1 The Main Program

- There are two ways to use Jex.
- Call Jex with file arguments. Then it will interpret the files one by one.

```
> java jex.Main file1.jex file2.jex
>
```

- Call Jex without file arguments. then it will go into interactive mode and you can type commands one by one.

```
> java jex.Main
jex> 3 + 5;
8
jex>
```

- In the implementation main distinguishes these cases and calls handleInput(Reader input) to do the real work.

2 The Main Program

```
public static void handleInput(Reader input) throws Exception {
    try {
        Parser parser = new Parser(new Scanner(input));
        Tree tree = (Tree)parser.parse().value;
        Analyzer.analyzeTree(tree, symbols, true, imports);
        if (Report.errCount == 0) {
            JexValue value = Interpreter.interpretTree(tree);
            if (value != null && !value.isVoid())
                System.out.println(value);
        }
    } catch (JexException exception) {
        System.out.println("");
        Report.error(exception.pos, exception.getMessage());
    }
    Report.reset();
}
```


3 Applications of Jex (as Program)

- If you are developing a Java program. Typically you would import the classes that you are developing and call their functions.

```
>java jex.Main
jex> import myNewMathPackage;
jex> MyNewMathClass.factorial(3);
6
jex>
```

- You can use Jex for writing test scripts. The file testNewMath.jex:

```
import myNewMathPackage;
if (MyNewMathClass.factorial(3) != 6)
    System.out.println("factorial–test failed");
```

  To test the package you would call:

```
>java jex.Main testNewMath.jex
```

- Especially for interactive programs this might be a good solution. They are sometimes difficult to test.

4 Applications of Jex (Scripts/Macros)

- You can call Jex from other Java programs. Include jex.Main.handleInput(**new** StringReader("System.out.println (5)")); into your code. This fragment calls Jex and lets it interpret "System.out.println (5)".
- If this doesn't look useful, maybe the next one does:

```
String userscript;
String[ ] macros;
... // set macros, userScript;
jex.Main.handleInput(new StringReader(userScript));
jex.Main.handleInput(new StringReader(macros[i]));
```

- An application could allow the user to attach Jex-scripts to buttons. In the case of a button-press event it would execute the Jex-script.
- Scripts that you write for an application can later be ported to Java and built into the application.
- Ports to Java should be very easy (You may need some casts).

5 Applications of Jex (Debugger)

- You can call the interpreter-loop from within your other Java program.

```
jex.Main.main(new String[ ]{});
```

  This allows you to interact with your program at certain points in the program.
- It might act as a tool for debugging. You can call it from your Java-program and then evaluate some expressions. However, you cannot access local variables.

6 Applications of Jex (Command Loop)

- Insert this code into your code

```
jex.Main.handleFile("commands.predef");
jex.Main.handleStdIn();
```

  Your file commands.predef:

```
import myNewMathPackage;
int fac(int n) {
        return MyNewMathClass.factorial(n)
}
```

- This version first defines a command fac(**int**) as a function in the file commands.predef. From the command-line the user can then call these functions.
- But the user can call everything. You might not want to have that.

7 Reflection

Q: Could you write A Cex/Cex++?

- Reflection is the ability to access the program at run-time.
- A language can support different degrees of reflection. It may allow:
  - Accessing variables by their name (as a String).
  - Calling functions or methods by their name.
  - Constructing new functions, methods, and variables.
  - Constructing new classes.
- Jex is quite flexible
  - One can create new variables and functions.
  - One can access classes and methods.
  - One cannot create new classes and methods.
  - However, you can write a new class, compile it and import it from Jex.

8 The Java Reflection Library

We could not write Jex if Java would not have support for reflection.

- Jex is built on top of the Java reflection library.
- In Java their exist classes Class, Method, Field, Constructor, to access these things at run-time.
- You can call methods on them:

```
public JexValue getField(String s) {
        Field f = obj.getClass().getField(s);
        Object res = f.get(obj);
        Class resType = f.getType();
        if (resType.isPrimitive())
                return new JexBasic(res, resType);
        else
                return new JexObject(res);
}
```

9 Reflection and Compilers

- If reflection is not part of the language it cannot be built in (C, C++).
- Interpreted languages often have reflection, because it is easy to implement for them.
- Compiled languages often do not have reflection.
- If a language allows reflection, this restricts the compiler.
  - The compiler cannot remove unused functions from the code.
  - At some points, variables cannot be kept in registers.
  - Constant expression may not be constant any more (they may be changed through reflection).
- On the other side, reflection makes a language very flexible.
- A language with reflection is easier to interface with other languages.