Code Weaver: An Aspect Oriented Approach to Merging Java Files Together
Michael Peterson
Faculty Advisor: Professor Heliotis

Introduction
- Aspect Oriented Programming (AOP) is a programming style to address specific areas of concern by a user
- Code Weaver is an AOP tool used to merge Java classes together
- Code Weaver is a re-imagining of IBM's Hyper/J

Hyper/J
- A tool created by IBM
- Allows users to define areas of concern
- User specifies ways to handle merging concerns together
- Several different ways to merge concerns
- Hyper/J is no longer updated and supported
- Serves as a basis for Code Weaver

Project Goals
- Reconstruct Hyper/J's fundamental core features
- Remove the complicated, unnecessary, and outdated parts of Hyper/J
- Simplify the learning process for Code Weaver
- Make Code Weaver intuitive enough to use as a teaching tool in an AOP class

Workflow
Code Weaver is split into two parts: the Parser and the Weaver (implemented by John Rivera)
- Parser interprets a Weave File
- Parser sends commands to the Weaver
- Weaver weaves the byte code of supplied Java classes
- Weaver sends new Java class location to Parser
- Parser displays location to user

Weave File
- The Weave File is input given by the user for the Parser to interpret and send to the Weaver
- 3 parts: File Path, File Imports, and Merge Cases

```
WeaveFile = Path Import* MergeCase*
Path = Path filepath;
Import = Import filename;
MergeCase = DefaultClassName(className, className, outputClassName) (CaseByCase)
DefaultClassName = MergeByClassName NonCorrespondingMerge OverrideByClassName
CaseByCase = Merge(functionName, functionName, outputFunctionName)
NonMerge(functionName); | Override(functionName, functionName);
```

Merge Cases
Merge Cases deal with how to merge two methods from separate classes together. They can be specified from a Default Case, which automatically matches two methods sharing a name together, or by a Case By Case, which allows for more specification and takes precedence.
- Merge Case - Merges two methods into one
- No Merge Case - Keeps two methods separated
- Override Case - Removes first method for the second

Analysis & Conclusion
- Successfully merges two classes into one
- Handles updating auxiliary classes
- Cannot handle non-void methods
- Remains simple and easy to use