1. **Sample solution.** The miniJava program and its corresponding AST are shown on the next page. The *Declaration* nodes in the AST are shown in red, *Identifier* nodes are green, and *Expression* nodes are blue. *Green arrows* link identifiers to their controlling declaration. The arrows are numbered in the order in which they are created. *Blue lines* show the type assignments at Expression nodes and are numbered in the order they are assigned.

2. Suppose we have the following type declarations for variables and functions

\[
\begin{align*}
+: \text{Int} \times \text{Int} & \rightarrow \text{Int} \\
+: \text{Real} \times \text{Real} & \rightarrow \text{Real} \\
*: \text{Int} \times \text{Int} & \rightarrow \text{Int} \\
*: \text{Real} \times \text{Real} & \rightarrow \text{Real} \\
*: \text{Int} \times \text{Int} \rightarrow \text{Int} & \rightarrow \text{Stmt} \\
r: \text{Real} & \\
i: \text{Int} & \\
\end{align*}
\]

Assume also that integer and real literals have a single (manifest) type (e.g. “2” is Int and “2.0” is Real). Use the type checking framework for simple ASTs for the following.

**(a)** Draw the AST and perform type assignment for the statement \( r = r \times 2 \)

**(b)** Draw the AST and perform type assignment for the expression \( i = 2 \times i \)

**Sample Solution**

**(a)** the statement \( r = r \times 2 \) has no type assignment, the multiplication operation cannot be applied.

**(b)** the expression \( i = 2 \times i \) has a unique type assignment
class Test {
    public int go() {
        Foo p = new Foo();
        this.p = p;
        return p.x;
    }
    public Foo p;
    class Foo {
        public int x;
    }
}