Creating a Layered Text Editor

Hawk Weisman
hawk@hawkweisman.me
http://hawkweisman.me

Department of Computer Science
Allegheny College

March 27, 2015

Text Editors Today

1. Terminal-Based

Text Editors Today

1. Terminal-Based
   - `vim(1)`: released 1991, based on `vi(1)` from 1978

Text Editors Today

1. Terminal-Based
   - `vim(1)`: released 1991, based on `vi(1)` from 1978
   - `emacs(1)`: released 1976

Text Editors Today

1. Terminal-Based
   - `vim(1)`: released 1991, based on `vi(1)` from 1978
   - `emacs(1)`: released 1976
   - Minimal featureset

Text Editors Today

1. Terminal-Based
   - `vim(1)`: released 1991, based on `vi(1)` from 1978
   - `emacs(1)`: released 1976
   - Minimal featureset
   - Extensible and hackable

Text Editors Today

1. Terminal-Based
   - `vim(1)`: released 1991, based on `vi(1)` from 1978
   - `emacs(1)`: released 1976
   - Minimal featureset
   - Extensible and hackable
2. GUI-Based

Text Editors Today

1. Terminal-Based
   - `vim(1)`: released 1991, based on `vi(1)` from 1978
   - `emacs(1)`: released 1976
   - Minimal featureset
   - Extensible and hackable
2. GUI-Based
   - IDEs (Eclipse, Visual Studio, IntelliJ IDEA, XCode ...)

Text Editors Today

1. Terminal-Based
    - `vim(1)`: released 1991, based on `vi(1)` from 1978
    - `emacs(1)`: released 1976
    - Minimal featureset
    - Extensible and hackable
2. GUI-Based
    - IDEs (Eclipse, Visual Studio, IntelliJ IDEA, XCode …)
        - Many features built in

Text Editors Today

1. Terminal-Based
   - `vim(1)`: released 1991, based on `vi(1)` from 1978
   - `emacs(1)`: released 1976
   - Minimal featureset
   - Extensible and hackable
2. GUI-Based
   - IDEs (Eclipse, Visual Studio, IntelliJ IDEA, XCode …)
     - Many features built in
     - Often language-specific

Text Editors Today

1. Terminal-Based
   - `vim(1)`: released 1991, based on `vi(1)` from 1978
   - `emacs(1)`: released 1976
   - Minimal featureset
   - Extensible and hackable
2. GUI-Based
   - IDEs (Eclipse, Visual Studio, IntelliJ IDEA, XCode ...)
     - Many features built in
     - Often language-specific
     - Information overload?

Text Editors Today

1. Terminal-Based
   - `vim(1)`: released 1991, based on `vi(1)` from 1978
   - `emacs(1)`: released 1976
   - Minimal featureset
   - Extensible and hackable
2. GUI-Based
   - IDEs (Eclipse, Visual Studio, IntelliJ IDEA, XCode ...)
     - Many features built in
     - Often language-specific
     - Information overload?
   - Text Editors (SublimeText, notepad++, TextMate ...)

Text Editors Tomorrow

- Many programmers still prefer terminal-based editors

Text Editors Tomorrow

- Many programmers still prefer terminal-based editors
- But programming in 2015 is very different from programming in 1976

Text Editors Tomorrow

- Many programmers still prefer terminal-based editors
- But programming in 2015 is very different from programming in 1976
- **Idea**: display information in *layers*

Text Editors Tomorrow

- Many programmers still prefer terminal-based editors
- But programming in 2015 is very different from programming in 1976
- **Idea**: display information in *layers*
  - First proposed by Gary Bernhardt in *A Whole New World*

Text Editors Tomorrow

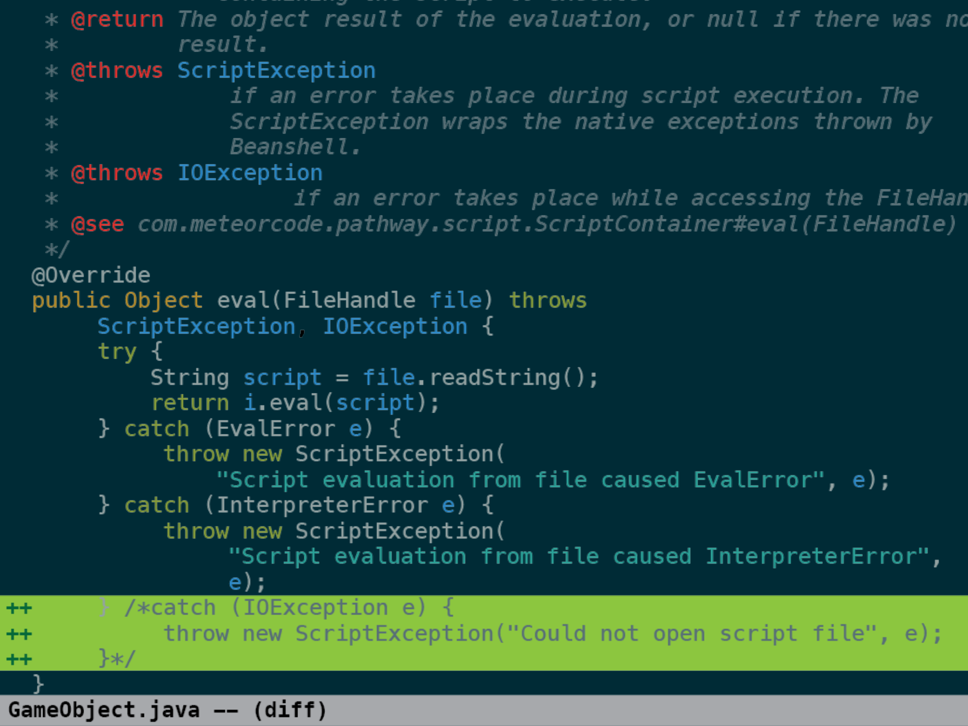
- Many programmers still prefer terminal-based editors
- But programming in 2015 is very different from programming in 1976
- **Idea**: display information in *layers*
    - First proposed by Gary Bernhardt in *A Whole New World*
    - **Modal editors:** multiplex keyboard commands

Text Editors Tomorrow

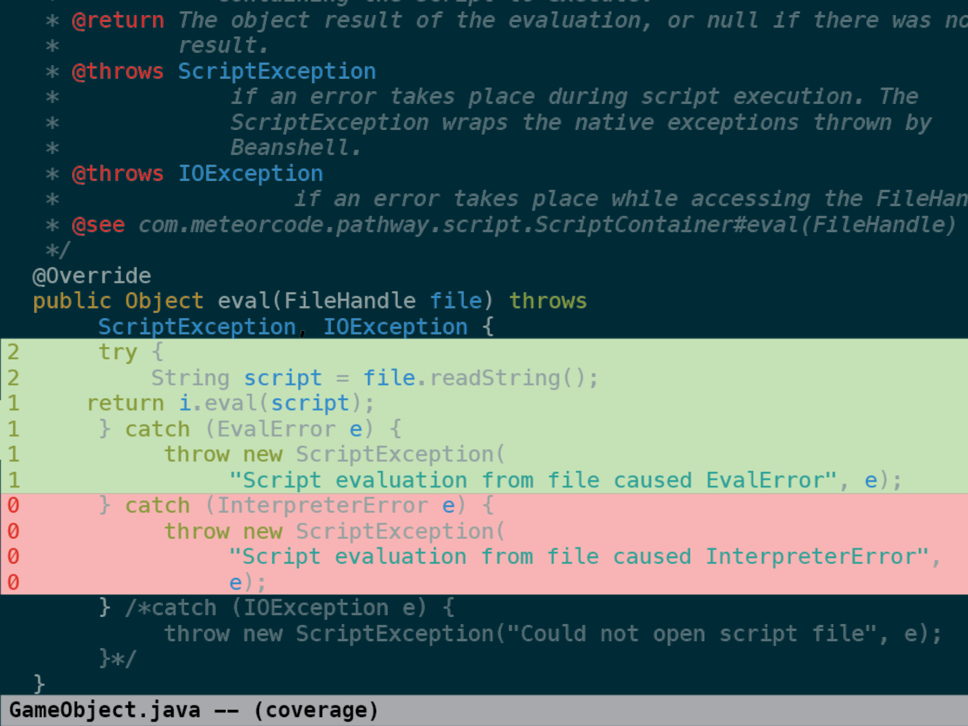
- Many programmers still prefer terminal-based editors
- But programming in 2015 is very different from programming in 1976
- **Idea**: display information in *layers*
  - First proposed by Gary Bernhardt in *A Whole New World*
  - **Modal editors:** multiplex keyboard commands
  - **Layered editors:** multiplex information display

```java
 *                 containing the script to execute.
 * @return The object result of the evaluation, or null if there was no
 *         result.
 * @throws ScriptException
 *             if an error takes place during script execution. The
 *             ScriptException wraps the native exceptions thrown by
 *             Beanshell.
 * @throws IOException
 *                  if an error takes place while accessing the FileHan
 * @see com.meteorcode.pathway.script.ScriptContainer#eval(FileHandle)
 */
@Override
public Object eval(FileHandle file) throws
     ScriptException, IOException {
     try {
         String script = file.readString();
         return i.eval(script);
     } catch (EvalError e) {
          throw new ScriptException(
              "Script evaluation from file caused EvalError", e);
     } catch (InterpreterError e) {
          throw new ScriptException(
               "Script evaluation from file caused InterpreterError",
               e);
     } /*catch (IOException e) {
          throw new ScriptException("Could not open script file", e);
     }*/
}
```

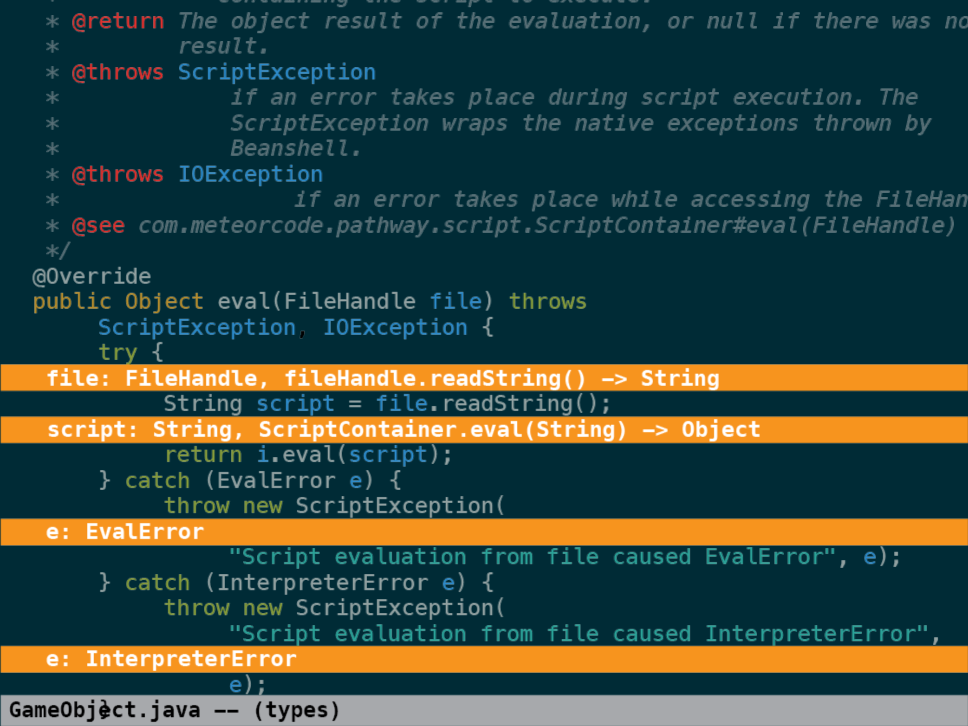
GameObject.java



```
   * @return The object result of the evaluation, or null if there was no
   *         result.
   * @throws ScriptException
   *             if an error takes place during script execution. The
   *             ScriptException wraps the native exceptions thrown by
   *             Beanshell.
   * @throws IOException
   *                  if an error takes place while accessing the FileHan
   * @see com.meteorcode.pathway.script.ScriptContainer#eval(FileHandle)
   */
  @Override
  public Object eval(FileHandle file) throws
       ScriptException, IOException {
       try {
           String script = file.readString();
           return i.eval(script);
       } catch (EvalError e) {
            throw new ScriptException(
                "Script evaluation from file caused EvalError", e);
       } catch (InterpreterError e) {
            throw new ScriptException(
                 "Script evaluation from file caused InterpreterError",
                 e);
++     } /*catch (IOException e) {
++          throw new ScriptException("Could not open script file", e);
++     }*/
  }
GameObject.java -- (diff)
```



```
   *         containing the script to execute.
   * @return The object result of the evaluation, or null if there was no
   *         result.
   * @throws ScriptException
   *             if an error takes place during script execution. The
   *             ScriptException wraps the native exceptions thrown by
   *             Beanshell.
   * @throws IOException
   *                 if an error takes place while accessing the FileHan
   * @see com.meteorcode.pathway.script.ScriptContainer#eval(FileHandle)
   */
  @Override
  public Object eval(FileHandle file) throws
      ScriptException, IOException {
2     try {
2         String script = file.readString();
1     return i.eval(script);
1     } catch (EvalError e) {
1         throw new ScriptException(
1             "Script evaluation from file caused EvalError", e);
0     } catch (InterpreterError e) {
0         throw new ScriptException(
0             "Script evaluation from file caused InterpreterError",
0             e);
      } /*catch (IOException e) {
          throw new ScriptException("Could not open script file", e);
      }*/
  }
GameObject.java -- (coverage)
```



```
   *               containing the script to execute.
   * @return The object result of the evaluation, or null if there was no
   *         result.
   * @throws ScriptException
   *             if an error takes place during script execution. The
   *             ScriptException wraps the native exceptions thrown by
   *             Beanshell.
   * @throws IOException
   *                  if an error takes place while accessing the FileHan
   * @see com.meteorcode.pathway.script.ScriptContainer#eval(FileHandle)
   */
  @Override
  public Object eval(FileHandle file) throws
       ScriptException, IOException {
       try {
 file: FileHandle, fileHandle.readString() -> String
            String script = file.readString();
 script: String, ScriptContainer.eval(String) -> Object
            return i.eval(script);
       } catch (EvalError e) {
            throw new ScriptException(
 e: EvalError
                 "Script evaluation from file caused EvalError", e);
       } catch (InterpreterError e) {
            throw new ScriptException(
                 "Script evaluation from file caused InterpreterError",
 e: InterpreterError
                 e);
GameObject.java -- (types)
```

Evaluation

1. User Studies

Evaluation

1. User Studies
   - Students in introductory programming courses (112, 210)

Evaluation

1. User Studies
   - Students in introductory programming courses (112, 210)
   - **Performance:** time to complete assignments

Evaluation

1. User Studies
    - Students in introductory programming courses (112, 210)
    - **Performance:** time to complete assignments
    - **Opinions:** satisfaction, ease of use, ease to learn

Evaluation

1. User Studies
   - Students in introductory programming courses (112, 210)
   - **Performance:** time to complete assignments
   - **Opinions:** satisfaction, ease of use, ease to learn
2. Plugin Developers

Evaluation

1. User Studies
   - Students in introductory programming courses (112, 210)
   - **Performance:** time to complete assignments
   - **Opinions:** satisfaction, ease of use, ease to learn
2. Plugin Developers
   - Open-source release

Evaluation

1. User Studies
   - Students in introductory programming courses (112, 210)
   - **Performance:** time to complete assignments
   - **Opinions:** satisfaction, ease of use, ease to learn
2. Plugin Developers
   - Open-source release
   - Collect feedback

Evaluation

1. User Studies
    - Students in introductory programming courses (112, 210)
    - **Performance:** time to complete assignments
    - **Opinions:** satisfaction, ease of use, ease to learn
2. Plugin Developers
    - Open-source release
    - Collect feedback
    - Compare lines of code with plugins for other editors

Questions?

Contact me: hi@hawkweisman.me

More information: `http://hawkweisman.me/notebook/ideas/2015/03/04/a-layer-based-text-editor/`