                  "module M {" +
87                  "  void foo() { }" +
88                  "  void foo() { }" +
89                  "}");
90      }
91
92      @Test
93      public void testFunctionNoNameClash() {
94          runtest("module M {" +
95                  "  void foo() { }" +
96                  "  void bar() { }" +
97                  "}");
```

```
 98     }
 99
100     @Test
101     public void testFunctionShadowingImport() {
102         runtest("module M {" +
103                 "  import N;" +
104                 "  void foo() {}" +
105                 "}",
106                 "module N {" +
107                 "  public void foo() {}" +
108                 "}");
109     }
110
111     @Test
112     public void testTypeNameClash() {
113         runtest(false,
114                 "module M {" +
115                 "   type foo = \"int\";" +
116                 "  int bar;" +
117                 "  type foo = \"boolean\";"   +
118                 "}");
119     }
120
121         @Test
122     public void testTypeNameNoClash() {
123         runtest("module M {" +
124                 "   type foo = \"int\";" +
125                 "  int bar;" +
126                 "  foo moo;" +
127                 "  type bar = \"boolean\";"   +
128                 "}");
129     }
130
131     @Test
132     public void testTypeShadowingImport() {
133         runtest("module M {" +
134                 "  import N;" +
135                 "  type foo = \"int\";" +
136                 "}",
137                 "module N {" +
138                 "  public type foo = \"boolean\";" +
139                 "}");
140     }
141
142     @Test
143     public void testVarNameClash() {
144         runtest(false,
145                 "module M {" +
146                 "  void foo() {" +
147                 "    int x;" +
```

```
148                 "    boolean x;" +
149                 "  }" +
150                 "}");
151     }
152
153     @Test
154     public void testFieldNameClash() {
155         runtest("module M {" +
156                 "  int x;" +
157                 "  void foo() {" +
158                 "    boolean x;" +
159                 "  }" +
160                 "}");
161     }
162
163     @Test
164     public void testFieldNameShadowingImport() {
165         runtest("module M {" +
166                 "  import N;" +
167                 "  int x;" +
168                 "}",
169                 "module N {" +
170                 "  public int x;" +
171                 "}");
172     }
173
174     @Test
175     public void testVarParameterNoNameClash() {
176         runtest("module M {" +
177                 "  void foo(int x) {" +
178                 "    boolean x;" +
179                 "  }" +
180                 "}");
181     }
182
183     @Test
184     public void testParameterNameClash() {
185         runtest(false,
186                 "module M {" +
187                 "  void foo(int x; boolean x) {" +
188                 "  }" +
189                 "}");
190     }
191
192     @Test
193     public void testVarFieldNoNameClash() {
194         runtest("module M {" +
195                 "  int x;" +
196                 "  void foo() {" +
197                 "    boolean x;" +
```

```
198                 "  }" +
199                 "}");
200     }
201 
202     @Test
203     public void testVarNoNameClash() {
204         runtest("module M {" +
205                 "  void foo() {" +
206                 "    int x;" +
207                 "    {" +
208                 "      boolean x;" +
209                 "    }" +
210                 "  }" +
211                 "}");
212     }
213 
214     @Test
215     public void testUndeclaredVariable() {
216         runtest(false,
217                 "module M {" +
218                     "  int foo() {" +
219                 "    return res;" +
220                 "  }" +
221                 "}");
222     }
223 
224     @Test
225     public void testUndeclaredFunctionName() {
226         runtest(false,
227                 "module M {" +
228                     "  int foo() {" +
229                 "    return bar(0);" +
230                 "  }" +
231                 "}");
232     }
233 
234     @Test
235     public void testUndefinedUserType() {
236         runtest(false,
237                 "module M {" +
238                 "  foo bar;" +
239                 "}");
240     }
241 
242     @Test
243     public void testBreakOutsideLoop() {
244         runtest(false,
245                 "module M {" +
246                 "  void foo() {" +
247                 "    break;" +
```

```
248                 "  }" +
249                 "}");
250     }
251 
252     @Test
253     public void testBreakInsideLoop() {
254         runtest("module M {" +
255                 "  void foo() {" +
256                 "    while(true)" +
257                 "      break;" +
258                 "  }" +
259                 "}");
260     }
261 
262     @Test
263     public void testBreakNestedInsideLoop() {
264         runtest("module M {" +
265                 "  void foo() {" +
266                 "    while(true) {" +
267                 "      if(true) {" +
268                 "        { break; }" +
269                 "      }" +
270                 "    }" +
271                 "  }" +
272                 "}");
273     }
274 
275     @Test
276     public void testBreakNestedOutsideLoop() {
277         runtest(false,
278                 "module M {" +
279                 "  void foo() {" +
280                 "    while(true) {}" +
281                 "    if(true) {" +
282                 "      { break; }" +
283                 "    }" +
284                 "  }" +
285                 "}");
286     }
287 
288     @Test
289     public void testLookupLocal() {
290         runtest("module M {" +
291                 "  int foo() {" +
292                 "    int res;" +
293                 "    res = 23;" +
294                 "    return res;" +
295                 "  }" +
296                 "}");
297     }
```

```
298
299     @Test
300     public void testLookupLocalBad() {
301         runtest(false,
302                 "module M {" +
303                 "  void foo() {" +
304                 "    res = 23;" +
305                 "    int res;" +
306                 "  }" +
307                 "}");
308     }
309
310     @Test
311     public void testLookupLocalNested() {
312         runtest("module M {" +
313                 "  int foo() {" +
314                 "    int res;" +
315                 "    res = 23;" +
316                 "    if(true)" +
317                 "      return res;" +
318                 "    return 0;" +
319                 "  }" +
320                 "}");
321     }
322
323     @Test
324     public void testLookupFunction() {
325         runtest("module M {" +
326                 "  int bar() { return 23; }" +
327                 "  int foo() {" +
328                 "    return bar();" +
329                 "  }" +
330                 "}");
331     }
332
333     @Test
334     public void testLookupImportedFunction() {
335         runtest("module M {" +
336                 "  import N;" +
337                 "  int foo() {" +
338                 "    return bar();" +
339                 "  }" +
340                 "}",
341                 "module N {" +
342                 "  public int bar() { return 23; }" +
343                 "}");
344     }
345
346      @Test
347     public void testLookupImportedFunctionFail() {
```

```
348         runtest(false,
349                 "module M {" +
350                 "  import N;" +
351                 "  int foo() {" +
352                 "    return bar();" +
353                 "  }" +
354                 "}",
355                 "module N {" +
356                 "  int bar() { return 23; }" +
357                 "}");
358     }
359
360     @Test
361     public void testLookupField() {
362         runtest("module M {" +
363                 "  int res;" +
364                 "  int foo() {" +
365                 "    return res;" +
366                 "  }" +
367                 "}");
368     }
369
370     @Test
371     public void testLookupImportedField() {
372         runtest("module M {" +
373                 "  import N;" +
374                 "  int foo() {" +
375                 "    return res;" +
376                 "  }" +
377                 "}",
378                 "module N {" +
379                 "  public int res;" +
380                 "}");
381     }
382
383     @Test
384     public void testLookupImportedFieldFail() {
385         runtest(false,
386                 "module M {" +
387                 "  import N;" +
388                 "  int foo() {" +
389                 "    return res;" +
390                 "  }" +
391                 "}",
392                 "module N {" +
393                 "  int res;" +
394                 "}");
395     }
396
397     @Test
```

```
398     public void testVoidField() {
399         runtest(false,
400                 "module M {" +
401                 "  void foo;" +
402                 "}");
403     }
404 
405     @Test
406     public void testVoidParameter() {
407         runtest(false,
408                 "module M {" +
409                 "  int foo(void x) {}" +
410                 "}");
411     }
412 
413     @Test
414     public void testVoidLocal() {
415         runtest(false,
416                 "module M {" +
417                 "  int foo() { void x; }" +
418                 "}");
419     }
420 
421     @Test
422     public void testBooleanIfCond() {
423         runtest("module M {" +
424                 "  int foo() {" +
425                 "    if(true)" +
426                 "      return 42;" +
427                 "    return 23;" +
428                 "  }" +
429                 "}");
430     }
431 
432     @Test
433     public void testCompIfCond() {
434         runtest("module M {" +
435                 "  int foo() {" +
436                 "    if(0<1+1)" +
437                 "      return 42;" +
438                 "    return 23;" +
439                 "  }" +
440                 "}");
441     }
442 
443     @Test
444     public void testNonBooleanIfCond() {
445         runtest(false,
446                 "module M {" +
447                 "  int foo() {" +
```

```
448                 "    if(0)" +
449                 "      return 42;" +
450                 "    return 23;" +
451                 "  }" +
452                 "}");
453     }
454 
455     @Test
456     public void testReturnTypeWrong() {
457         runtest(false,
458                 "module M {" +
459                 "  int foo() {" +
460                 "    return true;" +
461                 "  }" +
462                 "}");
463     }
464 
465     @Test
466     public void testVoidReturn() {
467         runtest("module M {" +
468                 "  void foo() {" +
469                 "    return;" +
470                 "  }" +
471                 "}");
472     }
473 
474     @Test
475     public void testExprReturn() {
476         runtest("module M {" +
477                 "  int foo() {" +
478                 "    return 0;" +
479                 "  }" +
480                 "}");
481     }
482 
483     @Test
484     public void testExprReturnVoid() {
485         runtest(false,
486                 "module M {" +
487                 "  boolean foo() {" +
488                 "    return;" +
489                 "  }" +
490                 "}");
491     }
492 
493     @Test
494     public void testVoidTypeReturnWrong() {
495         runtest(false,
496                 "module M {" +
497                 "  int bar() {}" +
```

```
498                 "  void foo() {" +
499                 "    return bar();" +
500                 "  }" +
501                 "}");
502     }
503 
504     @Test
505     public void testBooleanLoopCond() {
506         runtest("module M {" +
507                 "  int foo() {" +
508                 "    while(6>=0)" +
509                 "      return 42;" +
510                 "    return 23;" +
511                 "  }" +
512                 "}");
513     }
514 
515     @Test
516     public void testNonBooleanWhileCond() {
517         runtest(false,
518                 "module M {" +
519                 "  int foo() {" +
520                 "    while(5+5)" +
521                 "      return 42;" +
522                 "    return 23;" +
523                 "  }" +
524                 "}");
525     }
526 
527     @Test
528     public void testCorrectArrayExpression() {
529         runtest("module M {" +
530                 "  boolean foo(boolean[][] bss, int i, int j) {" +
531                 "    return bss[i][j];" +
532                 "  }" +
533                 "}");
534     }
535 
536     @Test
537     public void testIncorrectArrayBaseExpression() {
538         runtest(false,
539                 "module M {" +
540                 "  boolean foo() {" +
541                 "  boolean[][] bss;" +
542                 "  bss[0] = true;" +
543                 "  return false;" +
544                 "  }" +
545                 "}");
546     }
547 
```

```
548     @Test
549     public void testIncorrectArrayIndexExpression() {
550         runtest(false,
551                 "module M {" +
552                 "  boolean foo(boolean[][] bss, int i, boolean j) {"
    +
553                 "    return bss[i][j];" +
554                 "  }" +
555                 "}");
556     }
557
558     @Test
559     public void testArityMatch() {
560         runtest(
561                 "module M {" +
562                 "  int id(int x, int y) { return x+y; }" +
563                 "  int foo() { return id(23, 42); }" +
564                 "}");
565     }
566
567     @Test
568     public void testArityMismatchTooMany() {
569         runtest(false,
570                 "module M {" +
571                 "  int id(int a) { return a; }" +
572                 "  int foo() { return id(1, 1); }" +
573                 "}");
574     }
575
576     @Test
577     public void testArityMismatchTooFew() {
578         runtest(false,
579                 "module M {" +
580                 "  int plus(int x, int y) { return x+y; }" +
581                 "  int foo() { return plus(23); }" +
582                 "}");
583     }
584
585     @Test
586     public void testArgumentTypeMismatch() {
587         runtest(false,
588                 "module M {" +
589                 "  int plus(int x, int y) { return x+y; }" +
590                 "  int foo() { return plus(23, false); }" +
591                 "}");
592     }
593
594     @Test
595     public void testCorrectAssignment() {
596         runtest("module M {" +
```

```
597                 "  boolean foo() {" +
598                 "    boolean res;" +
599                 "    res = true;" +
600                 "    return res;" +
601                 "  }" +
602                 "}");
603     }
604 
605     @Test
606     public void testNonNumericBinary() {
607         runtest(false,
608                 "module M {" +
609                 "  int plus(boolean x) {" +
610                 "    return x + x;" +
611                 "  }" +
612                 "}");
613     }
614 
615     @Test
616     public void testComparison() {
617         runtest("module M {" +
618                 "  void foo(int x) {" +
619                 "    if(x == 42)" +
620                 "      x = 23;" +
621                 "  }" +
622                 "}");
623     }
624 
625     @Test
626     public void testInvalidComparison() {
627         runtest(false,
628                 "module M {" +
629                 "  void foo(int x) {" +
630                 "    if(x == true)" +
631                 "      x = 0;" +
632                 "  }" +
633                 "}");
634     }
635 
636     @Test
637     public void testInvalidArithmeticComparison() {
638         runtest(false,
639                 "module M {" +
640                 "  void foo(boolean x) {" +
641                 "    if(x > true)" +
642                 "      x = false;" +
643                 "  }" +
644                 "}");
645     }
646 
```

```
647     @Test
648     public void testUnaryOperator() {
649         runtest("module M {" +
650                 "  int foo() {" +
651                 "    return -23;" +
652                 "  }" +
653                 "}");
654     }
655
656     @Test
657     public void testInvalidUnaryOperator() {
658         runtest(false,
659                 "module M {" +
660                 "  int foo(boolean s) {" +
661                 "    return -s;" +
662                 "  }" +
663                 "}");
664     }
665
666     @Test
667     public void testArrayLiteral() {
668         runtest("module M {" +
669                 "  int[] foo() {" +
670                 "    return [23, 42];" +
671                 "  }" +
672                 "}");
673     }
674
675     @Test
676     public void testInvalidArrayLiteral() {
677         runtest(false,
678                 "module M {" +
679                 "  int[] foo() {" +
680                 "    return [true, 42];" +
681                 "  }" +
682                 "}");
683     }
684
685     @Test
686     public void testInvalidArrayLiteral2() {
687         runtest(false,
688                 "module M {" +
689                 "  void bar() {}" +
690                 "  int[] foo() {" +
691                 "    return [bar()];" +
692                 "  }" +
693                 "}");
694     }
695
696     @Test
```

```
697     public void testVoidArrayType() {
698         runtest(false,
699                 "module M {" +
700                 "  void[] bar;" +
701                 "}");
702     }
703
704 }
705
```