```
package mini;

import compiler.Failure;

/** Abstract syntax for assignment expressions.
 */
class Assign extends Expr {
    private Id lhs;
    private Expr rhs;
    Assign(Id lhs, Expr rhs) {
        this.lhs = lhs;
        this.rhs = rhs;
    }

    /** Print an indented description of this abstract syntax node,
     *  including a name for the node itself at the specified level
     *  of indentation, plus more deeply indented descriptions of
     *  any child nodes.
     */
    public void indent(IndentOutput out, int n) {
        out.indent(n, "Assign");
        lhs.indent(out, n+1);
        rhs.indent(out, n+1);
    }

    /** Output a description of this node (with id n), returning the
     *  next available node id after this node and all of its children
     *  have been output.
     */
    public int toDot(DotOutput dot, int n) {
        return rhs.toDot(dot, n,
               lhs.toDot(dot, n,
               dot.node("Assign", n)));
    }

    /** Return the type of value that will be produced when this
     *  expression is evaluated.
     */
    public Type typeOf(Context ctxt, VarEnv env)
      throws Failure {
        Type lt = lhs.typeOf(ctxt, env);
        Type rt = rhs.typeOf(ctxt,env);

        if (lt!=rt || lt.equal(Type.DOUBLE) && rt.equal(Type.INT)) {
            ctxt.report(new Failure("Types in assignment do not match or convert nicely"));
        }
        return rt;
    }
}
```

```java
package mini;

import compiler.Failure;

/** Abstract syntax for binary arithmetic expressions.
 */
abstract class BinArithExpr extends BinExpr {
    BinArithExpr(Expr left, Expr right) {
        super(left, right);
    }

    /** Return the type of value that will be produced when this
     *  expression is evaluated.
     */
    public Type typeOf(Context ctxt, VarEnv env)
      throws Failure {
        // Covers +, -, *, /
        Type lt = left.typeOf(ctxt, env);
        Type rt = right.typeOf(ctxt, env);
        if ( !(lt.equal(Type.INT) || lt.equal(Type.DOUBLE)) ) {
            throw new Failure("Lefthand arithmetic operands must be int or double.");
        }
        if ( !(rt.equal(Type.INT) || rt.equal(Type.DOUBLE)) ) {
            throw new Failure("Righthand arithmetic operands must be int or double.");
        }

        return lt;
    }
}
```

```
int c = 6;
double f;

f = c;
f = c + 4;
f = f + c;
f = f + 4;
```