Living in the Matrix with Bytecode Manipulation

QCon NY 2014

Ashley Puls

Senior Software Engineer
New Relic, Inc.

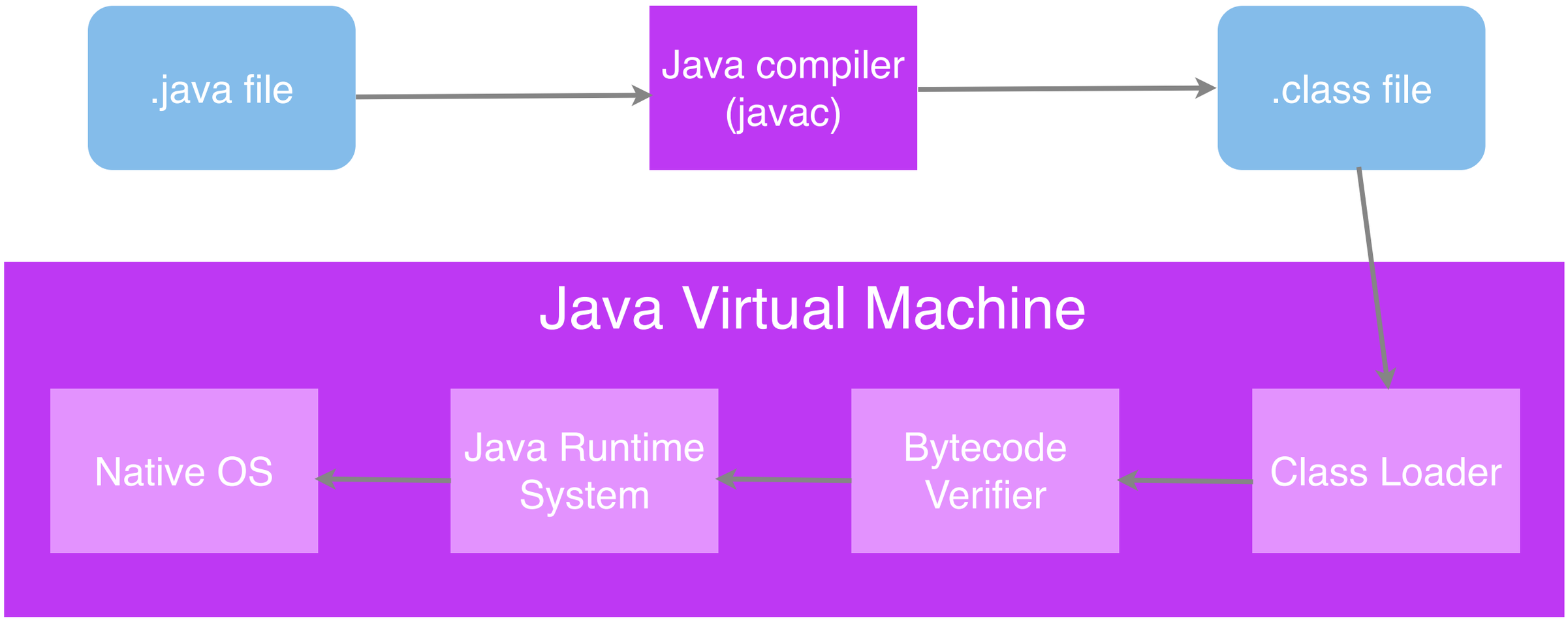
Follow Along

http://slidesha.re/1kZwCXr

Outline

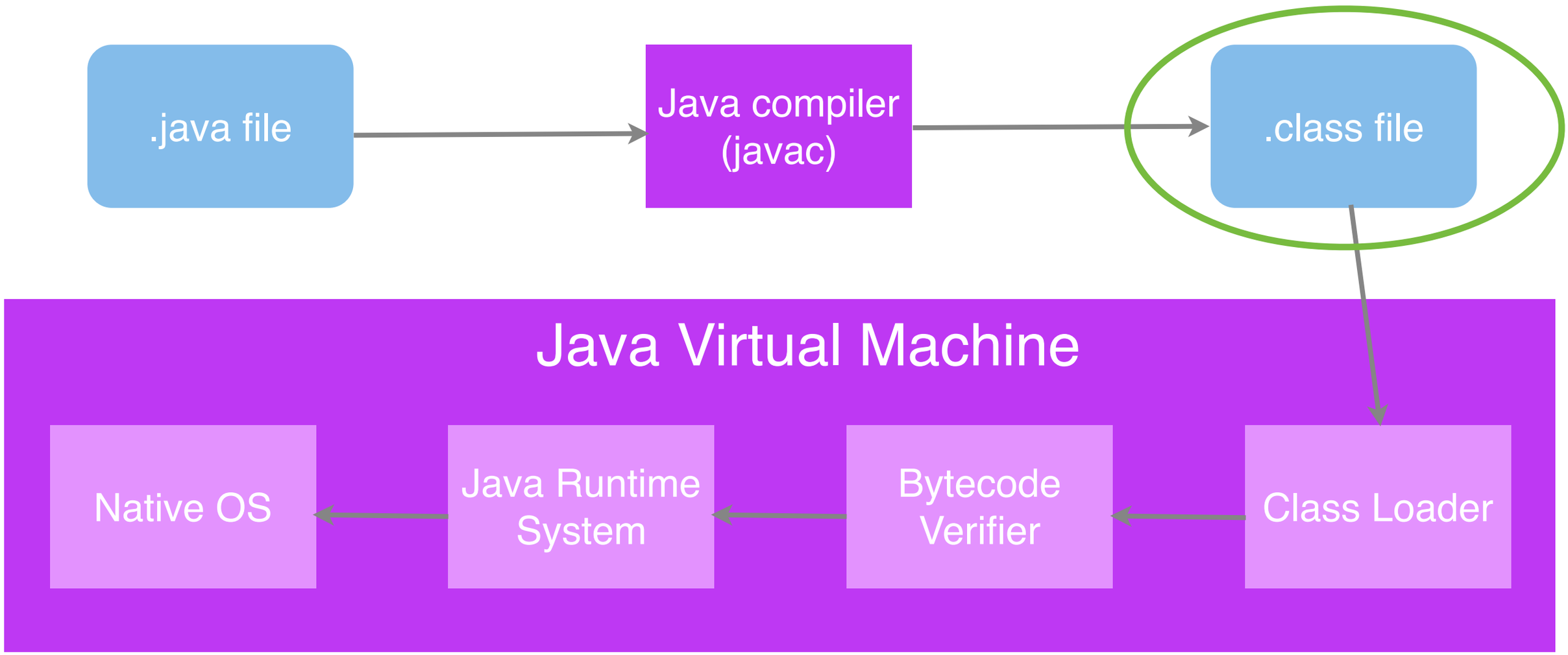
- What is bytecode?
- Why manipulate bytecode?
- What frameworks can be used to manipulate bytecode?
- Examine two frameworks in depth

What is Java bytecode?



Source: http://www.techlila.com/write-programs-linux/

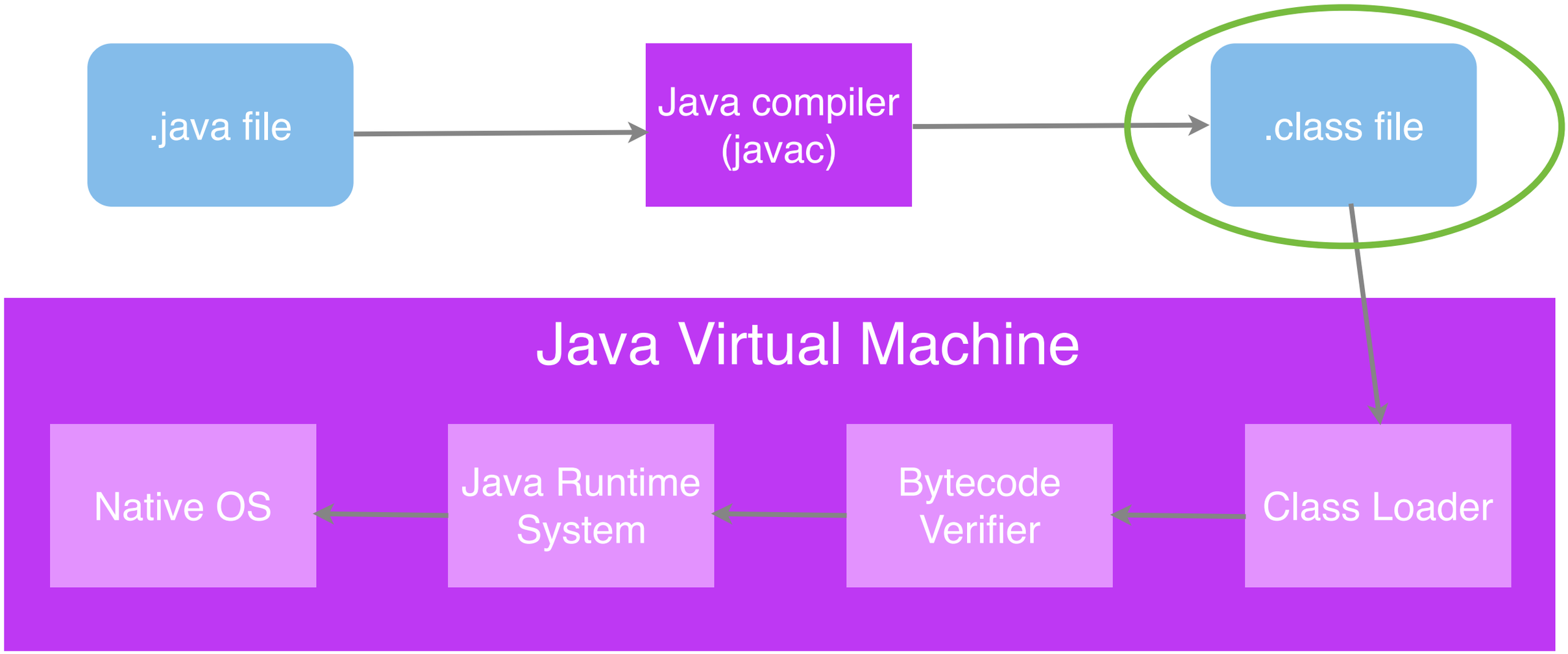
What is Java bytecode?



Source: http://www.techlila.com/write-programs-linux/

What is Java bytecode?

Instruction set of the Java Virtual Machine



Source: http://www.techlila.com/write-programs-linux/

Why learn about Java bytecode manipulation frameworks ?

Why learn about Java bytecode manipulation frameworks ?

- A bytecode manipulation framework is likely already on your stack
  - Spring
  - Hibernate
  - Groovy
  - Clojure
  - Eclipse
  - JRuby

Why learn about Java bytecode manipulation frameworks ?

- A bytecode manipulation framework is likely already on your stack
  - Spring
  - Hibernate
  - Groovy
  - Clojure
  - Eclipse
  - JRuby
- Its really fun!!!

Why learn about Java bytecode manipulation frameworks ?

- program analysis
  - find bugs in code
  - examine code complexity
- generate classes
  - proxies
  - remove access to certain APIs
  - compiler for another language like Scala
- transform classes without Java source code
  - profilers
  - optimization and obfuscation
  - additional logging

Why learn about Java bytecode manipulation frameworks ?

- program analysis
  - find bugs in code
  - examine code complexity
- generate classes
  - proxies
  - remove access to certain APIs
  - compiler for another language like Scala
- transform classes without Java source code
  - profilers
  - optimization and obfuscation
  - **additional logging**

Logging

Source: http://www.vforteachers.com/About_NetSupport.htm

Logging

Source: http://www.vforteachers.com/About_NetSupport.htm, http://inzolo.com/blog/tutorials/bank-accounts

Logging

Source: http://www.vforteachers.com/About_NetSupport.htm, http://upload.wikimedia.org/wikipedia/commons/d/d3/49024-SOS-ATM.JPG

Logging

Audit Log: a message should be logged every time an important method is called

Source: http://www.vforteachers.com/About_NetSupport.htm

Logging

Audit Log: a message should be logged every time an important method is called

log important input parameters to the method

Source: http://www.vforteachers.com/About_NetSupport.htm

Logging

```
public class BankTransactions {

   public static void main(String[] args) {
      BankTransactions bank = new BankTransactions();

      // login and add i dollars to account
      for (int i = 0; i < 100; i++) {
         String accountId = "account" + i;
         bank.login("password", accountId, "Ashley");
         bank.unimportantProcessing(accountId);
         bank.finalizeTransaction(accountId, Double.valueOf(i));
      }

      System.out.println(“Transactions completed”);
   }

}
```

Logging

```
public class BankTransactions {

   public static void main(String[] args) {
      BankTransactions bank = new BankTransactions();

      // login and add i dollars to each account
      for (int i = 0; i < 100; i++) {
         String accountId = "account" + i;
         bank.login("password", accountId, "Ashley");
         bank.unimportantProcessing(accountId);
         bank.finalizeTransaction(accountId, Double.valueOf(i));
      }

      System.out.println(“Transactions completed”);
   }

}
```

Logging

```
/**
 * A method annotation which should be used to indicate important methods whose
 * invocations should be logged.
 *
 */
public @interface ImportantLog {

    /**
     * The method parameter indexes whose values should be logged. For example,
     * if we have the method hello(int paramA, int paramB, int paramC), and we
     * wanted to log the values of paramA and paramC, then fields would be ["0",
     * "2"]. If we only want to log the value of paramB, then fields would be
     * ["1"].
     */
    String[] fields();
}
```

Logging

```
public void login(String password, String accountId, String userName) {
    // login logic
}
```

```
public void finalizeTransaction(String accountId, Double moneyToAdd) {
    // transaction logic
}
```

Logging

```
@ImportantLog(fields = { "1", "2" })
public void login(String password, String accountId, String userName) {
   // login logic
}
```

```
@ImportantLog(fields = { "0", "1" })
public void finalizeTransaction(String accountId, Double moneyToAdd) {
   // transaction logic
}
```

Logging

```
@ImportantLog(fields = { "1", "2" })
public void login(String password, String accountId, String userName) {
   // login logic
}
```

```
A call was made to method "login" on class "com/example/qcon/mains/BankTransactions".
    Important params:
        Index 1 value: ${accountId}
        Index 2 value: ${userName}
```

```
@ImportantLog(fields = { "0", "1" })
public void finalizeTransaction(String accountId, Double moneyToAdd) {
   // transaction logic
}
```

Logging

```
@ImportantLog(fields = { "1", "2" })
public void login(String password, String accountId, String userName) {
   // login logic
}
```

```
A call was made to method "login" on class "com/example/qcon/mains/BankTransactions".
    Important params:
        Index 1 value: ${accountId}
        Index 2 value: ${userName}
```

```
@ImportantLog(fields = { "0", "1" })
public void finalizeTransaction(String accountId, Double moneyToAdd) {
   // transaction logic
}
```

```
A call was made to method "finalizeTransaction" on class "com/example/qcon/
mains/BankTransactions".
    Important params:
        Index 0 value: ${accountId}
        Index 1 value: ${moneyToAdd}
```

Java Agent

- Ability to modify the bytecode without modifying the application source code
- Added in Java 1.5
- Docs: http://docs.oracle.com/javase/8/docs/api/java/lang/instrument/package-summary.html

Typical Java Process

```
java com/example/qcon/mains/BankTransactions
```

JVM

Classloader

```
BankTransactions.class


public static void main(String[] args)
```

```
Transactions completed
```

Java Agent

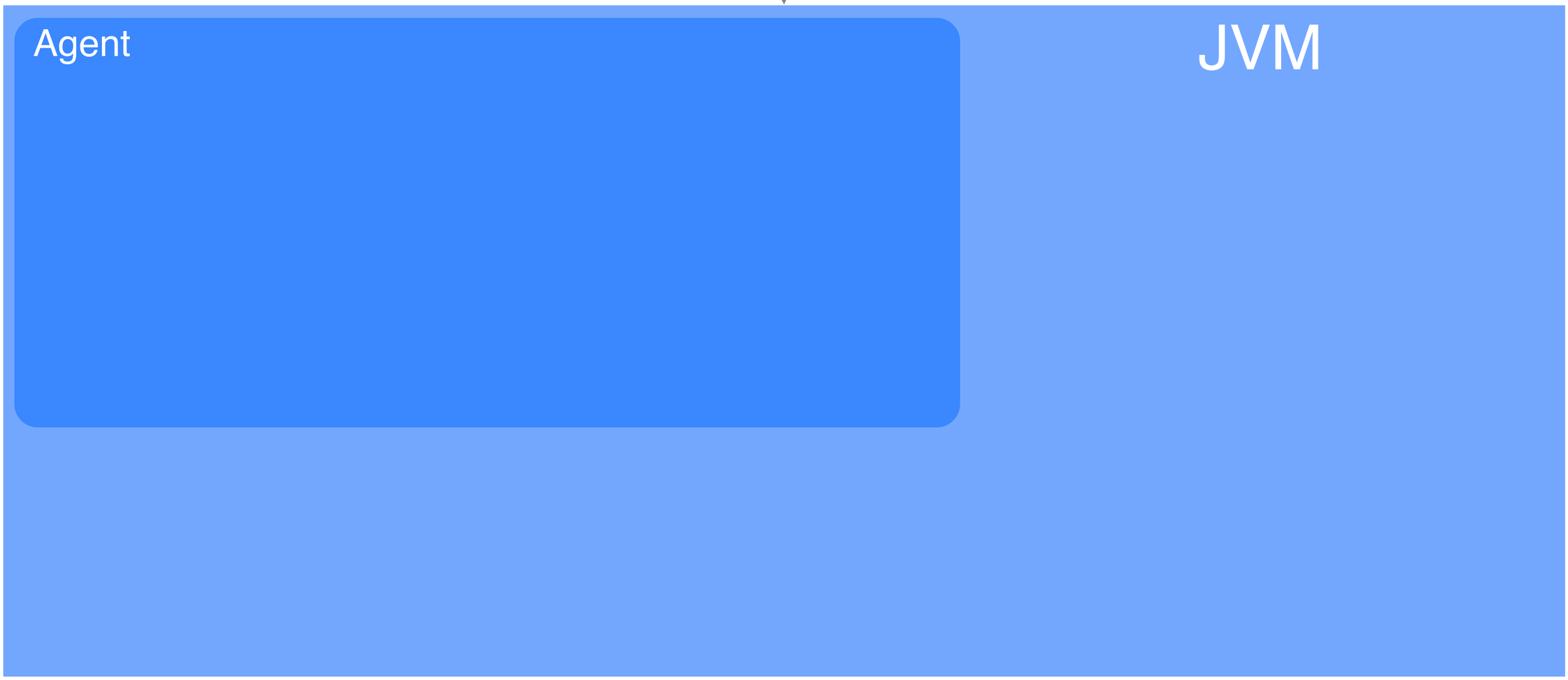
```
java -javaagent:/path/to/agent.jar com/example/qcon/mains/
BankTransactions
```

Java Agent

```
java -javaagent:/path/to/agent.jar com/example/qcon/mains/
BankTransactions
```

Java Agent

```
java -javaagent:/path/to/agent.jar com/example/qcon/mains/
BankTransactions
```



Java Agent

```
java -javaagent:/path/to/agent.jar com/example/qcon/mains/
BankTransactions
```

↓

Agent

```
Agent.class

  void premain(String agentArgs, Instrumentation inst)
```

```
MyTransformer.class

  byte[] transform( . . . , byte[] bankTransBytes)
```

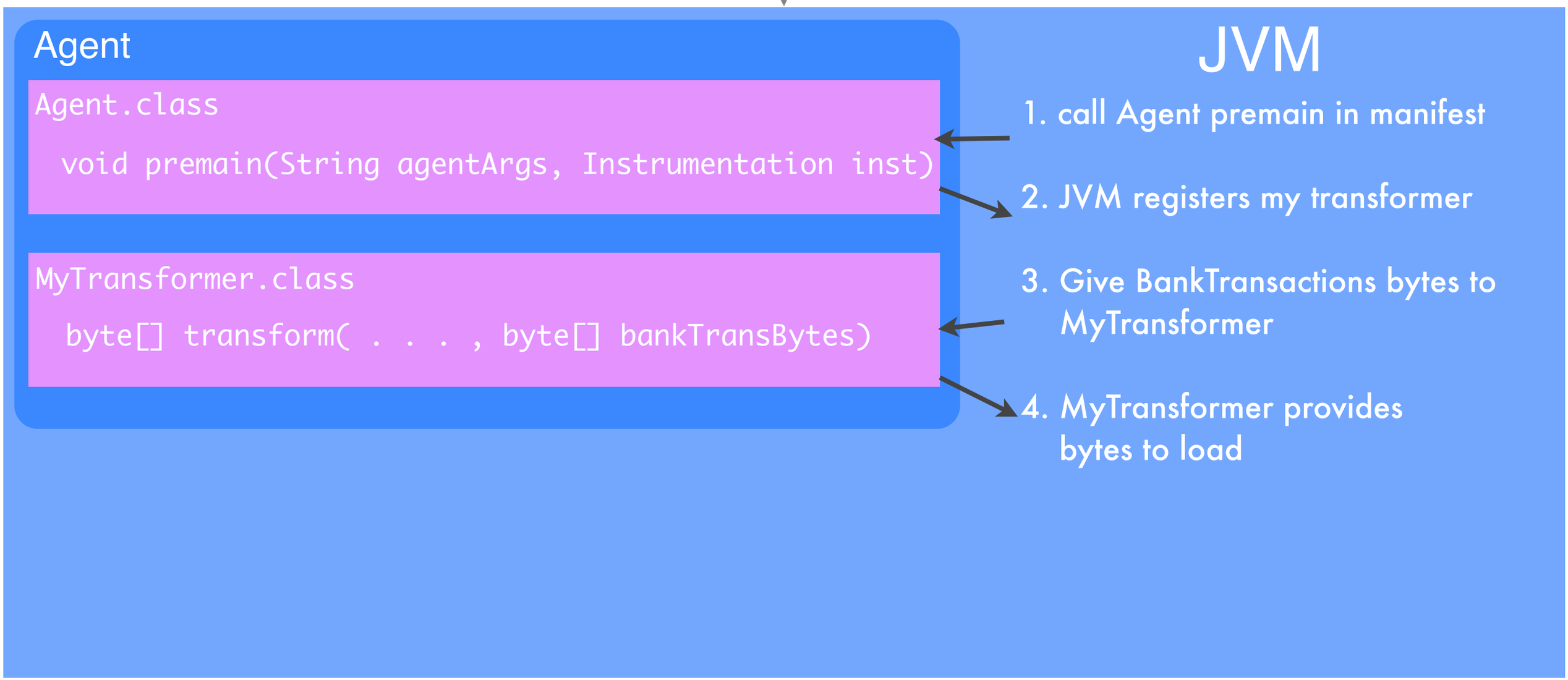
JVM

1. call Agent premain in manifest

2. JVM registers my transformer

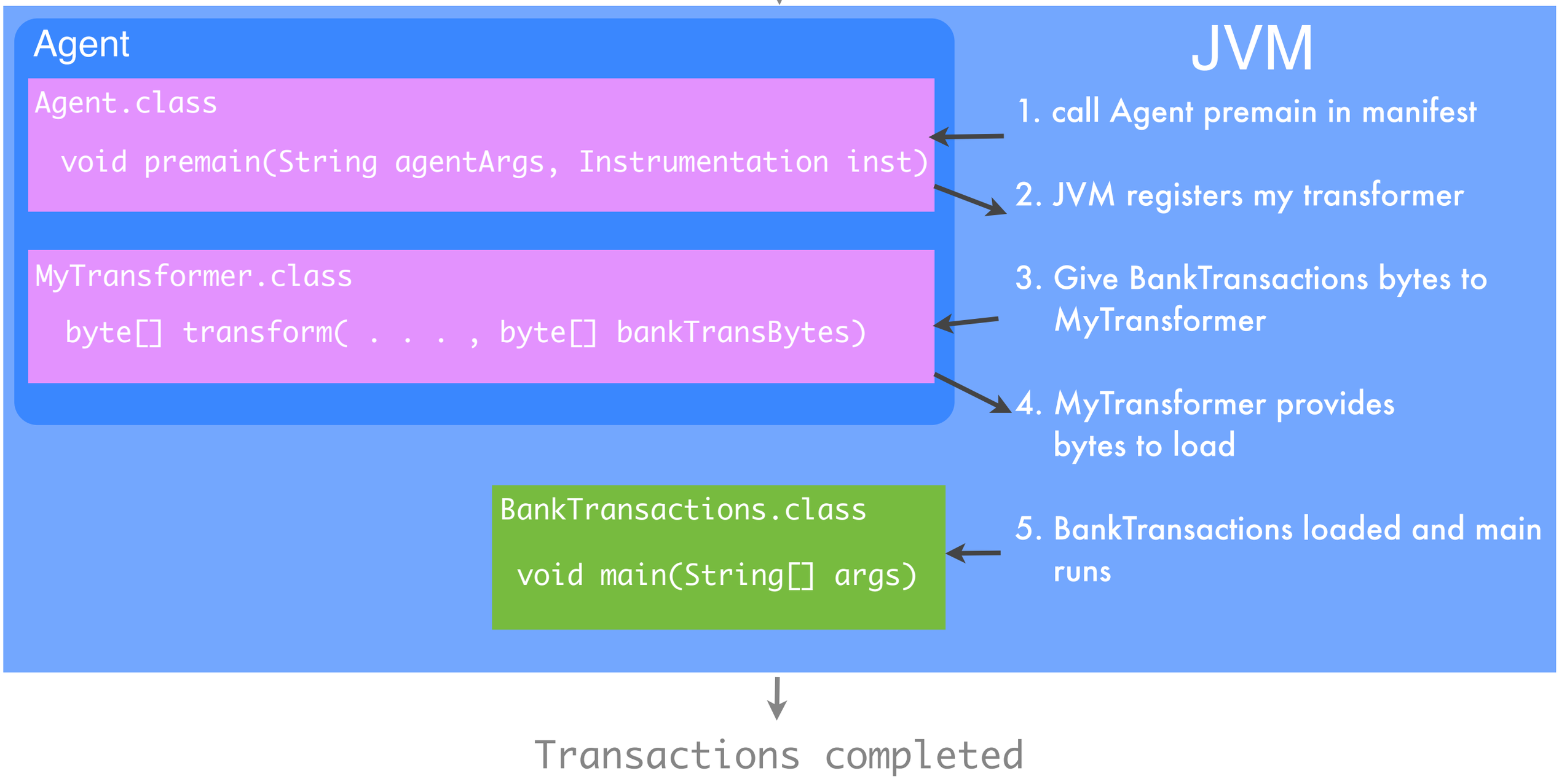
Java Agent

```
java -javaagent:/path/to/agent.jar com/example/qcon/mains/
BankTransactions
```



Java Agent

```
java -javaagent:/path/to/agent.jar com/example/qcon/mains/
BankTransactions
```




```
Transactions completed
```

lets code

```
Manifest:
      Premain-Class: com.example.qcon.agent.Agent
```

lets code

```
Manifest:
      Premain-Class: com.example.qcon.agent.Agent
```

```
package com.example.qcon.agent;

public class Agent {

   public static void premain(String agentArgs, Instrumentation inst) {
      System.out.println("Starting the agent");
      inst.addTransformer(new ImportantLogClassTransformer());
   }

}
```

lets code

```
Manifest:
      Premain-Class: com.example.qcon.agent.Agent
```

```
package com.example.qcon.agent;

public class Agent {

   public static void premain(String agentArgs, Instrumentation inst) {
      System.out.println("Starting the agent");
      inst.addTransformer(new ImportantLogClassTransformer());
   }

}
```

lets code

```
package com.example.qcon.agent;

import java.lang.instrument.ClassFileTransformer;
import java.lang.instrument.IllegalClassFormatException;

public class ImportantLogClassTransformer implements
ClassFileTransformer {

   public byte[] transform(ClassLoader loader, String className,
        Class classBeingRedefined, ProtectionDomain protectionDomain,
        byte[] classfileBuffer) throws IllegalClassFormatException {

      //TODO
      return null;
   }
}
```

lets code

```
package com.example.qcon.agent;

import java.lang.instrument.ClassFileTransformer;
import java.lang.instrument.IllegalClassFormatException;

public class ImportantLogClassTransformer implements
ClassFileTransformer {

   public byte[] transform(ClassLoader loader, String className,
        Class classBeingRedefined, ProtectionDomain protectionDomain,
        byte[] classfileBuffer) throws IllegalClassFormatException {

      //TODO
      return null;
   }
}
```

lets code

```
package com.example.qcon.agent;

import java.lang.instrument.ClassFileTransformer;
import java.lang.instrument.IllegalClassFormatException;

public class ImportantLogClassTransformer implements
ClassFileTransformer {

   public byte[] transform(ClassLoader loader, String className,
        Class classBeingRedefined, ProtectionDomain protectionDomain,
        byte[] classfileBuffer) throws IllegalClassFormatException {

      System.out.println(" Loading class: " + className);
      return null;
   }
}
```

lets code

```
Starting the agent
 Loading class: java/lang/invoke/MethodHandleImpl
 Loading class: java/lang/invoke/MemberName$Factory
 Loading class: java/lang/invoke/LambdaForm$NamedFunction
 Loading class: java/lang/invoke/MethodType$ConcurrentWeakInternSet
 Loading class: java/lang/invoke/MethodHandleStatics
 Loading class: java/lang/invoke/MethodHandleStatics$1
 Loading class: java/lang/invoke/MethodTypeForm
 Loading class: java/lang/invoke/Invokers
 Loading class: java/lang/invoke/MethodType$ConcurrentWeakInternSet$WeakEntry
 Loading class: java/lang/Void
 Loading class: java/lang/IllegalAccessException
 Loading class: sun/misc/PostVMInitHook
 Loading class: sun/launcher/LauncherHelper
 Loading class: java/util/concurrent/ConcurrentHashMap$ForwardingNode
 Loading class: sun/misc/URLClassPath$FileLoader$1
 Loading class: com/example/qcon/mains/BankTransactions
 Loading class: sun/launcher/LauncherHelper$FXHelper
```

lets code

```
package com.example.qcon.agent;

import java.lang.instrument.ClassFileTransformer;
import java.lang.instrument.IllegalClassFormatException;

public class ImportantLogClassTransformer implements
ClassFileTransformer {

   public byte[] transform(ClassLoader loader, String className,
        Class classBeingRedefined, ProtectionDomain protectionDomain,
        byte[] classfileBuffer) throws IllegalClassFormatException {

      System.out.println(" Loading class: " + className);
      return null;
   }
}
```

lets code

```
package com.example.qcon.agent;

import java.lang.instrument.ClassFileTransformer;
import java.lang.instrument.IllegalClassFormatException;

public class ImportantLogClassTransformer implements
ClassFileTransformer {

   public byte[] transform(ClassLoader loader, String className,
        Class classBeingRedefined, ProtectionDomain protectionDomain,
        byte[] classfileBuffer) throws IllegalClassFormatException {

      // manipulate the bytes here and then return them
      return null;
   }
}
```

Bytecode Manipulation Frameworks

- ASM: http://asm.ow2.org/
- BCEL: http://commons.apache.org/proper/commons-bcel/
- CGLib: https://github.com/cglib/cglib
- Javassist: http://www.csg.ci.i.u-tokyo.ac.jp/~chiba/javassist/
- Serp: http://serp.sourceforge.net/
- Cojen: https://github.com/cojen/Cojen/wiki
- Soot: http://www.sable.mcgill.ca/soot/

Bytecode Manipulation Frameworks

- **ASM: http://asm.ow2.org/**
- BCEL: http://commons.apache.org/proper/commons-bcel/
- CGLib: https://github.com/cglib/cglib
- **Javassist: http://www.csg.ci.i.u-tokyo.ac.jp/~chiba/javassist/**
- Serp: http://serp.sourceforge.net/
- Cojen: https://github.com/cojen/Cojen/wiki
- Soot: http://www.sable.mcgill.ca/soot/

Javassist

- Java Programming Assistant
- Subproject of JBoss
- Actively updated
  - Version 3.18.2 released May 2014
- Source code: https://github.com/jboss-javassist/javassist
- Documentation: http://www.csg.ci.i.u-tokyo.ac.jp/~chiba/javassist/tutorial/tutorial.html

Javassist

CtClass: represents a compile time class

ArrayList

HashMap

Javassist

CtClass: represents a compile time class

ClassPool: container of CtClass objects

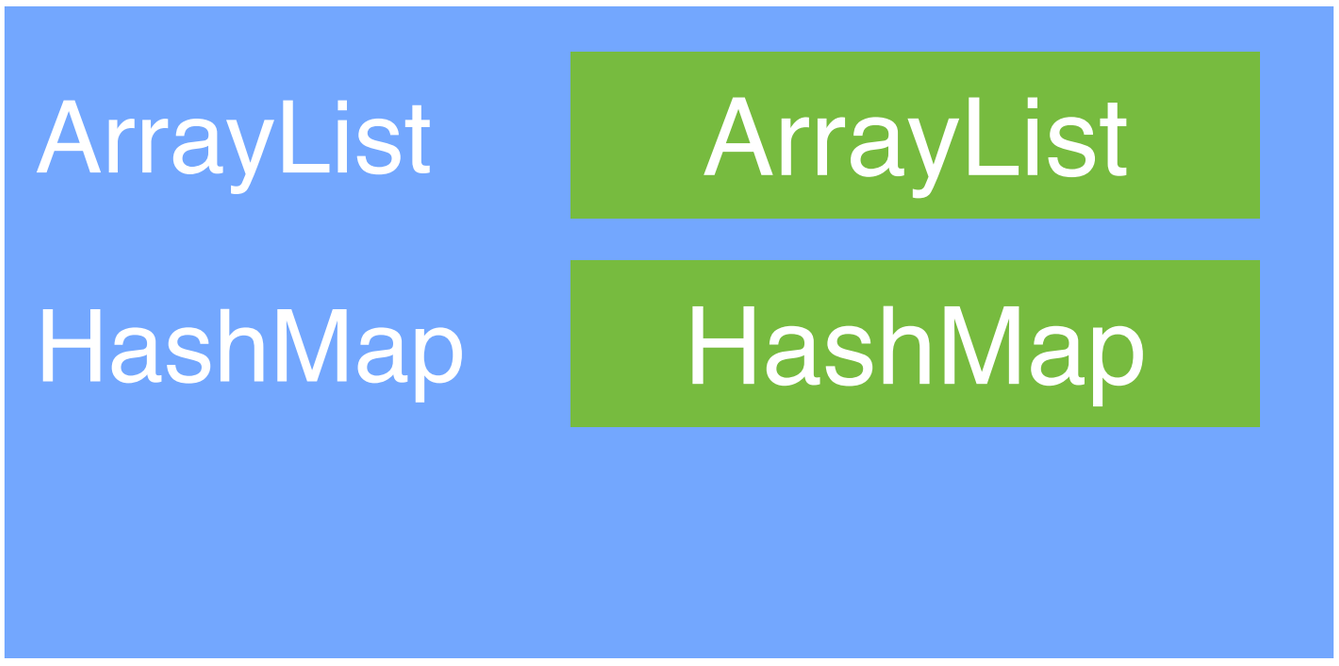
ArrayList

HashMap

Javassist

CtClass: represents a compile time class

ClassPool: container of CtClass objects



Javassist

CtClass: represents a compile time class

ClassPool: container of CtClass objects

ArrayList

HashMap

ArrayList

HashMap

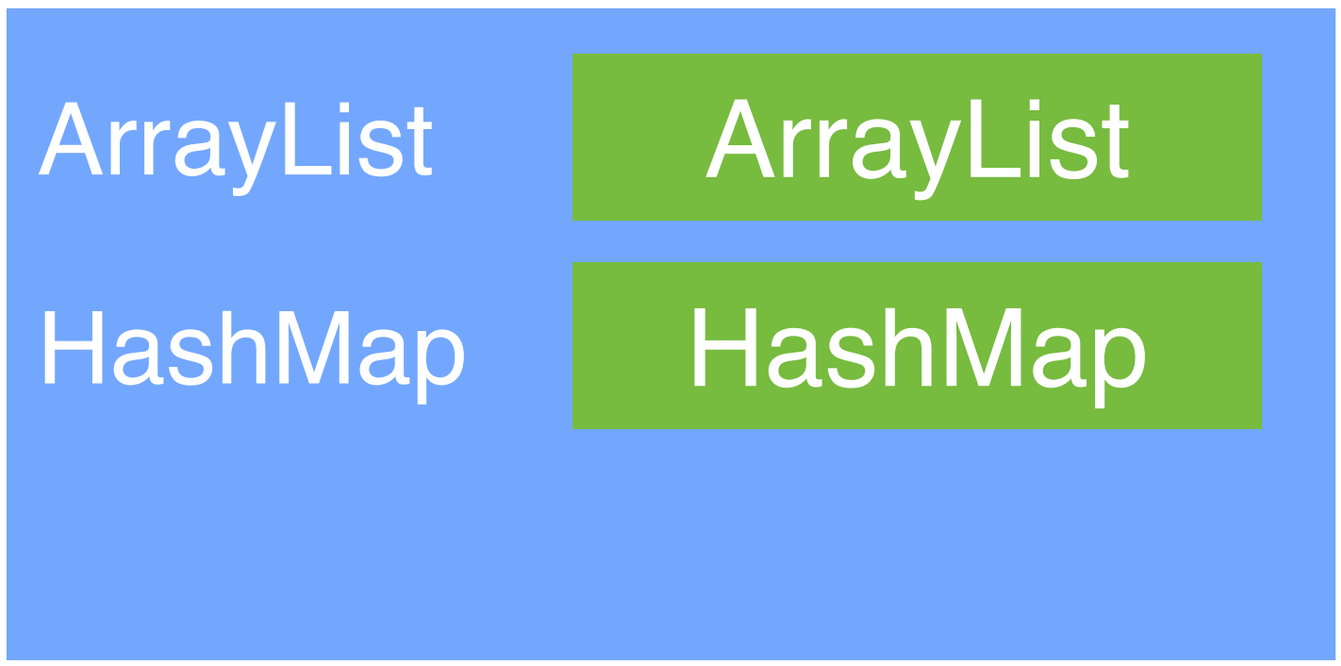
BankTrans

Javassist

CtClass: represents a compile time class

ClassPool: container of CtClass objects

ByteArrayClassPath: converts a byte[] to a CtClass



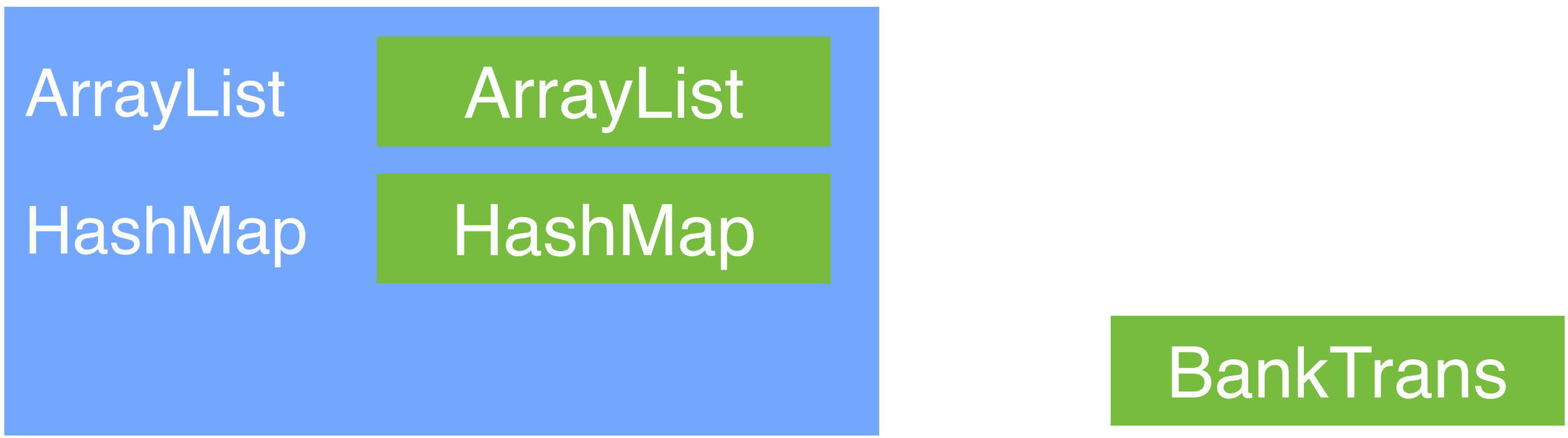
Javassist

CtClass: represents a compile time class

ClassPool: container of CtClass objects

ByteArrayClassPath: converts a byte[] to a CtClass

```
Classpool.getDefault().insertClassPath(new ByteArrayClassPath("BankTrans", classBytes);
```



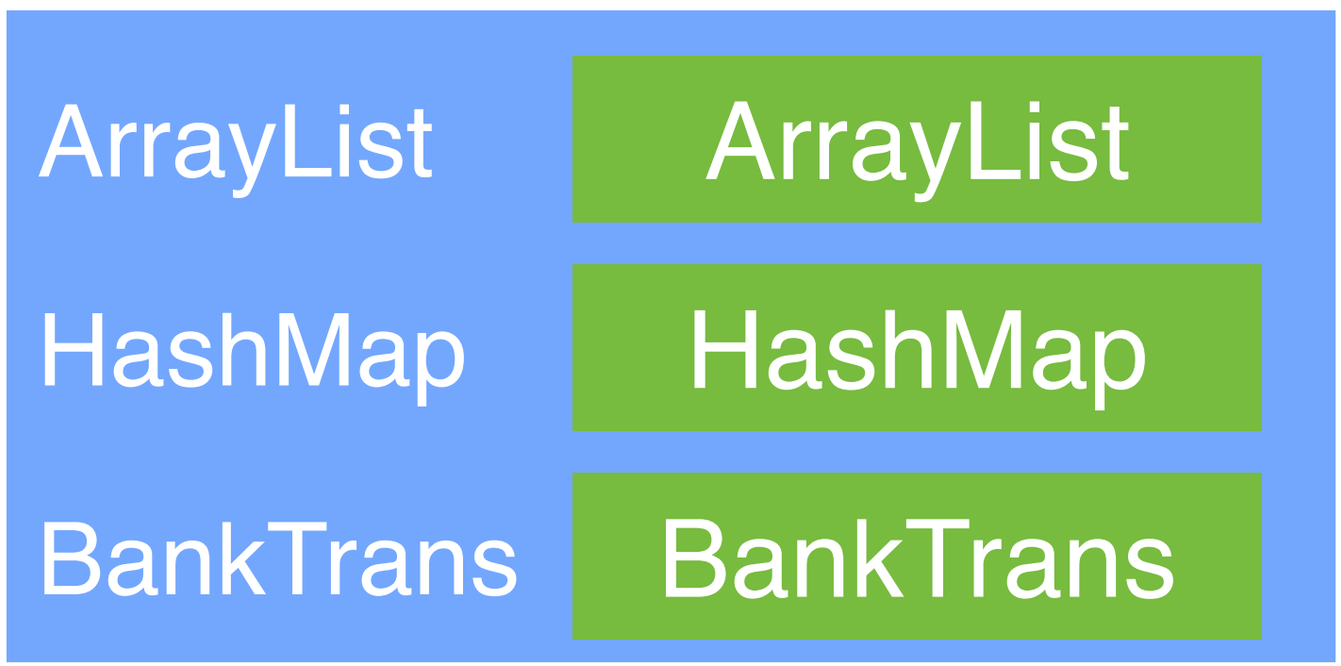
Javassist

CtClass: represents a compile time class

ClassPool: container of CtClass objects

ByteArrayClassPath: converts a byte[] to a CtClass

```
Classpool.getDefault().insertClassPath(new ByteArrayClassPath("BankTrans", classBytes);
```



Javassist

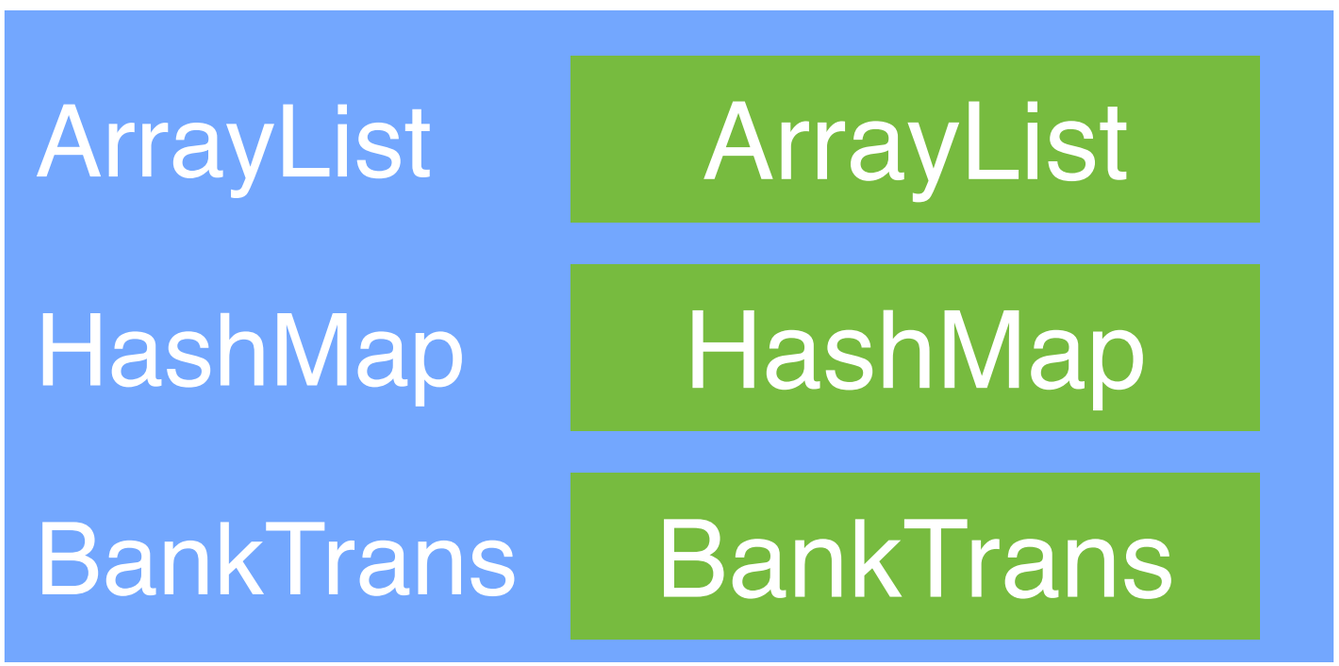
CtClass: represents a compile time class

ClassPool: container of CtClass objects

ByteArrayClassPath: converts a byte[] to a CtClass

```
Classpool.getDefault().insertClassPath(new ByteArrayClassPath(“BankTrans”, classBytes);

CtClass bank = Classpool.getDefault().get(“BankTrans”);
```



Javassist

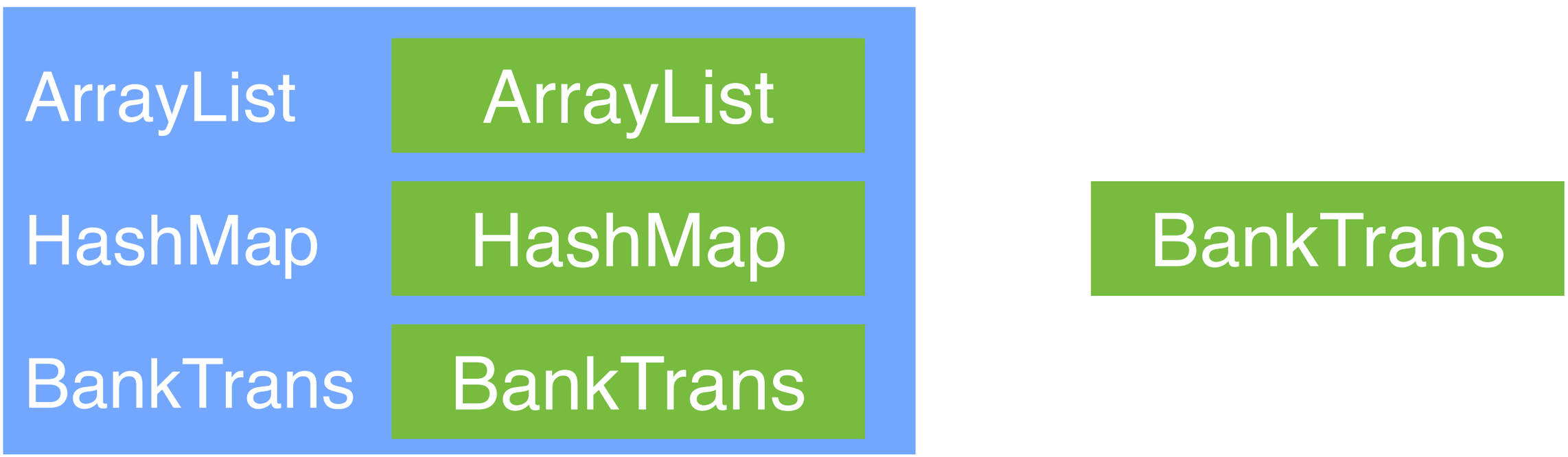
CtClass: represents a compile time class

ClassPool: container of CtClass objects

ByteArrayClassPath: converts a byte[] to a CtClass

```
Classpool.getDefault().insertClassPath(new ByteArrayClassPath(“BankTrans”, classBytes);

CtClass bank = Classpool.getDefault().get(“BankTrans”);
```



Javassist

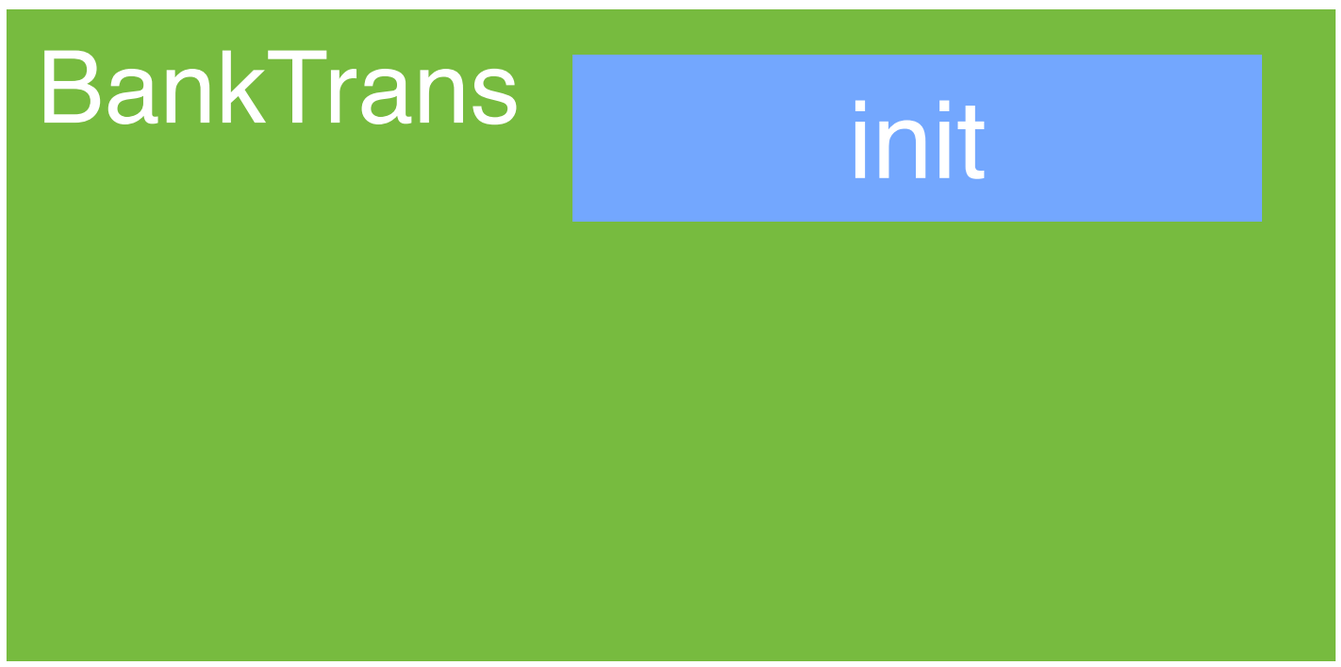
CtClass: represents a compile time class

BankTrans

Javassist

CtClass: represents a compile time class

CtConstructor: represents a compile time constructor

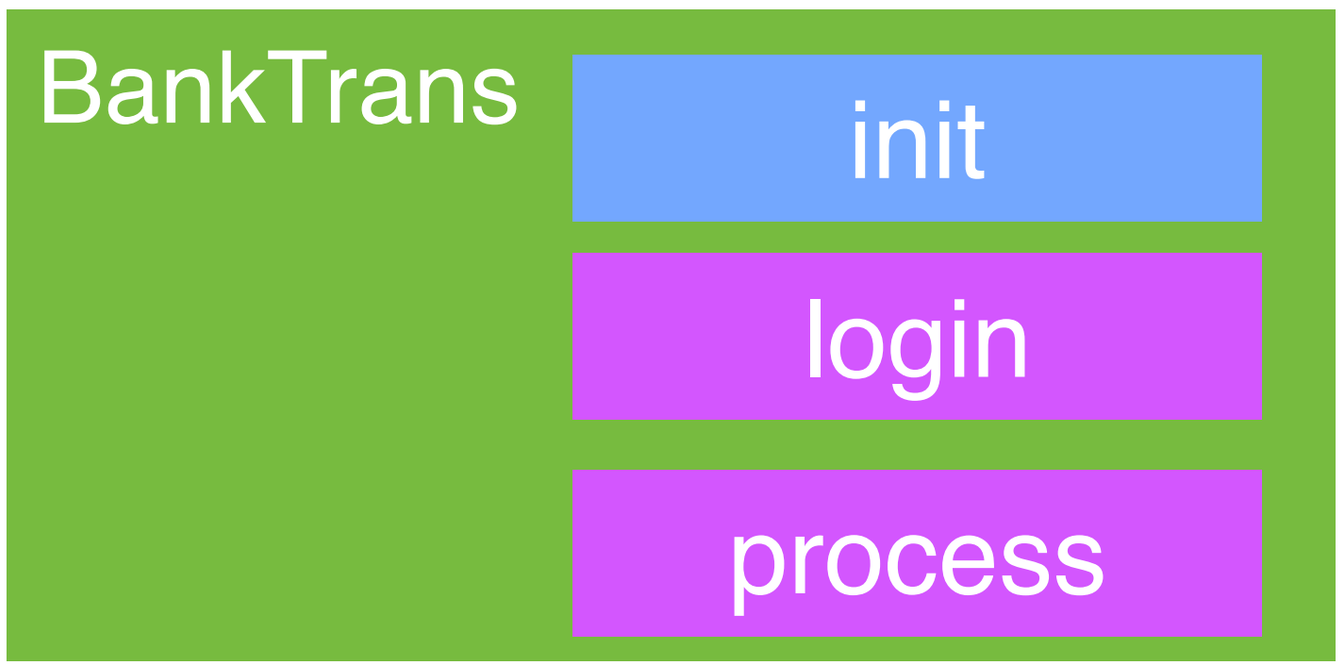


Javassist

CtClass: represents a compile time class

CtConstructor: represents a compile time constructor

CtMethod: represents a compile time class



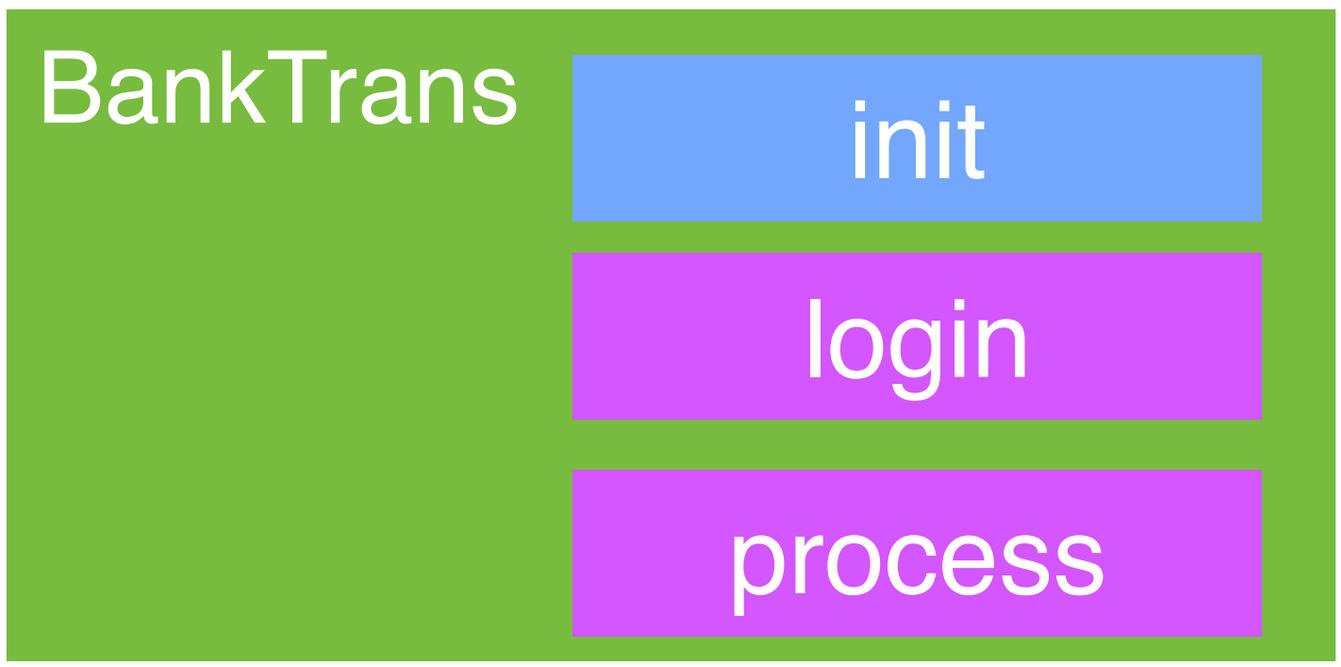
Javassist

CtClass: represents a compile time class

CtConstructor: represents a compile time constructor

CtMethod: represents a compile time method

```
CtMethod loginMethod = loginCtClass.getDeclaredMethod(“login”);
```



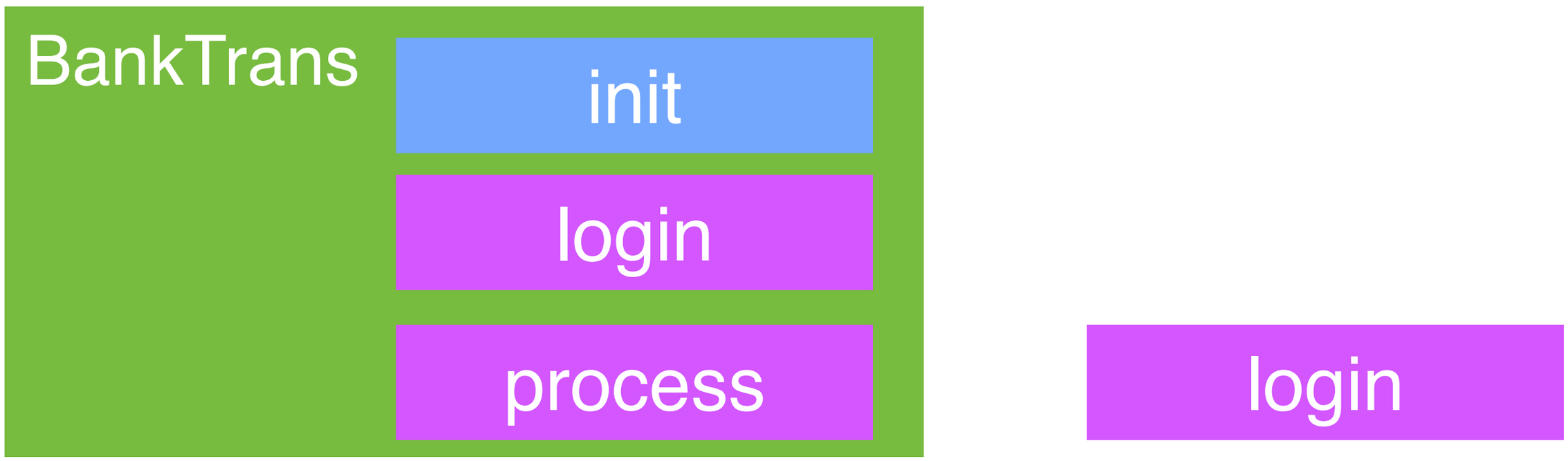
Javassist

CtClass: represents a compile time class

CtConstructor: represents a compile time constructor

CtMethod: represents a compile time method

```
CtMethod loginMethod = loginCtClass.getDeclaredMethod("login");
```



Javassist

CtMethod: represents a compile time method

login

Javassist

CtMethod: represents a compile time method

Modifications:

login

Javassist

CtMethod: represents a compile time method

Modifications:

putMethod.insertBefore(“System.out.println(\“before method\”);”);

putMethod.insertAfter(“System.out.println(\”after method\”);”);

login

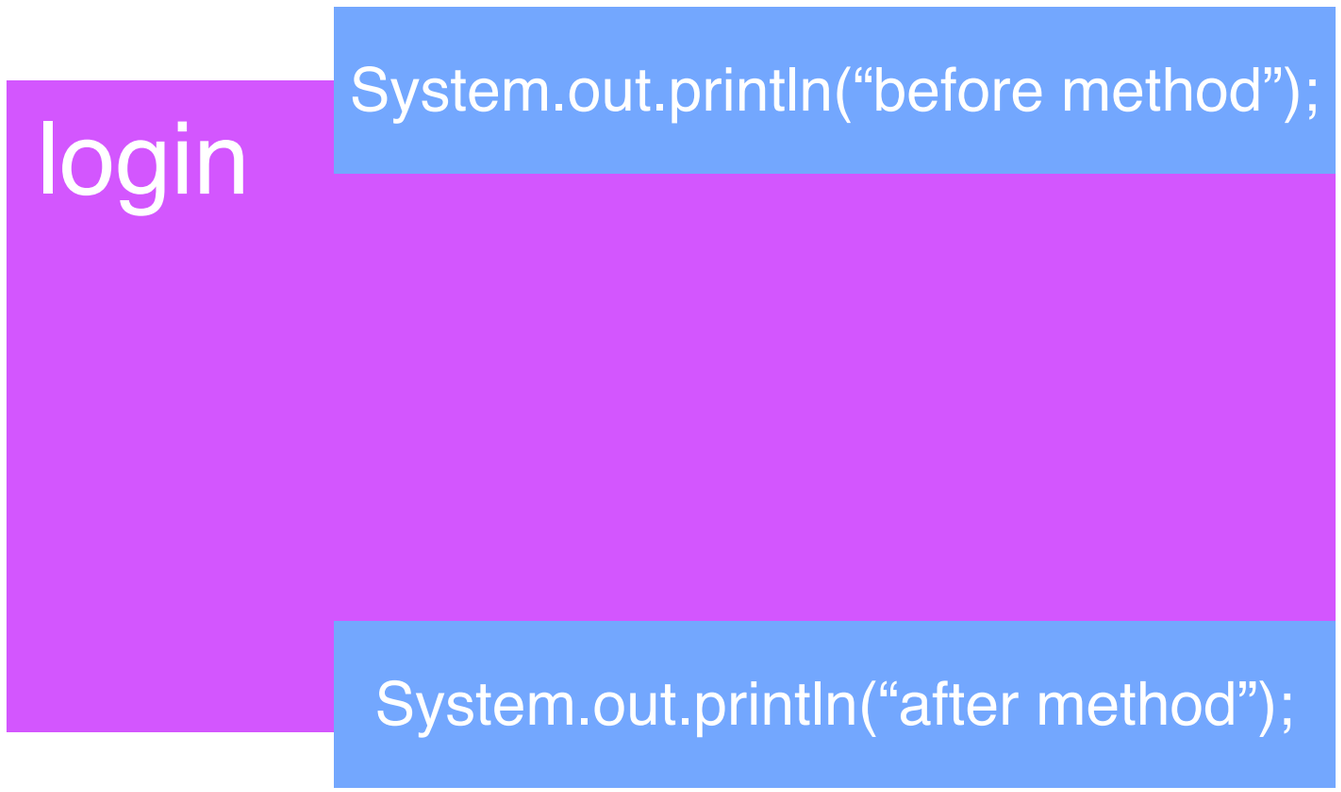
Javassist

CtMethod: represents a compile time method

Modifications:

putMethod.insertBefore(“System.out.println(\“before method\”);”);

putMethod.insertAfter(“System.out.println(\”after method\”);”);



Javassist

CtMethod: represents a compile time method

Modifications:

```
CtClass exceptionType = classpool.get(“java.io.IOException”);
putMethod.addCatch(“{System.out.println($e); throw $e}”, exceptionType);
```

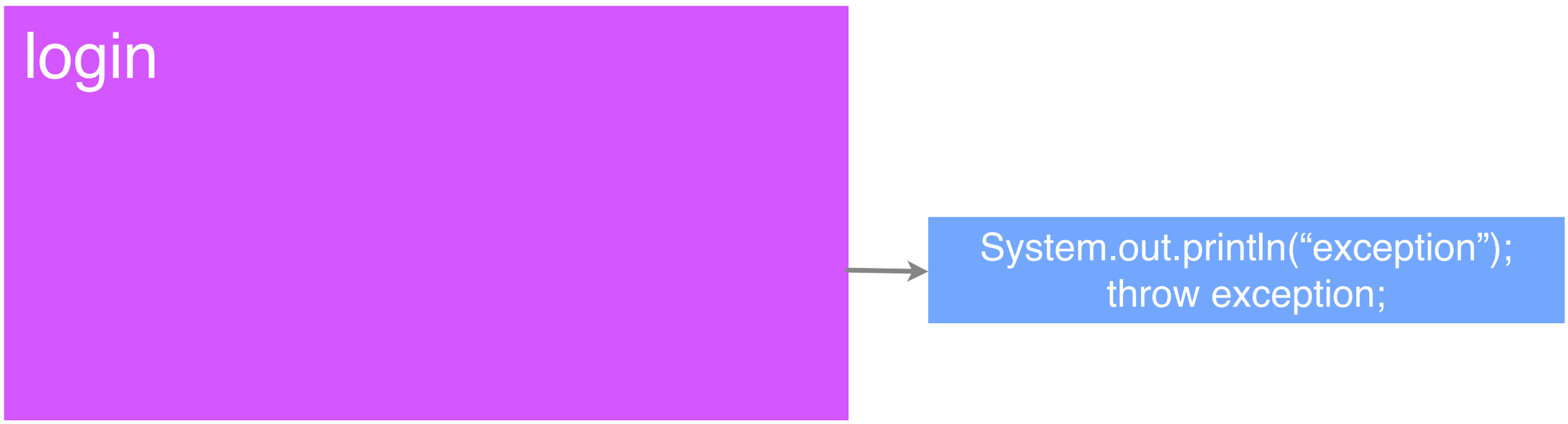
login

Javassist

CtMethod: represents a compile time method

Modifications:

```
CtClass exceptionType = classpool.get(“java.io.IOException”);
putMethod.addCatch(“{System.out.println($e); throw $e}”, exceptionType);
```



Javassist

CtMethod: represents a compile time method

Modifications:

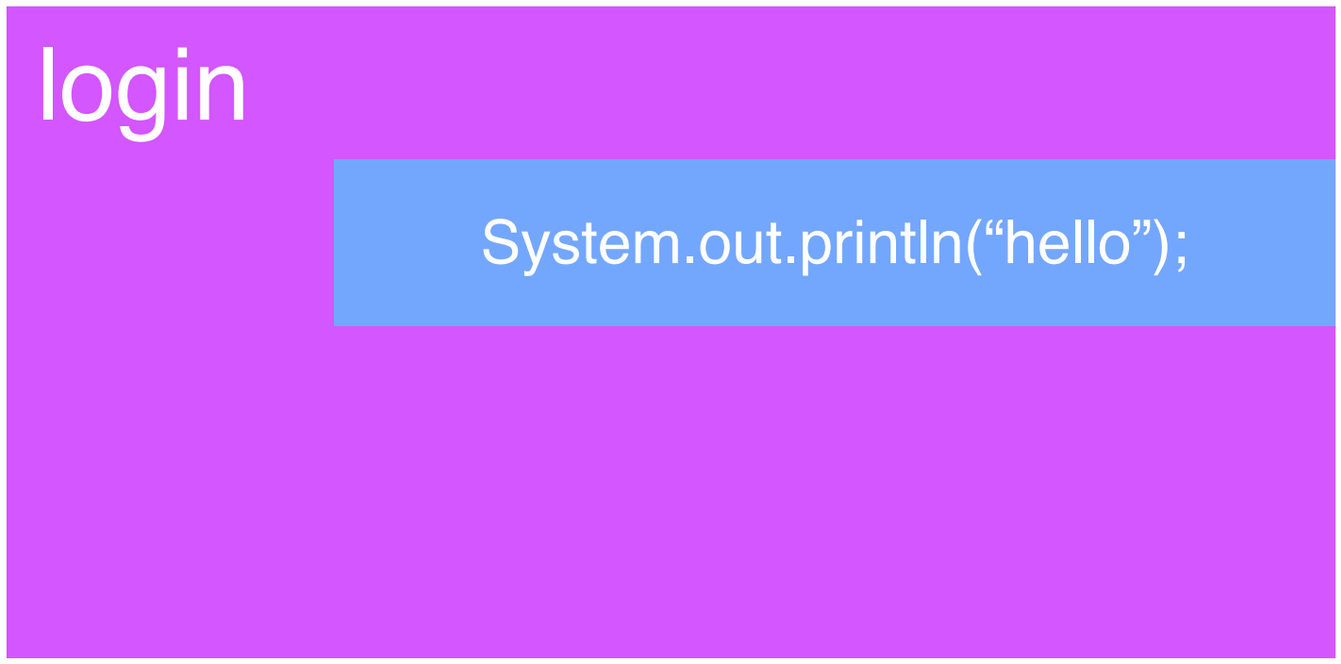
putMethod.insertAt(5, true, “System.out.println(\“hello\”);”);

Javassist

CtMethod: represents a compile time method

Modifications:

putMethod.insertAt(5, true, “System.out.println(\“hello\”);”);

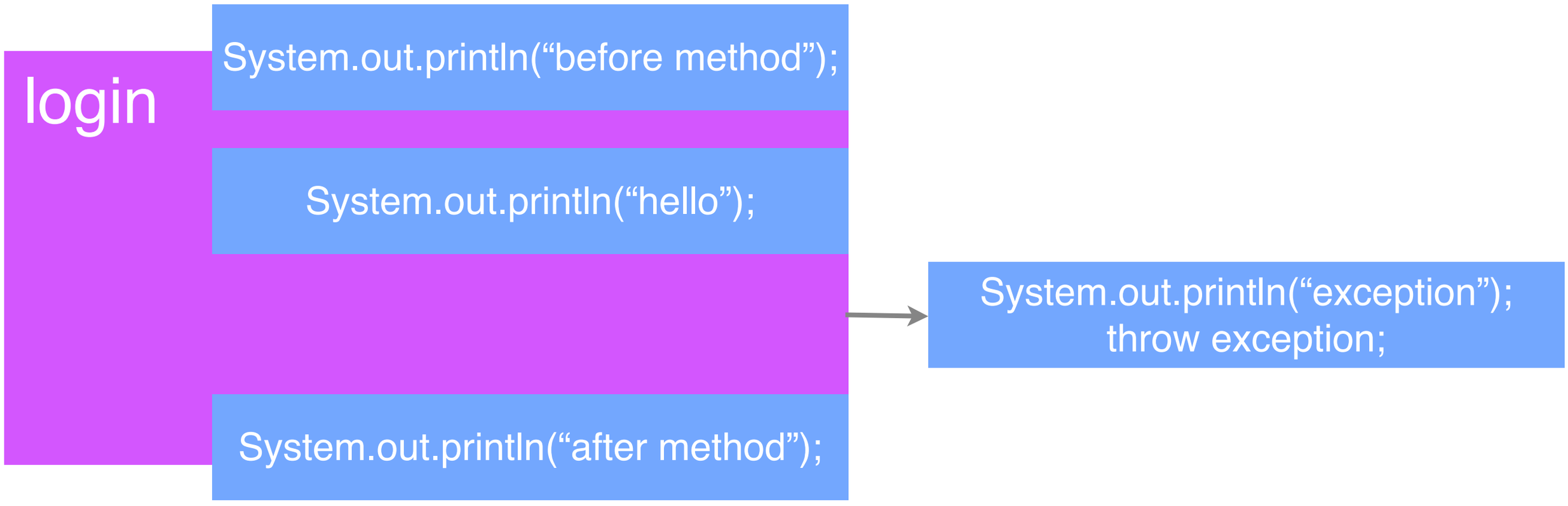


Javassist

CtMethod: represents a compile time method

Modifications:

insertBefore　　insertAfter　　insertAt　　addCatch



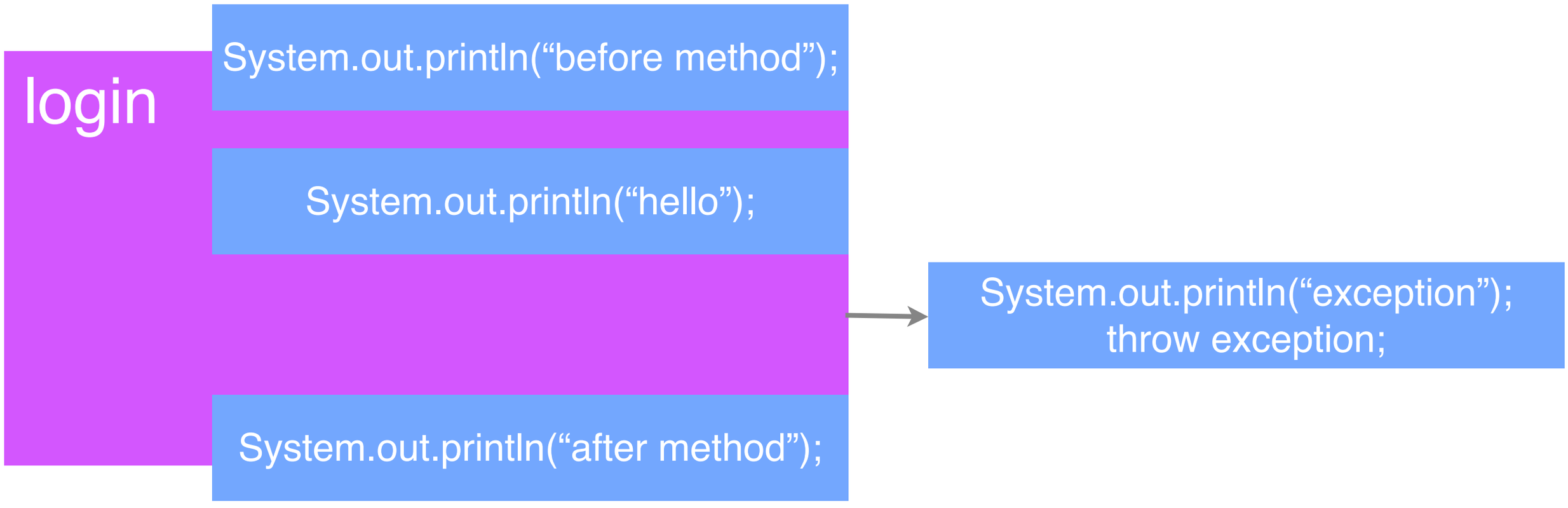
Javassist

CtMethod: represents a compile time method

Modifications:

insertBefore insertAfter insertAt addCatch

Freeze Class:



Javassist

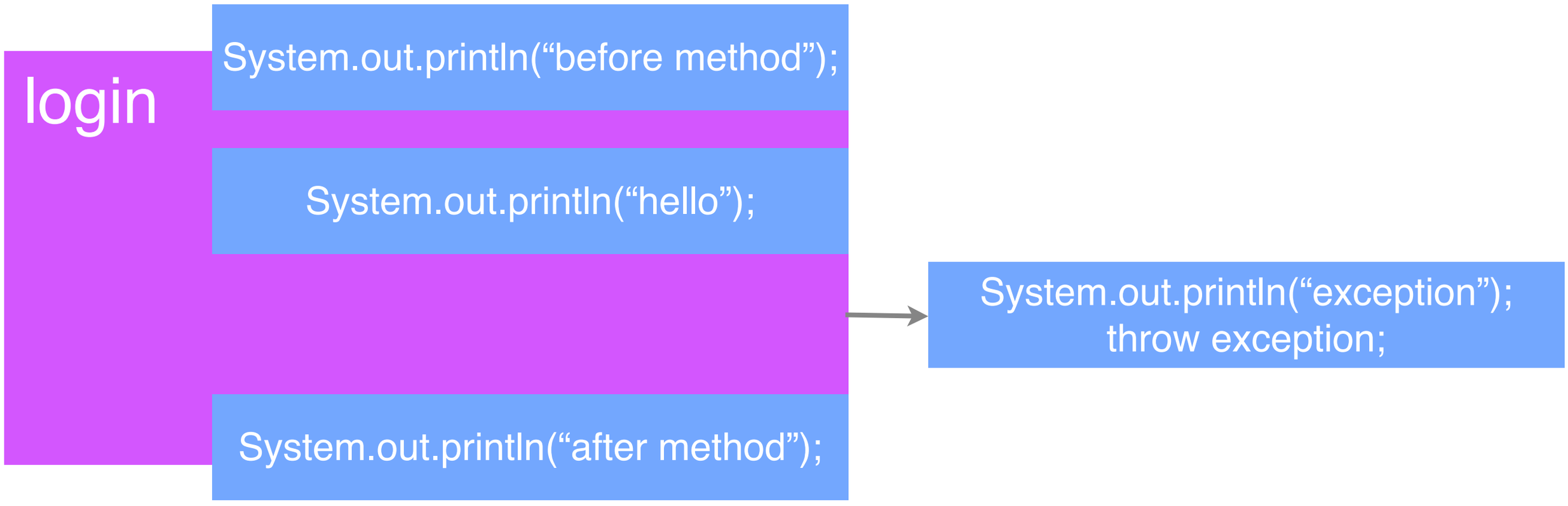
CtMethod: represents a compile time method

Modifications:

insertBefore insertAfter insertAt addCatch

Freeze Class:

writeFile() toClass() toBytecode()



lets code

lets code

```
package com.example.qcon.agent;

import java.lang.instrument.ClassFileTransformer;
import java.lang.instrument.IllegalClassFormatException;

public class ImportantLogClassTransformer implements ClassFileTransformer {

   public byte[] transform(ClassLoader loader, String className,
         Class classBeingRedefined, ProtectionDomain protectionDomain,
         byte[] classfileBuffer) throws IllegalClassFormatException {

     //TODO

      return null;
   }
}
```

lets code

```
package com.example.qcon.agent;

import java.lang.instrument.ClassFileTransformer;
import java.lang.instrument.IllegalClassFormatException;

public class ImportantLogClassTransformer implements ClassFileTransformer {

  public byte[] transform(ClassLoader loader, String className,
        Class classBeingRedefined, ProtectionDomain protectionDomain,
        byte[] classfileBuffer) throws IllegalClassFormatException {

    //TODO

     return null;
  }
}
```

lets code

```
public byte[] transform(ClassLoader loader, String className,
      Class classBeingRedefined, ProtectionDomain protectionDomain,
      byte[] cfbuffer) throws IllegalClassFormatException {


       1. convert byte array to a ct class object

       2. check each method of ct class for annotation @ImportantLog

       3. if @ImportantLog annotation present on method, then

           a. get important method parameter indexes

           b. add logging statement to beginning of the method


    return null;
}
```

lets code

```
public byte[] transform(ClassLoader loader, String className,
      Class classBeingRedefined, ProtectionDomain protectionDomain,
      byte[] cfbuffer) throws IllegalClassFormatException {

      pool.insertClassPath(new ByteArrayClassPath(className,
            classfileBuffer));
      CtClass cclass = pool.get(className.replaceAll("/", "."));

      2. check each method of ct class for annotation @ImportantLog

      3. if @ImportantLog annotation present on method, then

          a. get important method parameter indexes

          b. add logging statement to beginning of the method


   return null;
}
```

lets code

```
public byte[] transform(ClassLoader loader, String className,
      Class classBeingRedefined, ProtectionDomain protectionDomain,
      byte[] cfbuffer) throws IllegalClassFormatException {

      pool.insertClassPath(new ByteArrayClassPath(className,
            classfileBuffer));
      CtClass cclass = pool.get(className.replaceAll("/", "."));
      if (!cclass.isFrozen()) {
         for (CtMethod currentMethod : cclass.getDeclaredMethods()) {
             Annotation annotation = getAnnotation(currentMethod);

       3. if @ImportantLog annotation present on method, then

          a. get important method parameter indexes

          b. add logging statement to beginning of the method
         }
      }
    return null;
}
```

lets code

```
public byte[] transform(ClassLoader loader, String className,
      Class classBeingRedefined, ProtectionDomain protectionDomain,
      byte[] cfbuffer) throws IllegalClassFormatException {

      pool.insertClassPath(new ByteArrayClassPath(className,
            classfileBuffer));
      CtClass cclass = pool.get(className.replaceAll("/", "."));
      if (!cclass.isFrozen()) {
          for (CtMethod currentMethod : cclass.getDeclaredMethods()) {
              Annotation annotation = getAnnotation(currentMethod);
            if (annotation != null) {
               List<String> parameterIndexes = getParamIndexes(annotation);

            b. add logging statement to beginning of the method

            }
          }
      }
    return null;
}
```

lets code

```
public byte[] transform(ClassLoader loader, String className,
      Class classBeingRedefined, ProtectionDomain protectionDomain,
      byte[] cfbuffer) throws IllegalClassFormatException {

      pool.insertClassPath(new ByteArrayClassPath(className,
            classfileBuffer));
      CtClass cclass = pool.get(className.replaceAll("/", "."));
      if (!cclass.isFrozen()) {
          for (CtMethod currentMethod : cclass.getDeclaredMethods()) {
              Annotation annotation = getAnnotation(currentMethod);
            if (annotation != null) {
               List<String> parameterIndexes = getParamIndexes(annotation);
               currentMethod.insertBefore(createJavaString(
                     currentMethod, className, parameterIndexes));
            }
          }
          return cclass.toBytecode();
      }
    return null;
}
```

lets code

```
public byte[] transform(ClassLoader loader, String className,
      Class classBeingRedefined, ProtectionDomain protectionDomain,
      byte[] cfbuffer) throws IllegalClassFormatException {

      pool.insertClassPath(new ByteArrayClassPath(className,
            classfileBuffer));
      CtClass cclass = pool.get(className.replaceAll("/", "."));
      if (!cclass.isFrozen()) {
          for (CtMethod currentMethod : cclass.getDeclaredMethods()) {
              Annotation annotation = getAnnotation(currentMethod);
            if (annotation != null) {
               List<String> parameterIndexes = getParamIndexes(annotation);
               currentMethod.insertBefore(createJavaString(
                     currentMethod, className, parameterIndexes));
            }
          }
          return cclass.toBytecode();
      }
    return null;
}
```

lets code

```
private Annotation getAnnotation(CtMethod method) {
MethodInfo mInfo = method.getMethodInfo();
// the attribute we are looking for is a runtime invisible attribute
// use Retention(RetentionPolicy.RUNTIME) on the annotation to make it
// visible at runtime
AnnotationsAttribute attInfo = (AnnotationsAttribute) mInfo
      .getAttribute(AnnotationsAttribute.invisibleTag);
   if (attInfo != null) {
      // this is the type name meaning use dots instead of slashes
      return attInfo.getAnnotation("com.example.qcon.mains.ImportantLog");
   }
   return null;
}
```

lets code

```
public byte[] transform(ClassLoader loader, String className,
      Class classBeingRedefined, ProtectionDomain protectionDomain,
      byte[] cfbuffer) throws IllegalClassFormatException {

      pool.insertClassPath(new ByteArrayClassPath(className,
            classfileBuffer));
      CtClass cclass = pool.get(className.replaceAll("/", "."));
      if (!cclass.isFrozen()) {
          for (CtMethod currentMethod : cclass.getDeclaredMethods()) {
              Annotation annotation = getAnnotation(currentMethod);
            if (annotation != null) {
              List<String> parameterIndexes = getParamIndexes(annotation);
              currentMethod.insertBefore(createJavaString(
                    currentMethod, className, parameterIndexes));
            }
          }
          return cclass.toBytecode();
      }
    return null;
}
```

lets code

```
private List<String> getParamIndexes(Annotation annotation) {
   ArrayMemberValue fields = (ArrayMemberValue) annotation
         .getMemberValue("fields");
   if (fields != null) {
      MemberValue[] values = (MemberValue[]) fields.getValue();
      List<String> parameterIndexes = new ArrayList<String>();
      for (MemberValue val : values) {
         parameterIndexes.add(((StringMemberValue) val).getValue());
      }
      return parameterIndexes;
   }
   return Collections.emptyList();
}
```

lets code

```
public byte[] transform(ClassLoader loader, String className,
      Class classBeingRedefined, ProtectionDomain protectionDomain,
      byte[] cfbuffer) throws IllegalClassFormatException {

      pool.insertClassPath(new ByteArrayClassPath(className,
            classfileBuffer));
      CtClass cclass = pool.get(className.replaceAll("/", "."));
      if (!cclass.isFrozen()) {
          for (CtMethod currentMethod : cclass.getDeclaredMethods()) {
              Annotation annotation = getAnnotation(currentMethod);
            if (annotation != null) {
               List<String> parameterIndexes = getParamIndexes(annotation);
               currentMethod.insertBefore(createJavaString(
                     currentMethod, className, parameterIndexes));
            }
          }
          return cclass.toBytecode();
      }
    return null;
}
```

lets code

```
private String createJavaString(CtMethod currentMethod, String className,
      List<String> indexParameters) {
   StringBuilder sb = new StringBuilder();
   sb.append("{StringBuilder sb = new StringBuilder");
   sb.append("(\"A call was made to method '\");");
   sb.append("sb.append(\"");
   sb.append(currentMethod.getName());
   sb.append("\");sb.append(\"' on class '\");");
   sb.append("sb.append(\"");
   sb.append(className);
   sb.append("\");sb.append(\"'.\");");
   sb.append("sb.append(\"\\n    Important params:\");");

    . . .
```

lets code

```
private String createJavaString(CtMethod currentMethod, String className,
      List<String> indexParameters) {
   StringBuilder sb = new StringBuilder();
   sb.append("{StringBuilder sb = new StringBuilder");
   sb.append("(\"A call was made to method '\");");
   sb.append("sb.append(\"");
   sb.append(currentMethod.getName());
   sb.append("\");sb.append(\"' on class '\");");
   sb.append("sb.append(\"");
   sb.append(className);
   sb.append("\");sb.append(\"'.\");");
   sb.append("sb.append(\"\\n    Important params:\");");

    . . .
```

Put multiple statements inside brackets {}

lets code

```
private String createJavaString(CtMethod currentMethod, String className,
      List<String> indexParameters) {
    . . .
   for (String index : indexParameters) {
      try {
         int localVar = Integer.parseInt(index) + 1;
         sb.append("sb.append(\"\\n        Index: \");");
         sb.append("sb.append(\"");
         sb.append(index);
         sb.append("\");sb.append(\" value: \");");
         sb.append("sb.append($" + localVar + ");");
      } catch (NumberFormatException e) {
         e.printStackTrace();
      }
   }
   sb.append("System.out.println(sb.toString());}");
   return sb.toString();
}
```

lets code

```
private String createJavaString(CtMethod currentMethod, String className,
      List<String> indexParameters) {
    . . .
   for (String index : indexParameters) {
      try {
         int localVar = Integer.parseInt(index) + 1;
         sb.append("sb.append(\"\\n        Index: \");");
         sb.append("sb.append(\"");
         sb.append(index);
         sb.append("\");sb.append(\" value: \");");
         sb.append("sb.append($" + localVar + ");");
      } catch (NumberFormatException e) {
         e.printStackTrace();
      }
   }
   sb.append("System.out.println(sb.toString());}");
   return sb.toString();
}
```

$0, $1, $2, ... can be used to access "this" and the actual method parameters

lets code

```
public byte[] transform(ClassLoader loader, String className,
      Class classBeingRedefined, ProtectionDomain protectionDomain,
      byte[] cfbuffer) throws IllegalClassFormatException {

      pool.insertClassPath(new ByteArrayClassPath(className,
            classfileBuffer));
      CtClass cclass = pool.get(className.replaceAll("/", "."));
      if (!cclass.isFrozen()) {
          for (CtMethod currentMethod : cclass.getDeclaredMethods()) {
              Annotation annotation = getAnnotation(currentMethod);
            if (annotation != null) {
               List<String> parameterIndexes = getParamIndexes(annotation);
               currentMethod.insertBefore(createJavaString(
                     currentMethod, className, parameterIndexes));
            }
          }
          return cclass.toBytecode();
      }
    return null;
}
```

Javassist

- Positives
  - Simplicity
    - Do not have to write actual bytecode
    - Decent documentation
- Negatives
  - Generally slower than ASM
  - Less functionality

ASM

- Released in Open Source in 2002
- Actively updated
  - Version 5.0.3 released May 2014
- Two ASM libraries
  - Event based (SAX like)
    - Visitor design pattern
  - Object based (DOM like)
- Documentation: http://download.forge.objectweb.org/asm/asm4-guide.pdf

ASM

Modifiers, name, super class, interfaces	
Constant pool: numeric, string and type constants	
Source file name (optional)	
Enclosing class reference	
Annotation*	
Attribute*	
Inner class*	Name
Field*	Modifiers, name, type
	Annotation*
	Attribute*
Method*	Modifiers, name, return and parameter types
	Annotation*
	Attribute*
	Compiled code

Figure 2.1.: Overall structure of a compiled class (* means zero or more)

Source: http://download.forge.objectweb.org/asm/asm4-guide.pdf

ASM

```
public abstract class ClassVisitor {
  public ClassVisitor(int api);
  public ClassVisitor(int api, ClassVisitor cv);
  public void visit(int version, int access, String name,
      String signature, String superName, String[] interfaces);
  public void visitSource(String source, String debug);
  public void visitOuterClass(String owner, String name, String desc);
  AnnotationVisitor visitAnnotation(String desc, boolean visible);
  public void visitAttribute(Attribute attr);
  public void visitInnerClass(String name, String outerName,
      String innerName, int access);
  public FieldVisitor visitField(int access, String name, String desc,
      String signature, Object value);
  public MethodVisitor visitMethod(int access, String name, String desc,
      String signature, String[] exceptions);
  void visitEnd();
}
```

Figure 2.4.: The `ClassVisitor` class

Source: http://download.forge.objectweb.org/asm/asm4-guide.pdf

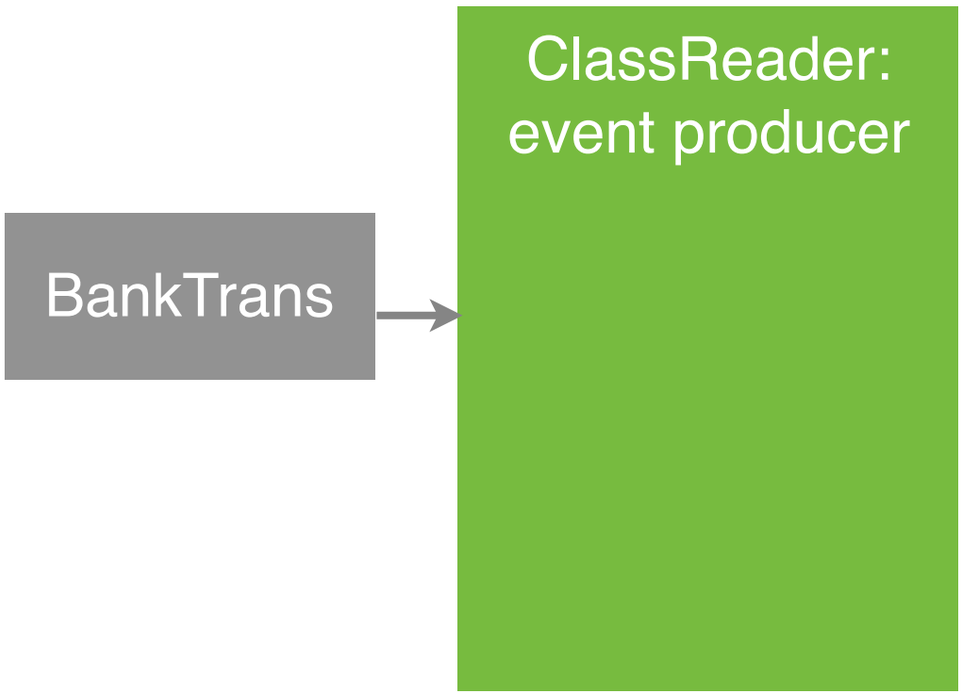
ASM

```
visit visitSource? visitOuterClass? ( visitAnnotation | visitAttribute )*
( visitInnerClass | visitField | visitMethod )*
visitEnd
```

Source: http://download.forge.objectweb.org/asm/asm4-guide.pdf

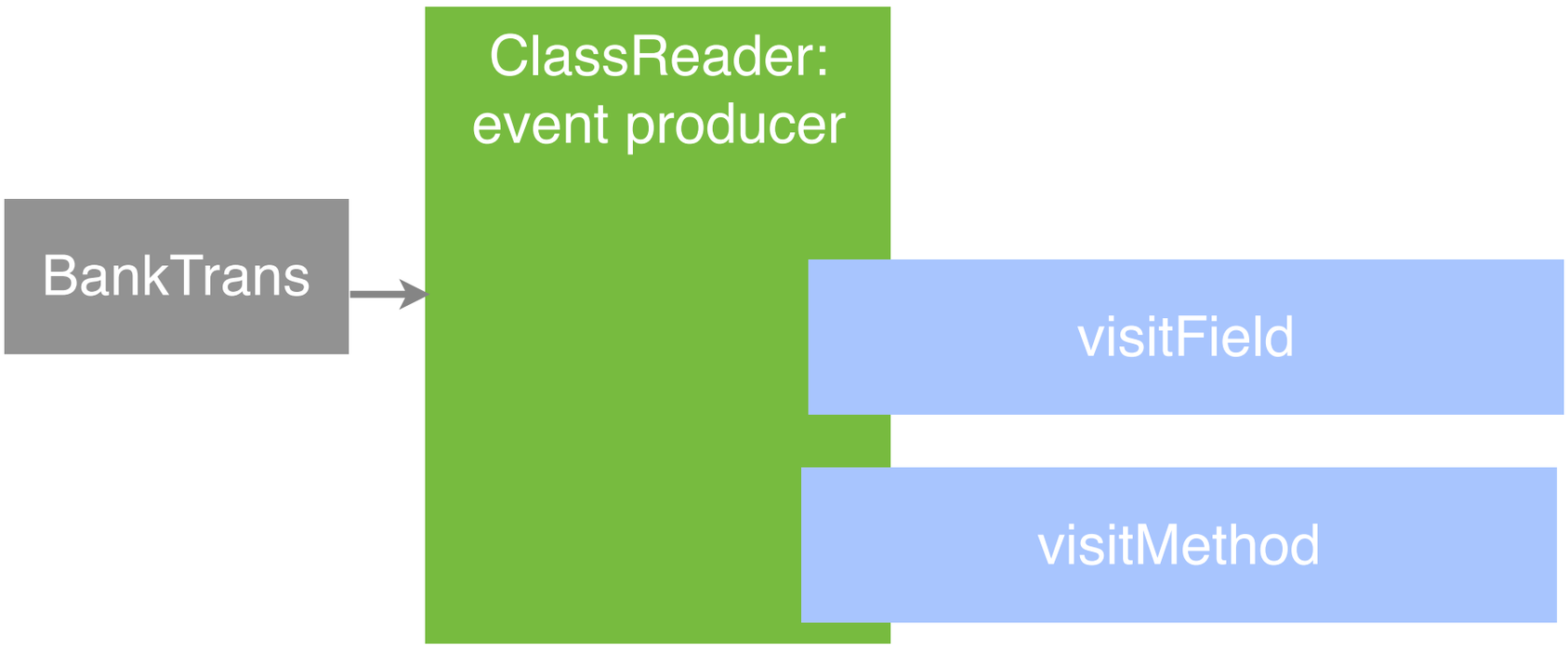
ASM

ClassReader: given a byte[], parses a compiled class



ASM

ClassReader: given a byte[], parses a compiled class

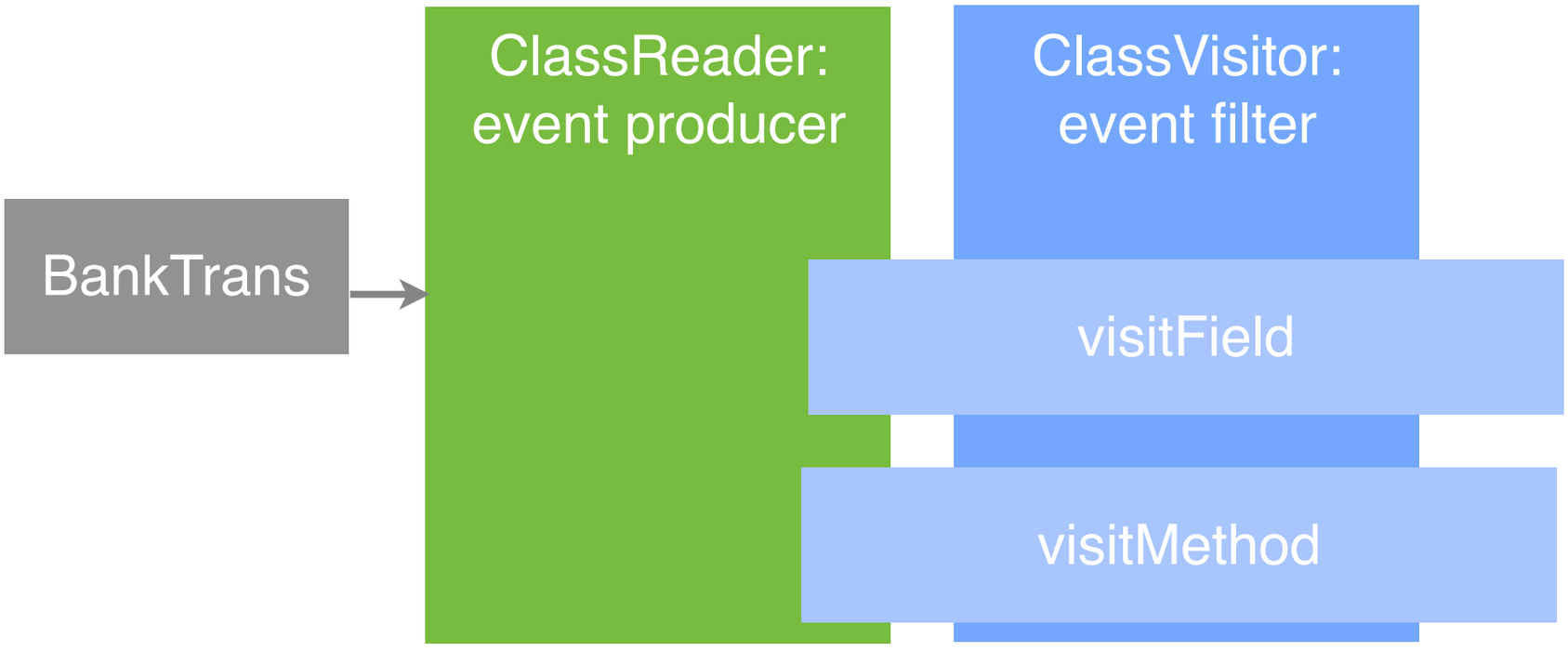


ASM

ClassReader: given a byte[], parses a compiled class

ClassVisitor: delegates class events, event filter

visitAttribute visitField visitMethod visitInnerClass visitEnd



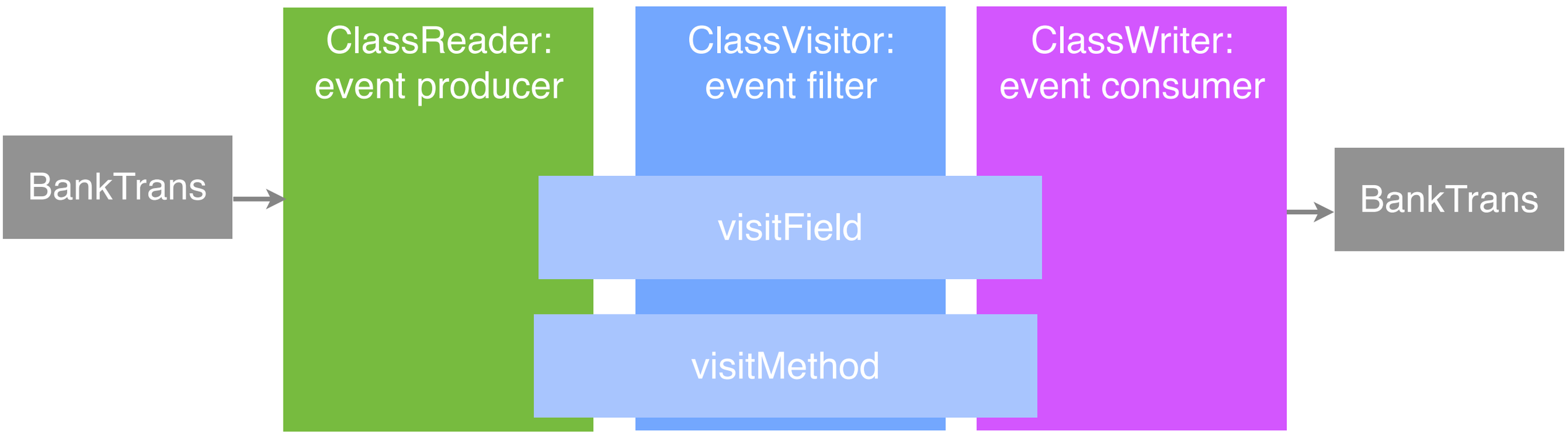
ASM

ClassReader: given a byte[], parses a compiled class

ClassVisitor: delegates class events, event filter

visitAttribute visitField visitMethod visitInnerClass visitEnd

ClassWriter: produces output byte[]



lets code

```
package com.example.qcon.agent;

import java.lang.instrument.Instrumentation;

public class Agent {

   public static void premain(String agentArgs, Instrumentation inst) {
      System.out.println("Starting the agent");
      inst.addTransformer(new ImportantLogClassTransformer());
   }

}
```

lets code

```
package com.example.qcon.agent;

import java.lang.instrument.Instrumentation;

public class Agent {

   public static void premain(String agentArgs, Instrumentation inst) {
      System.out.println("Starting the agent");
      inst.addTransformer(new ImportantLogClassTransformer());
   }

}
```

lets code

```
package com.example.qcon.agent;

import java.lang.instrument.ClassFileTransformer;
import java.lang.instrument.IllegalClassFormatException;

public class ImportantLogClassTransformer implements ClassFileTransformer {

   public byte[] transform(ClassLoader loader, String className,
         Class classBeingRedefined, ProtectionDomain protectionDomain,
         byte[] classfileBuffer) throws IllegalClassFormatException {

     //TODO

      return null;
   }
}
```

lets code

```
package com.example.qcon.agent;

import java.lang.instrument.ClassFileTransformer;
import java.lang.instrument.IllegalClassFormatException;

public class ImportantLogClassTransformer implements ClassFileTransformer {

  public byte[] transform(ClassLoader loader, String className,
        Class classBeingRedefined, ProtectionDomain protectionDomain,
        byte[] classfileBuffer) throws IllegalClassFormatException {

    //TODO

     return null;
  }
}
```

lets code

```
public byte[] transform(ClassLoader loader, String className,
      Class<?> classBeingRedefined, ProtectionDomain protectionDomain,
      byte[] classfileBuffer) throws IllegalClassFormatException {

      ClassReader cr = new ClassReader(classfileBuffer);
      ClassWriter cw = new ClassWriter(cr, ClassWriter.COMPUTE_FRAMES);

      cr.accept(cw, 0);
      return cw.toByteArray();
}
```

lets code

```
public byte[] transform(ClassLoader loader, String className,
      Class<?> classBeingRedefined, ProtectionDomain protectionDomain,
      byte[] classfileBuffer) throws IllegalClassFormatException {

      ClassReader cr = new ClassReader(classfileBuffer);
      ClassWriter cw = new ClassWriter(cr, ClassWriter.COMPUTE_FRAMES);
      ClassVisitor cv = new LogMethodClassVisitor(cw, className);
      cr.accept(cv, 0);
      return cw.toByteArray();
}
```

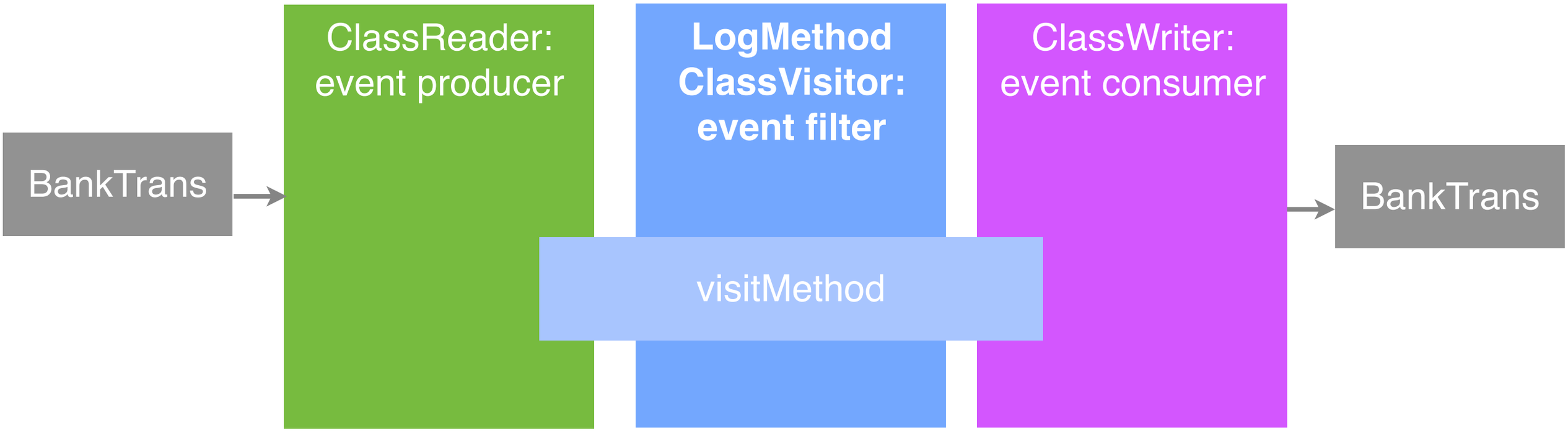
ASM

ClassReader: given a byte[], parses a compiled class

ClassVisitor: delegates class events, event filter

visitAttribute visitField visitMethod visitInnerClass visitEnd

ClassWriter: produces output byte[]



ASM

```
public abstract class ClassVisitor {
  public ClassVisitor(int api);
  public ClassVisitor(int api, ClassVisitor cv);
  public void visit(int version, int access, String name,
      String signature, String superName, String[] interfaces);
  public void visitSource(String source, String debug);
  public void visitOuterClass(String owner, String name, String desc);
  AnnotationVisitor visitAnnotation(String desc, boolean visible);
  public void visitAttribute(Attribute attr);
  public void visitInnerClass(String name, String outerName,
      String innerName, int access);
  public FieldVisitor visitField(int access, String name, String desc,
      String signature, Object value);
  public MethodVisitor visitMethod(int access, String name, String desc,
      String signature, String[] exceptions);
  void visitEnd();
}
```

Figure 2.4.: The `ClassVisitor` class

Source: http://download.forge.objectweb.org/asm/asm4-guide.pdf

lets code

```
public class LogMethodClassVisitor extends ClassVisitor {

    private String className;

    public LogMethodIfAnnotationVisitor(ClassVisitor cv, String pClassName) {
        super(Opcodes.ASM5, cv);
        className = pClassName;
    }

    @Override
    public MethodVisitor visitMethod(int access, String name, String desc,
            String signature, String[] exceptions) {

       // put our logic in here

    }
}
```

lets code

```
public class LogMethodClassVisitor extends ClassVisitor {

    private String className;

    public LogMethodIfAnnotationVisitor(ClassVisitor cv, String pClassName) {
        super(Opcodes.ASM5, cv);
        className = pClassName;
    }

    @Override
    public MethodVisitor visitMethod(int access, String name, String desc,
            String signature, String[] exceptions) {

       // put our logic in here

    }
}
```

ASM

Modifiers, name, super class, interfaces	
Constant pool: numeric, string and type constants	
Source file name (optional)	
Enclosing class reference	
Annotation*	
Attribute*	
Inner class*	Name
Field*	Modifiers, name, type
	Annotation*
	Attribute*
Method*	Modifiers, name, return and parameter types
	Annotation*
	Attribute*
	Compiled code

Figure 2.1.: Overall structure of a compiled class (* means zero or more)

Source: http://download.forge.objectweb.org/asm/asm4-guide.pdf

ASM

```
abstract class MethodVisitor { // public accessors ommited
  MethodVisitor(int api);
  MethodVisitor(int api, MethodVisitor mv);
  AnnotationVisitor visitAnnotationDefault();
  AnnotationVisitor visitAnnotation(String desc, boolean visible);
  AnnotationVisitor visitParameterAnnotation(int parameter,
      String desc, boolean visible);
  void visitAttribute(Attribute attr);
  void visitCode();
  void visitFrame(int type, int nLocal, Object[] local, int nStack,
      Object[] stack);
  void visitInsn(int opcode);
  void visitIntInsn(int opcode, int operand);
  void visitVarInsn(int opcode, int var);
  void visitTypeInsn(int opcode, String desc);
  void visitFieldInsn(int opc, String owner, String name, String desc);
  void visitMethodInsn(int opc, String owner, String name, String desc);
  void visitInvokeDynamicInsn(String name, String desc, Handle bsm,
      Object... bsmArgs);
  void visitJumpInsn(int opcode, Label label);
  void visitLabel(Label label);
  void visitLdcInsn(Object cst);
  void visitIincInsn(int var, int increment);
  void visitTableSwitchInsn(int min, int max, Label dflt, Label[] labels);
  void visitLookupSwitchInsn(Label dflt, int[] keys, Label[] labels);
  void visitMultiANewArrayInsn(String desc, int dims);
  void visitTryCatchBlock(Label start, Label end, Label handler,
      String type);
  void visitLocalVariable(String name, String desc, String signature,
      Label start, Label end, int index);
  void visitLineNumber(int line, Label start);
  void visitMaxs(int maxStack, int maxLocals);
  void visitEnd();
}
```

Source: http://download.forge.objectweb.o

ASM

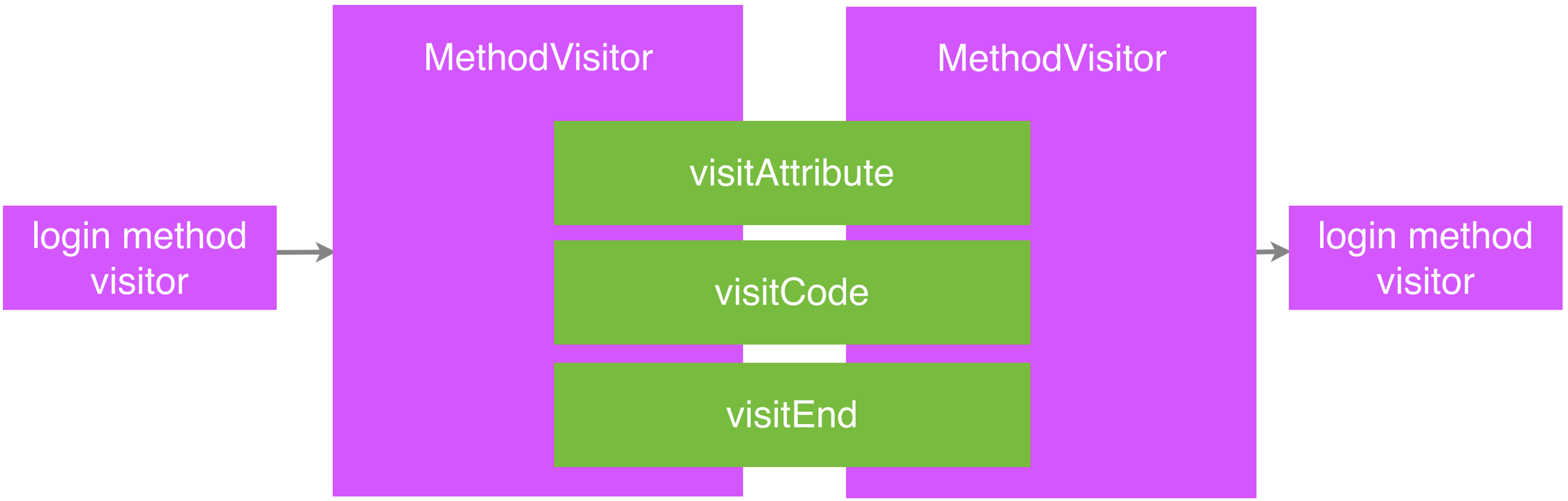
```
visitAnnotationDefault?
( visitAnnotation | visitParameterAnnotation | visitAttribute )*
( visitCode
  ( visitTryCatchBlock | visitLabel | visitFrame | visitXxxInsn |
    visitLocalVariable | visitLineNumber )*
  visitMaxs )?
visitEnd
```

Source: http://download.forge.objectweb.org/asm/asm4-guide.pdf

ASM

MethodVisitor: method event filter

visitAnnotation visitEnd visitCode visitTryCatchBlock



ASM

```
abstract class MethodVisitor { // public accessors ommited
  MethodVisitor(int api);
  MethodVisitor(int api, MethodVisitor mv);
  AnnotationVisitor visitAnnotationDefault();
  AnnotationVisitor visitAnnotation(String desc, boolean visible);
  AnnotationVisitor visitParameterAnnotation(int parameter,
      String desc, boolean visible);
  void visitAttribute(Attribute attr);
  void visitCode();
  void visitFrame(int type, int nLocal, Object[] local, int nStack,
      Object[] stack);
  void visitInsn(int opcode);
  void visitIntInsn(int opcode, int operand);
  void visitVarInsn(int opcode, int var);
  void visitTypeInsn(int opcode, String desc);
  void visitFieldInsn(int opc, String owner, String name, String desc);
  void visitMethodInsn(int opc, String owner, String name, String desc);
  void visitInvokeDynamicInsn(String name, String desc, Handle bsm,
      Object... bsmArgs);
  void visitJumpInsn(int opcode, Label label);
  void visitLabel(Label label);
  void visitLdcInsn(Object cst);
  void visitIincInsn(int var, int increment);
  void visitTableSwitchInsn(int min, int max, Label dflt, Label[] labels);
  void visitLookupSwitchInsn(Label dflt, int[] keys, Label[] labels);
  void visitMultiANewArrayInsn(String desc, int dims);
  void visitTryCatchBlock(Label start, Label end, Label handler,
      String type);
  void visitLocalVariable(String name, String desc, String signature,
      Label start, Label end, int index);
  void visitLineNumber(int line, Label start);
  void visitMaxs(int maxStack, int maxLocals);
  void visitEnd();
}
```

Source: http://download.forge.objectweb.o

lets code

```
public class LogMethodClassVisitor extends ClassVisitor {

   private String className;

   public LogMethodIfAnnotationVisitor(ClassVisitor cv, String pClassName) {
      super(Opcodes.ASM5, cv);
      className = pClassName;
   }

   @Override
   public MethodVisitor visitMethod(int access, String name, String desc,
         String signature, String[] exceptions) {

         MethodVisitor mv = super.visitMethod(access, name, desc, signature,
             exceptions);
         return new PrintMessageMethodVisitor(mv, name, className);

   }
}
```

lets code

```
public class PrintMessageMethodVisitor extends MethodVisitor {
	private String methodName;
	private String className;
	private boolean isAnnotationPresent;
	private List<String> parameterIndexes;

	public PrintMessageMethodVisitor(MethodVisitor mv, String methodName,
		String className) {

	}

	@Override
	public AnnotationVisitor visitAnnotation(String desc, boolean visible) {


	}

	@Override
	public void visitCode() {

	 }
}
```

lets code

```
public class PrintMessageMethodVisitor extends MethodVisitor {
      private String methodName;
      private String className;
      private boolean isAnnotationPresent;
      private List<String> parameterIndexes;

      public PrintMessageMethodVisitor(MethodVisitor mv, String methodName,
            String className) {
          // initialize instance variables
      }

      @Override
      public AnnotationVisitor visitAnnotation(String desc, boolean visible) {
          1. check method for annotation @ImportantLog
          2. if annotation present, then get important method param indexes
      }

      @Override
      public void visitCode() {
         3. if annotation present, add logging to beginning of the method
       }
}
```

lets code

```
public AnnotationVisitor visitAnnotation(String desc, boolean visible) {
    if ("Lcom/example/qcon/mains/ImportantLog;".equals(desc)) {
        isAnnotationPresent = true;
        return new AnnotationVisitor(Opcodes.ASM5,
                super.visitAnnotation(desc, visible)) {
            public AnnotationVisitor visitArray(String name) {
                if (“fields”.equals(name)) {
                    return new AnnotationVisitor(Opcodes.ASM5,
                            super.visitArray(name)) {
                        public void visit(String name, Object value) {
                            parameterIndexes.add((String) value);
                            super.visit(name, value);
                        }
                    };
                } else {
                    return super.visitArray(name);
                }
            }
        };
    }
    return super.visitAnnotation(desc, visible);
}
```

lets code

```
public class PrintMessageMethodVisitor extends MethodVisitor {
      private String methodName;
      private String className;
      private boolean isAnnotationPresent;
      private List<String> parameterIndexes;

      public PrintMessageMethodVisitor(MethodVisitor mv, String methodName,
            String className) {
          // initialize instance variables
      }

      @Override
      public AnnotationVisitor visitAnnotation(String desc, boolean visible) {
          1. check method for annotation @ImportantLog
          2. if annotation present, then get important method param indexes
      }

      @Override
      public void visitCode() {
         3. if annotation present, add logging statement to beginning of the method
       }
}
```

lets code

```
public void visitCode() {
    if (isAnnotationPresent) {
        // create string builder
        mv.visitFieldInsn(Opcodes.GETSTATIC, "java/lang/System", "out",
                "Ljava/io/PrintStream;");
        mv.visitTypeInsn(Opcodes.NEW, "java/lang/StringBuilder");
        mv.visitInsn(Opcodes.DUP);

        // add everything to the string builder
        mv.visitLdcInsn("A call was made to method \"");
        mv.visitMethodInsn(Opcodes.INVOKESPECIAL,
                "java/lang/StringBuilder", "<init>",
                "(Ljava/lang/String;)V", false);

        mv.visitLdcInsn(methodName);
        mv.visitMethodInsn(Opcodes.INVOKEVIRTUAL,
                "java/lang/StringBuilder", "append",
                "(Ljava/lang/String;)Ljava/lang/StringBuilder;", false);

. . .
```

What is Java bytecode?

use javap to examine the bytecode

What is Java bytecode?

```
public class PrintMessage {
    private String methodName;
    private String className;

    public PrintMessage(String mName, String cName) {
        methodName = mName;
        className = cName;
    }

    public void print() {
        StringBuilder sb = new StringBuilder();
        sb.append("A call was made to method \"");
        sb.append(methodName);
        sb.append("\" on class \"");
        sb.append(className);
        sb.append("\".");
        System.out.println(sb.toString());
    }
```

What is Java bytecode?

```
javap -c bin/com/example/qcon/mains/PrintMessage
```

What is Java bytecode?

```
javap -c bin/com/example/qcon/mains/PrintMessage

public void print();
    Code:
       0: new           #38                 // class java/lang/StringBuilder
       3: dup
       4: invokespecial #40                 // Method java/lang/
StringBuilder."<init>":()V
       7: astore_1
       8: aload_1
       9: ldc           #41                 // String A call was made to method \"
      11: invokevirtual #43                 // Method java/lang/
StringBuilder.append:(Ljava/lang/String;)Ljava/lang/StringBuilder;
      14: pop
      15: aload_1
      16: aload_0
      17: getfield      #14                 // Field methodName:Ljava/lang/String;
      20: invokevirtual #43                 // Method java/lang/
StringBuilder.append:(Ljava/lang/String;)Ljava/lang/StringBuilder;
 . . .
```

What is Java bytecode?

each line represents one JVM instruction

```
public void print();
    Code:
       0: new           #38                 // class java/lang/StringBuilder
       3: dup
       4: invokespecial #40                 // Method java/lang/
StringBuilder."<init>":()V
       7: astore_1
       8: aload_1
       9: ldc           #41                 // String A call was made to method \"
      11: invokevirtual #43                 // Method java/lang/
StringBuilder.append:(Ljava/lang/String;)Ljava/lang/StringBuilder;
      14: pop
      15: aload_1
      16: aload_0
      17: getfield      #14                 // Field methodName:Ljava/lang/String;
      20: invokevirtual #43                 // Method java/lang/
StringBuilder.append:(Ljava/lang/String;)Ljava/lang/StringBuilder;
 . . .
```

What is Java bytecode?

JVM instruction = 1 opcode with 0 or more operands

```
public void print();
    Code:
       0: new           #38                 // class java/lang/StringBuilder
       3: dup
       4: invokespecial #40                 // Method java/lang/
StringBuilder."<init>":()V
       7: astore_1
       8: aload_1
       9: ldc           #41                 // String A call was made to method \"
      11: invokevirtual #43                 // Method java/lang/
StringBuilder.append:(Ljava/lang/String;)Ljava/lang/StringBuilder;
      14: pop
      15: aload_1
      16: aload_0
      17: getfield      #14                 // Field methodName:Ljava/lang/String;
      20: invokevirtual #43                 // Method java/lang/
StringBuilder.append:(Ljava/lang/String;)Ljava/lang/StringBuilder;
 . . .
```

What is Java bytecode?

Opcode	Description
load	load a reference onto the stack from a local variable iload, lload, fload, dload, aload
store	stores a value from the stack into a local variable istore, lstore, fstore, dstore, astore
lcd	push a constant from a constant pool onto the stack
return	method return instruction ireturn, lreturn, freturn, dreturn, areturn, return
invokevirtual	invoke an instance method on an object
invokespecial	invokes an instance method requiring special handling such as an initialization method

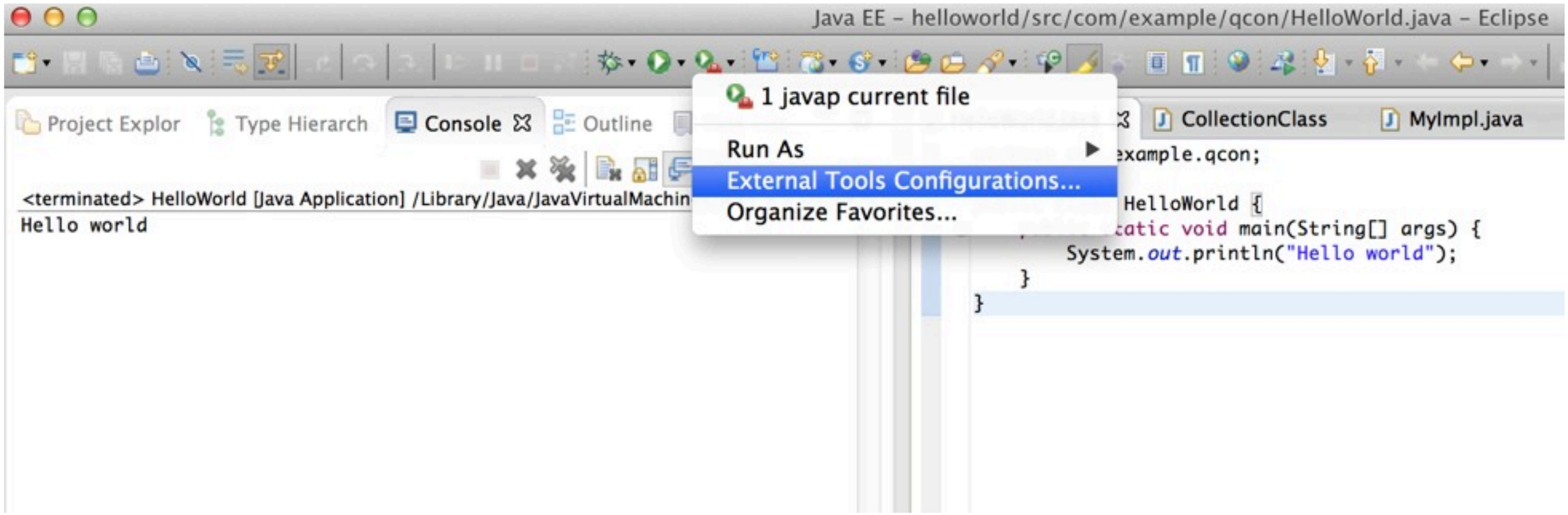
See the spec for more info: http://docs.oracle.com/javase/specs/jvms/se8/jvms8.pdf

What is Java bytecode?

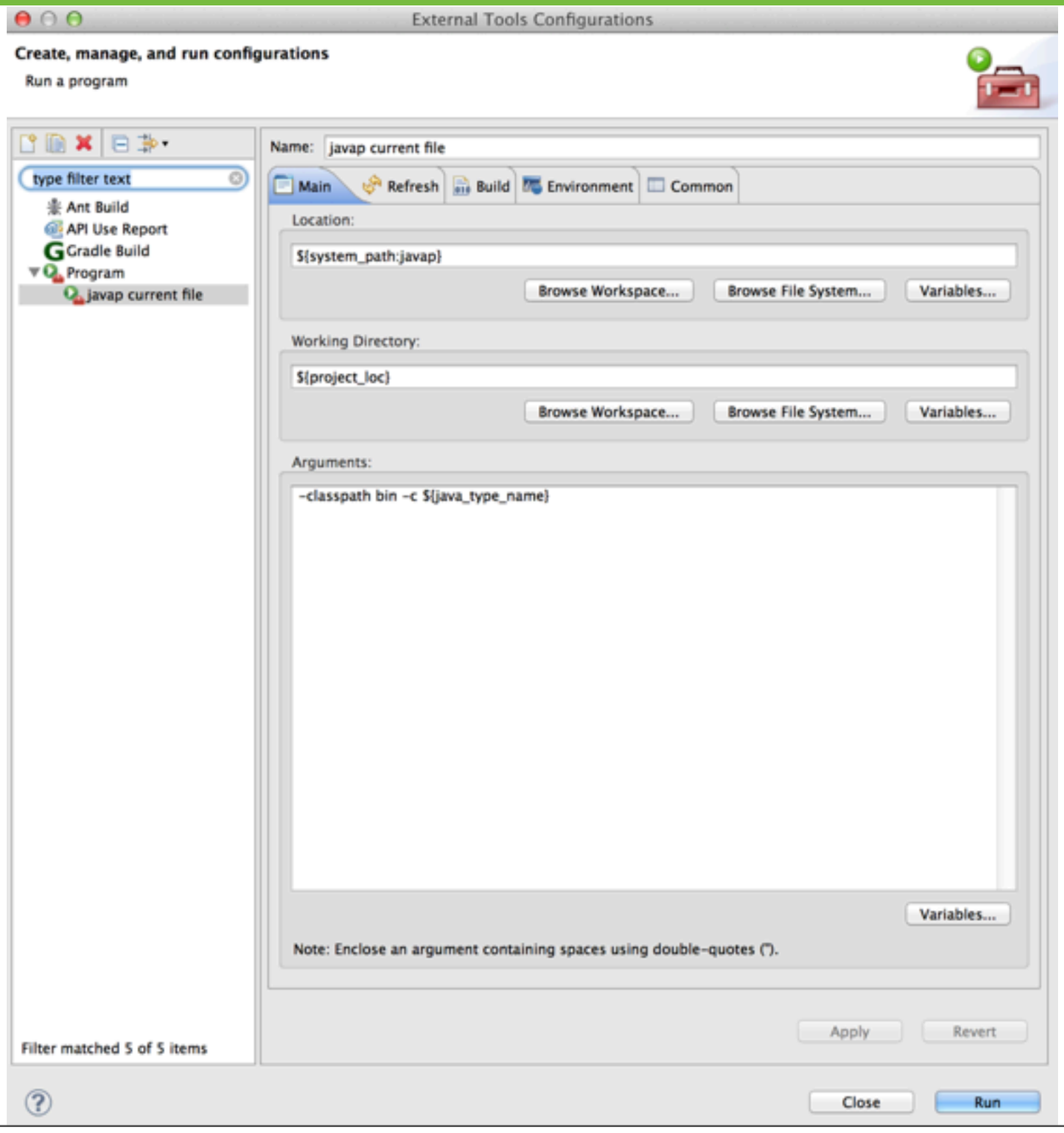
Opcode	Description
invokestatic	invoke a class (static) method on a named class
athrow	throws an exception
new	create a new class
newarray	create a new array
if<cond>	branch if int comparison with 0 succeeds ifeq, ifne, iflt, ifge, ifgt, ifle
dup	duplicate the top operand stack value

See the spec for more info: http://docs.oracle.com/javase/specs/jvms/se8/jvms8.pdf

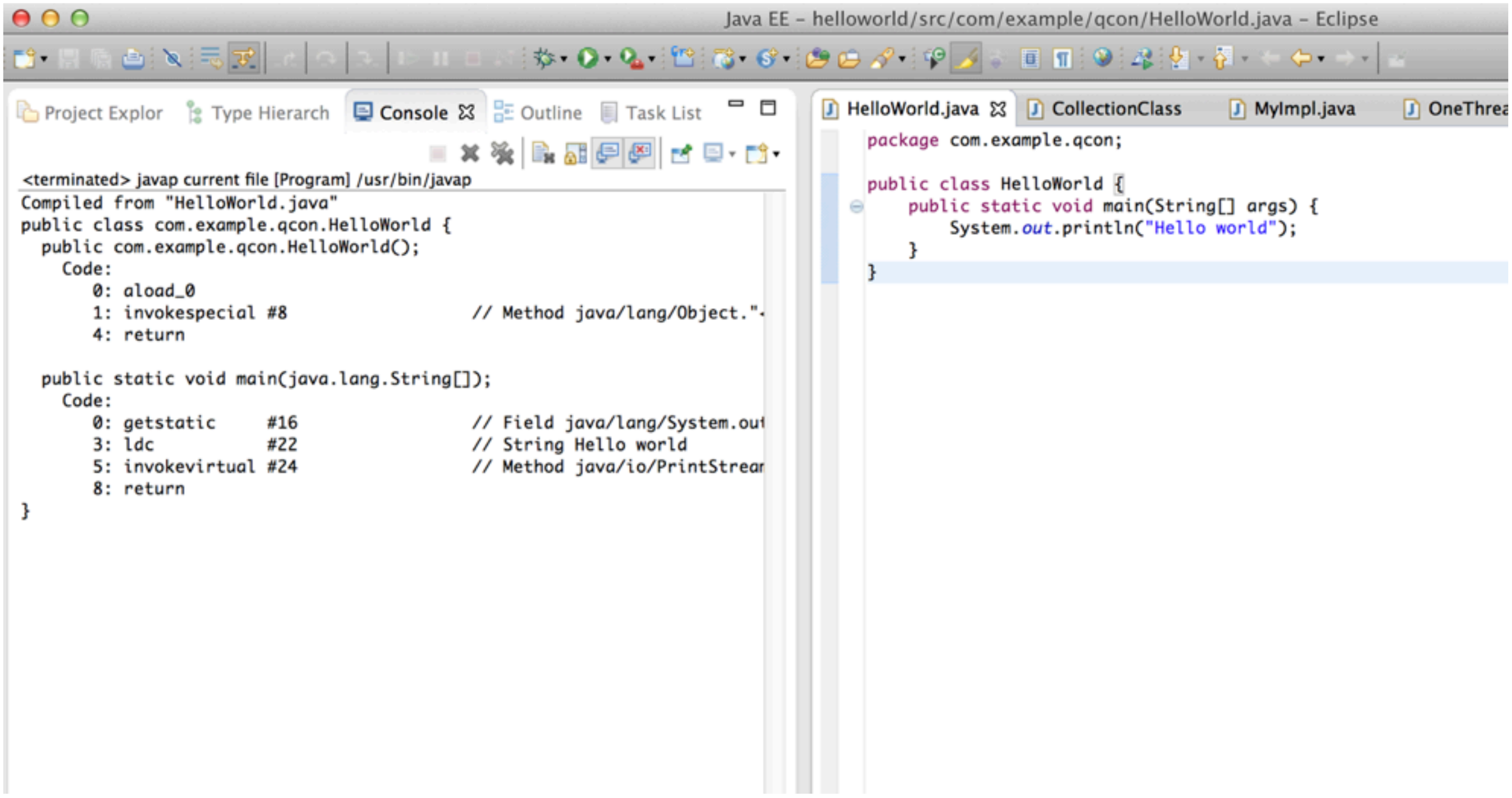
What is Java bytecode?



What is Java bytecode?



What is Java bytecode?



lets code

```
public void visitCode() {
    if (isAnnotationPresent) {
        // create string builder
        mv.visitFieldInsn(Opcodes.GETSTATIC, "java/lang/System", "out",
                "Ljava/io/PrintStream;");
        mv.visitTypeInsn(Opcodes.NEW, "java/lang/StringBuilder");
        mv.visitInsn(Opcodes.DUP);

        // add everything to the string builder
        mv.visitLdcInsn("A call was made to method \"");
        mv.visitMethodInsn(Opcodes.INVOKESPECIAL,
                "java/lang/StringBuilder", "<init>",
                "(Ljava/lang/String;)V", false);

        mv.visitLdcInsn(methodName);
        mv.visitMethodInsn(Opcodes.INVOKEVIRTUAL,
                "java/lang/StringBuilder", "append",
                "(Ljava/lang/String;)Ljava/lang/StringBuilder;", false);

. . .
```

ASM

- Positives
  - Speed and Size
  - Decent documentation
  - Depth of Functionality
  - Number of people using framework
- Negatives
  - Need to write actual byte code
  - longer developer ramp-up

Why learn about Java bytecode manipulation frameworks ?

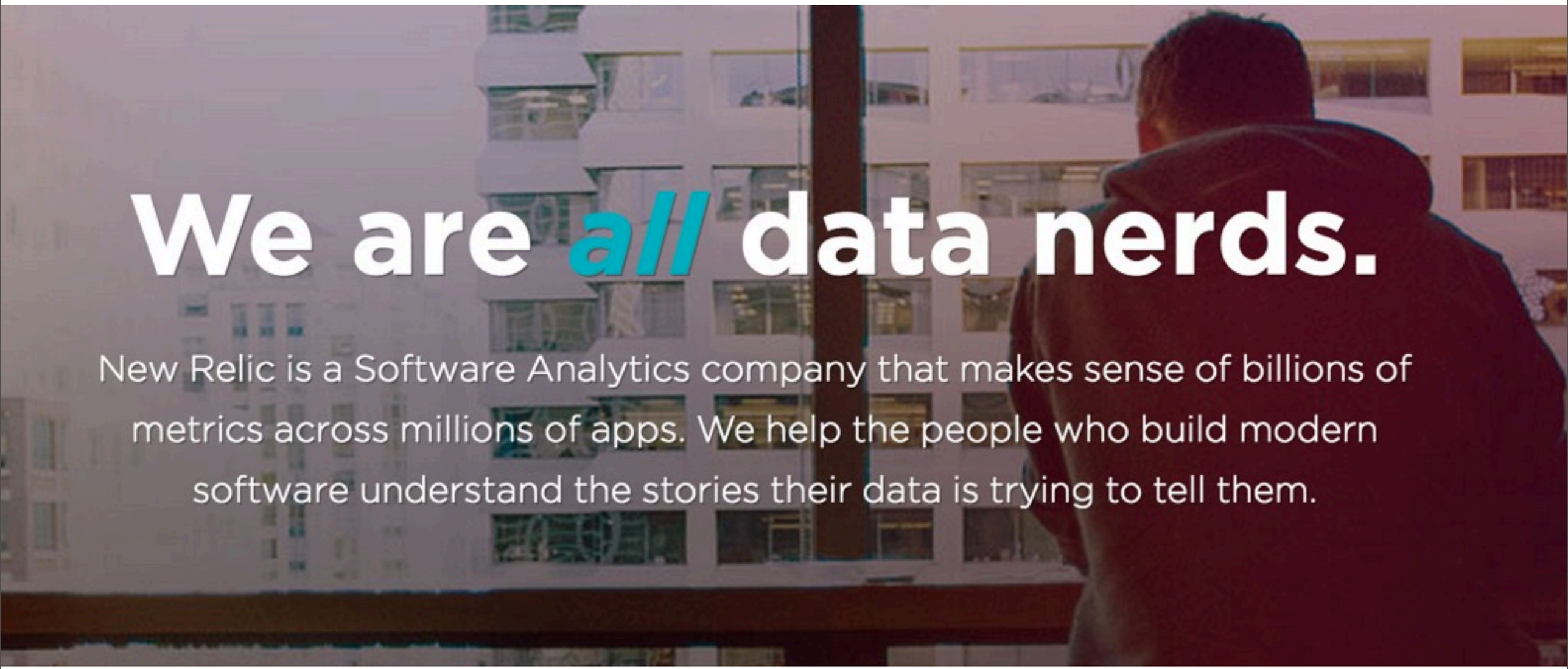
- program analysis
  - find bugs in code
  - examine code complexity
- generate classes
  - proxies
  - remove access to certain APIs
  - compiler for another language like Scala
- transform classes without Java source code
  - profilers
  - optimization and obfuscation
  - additional logging

Bytecode Manipulation Frameworks

- ASM: http://asm.ow2.org/
- BCEL: http://commons.apache.org/proper/commons-bcel/
- CGLib: https://github.com/cglib/cglib
- Javassist: http://www.csg.ci.i.u-tokyo.ac.jp/~chiba/javassist/
- Serp: http://serp.sourceforge.net/
- Cojen: https://github.com/cojen/Cojen/wiki
- Soot: http://www.sable.mcgill.ca/soot/

Make bytecode manipulation frameworks your friend.
They might save your job one day.

Source: http://www.kulfoto.com/dog-pictures/15799/mans-best-friend


We are all data nerds.
New Relic is a Software Analytics company that makes sense of billions of metrics across millions of apps. We help the people who build modern software understand the stories their data is trying to tell them.

Questions?

Source: http://alangregerman.typepad.com/.a/6a00d83516c0ad53ef017c3329aba8970b-800wi