```
-------------------------------------------------
CLASS NAME : In
SUPER-CLASS NAME :
FIELDS:
CONSTRUCTORS:
METHODS:
METHOD: 2, scan_float, In, public, static, FLOAT
Method parameters:
Variable Table:
Method Body:

Block([
])

METHOD: 1, scan_int, In, public, static, INT
Method parameters:
Variable Table:
Method Body:

Block([
])

-------------------------------------------------
CLASS NAME : Out
SUPER-CLASS NAME :
FIELDS:
CONSTRUCTORS:
METHODS:
METHOD: 4, print, Out, public, static, VOID
Method parameters: 2
Variable Table:
VARIABLE: 2, s, formal, STRING
Method Body:

Block([
])

METHOD: 3, print, Out, public, static, VOID
Method parameters: 2
Variable Table:
VARIABLE: 2, b, formal, BOOLEAN
Method Body:

Block([
])

METHOD: 2, print, Out, public, static, VOID
```

```
Method parameters: 2
Variable Table:
VARIABLE: 2, f, formal, FLOAT
Method Body:

Block([
])

METHOD: 1, print, Out, public, static, VOID
Method parameters: 2
Variable Table:
VARIABLE: 2, i, formal, INT
Method Body:

Block([
])
```

```
TEST CASE test001: Unary Negation

-------------------------------------------------
CLASS NAME : test001
SUPER-CLASS NAME :
FIELDS:
CONSTRUCTORS:
METHODS:
METHOD: 1, main, test001, private, instance, VOID
Method parameters:
Variable Table:
VARIABLE: 3, a, local, BOOLEAN
VARIABLE: 4, b, local, BOOLEAN
Method Body:

Block([
Expr( Assign(Variable(3), ConstantTrue( True )) )
, Expr( Assign(Variable(4), UnaryExpression(!, Variable(3))) )
])
```

```
TEST CASE test002: Unary Minus

-------------------------------------------------
CLASS NAME : test002
SUPER-CLASS NAME :
FIELDS:
CONSTRUCTORS:
METHODS:
METHOD: 1, main, test002, private, instance, VOID
Method parameters:
Variable Table:
VARIABLE: 3, a, local, INT
VARIABLE: 4, b, local, INT
Method Body:

Block([
Expr( Assign(Variable(3), ConstantInteger( 17 )) )
, Expr( Assign(Variable(4), UnaryExpression(-, Variable(3))) )
])
```

```
TEST CASE test003: Unary Plus

-------------------------------------------------
CLASS NAME : test003
SUPER-CLASS NAME :
FIELDS:
CONSTRUCTORS:
METHODS:
METHOD: 1, main, test003, private, instance, VOID
Method parameters:
Variable Table:
VARIABLE: 3, a, local, INT
VARIABLE: 4, b, local, INT
Method Body:

Block([
Expr( Assign(Variable(3), ConstantInteger( 19 )) )
, Expr( Assign(Variable(4), UnaryExpression(+, Variable(3))) )
])
```

```
TEST CASE test004: Binary GTE

--------------------------------------------------
CLASS NAME : test004
SUPER-CLASS NAME :
FIELDS:
CONSTRUCTORS:
METHODS:
METHOD: 1, main, test004, private, instance, VOID
Method parameters:
Variable Table:
VARIABLE: 3, a, local, BOOLEAN
Method Body:

Block([
Expr( Assign(Variable(3),
BinaryExpression(ConstantInteger( 17 ), >=,
ConstantInteger( 14 ))) )
])
```

```
TEST CASE test005: Binary GT

--------------------------------------------------
CLASS NAME : test005
SUPER-CLASS NAME :
FIELDS:
CONSTRUCTORS:
METHODS:
METHOD: 1, main, test005, private, instance, VOID
Method parameters:
Variable Table:
VARIABLE: 3, a, local, BOOLEAN
Method Body:

Block([
Expr( Assign(Variable(3),
BinaryExpression(ConstantInteger( 17 ), >,
ConstantInteger( 14 ))) )
])
```

```
TEST CASE test006: Binary LTE

-------------------------------------------------
CLASS NAME : test006
SUPER-CLASS NAME :
FIELDS:
CONSTRUCTORS:
METHODS:
METHOD: 1, main, test006, private, instance, VOID
Method parameters:
Variable Table:
VARIABLE: 3, a, local, BOOLEAN
Method Body:

Block([
Expr( Assign(Variable(3),
BinaryExpression(ConstantInteger( 17 ), <=,
ConstantInteger( 14 ))) )
])
```

```
TEST CASE test007: Binary LT

--------------------------------------------------
CLASS NAME : test007
SUPER-CLASS NAME :
FIELDS:
CONSTRUCTORS:
METHODS:
METHOD: 1, main, test007, private, instance, VOID
Method parameters:
Variable Table:
VARIABLE: 3, a, local, BOOLEAN
Method Body:

Block([
Expr( Assign(Variable(3),
BinaryExpression(ConstantInteger( 17 ), <,
ConstantInteger( 14 ))) )
])
```

```
TEST CASE test008: Binary NEQ

--------------------------------------------------
CLASS NAME : test008
SUPER-CLASS NAME :
FIELDS:
CONSTRUCTORS:
METHODS:
METHOD: 1, main, test008, private, instance, VOID
Method parameters:
Variable Table:
VARIABLE: 3, a, local, BOOLEAN
Method Body:

Block([
Expr( Assign(Variable(3),
BinaryExpression(ConstantInteger( 17 ), !=,
ConstantInteger( 14 ))) )
])
```

```
TEST CASE test009: Binary EQ

--------------------------------------------------
CLASS NAME : test009
SUPER-CLASS NAME :
FIELDS:
CONSTRUCTORS:
METHODS:
METHOD: 1, main, test009, private, instance, VOID
Method parameters:
Variable Table:
VARIABLE: 3, a, local, BOOLEAN
Method Body:

Block([
Expr( Assign(Variable(3),
BinaryExpression(ConstantInteger( 17 ), ==,
ConstantInteger( 14 ))) )
])
```

```
TEST CASE test010: Binary OR

-------------------------------------------------
CLASS NAME : test010
SUPER-CLASS NAME :
FIELDS:
CONSTRUCTORS:
METHODS:
METHOD: 1, main, test010, private, instance, VOID
Method parameters:
Variable Table:
VARIABLE: 3, a, local, BOOLEAN
Method Body:

Block([
Expr( Assign(Variable(3), BinaryExpression(ConstantTrue( True ),
||, ConstantFalse( False ))) )
])
```

```
TEST CASE test011: Binary AND

-------------------------------------------------
CLASS NAME : test011
SUPER-CLASS NAME :
FIELDS:
CONSTRUCTORS:
METHODS:
METHOD: 1, main, test011, private, instance, VOID
Method parameters:
Variable Table:
VARIABLE: 3, a, local, BOOLEAN
Method Body:

Block([
Expr( Assign(Variable(3), BinaryExpression(ConstantTrue( True ),
&&, ConstantFalse( False ))) )
])
```

```
TEST CASE test012: Binary DIVISION

--------------------------------------------------
CLASS NAME : test012
SUPER-CLASS NAME :
FIELDS:
CONSTRUCTORS:
METHODS:
METHOD: 1, main, test012, private, instance, VOID
Method parameters:
Variable Table:
VARIABLE: 3, a, local, INT
Method Body:

Block([
Expr( Assign(Variable(3),
BinaryExpression(ConstantInteger( 18 ), /,
ConstantInteger( 6 ))) )
])
```

```
TEST CASE test013: Binary MULTIPLICATION

--------------------------------------------------
CLASS NAME : test013
SUPER-CLASS NAME :
FIELDS:
CONSTRUCTORS:
METHODS:
METHOD: 1, main, test013, private, instance, VOID
Method parameters:
Variable Table:
VARIABLE: 3, a, local, INT
Method Body:

Block([
Expr( Assign(Variable(3),
BinaryExpression(ConstantInteger( 18 ), *,
ConstantInteger( 6 ))) )
])
```

```
TEST CASE test014: Binary SUBTRACTION

-------------------------------------------------
CLASS NAME : test014
SUPER-CLASS NAME :
FIELDS:
CONSTRUCTORS:
METHODS:
METHOD: 1, main, test014, private, instance, VOID
Method parameters:
Variable Table:
VARIABLE: 3, a, local, INT
Method Body:

Block([
Expr( Assign(Variable(3),
BinaryExpression(ConstantInteger( 18 ), -,
ConstantInteger( 6 ))) )
])
```

```
TEST CASE test015: Binary ADDITION

-------------------------------------------------
CLASS NAME : test015
SUPER-CLASS NAME :
FIELDS:
CONSTRUCTORS:
METHODS:
METHOD: 1, main, test015, private, instance, VOID
Method parameters:
Variable Table:
VARIABLE: 3, a, local, INT
Method Body:

Block([
Expr( Assign(Variable(3),
BinaryExpression(ConstantInteger( 18 ), +,
ConstantInteger( 6 ))) )
])
```

```
TEST CASE test016: INTEGER CONST Declaration and Assignment

--------------------------------------------------
CLASS NAME : test016
SUPER-CLASS NAME :
FIELDS:
CONSTRUCTORS:
METHODS:
METHOD: 1, main, test016, private, instance, VOID
Method parameters:
Variable Table:
VARIABLE: 3, a, local, INT
Method Body:

Block([
Expr( Assign(Variable(3), ConstantInteger( 18 )) )
])
```

```
TEST CASE test017: FLOAT CONST Declaration and Assignment

--------------------------------------------------
CLASS NAME : test017
SUPER-CLASS NAME :
FIELDS:
CONSTRUCTORS:
METHODS:
METHOD: 1, main, test017, private, instance, VOID
Method parameters:
Variable Table:
VARIABLE: 3, a, local, FLOAT
Method Body:

Block([
Expr( Assign(Variable(3), ConstantFloat( 17.4 )) )
])
```

TEST CASE test018: BOOLEAN TRUE CONST Declaration and Assignment

```
--------------------------------------------------
CLASS NAME : test018
SUPER-CLASS NAME :
FIELDS:
CONSTRUCTORS:
METHODS:
METHOD: 1, main, test018, private, instance, VOID
Method parameters:
Variable Table:
VARIABLE: 3, a, local, BOOLEAN
Method Body:

Block([
Expr( Assign(Variable(3), ConstantTrue( True )) )
])
```

```
TEST CASE test019: BOOLEAN FALSE CONST Declaration and
Assignment

------------------------------------------------------
CLASS NAME : test019
SUPER-CLASS NAME :
FIELDS:
CONSTRUCTORS:
METHODS:
METHOD: 1, main, test019, private, instance, VOID
Method parameters:
Variable Table:
VARIABLE: 3, a, local, BOOLEAN
Method Body:

Block([
Expr( Assign(Variable(3), ConstantFalse( False )) )
])
```

```
TEST CASE test020: STRING CONST Declaration and Assignment

-------------------------------------------------
CLASS NAME : test020
SUPER-CLASS NAME :
FIELDS:
CONSTRUCTORS:
METHODS:
METHOD: 1, main, test020, private, instance, VOID
Method parameters:
Variable Table:
VARIABLE: 3, a, local, string
Method Body:

Block([
Expr( Assign(Variable(3), ConstantString( "Hello" )) )
])
```

```
TEST CASE test021: AUTO-INCREMENT PRE

-------------------------------------------------
CLASS NAME : test021
SUPER-CLASS NAME :
FIELDS:
CONSTRUCTORS:
METHODS:
METHOD: 1, main, test021, private, instance, VOID
Method parameters:
Variable Table:
VARIABLE: 3, a, local, INT
Method Body:

Block([
Expr( Assign(Variable(3), ConstantInteger( 10 )) )
, Expr( AutoExpression(Variable(3), increment, pre) )
])
```

```
TEST CASE test022: AUTO-DECREMENT PRE

-------------------------------------------------
CLASS NAME : test022
SUPER-CLASS NAME :
FIELDS:
CONSTRUCTORS:
METHODS:
METHOD: 1, main, test022, private, instance, VOID
Method parameters:
Variable Table:
VARIABLE: 3, a, local, INT
Method Body:

Block([
Expr( Assign(Variable(3), ConstantInteger( 10 )) )
, Expr( AutoExpression(Variable(3), decrement, pre) )
])
```

```
TEST CASE test023: AUTO-INCREMENT POST

-------------------------------------------------
CLASS NAME : test023
SUPER-CLASS NAME :
FIELDS:
CONSTRUCTORS:
METHODS:
METHOD: 1, main, test023, private, instance, VOID
Method parameters:
Variable Table:
VARIABLE: 3, a, local, INT
Method Body:

Block([
Expr( Assign(Variable(3), ConstantInteger( 10 )) )
, Expr( AutoExpression(Variable(3), increment, post) )
])
```

```
TEST CASE test024: AUTO-DECREMENT POST

-------------------------------------------------
CLASS NAME : test024
SUPER-CLASS NAME :
FIELDS:
CONSTRUCTORS:
METHODS:
METHOD: 1, main, test024, private, instance, VOID
Method parameters:
Variable Table:
VARIABLE: 3, a, local, INT
Method Body:

Block([
Expr( Assign(Variable(3), ConstantInteger( 10 )) )
, Expr( AutoExpression(Variable(3), decrement, post) )
])
```

```
TEST CASE test025: IF Statement

-------------------------------------------------
CLASS NAME : test025
SUPER-CLASS NAME :
FIELDS:
CONSTRUCTORS:
METHODS:
METHOD: 1, main, test025, private, instance, VOID
Method parameters:
Variable Table:
VARIABLE: 3, a, local, INT
Method Body:

Block([
Expr( Assign(Variable(3), ConstantInteger( 0 )) )
, If( BinaryExpression(ConstantInteger( 10 ), >,
ConstantInteger( 5 )),
Block([
Expr( Assign(Variable(3), ConstantInteger( 10 )) )
])
 )
])
```

```
TEST CASE test026: IF ELSE Statement

-------------------------------------------------
CLASS NAME : test026
SUPER-CLASS NAME :
FIELDS:
CONSTRUCTORS:
METHODS:
METHOD: 1, main, test026, private, instance, VOID
Method parameters:
Variable Table:
VARIABLE: 3, a, local, INT
Method Body:

Block([
Expr( Assign(Variable(3), ConstantInteger( 0 )) )
, IfElse( BinaryExpression(ConstantInteger( 10 ), <,
ConstantInteger( 5 )),
Block([
Expr( Assign(Variable(3), ConstantInteger( 10 )) )
])
,
Block([
Expr( Assign(Variable(3), ConstantInteger( 20 )) )
])
 )
])
```

TEST CASE test027: DANGLING ELSE Statement

```
-------------------------------------------------
CLASS NAME : test027
SUPER-CLASS NAME :
FIELDS:
CONSTRUCTORS:
METHODS:
METHOD: 1, main, test027, private, instance, VOID
Method parameters:
Variable Table:
VARIABLE: 3, a, local, INT
Method Body:

Block([
Expr( Assign(Variable(3), ConstantInteger( 0 )) )
, If( BinaryExpression(ConstantInteger( 10 ), >,
ConstantInteger( 5 )),
IfElse( BinaryExpression(ConstantInteger( 11 ), >,
ConstantInteger( 6 )), Expr( Assign(Variable(3),
ConstantInteger( 10 )) ), Expr( Assign(Variable(3),
ConstantInteger( 20 )) ) ) )
])
```

```
TEST CASE test028: FOR Statement

-------------------------------------------------
CLASS NAME : test028
SUPER-CLASS NAME :
FIELDS:
CONSTRUCTORS:
METHODS:
METHOD: 1, main, test028, private, instance, VOID
Method parameters:
Variable Table:
VARIABLE: 3, a, local, INT
VARIABLE: 4, i, local, INT
Method Body:

Block([
Expr( Assign(Variable(3), ConstantInteger( 0 )) )
, For( Assign(Variable(4), ConstantInteger( 0 )),
BinaryExpression(Variable(4), <, ConstantInteger( 10 )),
AutoExpression(Variable(4), increment, post),
Block([
Expr( Assign(Variable(3), BinaryExpression(Variable(3), +,
ConstantInteger( 1 ))) )
])
 )
])
```

```
TEST CASE test029: WHILE Statement

-------------------------------------------------
CLASS NAME : test029
SUPER-CLASS NAME :
FIELDS:
CONSTRUCTORS:
METHODS:
METHOD: 1, main, test029, private, instance, VOID
Method parameters:
Variable Table:
VARIABLE: 3, a, local, INT
Method Body:

Block([
Expr( Assign(Variable(3), ConstantInteger( 0 )) )
, While( BinaryExpression(Variable(3), <, ConstantInteger( 10 ))
Block([
Expr( Assign(Variable(3), BinaryExpression(Variable(3), +,
ConstantInteger( 1 ))) )
])
 )
])
```

```
TEST CASE test030: STATIC METHOD Definition, RETURN Statment

-------------------------------------------------
CLASS NAME : test030
SUPER-CLASS NAME :
FIELDS:
CONSTRUCTORS:
METHODS:
METHOD: 1, add5, test030, private, instance, INT
Method parameters: 2
Variable Table:
VARIABLE: 2, n, formal, INT
Method Body:

Block([
Return( BinaryExpression(Variable(2), +, ConstantInteger( 5 )) )
])

METHOD: 2, main, test030, private, instance, VOID
Method parameters:
Variable Table:
VARIABLE: 3, a, local, INT
Method Body:

Block([
Expr( Assign(Variable(3), ConstantInteger( 0 )) )
])
```

```
TEST CASE test031: STATIC METHOD Invocation Statement

-------------------------------------------------
CLASS NAME : test031
SUPER-CLASS NAME :
FIELDS:
CONSTRUCTORS:
METHODS:
METHOD: 1, add5, test031, private, instance, INT
Method parameters: 2
Variable Table:
VARIABLE: 2, n, formal, INT
Method Body:

Block([
Return( BinaryExpression(Variable(2), +, ConstantInteger( 5 )) )
])

METHOD: 2, main, test031, private, instance, VOID
Method parameters:
Variable Table:
VARIABLE: 3, a, local, INT
Method Body:

Block([
Expr( Assign(Variable(3), MethodCall(Variable(-1), add5,
Arguments([ Variable(3)]))) )
])
```

```
TEST CASE test032: CLASS Definition, STATIC FIELD

--------------------------------------------------
CLASS NAME : Circle
SUPER-CLASS NAME :
FIELDS:
CONSTRUCTORS:
METHODS:
--------------------------------------------------
CLASS NAME : test032
SUPER-CLASS NAME :
FIELDS:
CONSTRUCTORS:
METHODS:
METHOD: 1, main, test032, private, instance, VOID
Method parameters:
Variable Table:
VARIABLE: 3, a, local, FLOAT
Method Body:

Block([
Expr( Assign(Variable(3), FieldAccess(Variable(-1), pi)) )
])
```

```
TEST CASE test033: CLASS Definition, CONSTRUCTOR Definition

--------------------------------------------------
CLASS NAME : Circle
SUPER-CLASS NAME :
FIELDS:
FIELD: 1, pi, Circle, private, instance, FLOAT
FIELD: 2, center, Circle, private, instance, FLOAT
FIELD: 3, radius, Circle, private, instance, FLOAT
CONSTRUCTORS:
CONSTRUCTOR: 1, private
Constructor paramters:
Variable Table:
VARIABLE: 2, c, formal, FLOAT
VARIABLE: 3, r, formal, FLOAT
Constructor Body:

Block([
Expr( Assign(FieldAccess(This, center), Variable(2)) )
, Expr( Assign(FieldAccess(This, radius), Variable(3)) )
, Expr( Assign(FieldAccess(Variable(-1), pi),
ConstantFloat( 3.14159 )) )
])

METHODS:
--------------------------------------------------
CLASS NAME : test033
SUPER-CLASS NAME :
FIELDS:
CONSTRUCTORS:
METHODS:
METHOD: 1, main, test033, private, instance, VOID
Method parameters:
Variable Table:
VARIABLE: 4, a, local, FLOAT
Method Body:

Block([
Expr( Assign(Variable(4), FieldAccess(Variable(-1), pi)) )
])
```

```
TEST CASE test034: CLASS Definition, CONSTRUCTOR Call

--------------------------------------------------
CLASS NAME : Circle
SUPER-CLASS NAME :
FIELDS:
FIELD: 1, pi, Circle, private, instance, FLOAT
FIELD: 2, radius, Circle, private, instance, FLOAT
CONSTRUCTORS:
CONSTRUCTOR: 1, private
Constructor paramters:
Variable Table:
VARIABLE: 2, r, formal, FLOAT
Constructor Body:

Block([
Expr( Assign(FieldAccess(This, radius), Variable(2)) )
, Expr( Assign(FieldAccess(Variable(-1), pi),
ConstantFloat( 3.14159 )) )
])

METHODS:
--------------------------------------------------
CLASS NAME : test034
SUPER-CLASS NAME :
FIELDS:
CONSTRUCTORS:
METHODS:
METHOD: 1, main, test034, private, instance, VOID
Method parameters:
Variable Table:
VARIABLE: 3, x, local, FLOAT
VARIABLE: 4, c, local, Circle
Method Body:

Block([
Expr( Assign(Variable(3), ConstantFloat( 5.0 )) )
, Expr( Assign(Variable(4), NewObject(Circle,
Arguments([ Variable(3)]))) )
])
```

```
TEST CASE test035: CLASS Definition, NON-STATIC METHOD
Definition

-----------------------------------------------------
CLASS NAME : Circle
SUPER-CLASS NAME :
FIELDS:
FIELD: 1, pi, Circle, private, instance, FLOAT
FIELD: 2, radius, Circle, private, instance, FLOAT
CONSTRUCTORS:
CONSTRUCTOR: 1, private
Constructor paramters:
Variable Table:
VARIABLE: 2, r, formal, FLOAT
Constructor Body:

Block([
Expr( Assign(FieldAccess(This, radius), Variable(2)) )
, Expr( Assign(FieldAccess(Variable(-1), pi),
ConstantFloat( 3.14159 )) )
])

METHODS:
METHOD: 1, area, Circle, private, instance, FLOAT
Method parameters:
Variable Table:
VARIABLE: 3, rSquared, local, FLOAT
VARIABLE: 4, result, local, FLOAT
Method Body:

Block([
Expr( Assign(Variable(3), BinaryExpression(FieldAccess(This,
radius), *, FieldAccess(This, radius))) )
, Expr( Assign(Variable(4), BinaryExpression(Variable(3), *,
FieldAccess(Variable(-1), pi))) )
, Return( Variable(4) )
])

-----------------------------------------------------
CLASS NAME : test035
SUPER-CLASS NAME :
FIELDS:
CONSTRUCTORS:
METHODS:
METHOD: 1, main, test035, private, instance, VOID
Method parameters:
Variable Table:
```

```
VARIABLE: 3, x, local, FLOAT
VARIABLE: 4, c, local, Circle
Method Body:

Block([
Expr( Assign(Variable(3), ConstantFloat( 5.0 )) )
, Expr( Assign(Variable(4), NewObject(Circle,
Arguments([ Variable(3)]))) )
])
```

```
TEST CASE test036: CLASS Definition, NON-STATIC METHOD
Invocation

-----------------------------------------------------
CLASS NAME : Circle
SUPER-CLASS NAME :
FIELDS:
FIELD: 1, pi, Circle, private, instance, FLOAT
FIELD: 2, radius, Circle, private, instance, FLOAT
CONSTRUCTORS:
CONSTRUCTOR: 1, private
Constructor paramters:
Variable Table:
VARIABLE: 2, r, formal, FLOAT
Constructor Body:

Block([
Expr( Assign(FieldAccess(This, radius), Variable(2)) )
, Expr( Assign(FieldAccess(Variable(-1), pi),
ConstantFloat( 3.14159 )) )
])

METHODS:
METHOD: 1, area, Circle, private, instance, FLOAT
Method parameters:
Variable Table:
VARIABLE: 3, rSquared, local, FLOAT
VARIABLE: 4, result, local, FLOAT
Method Body:

Block([
Expr( Assign(Variable(3), BinaryExpression(FieldAccess(This,
radius), *, FieldAccess(This, radius))) )
, Expr( Assign(Variable(4), BinaryExpression(Variable(3), *,
FieldAccess(Variable(-1), pi))) )
, Return( Variable(4) )
])

-----------------------------------------------------
CLASS NAME : test036
SUPER-CLASS NAME :
FIELDS:
CONSTRUCTORS:
METHODS:
METHOD: 1, main, test036, private, instance, VOID
Method parameters:
Variable Table:
```

```
VARIABLE: 3, x, local, FLOAT
VARIABLE: 4, a, local, FLOAT
VARIABLE: 5, c, local, Circle
Method Body:

Block([
Expr( Assign(Variable(3), ConstantFloat( 5.0 )) )
, Expr( Assign(Variable(5), NewObject(Circle,
Arguments([ Variable(3)]))) )
, Expr( Assign(Variable(4), MethodCall(Variable(5), area,
Arguments([ ]))) )
])
```

```
TEST CASE test037: Example in Assignment, tests EXTENDS

-------------------------------------------------
CLASS NAME : A
SUPER-CLASS NAME :
FIELDS:
FIELD: 1, x, A, private, instance, INT
CONSTRUCTORS:
CONSTRUCTOR: 1, private
Constructor paramters:
Variable Table:
Constructor Body:

Block([
Expr( Assign(FieldAccess(This, x), ConstantInteger( 0 )) )
])

METHODS:
METHOD: 1, f, A, private, instance, INT
Method parameters:
Variable Table:
Method Body:

Block([
Return( BinaryExpression(FieldAccess(This, x), +,
ConstantInteger( 1 )) )
])

METHOD: 2, g, A, private, instance, INT
Method parameters:
Variable Table:
VARIABLE: 3, i, local, INT
Method Body:

Block([
Expr( Assign(Variable(3), MethodCall(This, f, Arguments([ ]))) )
, Expr( AutoExpression(Variable(3), increment, post) )
, Return( Variable(3) )
])

-------------------------------------------------
CLASS NAME : B
SUPER-CLASS NAME : A
FIELDS:
FIELD: 2, y, B, private, instance, INT
FIELD: 3, s, B, private, instance, user(A)
CONSTRUCTORS:
```

```
CONSTRUCTOR: 1, private
Constructor paramters:
Variable Table:
Constructor Body:

Block([
Expr( Assign(FieldAccess(This, y), ConstantInteger( 2 )) )
, Expr( Assign(FieldAccess(This, s), NewObject(A,
Arguments([ ]))) )
])

METHODS:
METHOD: 1, f, B, private, instance, INT
Method parameters: 2
Variable Table:
VARIABLE: 2, k, formal, INT
Method Body:

Block([
Return( BinaryExpression(MethodCall(Super, f, Arguments([ ])),
+, Variable(2)) )
])
```

```
TEST CASE test038: hello_world.decaf

-------------------------------------------------
CLASS NAME : hello_world
SUPER-CLASS NAME :
FIELDS:
CONSTRUCTORS:
METHODS:
METHOD: 1, main, hello_world, private, instance, VOID
Method parameters:
Variable Table:
Method Body:

Block([
Expr( MethodCall(Variable(-1), print,
Arguments([ ConstantString( "Hello World!\n" )])) )
])
```

```
TEST CASE test039: nrfib.decaf

------------------------------------------------
CLASS NAME : nrfib
SUPER-CLASS NAME :
FIELDS:
CONSTRUCTORS:
METHODS:
METHOD: 1, main, nrfib, private, instance, VOID
Method parameters:
Variable Table:
VARIABLE: 3, n, local, INT
VARIABLE: 4, i, local, INT
VARIABLE: 5, fn, local, INT
VARIABLE: 6, fn_prev, local, INT
Method Body:

Block([
Expr( Assign(Variable(3), MethodCall(Variable(-1), scan_int,
Arguments([ ])))  )
, Expr( Assign(Variable(5), ConstantInteger( 1 ))  )
, Expr( Assign(Variable(6), ConstantInteger( 0 ))  )
, For( Assign(Variable(4), ConstantInteger( 1 )),
BinaryExpression(Variable(4), <, Variable(3)),
Assign(Variable(4), BinaryExpression(Variable(4), +,
ConstantInteger( 1 ))),
Block([
Expr( Assign(Variable(5), BinaryExpression(Variable(6), +,
Variable(5)))  )
, Expr( Assign(Variable(6), BinaryExpression(Variable(5), -,
Variable(6)))  )
])
 )
, Expr( MethodCall(Variable(-1), print,
Arguments([ ConstantString( "Fib = " )]))  )
, Expr( MethodCall(Variable(-1), print,
Arguments([ Variable(5)]))  )
, Expr( MethodCall(Variable(-1), print,
Arguments([ ConstantString( "\n" )]))  )
])
```

```
TEST CASE test040: rfib.decaf

-------------------------------------------------
CLASS NAME : rfib
SUPER-CLASS NAME :
FIELDS:
CONSTRUCTORS:
METHODS:
METHOD: 1, fib, rfib, private, instance, INT
Method parameters: 2
Variable Table:
VARIABLE: 2, n, formal, INT
Method Body:

Block([
IfElse( BinaryExpression(Variable(2), <=, ConstantInteger( 2 )),
Return( ConstantInteger( 1 ) ),
Return( BinaryExpression(MethodCall(Variable(-1), fib,
Arguments([ BinaryExpression(Variable(2), -,
ConstantInteger( 1 ))])), +, MethodCall(Variable(-1), fib,
Arguments([ BinaryExpression(Variable(2), -,
ConstantInteger( 2 ))])))  ) )
])

METHOD: 2, main, rfib, private, instance, VOID
Method parameters:
Variable Table:
VARIABLE: 3, n, local, INT
Method Body:

Block([
Expr( Assign(Variable(3), MethodCall(Variable(-1), scan_int,
Arguments([ ])))  )
, Expr( MethodCall(Variable(-1), print,
Arguments([ ConstantString( "Fib = " )]))  )
, Expr( MethodCall(Variable(-1), print,
Arguments([ MethodCall(Variable(-1), fib,
Arguments([ Variable(3)]))]))  )
, Expr( MethodCall(Variable(-1), print,
Arguments([ ConstantString( "\n" )]))  )
])
```

```
TEST CASE test041: IntList.decaf

-------------------------------------------------
CLASS NAME : IntList
SUPER-CLASS NAME :
FIELDS:
FIELD: 1, value, IntList, private, instance, INT
FIELD: 2, next, IntList, private, instance, user(IntList)
CONSTRUCTORS:
CONSTRUCTOR: 1, private
Constructor paramters:
Variable Table:
Constructor Body:

Block([
Skip()
])

METHODS:
METHOD: 1, create_list, IntList, private, instance,
user(IntList)
Method parameters: 3
Variable Table:
VARIABLE: 3, new_element, local, IntList
VARIABLE: 3, v, formal, INT
Method Body:

Block([
Expr( Assign(Variable(3), NewObject(IntList, Arguments([ ]))) )
, Expr( Assign(FieldAccess(Variable(3), value), Variable(3)) )
, Expr( Assign(FieldAccess(Variable(3), next),
ConstantNull( Null )) )
, Return( Variable(3) )
])

METHOD: 2, insert, IntList, private, instance, user(IntList)
Method parameters: 3
Variable Table:
VARIABLE: 4, new_element, local, IntList
VARIABLE: 3, v, formal, INT
Method Body:

Block([
Expr( Assign(Variable(4), MethodCall(Variable(-1), create_list,
Arguments([ Variable(3)]))) )
, Expr( Assign(FieldAccess(Variable(4), next), This) )
, Return( Variable(4) )
```

```
])

METHOD: 3, search, IntList, private, instance, BOOLEAN
Method parameters: 2
Variable Table:
VARIABLE: 2, v, formal, INT
Method Body:

Block([
IfElse( BinaryExpression(FieldAccess(This, value), ==,
Variable(2)),
Block([
Return( ConstantTrue( True ) )
])
, IfElse( BinaryExpression(FieldAccess(This, next), ==,
ConstantNull( Null )),
Block([
Return( ConstantFalse( False ) )
])
, Return( MethodCall(FieldAccess(This, next), search,
Arguments([ Variable(2)])) ) ) )
])

METHOD: 4, length, IntList, private, instance, INT
Method parameters:
Variable Table:
Method Body:

Block([
IfElse( BinaryExpression(FieldAccess(This, next), ==,
ConstantNull( Null )), Return( ConstantInteger( 1 ) ),
Return( BinaryExpression(ConstantInteger( 1 ), +,
MethodCall(FieldAccess(This, next), length, Arguments([ ])))  ) )
])
```

```
TEST CASE test042: Bank Accounts

-------------------------------------------------
CLASS NAME : BankAccount
SUPER-CLASS NAME :
FIELDS:
FIELD: 1, balance, BankAccount, private, instance, FLOAT
FIELD: 2, firstName, BankAccount, private, instance,
user(string)
FIELD: 3, lastName, BankAccount, private, instance, user(string)
CONSTRUCTORS:
CONSTRUCTOR: 1, private
Constructor paramters:
Variable Table:
VARIABLE: 2, fn, formal, string
VARIABLE: 3, ln, formal, string
Constructor Body:

Block([
Expr( Assign(FieldAccess(This, firstName), Variable(2)) )
, Expr( Assign(FieldAccess(This, lastName), Variable(3)) )
, Expr( Assign(FieldAccess(This, balance),
ConstantFloat( 0.0 )) )
])

METHODS:
METHOD: 1, deposit, BankAccount, private, instance, FLOAT
Method parameters: 2
Variable Table:
VARIABLE: 2, amount, formal, FLOAT
Method Body:

Block([
Expr( Assign(FieldAccess(This, balance),
BinaryExpression(FieldAccess(This, balance), +, Variable(2))) )
, Return( FieldAccess(This, balance) )
])

METHOD: 2, withdrawal, BankAccount, private, instance, FLOAT
Method parameters: 2
Variable Table:
VARIABLE: 2, amount, formal, FLOAT
Method Body:

Block([
IfElse( BinaryExpression(FieldAccess(This, balance), >,
Variable(2)),
```

```
Block([
Expr( Assign(FieldAccess(This, balance),
BinaryExpression(FieldAccess(This, balance), -, Variable(2))) )
])
,
Block([
Expr( MethodCall(Variable(-1), print,
Arguments([ ConstantString( "Insufficient funds in
account." )])) )
])
 )
, Return( FieldAccess(This, balance) )
])

METHOD: 3, get_balance, BankAccount, private, instance, FLOAT
Method parameters:
Variable Table:
Method Body:

Block([
Return( FieldAccess(This, balance) )
])

--------------------------------------------------
CLASS NAME : CheckingAccount
SUPER-CLASS NAME : BankAccount
FIELDS:
FIELD: 4, checkNumber, CheckingAccount, private, instance, INT
CONSTRUCTORS:
CONSTRUCTOR: 1, private
Constructor paramters:
Variable Table:
VARIABLE: 2, fn, formal, string
VARIABLE: 3, ln, formal, string
Constructor Body:

Block([
Expr( Assign(FieldAccess(This, firstName), Variable(2)) )
, Expr( Assign(FieldAccess(This, lastName), Variable(3)) )
, Expr( Assign(FieldAccess(This, balance),
ConstantFloat( 0.0 )) )
, Expr( Assign(FieldAccess(This, checkNumber),
ConstantInteger( 0 )) )
])

METHODS:
METHOD: 1, debit, CheckingAccount, private, instance, INT
```

```
Method parameters: 3
Variable Table:
VARIABLE: 4, ccn, local, INT
VARIABLE: 3, amount, formal, FLOAT
Method Body:

Block([
Expr( Assign(Variable(4), FieldAccess(This, checkNumber)) )
, IfElse( BinaryExpression(Variable(3), <, FieldAccess(This,
balance)),
Block([
Expr( AutoExpression(FieldAccess(This, checkNumber), increment,
post) )
, Expr( Assign(FieldAccess(This, balance),
BinaryExpression(FieldAccess(This, balance), -, Variable(3))) )
])
,
Block([
Expr( MethodCall(Variable(-1), print,
Arguments([ ConstantString( "Insufficient funds in
account." )])) )
])
 )
, Return( Variable(4) )
])

-------------------------------------------------
CLASS NAME : test042
SUPER-CLASS NAME :
FIELDS:
CONSTRUCTORS:
METHODS:
METHOD: 1, main, test042, private, instance, VOID
Method parameters:
Variable Table:
VARIABLE: 4, x, local, FLOAT
VARIABLE: 5, y, local, FLOAT
VARIABLE: 6, z, local, INT
VARIABLE: 7, w, local, INT
VARIABLE: 8, ca1, local, CheckingAccount
VARIABLE: 9, ca2, local, CheckingAccount
Method Body:

Block([
Expr( Assign(Variable(8), NewObject(CheckingAccount,
Arguments([ ConstantString( "Paul", "Blart" )]))) )
```

```
, Expr( Assign(Variable(9), NewObject(CheckingAccount,
Arguments([ ConstantString( "Michael", "Skarn" )]))) )
, Expr( Assign(Variable(4), MethodCall(Variable(8), deposit,
Arguments([ ConstantFloat( 10000.0 )]))) )
, Expr( Assign(Variable(5), MethodCall(Variable(9), deposit,
Arguments([ ConstantFloat( 100.0 )]))) )
, Expr( Assign(Variable(4), MethodCall(Variable(8), get_balance,
Arguments([ ]))) )
, Expr( Assign(Variable(5), MethodCall(Variable(9), get_balance,
Arguments([ ]))) )
, Expr( Assign(Variable(4), MethodCall(Variable(8), withdrawal,
Arguments([ ConstantFloat( 1000.0 )]))) )
, Expr( Assign(Variable(5), MethodCall(Variable(9), withdrawal,
Arguments([ ConstantFloat( 1000.0 )]))) )
, Expr( Assign(Variable(6), MethodCall(Variable(8), debit,
Arguments([ ConstantFloat( 1000.0 )]))) )
, Expr( Assign(Variable(7), MethodCall(Variable(9), debit,
Arguments([ ConstantFloat( 1000.0 )]))) )
])
```

```
TEST CASE error01: Duplicate Class Name

ERROR: CLASS 'error01' ALREADY EXISTS. (LINE: 16)

TEST CASE error02: Duplicate Field Name

ERROR: FIELD 'x' ALREADY EXISTS IN THIS CLASS. (LINE: 0)

TEST CASE error03: Duplicate Variable

ERROR: VARIABLE 'y' ALREADY EXISTS. (LINE: 1)
```