A. Excerpt from Java Language Reference

15.11 Field Access Expressions

A field access expression may access a field of an object or array, a reference to which is the value of either an expression or the special keyword super. (It is also possible to refer to a field of the current instance or current class by using a simple name; see §6.5.6.)

FieldAccess:
  Primary `.` *Identifier*
  `super .` *Identifier*
  ClassName `. super .` *Identifier*

The meaning of a field access expression is determined using the same rules as for qualified names (§6.6), but limited by the fact that an expression cannot denote a package, class type, or interface type.

15.11.1 Field Access Using a Primary

The type of the *Primary* must be a reference type *T*, or a compile-time error occurs. The meaning of the field access expression is determined as follows:

- If the identifier names several accessible member fields of type *T*, then the field access is ambiguous and a compile-time error occurs.
- If the identifier does not name an accessible member field of type *T*, then the field access is undefined and a compile-time error occurs.
- Otherwise, the identifier names a single accessible member field of type *T* and the type of the field access expression is the declared type of the field. At run time, the result of the field access expression is computed as follows:
  - If the field is `static`:
    - If the field is `final`, then the result is the value of the specified class variable in the class or interface that is the type of the *Primary* expression.
    - If the field is not `final`, then the result is a variable, namely, the specified class variable in the class that is the type of the *Primary* expression.
  - If the field is not `static`:
    - If the value of the *Primary* is `null`, then a `NullPointerException` is thrown.
    - If the field is `final`, then the result is the value of the specified instance variable in the object referenced by the value of the *Primary*.

If the field is not `final`, then the result is a variable, namely, the specified instance variable in the object referenced by the value of the *Primary*.

Note, specifically, that only the type of the *Primary* expression, not the class of the actual object referred to at run time, is used in determining which field to use.

Thus, the example: `class S { int x = 0; }` …

B. Quotes from Java Language Reference

1. The scope of a parameter of a method (§8.4.1) or constructor (§8.8.1) is the entire body of the method or constructor. The scope of a local variable declaration in a block (§14.4.2) is the rest of the block in which the declaration appears, starting with its own initializer (§14.4) and including any further declarators to the right in the local variable declaration statement.
2. When an exception is thrown, control is transferred from the code that caused the exception to the nearest dynamically-enclosing catch clause of a try statement (§14.19) that handles the exception.
3. The boolean type has two values, represented by the literals `true` and `false`, formed from ASCII letters. A boolean literal is always of type boolean.
4. An array access expression refers to a variable that is a component of an array.

   ArrayAccess:

   ExpressionName `[` *Expression* `]`

   PrimaryNoNewArray `[` *Expression* `]`
5. A variable that is declared final cannot be incremented, because when an access of a `final` variable is used as an expression, the result is a value, not a variable. Thus, it cannot be used as the operand of a postfix increment operator.
6. The left-hand operand of a binary operator appears to be fully evaluated before any part of the right-hand operand is evaluated.
7. Developers should take steps to avoid the possibility of two published packages having the same name by choosing *unique package names* for packages that are widely distributed.

C. Concept of Definite Assignment in Java

Each local variable (§14.4) and every blank `final` (§4.5.4) field (§8.3.1.2) must have a *definitely assigned* value when any access of its value occurs. An access to its value consists of the simple name, or the simple name qualified by `this`, of the variable occurring anywhere in an expression except as the left-hand operand of the simple assignment operator =. A Java compiler must carry out a specific conservative flow analysis to make sure that, for every access of a local variable or blank `final` field *f*, *f* is definitely assigned before the access; otherwise a compile-time error must occur.

Similarly, every blank `final` variable must be assigned at most once; it must be *definitely unassigned* when an assignment to it occurs. A Java compiler must carry out a specific conservative flow analysis to make sure that, for every assignment to a blank `final` variable, the variable is definitely unassigned before the assignment; otherwise a compile-time error must occur.

The remainder of this chapter is devoted to a precise explanation of the words *definitely assigned before* and *definitely unassigned before* . The idea behind definite assignment is that an assignment to the local variable or blank `final` field must occur on every possible execution path to the access. Similarly, the idea behind definite unassignment is that no other assignment to the `blank` final variable is permitted to occur on any possible execution path to an assignment. The analysis takes into account the structure of statements and expressions; it also provides a special treatment of the expression operators `&&`, `||`, and `?:`, and of boolean-valued constant expressions.