Bytecode Engineering & Software Development

Agenda

- Develop a Java application with a bytecode engineering toolkit
  - Manipulate Java bytecode, complied from the source code.

What is Reflection?

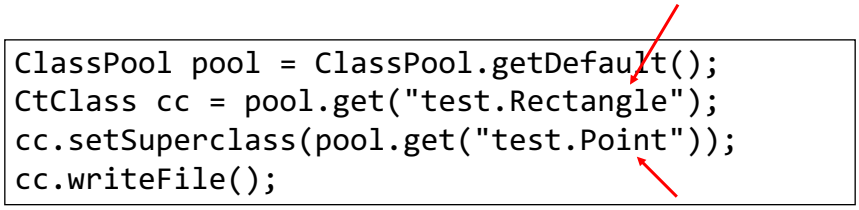
- Meaning
  - One of the meanings of reflection (in English) is
    - "introspection" : contemplation of one-self.
  - In other words, reflection is the ability of a thing to talk about itself.
- In computer science, (computational) **reflection** is the ability of a program to **examine and control its own implementation**.
- Computational reflection is good for
  - Extending a language during programming language design
  - Building learning systems, self - modifying and - optimizing applications
  - Building programming environments and advanced software development tools

Bytecode Engineering

- Bytecode manipulation.
- Editing bytecode, such as Java class files.
- E.g., adding a new class at runtime, or modifying a class file when the Java virtual machine loads it
- JDK introspects a program, but it is hard to modify program behavior at runtime.
- A bytecode engineering library – Javassist
  - http://jboss-javassist.github.io/javassist/
  - An extension to the reflection JDK APIs

An Example in Javassist

- Javassist
  - A class library for dealing with Java bytecode.
  - Java bytecode is stored in a binary file called a class file.



```
ClassPool pool = ClassPool.getDefault();
CtClass cc = pool.get("test.Rectangle");
cc.setSuperclass(pool.get("test.Point"));
cc.writeFile();
```

Program Reflection

- Java supports reflection APIs.
- Limited to introspection.
  - Monitoring data structures used in a program such as a class.
- Altering program behavior is very limited.
  - Instantiating a class, changing a field value, or invoking a method through reflection APIs.
- Javassist provides an extension of reflection APIs, structural reflection
  - Other toolkits, BCEL, JMangler, etc.
- Structural reflection
  - Allowing a program to change its structure, e.g., the definition of a class or a method.

Javassist – Bytecode Engineering Library

- The bytecode-level translation has advantages
  - Processing an off-the-shelf program or library
  - Applying program transformations on demand at load time
- Disadvantage of the bytecode translation
  - Hard to use detailed specifications (APIs) of the Java bytecode
- Javassist – Bytecode Translator Toolkit
  - Provide higher-level abstraction than raw bytecode
  - Source-level abstraction based on the reflective architecture
- JBoss (jboss.org), Hibernate (hibernate.org), Spring (spring.io)

High-Level Representation for Program Transformation

- Allowing the users to describe transformation with source-level vocabulary

```
CtClass cc = pool.get("test.Rectangle");
cc.setSuperclass(pool.get("test.Point"));
```

- Translating a Java class file into several objects
  - Representing a class, field, or method
- Accessing the "meta" objects for transformation
- Introducing a superclass or a new field to the class is performed through modifying these objects

High-Level Representation for Program Transformation

- The modifications are applied to the metaobjects
- Translating changes back into the modifications of the class file
- The transformation is reflected on the class definition
- The CtClass object
  - An object representing a class obtained from the given class file.
  - Similar to the java.lang.Class class of the standard reflection API.

Methods for Modifying a Class

- Inspecting data structures, such as a class, used in a program
  - The getName method declared in CtClass returns the name of the class,
  - The getSuperclass method returns the CtClass object representing the super class
  - getFields – CtField object representing a field
  - getMethods – CtMethod object representing a method
  - getConstructors – CtConstructor object representing a constructor

Methods in CtClass	Description
`void setName(String name)`	change the class name
`void setModifiers(int m)`	change the class modifiers such as `public`
`void setSuperclass(CtClass c)`	change the super class
`void setInterfaces(CtClass[] i)`	change the interfaces
`void addField(CtField f, String i)`	add a new field
`void addMethod(CtMethod m)`	add a new method
`void addConstructor(CtConstrutor c)`	add a new constructor

Methods for Modifying Program Elements

- The addMethod method
  - Adding a new method to the class.
  - The definition of the new method is given in the form of string.

```
CtClass cc = ...
CtMethod newMethod = CtNewMethod.make("void foo() { .. }", cc);
cc.addMethod(newMethod);
```

  - Compiling the source text into bytecode on the fly
  - Adding the bytecode into the class file

Methods for Modifying Program Elements

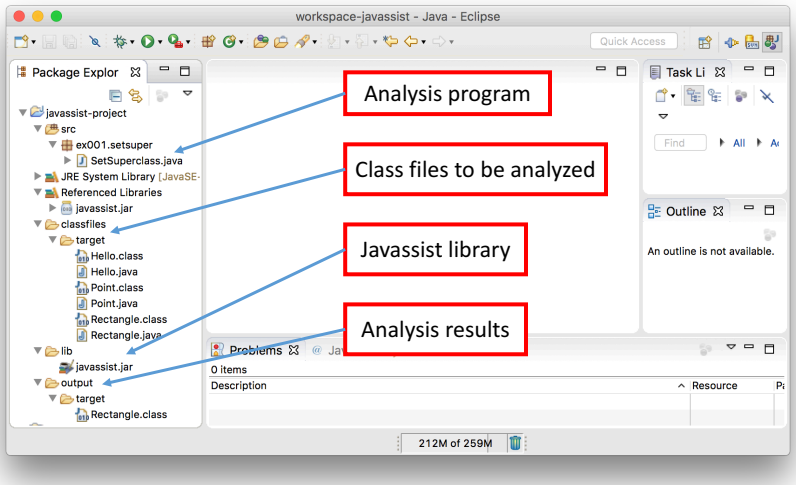
- The addField method
  - Adding a new field with source text representing the initial value
  - Compiling the source text
  - Inserting it in the constructor body
  - Initializing the field

```
CtClass ct = ...;
CtField f = new CtField(CtClass.intType, "z", ct);
pointClass.addField(f);
```

First Example to Set a Superclass

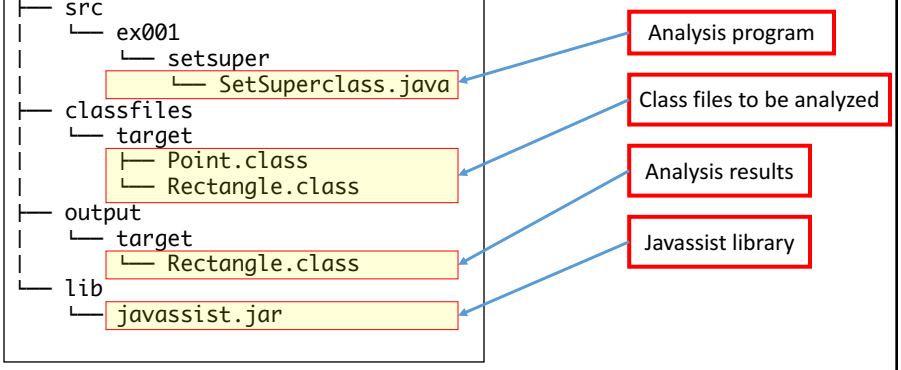
- Using Eclipse IDE
  - Eclipse – version 4.6, Neon
  - https://www.eclipse.org/downloads/packages/release/neon/3
  - Select a package, Eclipse IDE for RCP and RAP Developers
  - Run Eclipse and create an Java project, called "javassist-project" in Eclipse
- Installing a library named javassist.jar
  - https://github.com/jboss-javassist/javassist/releases
  - Select Javassist 3.21.0-GA
  - Download the above jar file into a new directory called "lib" in the "javassist-project"
  - Select "javassist.jar" > Build Path > Add to Build Path

Program Structure



Project Structure

```
javassist-project >
.
├── src
|   └── ex001
|       └── setsuper
|           └── SetSuperclass.java
├── classfiles
|   └── target
|       ├── Point.class
|       └── Rectangle.class
├── output
|   └── target
|       └── Rectangle.class
└── lib
    └── javassist.jar
```



Class Files To Be Analyzed

- Take as input these programs as bytecode (*.class files)

```
package target;

public class Rectangle {
   public int getVal() {
      return 1;
   }
}
```

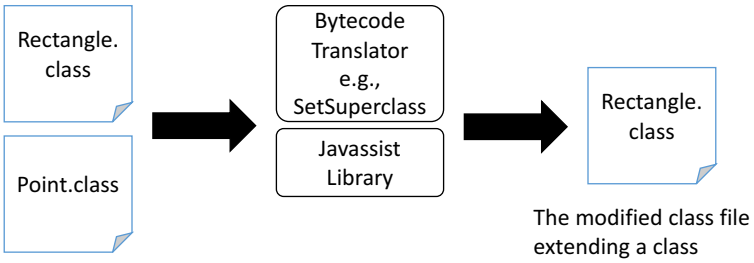
```
package target;

public class Point {
   int x, y;

   public void move(Integer dx,
                    Integer dy) {
      x += dx;  y += dy;
   }


   public void move(int dx, int dy) {
      x += dx;  y += dy;
   }
}
```

Overview



Example Program: SetSuperclass

- Access to bytecode "Rectangle" and "Point", modify "Rectangle", and save the modified class as a class file.

```
public class SetSuperclass {
   static String workDir   = System.getProperty("user.dir");
   static String outputDir = workDir + File.separator + "output";

   public static void main(String[] args) {
         ClassPool pool = ClassPool.getDefault(); (1)

         insertClassPath(pool); (2)

         CtClass cc = pool.get("target.Rectangle"); (3)
   (4)   setSuperclass(cc, "target.Point", pool);

   (5)   cc.writeFile(outputDir);
         System.out.println("[DBG] write output to: " + outputDir);
   }

   static void insertClassPath(ClassPool pool) { .. }
   static void setSuperclass(CtClass c, String s, ClassPool p) { .. }
}
```

The javassist.ClassPool Object

- This program first obtains a ClassPool object
  - Control bytecode modification with Javassist.
- The ClassPool is a container of CtClass representing a class file.
  - Read a class file on demand for constructing a CtClass object
  - Record the constructed object for responding later accesses.

```
public class SetSuperclass {
   static String workDir   = System.getProperty("user.dir");
   static String outputDir = workDir + File.separator + "output";

   public static void main(String[] args) {
         ClassPool pool = ClassPool.getDefault(); (1)

         insertClassPath(pool); (2)

         CtClass cc = pool.get("target.Rectangle"); (3)
   (4)   setSuperclass(cc, "target.Point", pool);

   (5)   cc.writeFile(outputDir);
```

Inserting ClassPath

- Searching for the input bytecode in the given directory path
  - E.g., "/path-a/path-b/../javassist-project/classfiles/"

```
javassist-project > ls classfiles/target/

Point.class      Rectangle.class
```

- insertClassPath(java.lang.String pathname)
  - Inserting a directory or a jar (or zip) file at the head of the search path.

```
   static void insertClassPath(ClassPool pool) throws NotFoundException {
      String strClassPath = workDir + File.separator + "classfiles";
      pool.insertClassPath(strClassPath);
      System.out.println("[DBG] insert classpath: " + strClassPath);
   }
```

Method ClassPool.get()

- Obtain a reference to a CtClass from a ClassPool object to modify the definition of a class
  - E.g., use method ClassPool.get() to obtain a reference to class "target.Rectangle"

```
public class SetSuperclass {
   static String workDir   = System.getProperty("user.dir");
   static String outputDir = workDir + File.separator + "output";

   public static void main(String[] args) {
         ClassPool pool = ClassPool.getDefault();  1

         insertClassPath(pool);  2

         CtClass cc = pool.get("target.Rectangle");  3
    4    setSuperclass(cc, "target.Point", pool);

    5    cc.writeFile(outputDir);
         System.out.println("[DBG] write output to: " + outputDir);
   }
```

Method ClassPool.get()

```
static void setSuperclass(CtClass curClass, String superClass, ClassPool pool) {
   curClass.setSuperclass(pool.get(superClass));
   println("[DBG] set superclass: " + curClass.getSuperclass().getName() + //
         ", subclass: " + curClass.getName());
}
```

```
public class SetSuperclass {
   static String workDir   = System.getProperty("user.dir");
   static String outputDir = workDir + File.separator + "output";

   public static void main(String[] args) {
         ClassPool pool = ClassPool.getDefault();  1

         insertClassPath(pool);  2

         CtClass cc = pool.get("target.Rectangle");  3
    4    setSuperclass(cc, "target.Point", pool);

    5    cc.writeFile(outputDir);
         System.out.println("[DBG] write output to: " + outputDir);
   }
```

Method CtClass.writeFile()

- The change is reflected on the original class file by a call to CtClass.writeFile().
- Translating the CtClass object into a class file
- Writing a class file on a local disk.

```
public class SetSuperclass {
   static String workDir   = System.getProperty("user.dir");
   static String outputDir = workDir + File.separator + "output";

   public static void main(String[] args) {
         ClassPool pool = ClassPool.getDefault();  1

         insertClassPath(pool);  2

         CtClass cc = pool.get("target.Rectangle");  3
    4    setSuperclass(cc, "target.Point", pool);

    5    cc.writeFile(outputDir);
         System.out.println("[DBG] write output to: " + outputDir);
   }
```