

## Home Work 3 – overview

The semantic analysis visitor has 22 checks to make (originally 24 but two will not be tested):

- ✓ 1. **The superclass of a class precedes it in the file:** when `class B extends A`, the class A was defined, and the definition of A must come before that of B. In particular, there are no cycles in the inheritance graph - **i.e., check to be sure that no class directly or indirectly extends itself**
- ✓ 2. The main class cannot be extended.
- ✓ 3. The same name cannot be used to name two classes.
4. The same name cannot be used for the same field in one class.
- ✓ ~~This includes fields defined in a class and a subclass (even though this is legal in real Java). I'm not sure exactly what this means, we'll to ask in forum.~~
- ✓ 5. The same name cannot be used for the same method in one class - overloading is not supported.
- ✓ 6. An overriding method matches the ancestor's method signature with the same name (same number of arguments, same static type of arguments, a covariant static return type). Note that overloading is not supported.
- ✓ 7. A type declaration of a reference type of A refers to classes that are defined somewhere in the file (either before or after the same class, or to the same class itself).
- ✓ 8. `new A()` is invoked for a class A that is defined somewhere in the file (either before or after the same class, or to the same class itself).
- ✓ 9. In method invocation, the static type of the object is a reference type (not `int`, `bool`, or `int[]`).
- ✓ 10. A method call is to a method that was defined in the class according to the static type of the object, and further, the type of the actual parameters matches the definition. Namely, in `e.f(a_1, ..., a_k)`, the method `f` was defined in class A where A is the static type of `e`, it has k arguments, and the static type of each `a_i` matches (i.e. is a subtype of) the type of the i'th formal parameter of said definition.
- ✓ 11. A method call is invoked on expression `e` which is either `this`, a `new` expression, or a reference to a local variable, formal parameter or a field.
- ✓ 12. The static type of the object on which `length` invoked is `int[]`.
- ✓ 13. A reference in an expression to a variable (i.e., not in a role of a method name in a call or a class name in `new`) is to a local variable or formal parameter defined in the current method, or to a field defined in the current class or its superclasses.
- ✓ 14. Every local variable is definitely initialized (assigned to) before it is used. See [here](#).

currently todo

no field overriding, only hiding by local variable

- ✓ 15. In an assignment `x = e`, the static type of `e` is valid according to the declaration of `x`. Note subtyping!
- ✓ 16. In `if` and `while`, the condition is boolean.
- ✓ 17. The static type of `e` in `return e` is valid according to the definition of the current method. Note subtyping!
- ✓ 18. The argument to `System.out.println` is of type `int`.
- ✓ 19. The arguments to the predefined operators (`&&`, `<`, `!`, `+`, `-`, `*` etc.) are of the correct type.
- ✓ 20. In an array access `x[e]`, `x` is `int[]` and `e` is an `int`.
- ✓ 21. In an assignment to an array `x[e1] = e2`, `x` is `int[]`, `e1` is an `int` and also `e2` is an `int`.
- ✓ 22. Variable redeclaration is forbidden - the same name cannot be used for declarations of two local variables or formal parameters.
- ✓ 23. **updated** In an array allocation `new int[e]`, `e` is an `int`.

We'll to implement some helpers:

- `BinaryExpr --> Type`
- `VarName --> StaticType`
- `(ClassName A, ClassName B) --> is B a legal instance of A`
- And probably more

Idea (general pseudo code):

1. `mainName <- name of main class`
2. `for classDecl in program.classDecls():`
3.     `check if extends main`
4.     `check if superClass is defined before it`
5.     `check if replecates a diffarent class name`
6.     `for VarDecl in classDecl.fiels():`
7.         `check if replecates a feild name`
8.         `if VarDecl.type() is RefType:`
9.             `check if type is defined as a class name`
10.     `for MethodDecl in classDecl.methodDecls():`
11.         `check for overloading`
12.         `if override check for signature match`
13.         `for formal args and varDecls check for double naming and valid definitions`
14.         `check that type of the retExpr matches decleration`
15.         `for “if” and “while” statements – check cond is boolean`
16.         `for expressions:`
17.             `binary – check type`
18.             `assignment – check type`
19.             `Identifer:`
20.                 `checl that is defined`
21.                 `if in lv of assignment check for inisialization`
22.             `NotExpr – check type`
23.             `MethodCall:`
24.                 `check for valid owner`
25.                 `owner has method defined`
26.                 `correct types of actuals`
27.             `arrayLength – check that owner is of static type int[]`
28.             `System.out.println – actual is of type int`
29.             `ArrayAccess and ArrayAssign – check types`