The J- Language (Static) Semantics

Version 1.3 (4/5/07)

We define here the syntactic restrictions and static semantics of the simple language **J-** used in 2006 for 3516ICT assignments.

Most of these restrictions and semantics can be inferred from your knowledge of Java and from the description of J- in Assignment 1.

The language used in what follows is terse and attempts to be precise. There is some deliberate repetition so that the semantics matches the grammar.

Types

Expressions in J- have type *void*, *null*, *boolean*, *integer*, *double*, *string*, *list*, *system*, *object*, or some user-defined type. These are internal types. Apart from *void*, they correspond to classes `Null`, `Boolean`, *etc*.

The types in J- (excluding *void*) form a directed acyclic graph, with type *object* at the top (every type is a subtype of *object*) and type *null* at the bottom (*null* is a subtype of every type). Type *integer* is a direct subtype of type *double*.

If class `C` is defined as an extension of class `D`, then type *C* is a subtype of type *D*,

The subtype relationship on the set of types is a partial order, *i.e.*, it is reflexive, antisymmetric and transitive.

It is not possible for programs to declare variables or parameters of classes `Null` or `System`. It is not possible for programs to declare extensions of classes `Null`, `Boolean`, `Integer`, `Double`, `String` or `System`.

Conditions

Every legal `J-` program must satisfy the following conditions. Conditions followed by (*) are recommended rather than required, and should generate warnings rather than errors when violated.

1. Every program must have at least one class declaration. (*)
2. Every class declared in a program must have a distinct identifier.
3. No class declaration may define the class identifier `Null`, `Boolean`, `Integer`, `Double`, `String`, `List`, `System` or `Object`.
4. At least one class declaration in a program must contain a declaration of a void method with no parameters. (*)
5. Every identifier used in a range[1] must be declared in that range or some textually enclosing range.
6. The declaration associated with the use of an identifier I in range R is the declaration of I in R or the innermost range containing R that has a declaration of I.

[1]See the definition below.

7. No field or variable declared in a given range can be used in that range before its declaration. However, mutually recursive classes and methods are allowed.

8. No identifier can be be declared twice in the same range. (In particular, an identifier cannot be defined to be both a field and a method in the same class declaration.)

9. A declaration of an identifier in a range hides declarations of the same identifier in textually enclosing ranges.

10. Every constructor declaration in a class declaration must have the same identifier as the class. No constructor declaration may contain a return statement.

11. Every class `C` has a default constructor of no arguments (named `C`) implicity declared by the compiler if the class declaration does not contain a constructor.

12. No field or method in a class declaration can have the same identifier as the class.

13. Every field or method in a class declaration may be used in every subtype of the class, unless it is overridden by a definition in some intervening subclass.

14. The type of a `constant` is determined as follows:

    (a) The type of an occurrence of `null` is *null*.

    (b) The type of an occurrence of `this` is the type of the innermost class declaration containing the occurrence.

    (c) The type of a constant of type *boolean*, *integer*, *double* or *string* is that type.

15. The type of a `variable` is determined as follows:[2]

    (a) The type of an identifier occurrence is the type assigned to the identifier in the declaration associated with the identifier occurrence.

    (b) The type of a variable $id_1 . \ldots . id_n$ (*e.g.*, `person.address.postcode`) is the type of the field id_n, in the type of the variable $id_1 . \ldots . id_{n-1}$, which must be a non-*void* type with a field id_n. A similar definition applies if any field id_k in such a variable is replaced by a method call $m(a_1, \ldots, a_p)$.

16. The type of any other `object` is determined as follows:

    (a) The type of an occurrence of `this` is the type of the innermost class declaration containing the occurrence.

    (b) The type of a constructor call is the type of the instance being constructed.

    (c) The type of a method call is the return type of the method.

    (d) The type of a parenthesized expression is the type of the expression.

17. The type of any other expression is determined as follows:

[2]Yes, this definition is recursive. It may require further elaboration.

(a) The type of an identifier occurrence is the type assigned to the identifier in the declaration associated with the identifier occurrence.

(b) The type of a constructor call is the type of the instance being constructed.

(c) The type of a method call is the return type of the method declaration associated with the method call.

(d) The type of an "assignment expression" (`variable "=" expression`) is the type of the expression.

(e) The type of a "relop expression" (`expression relop expression`) is *boolean*.

(f) Both operands in a "relop expression" with operator `"<"`, `"<="`, `">"` or `">="` must be of type *integer* or *double*.[3]

(g) Operands in a relop expression with operator `"=="` or `"!="` may be of any type.

(h) Both operands in an arithmetic expression must have type *integer* or *double*.[4] The type of the expression is determined by the types of the operands: if both are *integer*, the type of the expression is *integer*; if either are *double*, the type of the expression is *double*.

(i) Both operands in a "%" expression must have type *integer*.

(j) The type of a method call is the return type of the method.

(k) The type of a parenthesized expression is the type of the expression.

(l) The type of a "cast expression" (`"[" type-specifier "]" expression`) is the type of the type-specifier.

(m) The type of the type specifier in a "cast expression" must be a subtype of the type of the expression in the "cast expression".[5]

18. In every assignment expression, the type of the expression must be a subtype of the type of the variable.

19. In every constructor call or method call, the number of arguments must be the same as the number of parameters in the constructor declaration or method declaration associated with the constructor call or method call.

20. In every constructor call or method call, the type of each argument must be a subtype of the type of the corresponding parameter in the constructor declaration or method declaration associated with the constructor call or method call.

21. Every method declaration that has a non-void return type must contain a return statement with an operand whose type is a subtype of the return type of the method. In every method that has a void return type, no return statement can have an operand.

22. In every conditional statement and iteration statement, the expression must have type *boolean*.

[3]A runtime error occurs if either operand has value `null`.

[4]A runtime error occurs if either operand has value `null`.

[5]It is a runtime error if the type of the value of the expression is not a subtype of the type specifier.

Predefined classes

J- has several predefined classes and one predefined (global) variable as indicated below. In addition, the reserved word `this` denotes an instance of class `C` in within the declaration of class `C` and the reserved word `null` denotes the unique instance of class `Null`. word

```
class Object {
    Boolean equals(Object o);
}

class System {
    Boolean readBoolean();
    Integer readInteger();
    Double readDouble();
    String readString();

    void print(Object o);
    void println(Object o);
}

class String {
    Integer length();
    Integer compareTo(String s);
    Integer indexOf(String s);
    String subString(Integer from, Integer to);
    String append(String s);
}

class List {
    Integer size();
    Boolean contains(Object o);
    Object get(Integer i);
    Object set(Integer i, Object o);
    void add(Integer i, Object o);
    Object remove(Integer i);
}

class Double {
    Integer round();
    Integer truncate();
}

System system;
```

In addition, the reserved word `this` denotes an instance of class `C` within the declaration of class `C`, and the reserved word `null` denotes the unique instance of class `Null`.

Definitions

1. A *range* is a program, a class declaration, a constructor declaration, a method declaration or a compound statement. The range of a constructor or method declaration includes both the parameters and the local variables in its compound statement. (A method name belongs to the enclosing class declaration.)

Comments

1. There are no restrictions on operands of == and `!=` operators. This may not be the best definition.
2. The expression `123 + list.get(0)` is illegal. The expression `123 + [Integer]list.get(0)` is legal, but will result in a runtime error if `list.get(0)` is not an integer.
3. The semantic conditions omit a necessary restriction on methods in subclasses that override methods of the same name in superclasses.
4. The semantic conditions omit the following restriction: Identifier `system` and reserved words `this` and `null` are effectively constants: they may not occur as the variable in an assignment expression (nor as parameters in a method declaration).
5. The semantic conditions omit the restriction that classes `Null`, `Boolean`, `Integer`, `Double`, `String` and `System` do not have constructors.

Change history

Released. 3/5/2006.
Added reference to type *void*. 5/5/20006.
Added several clarifications. 7/5/2006.
Updated for 2007. 4/5/2007.