This is coursework 2 revision guidance for students.

MiniJava typechecker

```
---------------------minijava.jj---------------------------------
```

First minijava.jj has to be modified to put in the new operators (and
the do-while from the first coursework if you didn't already - see
Coursework 1 model answer).

Tokens have to be added:

```
| < DIVIDE: "/" >       /* Added by DB */
| < GT: ">" >           /* Added by DB */
| < OR: "||" >          /* Added by DB */
| < SIF: "?" >          /* Added by DB */
| < SELSE: ":" >        /* Added by DB */
```

and the grammar changed:

```
Exp Expression() :
{ Exp e; }
{
...
|
  LOOKAHEAD( PrimaryExpression() "||" )          /* Added by DB */
  e=OrExpression()
|
  LOOKAHEAD( PrimaryExpression() ">" )           /* Added by DB */
  e=CompareGTExpression()
|
  LOOKAHEAD( PrimaryExpression() "/" )           /* Added by DB */
  e=DivideExpression()
|
  LOOKAHEAD( PrimaryExpression() "?" )           /* Added by DB */
  e=ShortCutIfElseExpression()
...

Exp OrExpression() :                    /* Added by DB */
{ Exp e1,e2; }
{
  e1=PrimaryExpression() "||" e2=PrimaryExpression()
  { return new Or(e1,e2); }
}

Exp CompareGTExpression() :             /* Added by DB */
{ Exp e1,e2; }
{
  e1=PrimaryExpression() ">" e2=PrimaryExpression()
  { return new GreaterThan(e1,e2); }
}

Exp DivideExpression() :                /* Added by DB */
{ Exp e1,e2; }
{
  e1=PrimaryExpression() "/" e2=PrimaryExpression()
  { return new Divide(e1,e2); }
}

Exp ShortCutIfElseExpression() :                /* Added by DB */
{ Exp e1,e2,e3; }
{
  e1=PrimaryExpression() "?" e2=PrimaryExpression() ":"
      e3 = PrimaryExpression()
  { return new ShortCutIfElse(e1,e2,e3); }
}
--------------------------------------------------------------
```

Then syntax tree classes have to be created for each new operator (headers
missing here - same as other tree classes):

```
public class Or extends Exp {   /* Added by DB */
  public Exp e1,e2;

  public Or(Exp ae1, Exp ae2) {
    e1=ae1; e2=ae2;
  }

  public void accept(Visitor v) {
    v.visit(this);
  }

  public Type accept(TypeVisitor v) {
    return v.visit(this);
  }
}

public class Divide extends Exp {       /* Added by DB */
  public Exp e1,e2;

  public Divide(Exp ae1, Exp ae2) {
    e1=ae1; e2=ae2;
  }

  public void accept(Visitor v) {
    v.visit(this);
  }

  public Type accept(TypeVisitor v) {
    return v.visit(this);
  }
}

public class GreaterThan extends Exp {  /* Added by DB */
  public Exp e1,e2;

  public GreaterThan(Exp ae1, Exp ae2) {
    e1=ae1; e2=ae2;
  }

  public void accept(Visitor v) {
    v.visit(this);
  }

  public Type accept(TypeVisitor v) {
    return v.visit(this);
  }
}

public class ShortCutIfElse extends Exp {       /* Added by DB */
  public Exp e1, e2, e3;

  public ShortCutIfElse(Exp ec, Exp et, Exp ee) {
    e1=ec; e2=et; e2=ee;
  }

  public void accept(Visitor v) {
    v.visit(this);
  }

  public Type accept(TypeVisitor v) {
    return v.visit(this);
  }
}

------------------------------------------------------------------

Then the pretty printer (DBPrettyVisitor.java) needs to print the new operators
- the binary ones are just the same as And etc, the if-else is:

  // Exp e1,e2,e3;
  public void visit(ShortCutIfElse n) {         /* Added by DB */
    System.out.print("(");
    n.e1.accept(this);
    System.out.print(" ? ");
    n.e2.accept(this);
    System.out.print(" : ");
    n.e2.accept(this);
    System.out.print(")");
  }

-------------------------------------------------------------------

Then the visitor interfaces must be fixed up:

--Visitor.java------------
  public void visit(Or n);              /* Added by DB */
  public void visit(GreaterThan n);     /* Added by DB */
  public void visit(Divide n);          /* Added by DB */
  public void visit(ShortCutIfElse n);  /* Added by DB */
  public void visit(DoWhile n);         /* Added by DB */
--TypeVisitor.java---
  public Type visit(Or n);              /* Added by DB */
  public Type visit(GreaterThan n);     /* Added by DB */
  public Type visit(Divide n);          /* Added by DB */
  public Type visit(ShortCutIfElse n);  /* Added by DB */
  public Type visit(DoWhile n);         /* Added by DB */

--------------------------------------------------------------------

Now the generic visitors themselves:

-----DepthFirstVisitor.java--------

  // Exp e1,e2;
  public void visit(Or n) {     /* Added by DB */
    n.e1.accept(this);
    n.e2.accept(this);
  }
....Divide and GreaterThan similar....

  // Exp e1,e2,e3;
  public void visit(ShortCutIfElse n) {         /* Added by DB */
    n.e1.accept(this);
    n.e2.accept(this);
    n.e3.accept(this);
  }

  // Statement s;
  // Exp e;
  public void visit(DoWhile n) { /* Added by DB */
    n.s.accept(this);
    n.e.accept(this);
  }

----TypeDepthFirstVisitor.java------

  // Exp e1,e2;
  public Type visit(Or n) {     /* Added by DB */
    n.e1.accept(this);
    n.e2.accept(this);
    return null;
  }
....Divide and GreaterThan similar....

  // Exp e1,e2,e3;
  public Type visit(ShortCutIfElse n) { /* Added by DB */
    n.e1.accept(this);
    n.e2.accept(this);
    n.e3.accept(this);
    return null;
  }

  // Statement s;
  // Exp e;
  public Type visit(DoWhile n) { /* Added by DB */
    n.s.accept(this);
    n.e.accept(this);
    return null;
  }
--------------------------------------------------------------------

And finally the typechecking itself!

The new operators - we have to check that the two sides
of an Or are booleans, the two sides of Divide are integers, the
two sides of GreaterThan are integers, and that the first expression
in the if-else is a boolean and then the other two expressions
are compatible (ie the same according to OO rules):

--------TypeCheckExpVisitor.java---------
  // Exp e1,e2;
  public Type visit(Or n) {     /* Added by DB */
    if (! (n.e1.accept(this) instanceof BooleanType) ) {
       System.out.println("Left side of Or must be of type integer");
       System.exit(-1);
    }
    if (! (n.e2.accept(this) instanceof BooleanType) ) {
       System.out.println("Right side of Or must be of type integer");
       System.exit(-1);
    }
    return new BooleanType();
  }

  // Exp e1,e2;
  public Type visit(GreaterThan n) {    /* Added by DB */
    if (! (n.e1.accept(this) instanceof IntegerType) ) {
       System.out.println("Left side of GreaterThan must be of type integer");
       System.exit(-1);
    }
    if (! (n.e2.accept(this) instanceof IntegerType) ) {
       System.out.println("Right side of GreaterThan must be of type integer");
       System.exit(-1);
    }
    return new BooleanType();
  }

  // Exp e1,e2;
  public Type visit(Divide n) {         /* Added by DB */
    if (! (n.e1.accept(this) instanceof IntegerType) ) {
       System.out.println("Left side of Divide must be of type integer");
       System.exit(-1);
    }
    if (! (n.e2.accept(this) instanceof IntegerType) ) {
       System.out.println("Right side of Divide must be of type integer");
       System.exit(-1);
    }
    return new IntegerType();
  }

  // Exp e1,e2,e3;
  public Type visit(ShortCutIfElse n) {
    if (! (n.e1.accept(this) instanceof BooleanType) ) {
       System.out.println("First expression in shortcut" +
                          " if-else must be of type boolean");
       System.exit(-1);
    }
    Type t1 = n.e2.accept(new TypeCheckExpVisitor() );
    Type t2 = n.e3.accept(new TypeCheckExpVisitor() );
    if (TypeCheckVisitor.symbolTable.compareTypes(t1,t2)==false){
        System.out.println("Second and third expressions in shortcut" +
                           " if-else must be of same type");
        System.exit(0);
    }
    return t1;
  }
------end of TypeCheckExpVisitor.java---------------
```

And finally the do-while typecheck. We just have to see that the
expression is a boolean.

```
------TypeCheckVisitor.java-------------------------
public void visit(DoWhile n) {        /* Added by DB */
    n.s.accept(this);
    if (! (n.e.accept(new TypeCheckExpVisitor()) instanceof BooleanType) ) {
       System.out.println("The condition of do-while must be"+
                          " of type boolean");
       System.exit(-1);
    }
  }
------end of TypeCheckVisitor.java-------------------------
```

End of Guidance