```
public static void main(String[] args) {
        ClassPool pool = ClassPool.getDefault();
        boolean useRuntimeClass = true;
        if (useRuntimeClass) {
            ClassClassPath classPath = new
ClassClassPath(new Rectangle().getClass());
            pool.insertClassPath(classPath);
        } else {
            String strClassPath = workDir +
"\\bin";
            pool.insertClassPath(strClassPath);
        }
        CtClass cc = pool.get("target.Rectangle");

cc.setSuperclass(pool.get("Target.Point"));//takes
a CtClass
        cc.writeFile(outputDir);//outputDir is a
string

         ClassPool pool = ClassPool.getDefault();
         boolean useRuntimeClass = true;
         if (useRuntimeClass) {
           ClassClassPath classPath = new
ClassClassPath(new Rectangle().getClass());
            pool.insertClassPath(classPath);
         } else {
              String strClassPath = workDir +
         "\\bin";
               pool.insertClassPath(strClassPath);
         CtClass cc = pool.get("target.Rectangle");

curClass.setSuperclass(pool.get(superClass));
         cc.writeFile(outputDir);

         ClassPool pool = ClassPool.getDefault();
         CtClass cc = pool.makeClass(newClassName);
         cc.writeFile(outputDir);
         CtClass ccInterface =
pool.makeInterface(newInterfaceName);
         ccInterface.writeFile(outputDir);

         ClassPool pool = ClassPool.getDefault();
      String strClassPath = outputDir;
      pool.insertClassPath(strClassPath);
         CtClass ccPoint2 =
pool.makeClass("Point2");
         ccPoint2.writeFile(outputDir);
         CtClass ccRectangle2 =
pool.makeClass("Rectangle2");
         ccRectangle2.writeFile(outputDir);
         // ccRectangle2.defrost(); //
modifications of the class definition will be
permitted.
         ccRectangle2.setSuperclass(ccPoint2);
         ccRectangle2.writeFile(outputDir);

        CtMethod m = cc.getDeclaredMethod("say");
        m.insertBefore("{
System.out.println(\"Hello.say:\"); }");
        Class<?> c = cc.toClass();
        Hello h = (Hello) c.newInstance();
        h.say();

   private static String       workDir           =
System.getProperty("user.dir");
   private static final String TARGET_POINT     =
"target.Point";
   private static final String TARGET_RECTANGLE =
"target.Rectangle";
         ClassPool cp = ClassPool.getDefault();
      String strClassPath = workDir +
File.separator + "classfiles";
      pool.insertClassPath(strClassPath);
         Loader cl = new Loader(cp);
         CtClass cc = cp.get(TARGET_RECTANGLE);
         cc.setSuperclass(cp.get(TARGET_POINT));
         Class<?> c =
cl.loadClass(TARGET_RECTANGLE);
         Object rect = c.newInstance();
         System.out.println("[DBG] rect object: " +
rect);
         Class<?> rectClass = rect.getClass();
         Method m =
rectClass.getDeclaredMethod("getVal", new Class[]
{});
         System.out.println("[DBG] method: " + m);
         System.out.println("[DBG] result: " +
m.invoke(rect, new Object[] {}));

   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
});
   }
   private ClassPool pool;
   public SampleLoader() throws NotFoundException {
      pool = new ClassPool();
      pool.insertClassPath(inputDir); //
MyApp.class must be there.
   }
public static void main(String[] args) throws
Throwable {
      SubstituteMethodBody s = new
SubstituteMethodBody();
      Class<?> c = s.loadClass(TARGET_MY_APP);
      Method mainMethod =
c.getDeclaredMethod("main", new Class[] {
String[].class });
      mainMethod.invoke(null, new Object[] { args
});
   }

   protected Class<?> findClass(String name) throws
ClassNotFoundException {
      CtClass cc = null;
      try {
         cc = pool.get(name);
         cc.instrument(new ExprEditor() {
            public void edit(MethodCall m) throws
CannotCompileException {
               String className = m.getClassName();
               String methodName =
m.getMethodName();

               if (className.equals(TARGET_MY_APP)
&& methodName.equals(DRAW_METHOD)) {
                  System.out.println("[Edited by
ClassLoader] method name: " + methodName + ", line:
" + m.getLineNumber());
                  m.replace("{"//
                        + "$proceed($$); "//
                        + "}");
               } else if
(className.equals(TARGET_MY_APP) &&
methodName.equals(MOVE_METHOD)) {
                  System.out.println("[Edited by
ClassLoader] method name: " + methodName + ", line:
" + m.getLineNumber());
                  m.replace("{" //
                        + "$1 = 0; " //
                        + "$proceed($$); " //
                        + "}");
               }
            }
         });
         byte[] b = cc.toBytecode();
         return defineClass(name, b, 0, b.length);


static String workDir   =
System.getProperty("user.dir");
ClassPool pool = ClassPool.getDefault();
         pool.insertClassPath(inputDir);
         CtClass cc = pool.get("target.Point");
         CtMethod m = cc.getDeclaredMethod("move");
         m.insertBefore("{
System.out.println(\"[DBG] param1: \" + $1); " + //
               "System.out.println(\"[DBG] param2:
\" + $2); }");
```

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

        ClassPool defaultPool =
ClassPool.getDefault();
        defaultPool.insertClassPath(INPUT_PATH);
        CtClass cc = defaultPool.get(TARGET_MYAPP);
        CtMethod m =
cc.getDeclaredMethod(FACT_METHOD);
        m.useCflow(FACT_METHOD);
        m.insertBefore("if ($cflow(fact) == 0)" +
System.lineSeparator() + //
               "System.out.println(\"[MyAppFact
Inserted] fact \" + $1);");
        cc.writeFile(OUTPUT_PATH);
        InsertMethodBodyCflow s = new
InsertMethodBodyCflow();//pool = new
ClassPool();pool.insertClassPath(OUTPUT_PATH); //
TARGET must be there.
        Class<?> c = s.loadClass(TARGET_MYAPP);
        Method mainMethod =
c.getDeclaredMethod("main", new Class[] {
String[].class });
        mainMethod.invoke(null, new Object[] { args
});
        //findClass method:cc =
pool.get(name);byte[] b = cc.toBytecode();return
defineClass(name, b, 0, b.length);

        SubstituteMethodBody s = new
SubstituteMethodBody();// pool = new
ClassPool();pool.insertClassPath(new
ClassClassPath(new
java.lang.Object().getClass()));pool.insertClassPat
h(INPUT_PATH); // "target" must be there.
        Class<?> c = s.loadClass(TARGET_MY_APP);
        Method mainMethod =
c.getDeclaredMethod("main", new Class[] {
String[].class });
        mainMethod.invoke(null, new Object[] { args
});
        cc = pool.get(name);
        cc.instrument(new ExprEditor() {
            public void edit(MethodCall m) throws
CannotCompileException {
            }
         byte[] b = cc.toBytecode();
         return defineClass(name, b, 0, b.length);

            FieldAcess s = new FieldAcess();//pool
= new ClassPool();pool.insertClassPath(new
ClassClassPath(new
java.lang.Object().getClass()));pool.insertClassPat
h(INPUT_PATH); // TARGET must be there.
            Class<?> c =
s.loadClass(TARGET_MY_APP);
            Method mainMethod =
c.getDeclaredMethod("main", new Class[] {
String[].class });
            mainMethod.invoke(null, new Object[] {
args });

            NewExprAccess s = new NewExprAccess();
            Class<?> c =
s.loadClass(TARGET_MY_APP2);
            Method mainMethod =
c.getDeclaredMethod("main", new Class[] {
String[].class });
      mainMethod.invoke(null, new Object[] { args
});
            cc = pool.get(name);
         cc.instrument(new ExprEditor() {
                public void edit(NewExpr newExpr)
throws CannotCompileException {
                StringBuilder code = new
StringBuilder();
                code.append("\"y: \" + " +
"$_.y);\n }\n");
                // System.out.println(code);
                newExpr.replace(code.toString());
```

```
        String src = "public void xmove(int dx) { x
+= dx; }";
        CtMethod newMethod = CtNewMethod.make(src,
cc);
        cc.addMethod(newMethod);
        cc.writeFile(outputDir);

        CtMethod newMethod = CtNewMethod.make(src,
cc, "this", "move");
        cc.addMethod(newMethod);
        cc.writeFile(outputDir);

        ClassPool pool = ClassPool.getDefault();
        pool.insertClassPath(inputDir);
        CtMethod newMethod = new
CtMethod(CtClass.intType, "move", new CtClass[] {
CtClass.intType }, cc);
        cc.addMethod(newMethod);
        newMethod.setBody("{ x += $1; return x;}");
        cc.setModifiers(cc.getModifiers() &
~Modifier.ABSTRACT); cc.writeFile(outputDir);

ClassPool pool = ClassPool.getDefault();
         pool.insertClassPath(inputDir);
         CtClass cc = pool.get("target.Point");
         String src = "public void xmove(int dx) {
x += dx; }";
         CtMethod newMethod = CtNewMethod.make(src,
cc);
         cc.addMethod(newMethod);
         cc.writeFile(outputDir);

         ClassPool pool = ClassPool.getDefault();
         pool.insertClassPath(inputDir);
         CtClass cc = pool.get("target.Point");
         String src = "public void ymove(int dy) {
$proceed(0, dy); }";
         CtMethod newMethod = CtNewMethod.make(src,
cc, "this", "move");
         cc.addMethod(newMethod);
         cc.writeFile(outputDir);
         System.out.println("[DBG] write output to:
" + outputDir);

         ClassPool pool = ClassPool.getDefault();
         pool.insertClassPath(inputDir);
         CtClass cc = pool.get("target.Point");
         CtMethod newMethod = new
CtMethod(CtClass.intType, "move", new CtClass[] {
CtClass.intType }, cc);
         cc.addMethod(newMethod);
         newMethod.setBody("{ x += $1; return
x;}");
         cc.setModifiers(cc.getModifiers() &
~Modifier.ABSTRACT);
         cc.writeFile(outputDir);

         ClassPool pool = ClassPool.getDefault();
         pool.insertClassPath(inputDir);
         CtClass cc = pool.get("target.Point");
         CtMethod m = CtNewMethod.make("public
abstract int m(int i);", cc);
         CtMethod n = CtNewMethod.make("public
abstract int n(int i);", cc);
         cc.addMethod(m);
         cc.addMethod(n);
         m.setBody("{ return ($1 <= 0) ? 1 : (n($1
- 1) * $1); }");
         n.setBody("{ return m($1); }");
         cc.setModifiers(cc.getModifiers() &
~Modifier.ABSTRACT);

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