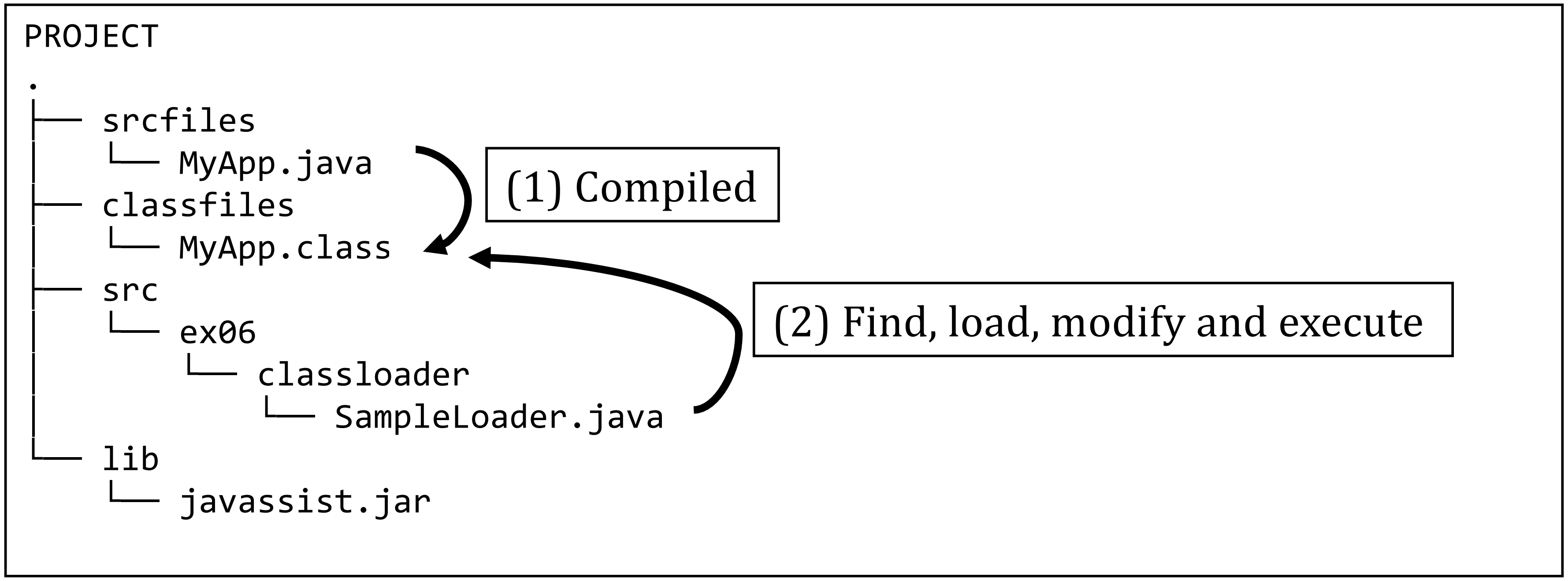
Develop a Class Loader

- Implement a class loader that can load a particular program, while modifying the program structure.



```
PROJECT
.
├── srcfiles
│   └── MyApp.java
├── classfiles
│   └── MyApp.class
├── src
│   └── ex06
│       └── classloader
│           └── SampleLoader.java
└── lib
    └── javassist.jar
```

Lab Practice

Import an Example Eclipse Project

- Download
  - Canvas > Module > Download > Code Examples > *Today's Date*
  - javassist-project-MMDD.zip
- Import
  - Import an archive file, javassist-project-MMDD.zip
    - Menu > Import > Existing Projects into Workspace > Select archive file
- Run the program
  - Select the program > Context Menu > Run As > Java Application

Original Program, MyApp

- MyApp.getClass().getField() – Return a Field object that reflects the specified public member field of the class.
- Throws: NoSuchFieldException, NullPointerException, and SecurityException

```java
import java.lang.reflect.Field;

public class MyApp {
   public static void main(String[] args) {
      MyApp localMyApp = new MyApp();
      localMyApp.foo();
      System.out.println(localMyApp.getClass().getField("hiddenValue").getName());
   }

   public void foo() {
      System.out.println("Called foo.");
   }
}
```

Original and Modified Versions

```
public class MyApp {
   public static void main(String[] args) {
      MyApp localMyApp = new MyApp();
      localMyApp.foo();
      System.out.println(localMyApp.getClass().getField("hiddenValue").getName());
   }

   public void foo() { .. }
}
```

```
public class MyApp {
   public int hiddenValue;

   public static void main(String[] args) {
      MyApp localMyApp = new MyApp();
      localMyApp.foo();
      System.out.println(localMyApp.getClass().getField("hiddenValue").getName());
   }

   public void foo() { .. }
}
```

Retrieving Class Objects

- java.lang.Class
  - The reflection operation's entry point
- Invoke appropriate methods on Class.

1. Object.getClass()
   - Get the Class object

```
MyApp localMyApp = new MyApp();
localMyApp.getClass().getField("hiddenValue").getName()
```

2. The .class syntax
   - The type is available
   - Obtain a Class by appending ".class"

```
MyApp.class.getField("hiddenValue").getName()
```

Retrieving Class Objects

3. Class.forName()
    - Fully-qualified name of a class is available
        - Static method Class.forName()
        - Cannot be used for primitive types
    - Class.getName()
        - The syntax for names of array classes
            - e.g., (new int[3]).getClass().getName() returns "[I;"
        - Applicable to references and primitive types

```
Class.forName("MyApp").getField("hiddenValue").getName();
```

Implement a Class Loader

(1) Inherit the class java.lang.ClassLoader

(3) Execute MyApp.main() method

(2) Override the method findClass

```java
public class SampleLoader extends ClassLoader {
   private ClassPool pool;

   public static void main(String[] args) {
      SampleLoader s = new SampleLoader();
      Class<?> c = s.loadClass("MyApp");
      c.getDeclaredMethod("main", new Class[] { String[].class }).
           invoke(null, new Object[] { args });
   }

   public SampleLoader() throws NotFoundException {
      pool = new ClassPool();
      pool.insertClassPath(inputDir);
   }

   protected Class<?> findClass(String name) {
   /* Finds a specified class.
    * The bytecode for this class can be modified.
    */
   }
}
```

The ClassLoader.loadClass Method

- A class loader
  - Responsible for loading classes.
- Attempt to locate or generate data for a definition for the class.
- A reference to the ClassLoader
- ClassLoader.loadClass() invokes
  - (1) "findLoadedClass()",
  - (2) "loadClass()" on the parent class loader, and
  - (3) "findClass()"

```java
public class SampleLoader extends ClassLoader {

   public static void main(String[] args) throws Throwable {
      SampleLoader s = new SampleLoader();
      Class<?> c = s.loadClass("MyApp");
```

Class<?>.getDeclaredMethod().invoke()

- Class<?>.getDeclaredMethod()
  - Return a Method object.
- java.lang.reflect.Method.invoke()
  - Invoke the underlying method.

```
public class SampleLoader extends ClassLoader {

   public static void main(String[] args) throws Throwable {
      SampleLoader s = new SampleLoader();
      Class<?> c = s.loadClass("MyApp");
      c.getDeclaredMethod("main", new Class[] { String[].class }).
           invoke(null, new Object[] { args });
   }
}
```

```
Class<?> c = Class.forName("OtherExampleApp");
Object t = c.newInstance();
Object o = m.invoke(t, ..);
```

ClassPool

- A container of CtClass objects.
  - A CtClass object must be obtained from this object.
- ClassPool.get() –
  - Search various sources represented by ClassPath to find a class file.

```java
public class SampleLoader extends ClassLoader {

   private ClassPool pool;

   public SampleLoader() throws NotFoundException {
      pool = new ClassPool();
      pool.insertClassPath(inputDir);
   }

   protected Class<?> findClass(String name) {
      CtClass cc = pool.get(name);
// (1) Find a specified class and (2) modify the bytecode for the identified class.
   }
}
```

(1) The Object container

(2) Extending the class search feature

ClassLoader.findClass()

- Subclasses of ClassLoader
  - The feature that the JVM dynamically loads classes.
- JVM loads classes from the local file system platform-dependently.
  - E.g., JVM loads classes by using the CLASSPATH variable on Linux.

```
   Class<?> findClass(String name) {
         CtClass cc = pool.get(name);
         if (name.equals("MyApp")) {
            CtField f = new CtField(CtClass.intType, "hiddenValue", cc);
            f.setModifiers(Modifier.PUBLIC);
            cc.addField(f);
         }
         byte[] b = cc.toBytecode();
         return defineClass(name, b, 0, b.length);
   }
```

Adding a CtField Object

- An instance of CtField represents a field.
- CtClass.addField() – Add the created field
- CtClass.toBytecode() – Convert the class to a class file.
- ClassLoader.defineClass() – Convert into an instance of Class type.

```
Class<?> findClass(String name) {
      CtClass cc = pool.get(name);
      if (name.equals("MyApp")) {
         CtField f = new CtField(CtClass.intType, "hiddenValue", cc);
         f.setModifiers(Modifier.PUBLIC);
         cc.addField(f);
      }
      byte[] b = cc.toBytecode();
      return defineClass(name, b, 0, b.length);
}
```