Using Statement and Program
Statement and Program

- The Statement and Program component families for the BL language are similar to what a Java compiler uses to represent a Java program.
- Consider *refactoring* of Java or BL programs, e.g.:
  - Simplifying if-then-else constructs
  - Renaming methods/instructions
The Statement and Program component families for the BL language are similar to what a Java compiler uses to represent a Java program.

- Consider *refactoring* of Java or BL programs, e.g.:
  - Simplifying if-then-else constructs
  - Renaming methods/instructions

*Refactoring* means restructuring code (presumably to improve its readability, etc.) without changing its behavior.
simplifyIfElse for Statement

```c
void simplifyIfElse(Statement s)
```

- Refactors `s` so that every `IF_ELSE` statement with a negated condition (`NEXT_IS_NOT_EMPTY`, `NEXT_IS_NOT_ENEMY`, `NEXT_IS_NOT_FRIEND`, `NEXT_IS_NOT_WALL`) is replaced by an equivalent `IF_ELSE` with the opposite condition and the “then” and “else” `BLOCK`s switched. Every other statement is left unmodified.
simplifyIfElse for Statement

```c
void simplifyIfElse(Statement s)
{
    // Updates: s
    // Ensures:
    s = [#s refactored so that IF_ELSE statements with "not" conditions are simplified so the "not" is removed]
```
The Idea

• Before:

• After:
The same transformation happens for every `IF_ELSE` statement in the entire statement s.
simplifyIfElse for Program

```c
void simplifyIfElse(Program p)

• Refactors p so that every IF_ELSE statement in the program body or the body of any user-defined instruction with a negated condition (NEXT_IS_NOT_EMPTY, NEXT_IS_NOT_ENEMY, NEXT_IS_NOT_FRIEND, NEXT_IS_NOT_WALL) is replaced by an equivalent IF_ELSE with the opposite condition and the “then” and “else” BLOCKs switched. Everything else is left unmodified.
```
simplifyIfElse for Program

```java
void simplifyIfElse(Program p)

• Updates: p
• Ensures:
  p = [#p refactored so that IF_ELSE statements with "not" conditions are simplified so the "not" is removed]
```
Challenge

• Assume you are given an implementation of \texttt{simplifyIfElse(Statement)}

• Write an implementation of \texttt{simplifyIfElse(Program)}
Structure of the Code

{

/*
* For each user-defined instruction body, simplify its IF_ELSE statements
*/

/*
* For the program body, simplify its IF_ELSE statements
*/

}
Structure of the Code

```java
{  
    Map<String, Statement> ctxt = p.newContext();  
    Map<String, Statement> c = p.newContext();  
    p.swapContext(ctxt);  
    while (ctxt.size() > 0) {  
        Map.Pair<String, Statement> instr = ctxt.removeAny();  
        // simplify IF_ELSE for body of instr (a Statement)  
        c.add(name, body);  
    }  
    p.swapContext(c);  
    Statement pBody = p.newBody();  
    p.swapBody(pBody);  
    // simplify IF_ELSE for pBody (a Statement)  
    p.swapBody(pBody);  
}
```
Structure of the Code

```java
{  
    Map<String, Statement> ctxt = p.newContext();
    Map<String, Statement> c = p.newContext();
    p.swapContext(ctxt);
    while (ctxt.size() > 0) {
        Map.Pair<String, Statement> instr = ctxt.removeAny();
        // simplify IF_ELSE for body of instr (a Statement)
        c.add(name, body);
    }
    p.swapContext(c);  
    Statement pBody = p.newBody();
    p.swapBody(pBody);
    // simplify IF_ELSE for pBody (a Statement)
    p.swapBody(pBody);
    }
```

We need to use the `removeAny` style for this loop (not the `iterator` style) because we need to modify the `Map` in the loop body.
Another Refactoring Idea

• Eclipse allows you to change the name of a Java method, including all calls to it, with a single refactoring step
  – See: Refactor >> Rename ...

• You can write code to do the same thing for a BL program; how?
renameInstruction for Statement

```java
void renameInstruction(Statement s, String oldName, String newName)
```

- Refactors `s` by renaming every occurrence of instruction `oldName` to `newName`. Every other statement is left unmodified.
- Updates: `s`
- Requires: `newName` is a valid IDENTIFIER
- Ensures:
  
  ```
  s = [#s refactored so that every occurrence of instruction oldName is replaced by newName]
  ```
The Idea

• Before:

```
CALL oldName
```

• After:

```
CALL newName
```
renameInstruction for Program

void renameInstruction(Program p,
                        String oldName, String newName)

• Refactors p by renaming instruction oldName, and every call to it, to newName. Everything else is left unmodified.
• Updates: p
• Requires:
  oldName is in DOMAIN(p.context) and
  [newName is a valid IDENTIFIER but not the name of a primitive instruction in the BL language] and
  newName is not in DOMAIN(p.context)

• Ensures:
  p = [#p refactored so that instruction oldName and every call to it are replaced by newName]
Example: Rename \textit{bar} to \textit{car}

\begin{verbatim}
PROGRAM Example IS
  INSTRUCTION foo IS
    IF random THEN
      bar
    ELSE
      turnleft
    END IF
  END foo

INSTRUCTION bar IS
  turnright
  turnright
END bar

BEGIN
  WHILE true DO
    IF next-is-empty THEN
      move
    ELSE
      IF next-is-enemy THEN
        bar
      ELSE
        foo
      END IF
    END IF
  END WHILE
END Example
\end{verbatim}
Example: Rename \textit{bar} to \textit{car}

\begin{verbatim}
PROGRAM Example IS
  INSTRUCTION foo IS
    IF random THEN
      \textit{bar}
    ELSE
      turnleft
    END IF
  END foo

INSTRUCTION \textit{bar} IS
  turnright
  turnright
END bar

BEGIN
  WHILE true DO
    IF next-is-empty THEN
      move
    ELSE
      IF next-is-enemy THEN
        bar
      ELSE
        foo
      END IF
    END IF
  END WHILE
END Example
\end{verbatim}
Example: Rename \textit{bar} to \textit{car}

\begin{verbatim}
PROGRAM Example IS
  INSTRUCTION foo IS
    IF random THEN
      car
    ELSE
      turnleft
    END IF
  END foo

  INSTRUCTION car IS
    turnright
    turnright
  END car

BEGIN
  WHILE true DO
    IF next-is-empty THEN
      move
    ELSE
    END IF
    IF next-is-enemy THEN
      car
    ELSE
      foo
    END IF
  END WHILE
END Example
\end{verbatim}
Challenge

• Assume you are *given* an implementation of `renameInstruction(Statement, String, String)`

• Write an implementation of
  `renameInstruction(Program, String, String)`
Draw AST from Code

• What is the value of instr at the end?

Statement instr = new Statement1();
Statement eblk = instr.newInstance();
instr.assembleCall("move");
eblk.addToBlock(0, instr);
Statement tblk = instr.newInstance();
instr.assembleCall("infect");
tblk.addToBlock(0, instr);
instr.assembleCall("turnleft");
tblk.addToBlock(1, instr);
instr.assembleIfElse(Condition.NEXT_IS_ENEMY, tblk, eblk);