



# The Trait Extractor: A Tool for Generating Traits in Java

Emerson Murphy-Hill and Andrew P. Black  
Portland State University



The research supported by the National Science Foundation for the Research Experience for Undergraduates. Grant numbers: 0334616 and 0366323.

**Purpose**  
Create a tool to assist trait-building in Java within the Eclipse IDE.

**Procedure**

- Build tool as a plugin to the Eclipse IDE
- Leverage built-in Java source manipulation library
- Integrate with pre-existing trait plugin
- Model functionality after Smalltalk version of tool

**Objective**  
Create a tool that facilitates turning a class into a trait.

**What's a Trait?**

- A collection of pure methods
- Like a class, but without state or a superclass
- Reusable building blocks for "use" in classes
- Alternative solutions: duplication, multiple inheritance, and mixins

**A Need for the Tool**  
Traits are usually created from pre-existing code in classes, rather than from scratch.

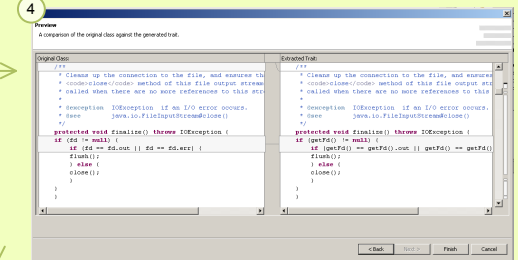
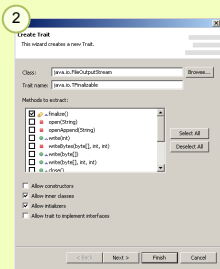
**Findings**  
Traits in Java must support features not present in the Smalltalk implementation, including constructors, initializers, and inner classes.

**Future Directions**

- Use tool in context of major software project
- Allow complete refactoring: integrate trait into original class
- Allow explicit super calls in traits, as in super.method()

1 **class**  
java.io.FileOutputStream

5 **trait**  
java.io.TFinalizable



What the User Sees

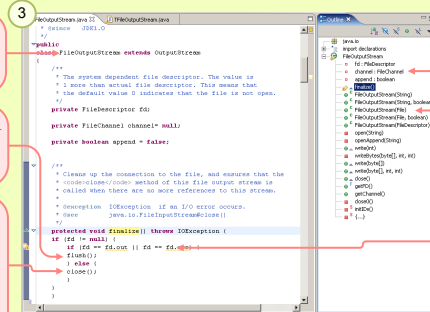
What The Tool Does

**The Alternative**  
Without this tool, users would have to copy and modify the class manually, at the cost of possible compilation errors and unanticipated runtime bugs.

**Change Class Signature**  
Original class must be made abstract, so that "required" methods are allowed. Class name is changed. Superclass is removed.  
`public abstract class TFileOutputStream`

**Called Super Methods Overridden by Abstract Methods**  
Super methods which are called by the defined methods are overridden with local abstract methods.  
`OutputStream >> public void flush() throws IOException`  
`TFinalizable >> public abstract void flush() throws IOException`

**Deleted Methods Replaced By Abstract Methods**  
Methods that are referenced, but the user does not desire to be defined in the trait, are made abstract. These are called "required" methods.  
`public void close() throws IOException {`  
`if (channel != null)`  
`channel.close();`  
`close0();`  
`}`  
`public abstract void close() throws IOException;`



**Removal of Fields**  
All fields, both static and instance, are deleted.

**Removal of Non-Selected Members**  
Members that were not selected by the user are removed (methods, initializers, and constructors).

**Field References Replaced by Method References**  
Referenced fields, which are now invalid, are replaced by method references. Abstract methods are added to make the reference valid. These are called "required" methods.  
`if (fd != null) {`  
`if (fd == fd.out || fd == fd.err) { ... }`  
`}`  
`if (getFd() != null) {`  
`if (getFd() == getFd().out || getFd() == getFd().err) { ... }`  
`}`  
`protected abstract FileDescriptor getFd();`

**Performance**  
The entire operation generally takes up to 10 seconds on a typical development machine, but performance is not the primary concern.