Reflection

António Menezes Leitão

March 7, 2022

Computational System

Definition

Computational System: a system that reasons about and acts upon a given domain.

Definition

The domain is represented by the internal structures of the system:

- Data representing entities and relations.
- Program prescribing data manipulation.

Notes

- A program **is not** a computational system.
- A program describes (part of) a computational system.
- A running program is a computational system.

Computational Meta-System

Definition

Computational Meta-System: a computational system that has as domain another computational system (called the **Object System**).

Definition

A computational meta-system operates on data that represents the computational object-system.

Examples

- A debugger is a computational meta-system.
- A profiler is a computational meta-system.
- A (classic) compiler **is not** a computational meta-system (its domain is a program and not a computational system)

Reflection

Definition

Reflection: the process of reasoning about and/or acting upon oneself.

Definition

Reflective System: a meta-system that has itself as object-system.

Definition

A reflective system is a system that can represent and manipulate its own structure and behavior at run time.

Two *levels* of Reflection

Definition

Introspection: the ability of a program to *examine* its own structure and behavior.

Definition

Intercession: the ability of a program to *modify* its own structure and behavior.

Examples

- **Introspection**: How many parameters has the function `foo`?
- **Intercession**: Change the class of this instance to `Bar`!

Different Definitions

Definition

Self-Modification: the ability of a program to modify its own structure.

Definition

Intercession: the ability to modify the semantics of the underlying programming language from within the language itself.

Examples

Language	Introspection	Self-Modification	Intercession
Java	Yes	No	No
Java Debugger Interface	Yes	Limited	No
Smalltalk	Yes	Yes	Very Limited
CLOS	Yes	Yes	Yes

Reification

Definition

Reification: the creation of an entity that represents, in the meta-system, an entity of the object-system. Reification is a pre-condition for reflection.

Examples

- What is the class of this instance? $\Rightarrow$ reification of classes.
- Which are the methods of this class? $\Rightarrow$ reification of methods.
- What was the call chain that caused this bug? $\Rightarrow$ reification of the *stack*.
- Which are the values of the free variables of this function? $\Rightarrow$ reification of the lexical environment.

Reification

Notes

- There is a relation between an entity and its reification. Any change in one causes a corresponding change in the other.
- A reified entity is a **first-class** entity. Not all first-class entities must be reifiable entities.

Examples

- In Scheme, Common Lisp, and Emacs Lisp, a function is a first-class entity.
- In Scheme, a function cannot be introspected.
- In Common Lisp, a function can be (partially) introspected (`function-lambda-expression`, `disassemble`, `ed`).
- In Emacs Lisp, a (non-compiled) function can be introspected.

Two *levels* of Reification

Definition

Structural Reification: the ability of a system to reify its own *structure*.

Definition

Behavioral (or computational) Reification: the ability of a system to reify its own *execution*.

Examples

- Which are the instance variables of this class? $\Rightarrow$ structural reification.
- Which are the active *error handlers* at this moment? $\Rightarrow$ behavioral reification.

Reification

Notes

- Behavioral reification is harder to implement than structural reification.
- Intercession over behavioral reification makes compilation harder.

Issues

- How to formalize the semantics of a language that can change during program execution?
- How to reify while preserving efficiency?
- How to compile programs whose semantics can change during execution?

Programming Languages *vs* Programming Environments

Definition

A **Programming Language** allows the description of computational processes.

Definition

A **Program** is the description of a computational process written in a programming language.

Definition

A **Programming Environment** is a computational system that helps the development of programs.

Programming Languages *vs* Programming Environments

Notes

- Not all programming languages provide sufficient reflection mechanisms.
- But most programming environments provide alternative mechanisms for the same purpose.
- When a program can be executed separately from the programming environment, it is important to distinguish between mechanisms provided by the programming language and mechanisms provided by the programming environment.

Important Difference

- The programming language is always available.
- The programming environment is available only during development.

Reflective Architecture

Definition

A programming language is said to have a **Reflective Architecture** if it recognizes reflection as a fundamental programming concept and thus provides tools for handling reflective computation explicitly.

Consequences

- The language processor provides data that represents the system itself.
- Programs can describe reflective computations over such data.
- There is a relation between the data and the aspects of the system that it represents.
- Modifications in the data cause modifications in the state and behavior of the system.

Languages with Reflective Architecture

Lisp

- Motto: *"Equivalence between program and data"*.
- Allows introspection of programs as if they are data.
- Allows construction of programs from data.

Smalltalk

- Motto: *"Objects everywhere"*.
- Same approach (classes and methods) used for building everything, including compiler, virtual machine, IDE, etc.
- Allows introspection (and limited intercession) of classes, instances, compiler, *stack*, etc.

Java

Features

- Syntactically, a descendant of C++.
- Semantically, a descendant of Smalltalk.
- Huge library.
- Widely used.
- Started without any reflective capabilities.
- Each new version includes more reflective capabilities.
- Nowadays, allows structural introspection and limited forms of behavioral intercession.
- Reflection operates over the language elements: classes, fields, constructors, methods, etc.

Reflection in Java

Constructors and non-private methods of a class

```
import java.lang.reflect.*;

public class PrintClass {

    public static void main(String[] args)
      throws ClassNotFoundException {
        if (args.length != 1) {
            System.err.println("Usage: java PrintClass <class>");
            System.exit(1);
        } else {
            dumpClass(Class.forName(args[0]));
        }
    }

    static void dumpClass(Class c) {
        …
    }
}
```

Reflection in Java

Constructors and non-private methods of a class

```
static void dumpClass(Class c) {
    System.out.println(c + " {");

    for (Constructor con : c.getConstructors()) {
        System.out.println("    " + con);
    }

    for (Method m : c.getDeclaredMethods()) {
        if (! Modifier.isPrivate(m.getModifiers())) {
            System.out.println("    " + m);
        }
    }

    System.out.println("}");
}
```

Reflection in Java

Constructors and non-private methods of a class

```
$ java PrintClass java.lang.Object
class java.lang.Object {
    public java.lang.Object()
    public native int java.lang.Object.hashCode()
    public final native java.lang.Class java.lang.Object.getClass()
    …
    public boolean java.lang.Object.equals(java.lang.Object)
    public final native void java.lang.Object.notify()
    public final native void java.lang.Object.notifyAll()
    public java.lang.String java.lang.Object.toString()
}
```

Reflection in Java

Types

- Primitive Types: `boolean`, `byte`, `short`, `int`, `long`, `char`, `float`, and `double`.
- Reference Types: `java.lang.String`, `java.io.Serializable`, `java.lang.Integer`, and all the others.

Reified Types

- For each (primitive or reference) type, there is an (unique) instance of the class `java.lang.Class` that represents that type.
- The `java.lang.Class` class contains methods that:
  - provide information about the class (methods, variables, etc.),
  - create instances of the class,
  - change variables and call methods.

Reflection in Java

To obtain an instance of `java.lang.Class`

- From an object *foo*:
  foo`.getClass()`
- From a type *Bar*:
  Bar`.class`
- From the name of a type "*foo.bar.Baz*" (if not found, throws the *Checked exception* `ClassNotFoundException`):
  `Class.forName("`*foo.bar.Baz*`")`

Example

```
"I am a string".getClass()

String.class

Class.forName("java.lang.String")
```

Reflection in Java

Important methods of class `Class`

- `boolean isPrimitive()`
  Determines if the type represented by the receiver is a primitive type.
- `boolean isInterface()`
  Determines if the type represented by the receiver represents an interface type.
- `boolean isArray()`
  Determines if the type represented by the receiver is an array class.
- `Class getComponentType()`
  Returns the Class representing the component type of the array class represented by the receiver.
- `String getName()`
  Returns the name of the entity (class, interface, array class, primitive type, or void) represented by the receiver, as a String.

Reflection in Java

Important methods of class `Class`

- `Package getPackage()`
  Gets the package of type represented by the receiver.
- `int getModifiers()`
  Returns the Java language modifiers for the type represented by the receiver, encoded in an integer.
- `Class getSuperclass()`
  Returns the Class representing the superclass of the class represented by the receiver.
- `Class[] getInterfaces()`
  Determines the Classes representing the interfaces implemented by the class or interface represented by the receiver.
- `Class getDeclaringClass()`
  If the class or interface represented by the receiver is a member of another class, returns the Class object representing that class.

Reflection in Java

Important methods of class `Class`

- `Class[] getClasses()`
  Returns an array containing `Classes` representing all the public classes and interfaces members of the class represented by the receiver.
- `Field[] getFields()`
  Returns an array containing `Fields` representing all the accessible public fields of the class or interface represented by the receiver.
- `Constructor[] getConstructors()`
  Returns an array containing `Constructors` representing all the public constructors of the class represented by the receiver.
- `Method[] getMethods()`
  Returns an array containing `Methods` representing all the public member methods of the class or interface represented by the receiver, including those declared by the class or interface and those inherited from superclasses and superinterfaces.

Reflection in Java

Important methods of class `Class`

- `Class[] getDeclaredClasses()`
  Returns an array of `Classes` representing all the classes and interfaces declared as members of the class represented by the receiver.
- `Field[] getDeclaredFields()`
  Returns an array of `Fields` reflecting all the fields declared by the class or interface represented by the receiver.
- `Constructor[] getDeclaredConstructors()`
  Returns an array of `Constructors` representing all the constructors declared by the class represented by the receiver.
- `Method[] getDeclaredMethods()`
  Returns an array of `Methods` reflecting all the methods declared by the class or interface represented by the receiver.

Reflection in Java

Important methods of class `Class`

- `Field getField(String name)`
  Returns a `Field` that represents the specified public member field of the class or interface represented by the receiver.
- `Field getDeclaredField(String name)`
  Returns a `Field` representing the specified declared field of the class or interface represented by the receiver.
- `Constructor getConstructor(Class[] types)`
  Returns a `Constructor` that represents the specified public constructor of the class represented by the receiver.
- `Constructor getDeclaredConstructor(Class[] types)`
  Returns a `Constructor` that represents the specified constructor of the class represented by the receiver.

Reflection in Java

Important methods of class `Class`

- `Method getMethod(String name, Class[] types)`
  Returns a `Method` that represents the specified public member method of the class or interface represented by the receiver.
- `Method getDeclaredMethod(String name, Class[] types)`
  Returns a `Method` that represents the specified declared method of the class or interface represented by the receiver.
- `boolean isAssignableFrom(Class cls)`
  Determines if the class or interface represented by the receiver is either the same as, or is a superclass or superinterface of, the class or interface represented by the specified `Class` parameter.
- `Object newInstance()`
  Creates a new instance of the class represented by the receiver.
- `boolean isInstance(Object obj)`
  Determines if the specified Object is assignment-compatible with the type represented by the receiver.

Reflection in Java

Beware of the *package*

- `java.lang.Object`
- `java.lang.Class`
- `java.lang.Package`
- `java.lang.reflect.Field`
- `java.lang.reflect.Constructor`
- `java.lang.reflect.Method`
- `java.lang.reflect.Modifier`

Example: Write an interface from a set of classes

Classes

Cars

```
public class Car {

    void fillTank() { ... }

    void move() { ... }

    void stop() { ... }
}
```

Bicycles

```
public class Bicycle {

    void move() { ... }

    void stop() { ... }

    void drop() { ... }
}
```

Interface

Example: Write an interface from a set of classes

Classes

Cars

```
public class Car {

    void fillTank() { ... }

    void move() { ... }

    void stop() { ... }
}
```

Bicycles

```
public class Bicycle {

    void move() { ... }

    void stop() { ... }

    void drop() { ... }
}
```

Interface

```
$ java CommonInterface MovableThing Car Bicycle
```

Example: Write an interface from a set of classes

Classes

Cars

```
public class Car {

    void fillTank() { ... }

    void move() { ... }

    void stop() { ... }
}
```

Bicycles

```
public class Bicycle {

    void move() { ... }

    void stop() { ... }

    void drop() { ... }
}
```

Interface

```
$ java CommonInterface MovableThing Car Bicycle
public interface MovableThing  {
    public abstract void stop ();
    public abstract void move ();
}
```

Example: Write an interface from a set of classes

CommonInterface

```
import java.lang.reflect.*;
import java.io.*;

public class CommonInterface {

    protected static void print(String text) {
        System.out.print(text);
    }
    protected static void println(String text) {
        System.out.println(text);
    }
    protected static void printsp(String text) {
        System.out.print(text);
        System.out.print(" ");
    }

    …
}
```

Example: Write an interface from a set of classes

CommonInterface

```
public static void main(String[] args) {
    if (args.length > 1) {
        commonInterface(
            args[0],
            getAllClasses(Arrays.stream(args, 1, args.length)));
    } else {
        println("Usage: java CommonInterface <interface> <class>+");
        System.exit(1);
    }
}
```

Example: Write an interface from a set of classes

CommonInterface

```
static Class getClass(String className) {
    try {
        return (Class.forName(className));
    } catch (ClassNotFoundException cnfe) {
        System.err.println("Class '" + className + "' not found");
        System.exit(1);
        return null;
    }
}

static Stream<Class> getAllClasses(Stream<String> classNames) {
    return classNames.map(CommonInterface::getClass);
}

static void commonInterface(String name, Stream<Class> fromClasses) {
    printsp("public interface");
    printsp(name);
    println(" {");
    printMethods(commonMethods(fromClasses));
    println("}");
}
```

Example: Write an interface from a set of classes

CommonInterface

```
static Stream<Method> commonMethods(Stream<Class> classes) {
    return classes
        .map(c -> Stream.of(c.getDeclaredMethods()))
        .reduce(CommonInterface::intersection)
        .orElse(Stream.empty());
}

static Stream<Method> intersection(Stream<Method> methods,
                                   Stream<Method> otherMethods) {
    return methods.filter(m -> containsMethod(otherMethods, m));
}

static boolean containsMethod(Stream<Method> methods, Method m) {
    return methods.anyMatch(m2 -> equalSignature(m, m2));
}

static boolean equalSignature(Method m1, Method m2) {
    return m1.getName().equals(m2.getName()) &&
       Arrays.equals(m1.getParameterTypes(), m2.getParameterTypes());
}
```

Example: Write an interface from a set of classes

CommonInterface

```
static void printMethods(Stream<Method> methods) {
    methods.forEach(method -> {
        // Avoid methods added by the compiler (e.g., for covariant
        // return types or to ensure overriding of generic types)
        if (! method.isBridge()) {
            int mods = method.getModifiers();
            if (Modifier.isPublic(mods)) {
                printsp("    public abstract");
                printsp(toTypeName(method.getReturnType()));
                printsp(method.getName());
                printParamList(method.getParameterTypes());
                printThrows(method.getExceptionTypes());
                println(";");
            }
        }
     });
}
static String toTypeName(Class classObj) {
    return classObj.isArray() ?
        toTypeName(classObj.getComponentType()) + "[]" :
        classObj.getName();
}
```

Example: Write an interface from a set of classes

CommonInterface

```
static void printParamList(Class[] argTypes) {
    print("(");
    for(int i = 0; i < argTypes.length; i++) {
        if (i > 0) {
            printsp(",");
        }
        print(toTypeName(argTypes[i]) + " arg" + i);
    }
    print(")");
}

static void printThrows(Class[] excepTypes) {
    if (excepTypes.length > 0) {
        printsp(" throws");
        for(int i = 0; i < excepTypes.length; i++) {
            if (i > 0) {
                printsp(",");
            }
            print(toTypeName(excepTypes[i]));
        }
    }
}
```

Example: Multiple Dispatch

Example

```
class Shape {
}

class Line extends Shape {
}

class Circle extends Shape {
}

class Device {
    public void draw(Shape s) {
        System.err.println("draw what where?");
    }
    public void draw(Line l) {
        System.err.println("draw a line where?");
    }
    public void draw(Circle c)  {
        System.err.println("draw a circle where?");
    }
}
```

Example: Multiple Dispatch

Example

```
class Screen extends Device {

    public void draw(Shape s) {
        System.err.println("draw what on screen?");
    }

    public void draw(Line l) {
        System.err.println("drawing a line on screen!");
    }

    public void draw(Circle c) {
        System.err.println("drawing a circle on screen!");
    }
}
```

Example: Multiple Dispatch

Example

```
class Printer extends Device {

    public void draw(Shape s) {
        System.err.println("draw what on printer?");
    }

    public void draw(Line l) {
        System.err.println("drawing a line on printer!");
    }

    public void draw(Circle c) {
        System.err.println("drawing a circle on printer!");
    }
}
```

Example: Multiple Dispatch

Question: *What is the output?*

```
Device[] devices = new Device[] { new Screen(), new Printer() };
Shape[] shapes = new Shape[] { new Line(), new Circle() };
for (Device device : devices) {
    for (Shape shape : shapes) {
        device.draw(shape);
    }
}
```

Example: Multiple Dispatch

Question: *What is the output?*

```
Device[] devices = new Device[] { new Screen(), new Printer() };
Shape[] shapes = new Shape[] { new Line(), new Circle() };
for (Device device : devices) {
    for (Shape shape : shapes) {
        device.draw(shape);
    }
}
```

Answer: *Output*

```
draw what on screen?
draw what on screen?
draw what on printer?
draw what on printer?
```

Example: Multiple Dispatch

Question: *What is the output?*

```
Device[] devices = new Device[] { new Screen(), new Printer() };
Shape[] shapes = new Shape[] { new Line(), new Circle() };
for (Device device : devices) {
    for (Shape shape : shapes) {
        device.draw(shape);
    }
}
```

Answer: *Output*

```
draw what on screen?
draw what on screen?
draw what on printer?
draw what on printer?
```

Bug/Feature

Java uses dynamic dispatch for the receiver and static dispatch (overloading) for the arguments.

Example: Multiple Dispatch

Solution: *TypeCasts*

```
class Device {

    public void draw(Shape s) {
        if (s instanceof Line) {
            draw((Line)s);
        } else if (s instanceof Circle) {
            draw((Circle)s);
        } else {
            System.err.println("draw what where?");
        }
    }

    public void draw(Line l) {
        System.err.println("draw a line where?");
    }

    public void draw(Circle c)  {
        System.err.println("draw a circle where?");
    }
}
```

Example: Multiple Dispatch

Solution: *TypeCasts*

```
class Screen extends Device {

    public void draw(Line l) {
        System.err.println("drawing a line on screen!");
    }

    public void draw(Circle c) {
        System.err.println("drawing a circle on screen!");
    }
}

class Printer extends Device {

    public void draw(Line l) {
        System.err.println("drawing a line on printer!");
    }

    public void draw(Circle c) {
        System.err.println("drawing a circle on printer!");
    }
}
```

Example: Multiple Dispatch

Problems

- It is more efficient to draw instances of `Line` (just one test) than instances of `Circle` (two tests).
- When subclasses of `Shape` form a hierarchy we need to carefully think about the order of the tests in method `draw`.
- Every time we define a new subclass of `Shape`, we need to modify the method `draw` (and rethink the order of the tests).

Example: Multiple Dispatch

Problems

- It is more efficient to draw instances of `Line` (just one test) than instances of `Circle` (two tests).
- When subclasses of `Shape` form a hierarchy we need to carefully think about the order of the tests in method `draw`.
- Every time we define a new subclass of `Shape`, we need to modify the method `draw` (and rethink the order of the tests).

Solution: Multiple dispatch

- Method calls are dynamically dispatched based on the runtime type of more than one of its arguments.
- Java uses single dispatch + overloading, CLOS uses multiple dispatch.

Example: Multiple Dispatch

Solution: Double Dispatch

```
abstract class Device {

    public abstract void draw(Shape s);
}

class Screen extends Device {

    public void draw(Shape s) {
        s.drawOnScreen(this);
    }
}

class Printer extends Device {

    public void draw(Shape s) {
        s.drawOnPrinter(this);
    }
}
```

Example: Multiple Dispatch

Solution: Double Dispatch

```
abstract class Shape {
    public abstract void drawOnScreen(Screen s);
    public abstract void drawOnPrinter(Printer p);
}

class Line extends Shape {
    public void drawOnScreen(Screen s) {
        System.err.println("drawing a line on screen!");
    }
    public void drawOnPrinter(Printer p) {
        System.err.println("drawing a line on printer!");
    }
}

class Circle extends Shape {
    public void drawOnScreen(Screen s) {
        System.err.println("drawing a circle on screen!");
    }
    public void drawOnPrinter(Printer p) {
        System.err.println("drawing a circle on printer!");
    }
}
```

Example: Multiple Dispatch

Solution: Double Dispatch

```
abstract class Shape {
    public abstract void drawOnScreen(Screen s);
    public abstract void drawOnPrinter(Printer p);
}

class Line extends Shape {
    public void drawOnScreen(Screen s) {
        System.err.println("drawing a line on screen!");
    }
    public void drawOnPrinter(Printer p) {
        System.err.println("drawing a line on printer!");
    }
}

class Circle extends Shape {
    public void drawOnScreen(Screen s) {
        System.err.println("drawing a circle on screen!");
    }
    public void drawOnPrinter(Printer p) {
        System.err.println("drawing a circle on printer!");
    }
}
```

Example: Multiple Dispatch

Solution: Double Dispatch + Overloading

```
abstract class Shape {
    public abstract void draw(Screen s);
    public abstract void draw(Printer p);
}

class Line extends Shape {
    public void draw(Screen s) {
        System.err.println("drawing a line on screen!");
    }
    public void draw(Printer p) {
        System.err.println("drawing a line on printer!");
    }
}

class Circle extends Shape {
    public void draw(Screen s) {
        System.err.println("drawing a circle on screen!");
    }
    public void draw(Printer p) {
        System.err.println("drawing a circle on printer!");
    }
}
```

Example: Multiple Dispatch

Solution: Double Dispatch

```
abstract class Device {

    public abstract void draw(Shape s);
}

class Screen extends Device {

    public void draw(Shape s) {
        s.drawOnScreen(this);
    }
}

class Printer extends Device {

    public void draw(Shape s) {
        s.drawOnPrinter(this);
    }
}
```

Example: Multiple Dispatch

Solution: Double Dispatch + Overloading

```
abstract class Device {

    public abstract void draw(Shape s);
}

class Screen extends Device {

    public void draw(Shape s) {
        s.draw(this);
    }
}

class Printer extends Device {

    public void draw(Shape s) {
        s.draw(this);
    }
}
```

Example: Multiple Dispatch

Problems

- Requires program restructuring.
- It is easy to create a new type of `Shape` but creating a new type of `Device` entails adding methods to all types of `Shape`.
- Each subclasse of `Device` needs its own copy of method `draw`.
- Fixed dispatch order: first, by type of `Device`, then, by type of `Shape`.
- Generalization to triple, quadruple, etc., dispatch causes a combinatorial explosion of methods.

Example: Multiple Dispatch

Problems

- Requires program restructuring.
- It is easy to create a new type of `Shape` but creating a new type of `Device` entails adding methods to all types of `Shape`.
- Each subclasse of `Device` needs its own copy of method `draw`.
- Fixed dispatch order: first, by type of `Device`, then, by type of `Shape`.
- Generalization to triple, quadruple, etc., dispatch causes a combinatorial explosion of methods.

Solution

User-defined method call mechanism.

Example: Multiple Dispatch

Dynamic Invocation

```
class Device {

    public void draw(Shape s) {
        invoke(this, "draw", s);
    }
    static Object invoke(Object receiver, String name, Object arg) {
        try {
            Method method = bestMethod(receiver.getClass(),
                                       name,
                                       arg.getClass());
            return method.invoke(receiver, arg);
        } catch (NoSuchMethodException e) {
            throw new RuntimeException(e);
        } catch (IllegalAccessException e) {
            throw new RuntimeException(e);
        } catch (InvocationTargetException e) {
            throw new RuntimeException(e);
        }
    }
    …
}
```

Example: Multiple Dispatch

Dynamic Invocation – In Java 7

```
class Device {

    public void draw(Shape s) {
        invoke(this, "draw", s);
    }
    static Object invoke(Object receiver, String name, Object arg) {
        try {
            Method method = bestMethod(receiver.getClass(),
                                       name,
                                       arg.getClass());
            return method.invoke(receiver, arg);
        } catch (NoSuchMethodException |
                 IllegalAccessException |
                 InvocationTargetException e) {
            throw new RuntimeException(e);
        }
    }
    …
}
```

Example: Multiple Dispatch

Dynamic Invocation

```
class Device {

    …

    static Method bestMethod(Class type, String name, Class argType)
        throws NoSuchMethodException {
        try {
            return type.getMethod(name, argType);
        } catch (NoSuchMethodException e) {
            if (argType == Object.class) {
                throw e;
            } else {
                return bestMethod(type,
                                  name,
                                  argType.getSuperclass());
            }
        }
    }
}
```

Example: Multiple Dispatch

Problems

- `getMethod` can only access **public** methods.
- `bestMethod` can only access public methods with a single parameter.
- We are only dealing with double dispatch.
- We are "climbing" the class hierarchy but not the interface hierarchy.
- We are not dealing with *boxing/unboxing*.
- We are not dealing with variable arity methods.

Example: Multiple Dispatch

Problems

- `getMethod` can only access **public** methods.
- `bestMethod` can only access public methods with a single parameter.
- We are only dealing with double dispatch.
- We are "climbing" the class hierarchy but not the interface hierarchy.
- We are not dealing with *boxing/unboxing*.
- We are not dealing with variable arity methods.

Solution

- More work!
- MultiJava, Maya

Example: Multiple Dispatch

What about C#?

```
class Shape {
}

class Line : Shape {
}

class Circle : Shape {
}

class Device {
    public void Draw(Shape s) {
        Console.WriteLine("draw what where?");
    }
    public void Draw(Line l) {
        Console.WriteLine("draw a line where?");
    }
    public void Draw(Circle c)  {
        Console.WriteLine("draw a circle where?");
    }
}
```

Example: Multiple Dispatch

What about C#?

```
class Screen : Device {

    public void Draw(Shape s) {
        Console.WriteLine("draw what on screen?");
    }

    public void Draw(Line l) {
        Console.WriteLine("drawing a line on screen!");
    }

    public void Draw(Circle c) {
        Console.WriteLine("drawing a circle on screen!");
    }
}
```

Example: Multiple Dispatch

What about C#?

```
class Printer : Device {

    public void Draw(Shape s) {
        Console.WriteLine("draw what on printer?");
    }

    public void Draw(Line l) {
        Console.WriteLine("drawing a line on printer!");
    }

    public void Draw(Circle c) {
        Console.WriteLine("drawing a circle on printer!");
    }
}
```

Example: Multiple Dispatch

What about C#?

```
Device[] devices = new Device[] { new Screen(), new Printer() };
Shape[] shapes = new Shape[] { new Line(), new Circle() };
foreach (Device device in devices) {
    foreach (Shape shape in shapes) {
        device.Draw(shape);
    }
}
```

Example: Multiple Dispatch

What about C#?

```
Device[] devices = new Device[] { new Screen(), new Printer() };
Shape[] shapes = new Shape[] { new Line(), new Circle() };
foreach (Device device in devices) {
    foreach (Shape shape in shapes) {
        device.Draw(shape);
    }
}
```

Output: Even less dynamic than Java!

```
draw what where?
draw what where?
draw what where?
draw what where?
```

Example: Multiple Dispatch

What about C#, using virtual and override?

```
class Shape {
}

class Line : Shape {
}

class Circle : Shape {
}

class Device {
    public virtual void Draw(Shape s) {
        Console.WriteLine("draw what where?");
    }
    public virtual void Draw(Line l) {
        Console.WriteLine("draw a line where?");
    }
    public virtual void Draw(Circle c)  {
        Console.WriteLine("draw a circle where?");
    }
}
```

Example: Multiple Dispatch

What about C#, using virtual and override?

```
class Screen : Device {

    public override void Draw(Shape s) {
        Console.WriteLine("draw what on screen?");
    }

    public override void Draw(Line l) {
        Console.WriteLine("drawing a line on screen!");
    }

    public override void Draw(Circle c) {
        Console.WriteLine("drawing a circle on screen!");
    }
}
```

Example: Multiple Dispatch

What about C#, using virtual and override?

```
class Printer : Device {

    public override void Draw(Shape s) {
        Console.WriteLine("draw what on printer?");
    }

    public override void Draw(Line l) {
        Console.WriteLine("drawing a line on printer!");
    }

    public override void Draw(Circle c) {
        Console.WriteLine("drawing a circle on printer!");
    }
}
```

Example: Multiple Dispatch

What about C#, using `virtual` and `override`?

```
Device[] devices = new Device[] { new Screen(), new Printer() };
Shape[] shapes = new Shape[] { new Line(), new Circle() };
foreach (Device device in devices) {
    foreach (Shape shape in shapes) {
        device.Draw(shape);
    }
}
```

Example: Multiple Dispatch

What about C#, using virtual and override?

```
Device[] devices = new Device[] { new Screen(), new Printer() };
Shape[] shapes = new Shape[] { new Line(), new Circle() };
foreach (Device device in devices) {
    foreach (Shape shape in shapes) {
        device.Draw(shape);
    }
}
```

Output: Same as Java!

```
draw what on screen?
draw what on screen?
draw what on printer?
draw what on printer?
```

Example: Multiple Dispatch

What about C#, using virtual, override, and dynamic?

```
Device[] devices = new Device[] { new Screen(), new Printer() };
Shape[] shapes = new Shape[] { new Line(), new Circle() };
foreach (Device device in devices) {
    foreach (Shape shape in shapes) {
        device.Draw((dynamic)shape);
    }
}
```

Example: Multiple Dispatch

What about C#, using virtual, override, and dynamic?

```
Device[] devices = new Device[] { new Screen(), new Printer() };
Shape[] shapes = new Shape[] { new Line(), new Circle() };
foreach (Device device in devices) {
    foreach (Shape shape in shapes) {
        device.Draw((dynamic)shape);
    }
}
```

Output: Finally!

```
drawing a line on screen!
drawing a circle on screen!
drawing a line on printer!
drawing a circle on printer!
```

Example: Multiple Dispatch

What about C#, using `virtual`, `override`, and `dynamic`?

```
Device[] devices = new Device[] { new Screen(), new Printer() };
Shape[] shapes = new Shape[] { new Line(), new Circle() };
foreach (Device device in devices) {
    foreach (Shape shape in shapes) {
        device.Draw((dynamic)shape);
    }
}
```

Output: Finally!

```
drawing a line on screen!
drawing a circle on screen!
drawing a line on printer!
drawing a circle on printer!
```

It's all about tradeoffs

`dynamic` is five times slower than cascaded `if`s or Double Dispatch

Emacs Lisp

Example

```
> (defun foo (x)                                    ;function definition
    (+ x 3))
```

Emacs Lisp

Example

```
> (defun foo (x)                                    ;function definition
    (+ x 3))
foo
```

Emacs Lisp

Example

```
> (defun foo (x)                                    ;function definition
    (+ x 3))
foo
> (foo 4)                                           ;function call
```

Emacs Lisp

Example

```
> (defun foo (x)                                    ;function definition
    (+ x 3))
foo
> (foo 4)                                           ;function call
7
```

Emacs Lisp

Example

```
> (defun foo (x)                                ;function definition
    (+ x 3))
foo
> (foo 4)                                       ;function call
7
> (symbol-function 'foo)                        ;the foo function
```

Emacs Lisp

Example

```
> (defun foo (x)                                    ;function definition
    (+ x 3))
foo
> (foo 4)                                           ;function call
7
> (symbol-function 'foo)                            ;the foo function
(lambda (x)
  (+ x 3))
```

Emacs Lisp

Example

```
> (defun foo (x)                                    ;function definition
    (+ x 3))
foo
> (foo 4)                                           ;function call
7
> (symbol-function 'foo)                            ;the foo function
(lambda (x)
  (+ x 3))
> (cadr (symbol-function 'foo))                     ;function parameters
```

Emacs Lisp

Example

```
> (defun foo (x)                          ;function definition
    (+ x 3))
foo
> (foo 4)                                 ;function call
7
> (symbol-function 'foo)                  ;the foo function
(lambda (x)
  (+ x 3))
> (cadr (symbol-function 'foo))           ;function parameters
(x)
```

Emacs Lisp

Example

```
> (defun foo (x)                          ;function definition
    (+ x 3))
foo
> (foo 4)                                 ;function call
7
> (symbol-function 'foo)                  ;the foo function
(lambda (x)
  (+ x 3))
> (cadr (symbol-function 'foo))           ;function parameters
(x)
> (caaddr (symbol-function 'foo))         ;called function
```

Emacs Lisp

Example

```
> (defun foo (x)                          ;function definition
    (+ x 3))
foo
> (foo 4)                                 ;function call
7
> (symbol-function 'foo)                  ;the foo function
(lambda (x)
  (+ x 3))
> (cadr (symbol-function 'foo))           ;function parameters
(x)
> (caaddr (symbol-function 'foo))         ;called function
+
```

Emacs Lisp

Example

```
> (defun foo (x)                                    ;function definition
    (+ x 3))
foo
> (foo 4)                                           ;function call
7
> (symbol-function 'foo)                            ;the foo function
(lambda (x)
  (+ x 3))
> (cadr (symbol-function 'foo))                     ;function parameters
(x)
> (caaddr (symbol-function 'foo))                   ;called function
+
> (setcar (caddr (symbol-function 'foo)) '-)        ;let's change it
```

Emacs Lisp

Example

```
> (defun foo (x)                                    ;function definition
    (+ x 3))
foo
> (foo 4)                                           ;function call
7
> (symbol-function 'foo)                            ;the foo function
(lambda (x)
  (+ x 3))
> (cadr (symbol-function 'foo))                     ;function parameters
(x)
> (caaddr (symbol-function 'foo))                   ;called function
+
> (setcar (caddr (symbol-function 'foo)) '-) ;let's change it
-
```

Emacs Lisp

Example

```
> (defun foo (x)                                    ;function definition
    (+ x 3))
foo
> (foo 4)                                           ;function call
7
> (symbol-function 'foo)                            ;the foo function
(lambda (x)
  (+ x 3))
> (cadr (symbol-function 'foo))                     ;function parameters
(x)
> (caaddr (symbol-function 'foo))                   ;called function
+
> (setcar (caddr (symbol-function 'foo)) '-)        ;let's change it
-
> (foo 4)                                           ;function call
```

Emacs Lisp

Example

```
> (defun foo (x)                                    ;function definition
    (+ x 3))
foo
> (foo 4)                                           ;function call
7
> (symbol-function 'foo)                            ;the foo function
(lambda (x)
  (+ x 3))
> (cadr (symbol-function 'foo))                     ;function parameters
(x)
> (caaddr (symbol-function 'foo))                   ;called function
+
> (setcar (caddr (symbol-function 'foo)) '-)        ;let's change it
-
> (foo 4)                                           ;function call
1
```

Emacs Lisp

Example

```
> (defun foo (x)                                    ;function definition
    (+ x 3))
foo
> (foo 4)                                           ;function call
7
> (symbol-function 'foo)                            ;the foo function
(lambda (x)
  (+ x 3))
> (cadr (symbol-function 'foo))                     ;function parameters
(x)
> (caaddr (symbol-function 'foo))                   ;called function
+
> (setcar (caddr (symbol-function 'foo)) '-)        ;let's change it
-
> (foo 4)                                           ;function call
1
> (symbol-function 'foo)                            ;the 'new' foo function
```

Emacs Lisp

Example

```
> (defun foo (x)                                    ;function definition
    (+ x 3))
foo
> (foo 4)                                           ;function call
7
> (symbol-function 'foo)                            ;the foo function
(lambda (x)
  (+ x 3))
> (cadr (symbol-function 'foo))                     ;function parameters
(x)
> (caaddr (symbol-function 'foo))                   ;called function
+
> (setcar (caddr (symbol-function 'foo)) '-)        ;let's change it
-
> (foo 4)                                           ;function call
1
> (symbol-function 'foo)                            ;the 'new' foo function
(lambda (x)
  (- x 3))
```

Emacs Lisp

Definition (Trace)

- A form of behavioral introspection.
- On each function call, print the arguments and result.
- To simplify, we will omit the result.

Trace

```
> (defun fact (n)
    (if (= n 0)
      1
      (* n (fact (- n 1)))))
```

Emacs Lisp

Definition (Trace)

- A form of behavioral introspection.
- On each function call, print the arguments and result.
- To simplify, we will omit the result.

Trace

```
> (defun fact (n)
    (if (= n 0)
      1
      (* n (fact (- n 1)))))
fact
```

Emacs Lisp

Definition (Trace)

- A form of behavioral introspection.
- On each function call, print the arguments and result.
- To simplify, we will omit the result.

Trace

```
> (defun fact (n)
    (if (= n 0)
      1
      (* n (fact (- n 1)))))
fact
> (fact 4)
```

Emacs Lisp

Definition (Trace)

- A form of behavioral introspection.
- On each function call, print the arguments and result.
- To simplify, we will omit the result.

Trace

```
> (defun fact (n)
    (if (= n 0)
      1
      (* n (fact (- n 1)))))
fact
> (fact 4)
24
```

Emacs Lisp

Definition (Trace)

- A form of behavioral introspection.
- On each function call, print the arguments and result.
- To simplify, we will omit the result.

Trace

```
> (defun fact (n)
    (if (= n 0)
      1
      (* n (fact (- n 1)))))
fact
> (fact 4)
24
> (trace 'fact)
```

Emacs Lisp

Definition (Trace)

- A form of behavioral introspection.
- On each function call, print the arguments and result.
- To simplify, we will omit the result.

Trace

```
> (defun fact (n)
    (if (= n 0)
      1
      (* n (fact (- n 1)))))
fact
> (fact 4)
24
> (trace 'fact)
fact
```

Emacs Lisp

Definition (Trace)

- A form of behavioral introspection.
- On each function call, print the arguments and result.
- To simplify, we will omit the result.

Trace

```
> (defun fact (n)
    (if (= n 0)
      1
      (* n (fact (- n 1)))))
fact
> (fact 4)
24
> (trace 'fact)
fact
> (fact 4)
```

Emacs Lisp

Definition (Trace)

- A form of behavioral introspection.
- On each function call, print the arguments and result.
- To simplify, we will omit the result.

Trace

```
> (defun fact (n)
    (if (= n 0)
      1
      (* n (fact (- n 1)))))
fact
> (fact 4)
24
> (trace 'fact)
fact
> (fact 4)
(fact 4)->(fact 3)->(fact 2)->(fact 1)->(fact 0)->24
```

Emacs Lisp

Implementations

- Detection of the call and tracing done by the interpreter.
- Tracing code injected in the function.
- Function redefined to include tracing code.

Emacs Lisp

Implementations

- Detection of the call and tracing done by the interpreter.
- Tracing code injected in the function.
- Function redefined to include tracing code.

Trace by injection

```
(defun trace (name)
```

Emacs Lisp

Implementations

- Detection of the call and tracing done by the interpreter.
- Tracing code injected in the function.
- Function redefined to include tracing code.

Trace by injection

```
(defun trace (name)
  (let ((old-lambda (symbol-function name)))
```

Emacs Lisp

Implementations

- Detection of the call and tracing done by the interpreter.
- Tracing code injected in the function.
- Function redefined to include tracing code.

Trace by injection

```
(defun trace (name)
  (let ((old-lambda (symbol-function name)))
    (let ((new-lambda (traced-lambda old-lambda)))
```

Emacs Lisp

Implementations

- Detection of the call and tracing done by the interpreter.
- Tracing code injected in the function.
- Function redefined to include tracing code.

Trace by injection

```
(defun trace (name)
  (let ((old-lambda (symbol-function name)))
    (let ((new-lambda (traced-lambda old-lambda)))
      (setcdr old-lambda (cdr new-lambda))
```

Emacs Lisp

Implementations

- Detection of the call and tracing done by the interpreter.
- Tracing code injected in the function.
- Function redefined to include tracing code.

Trace by injection

```
(defun trace (name)
  (let ((old-lambda (symbol-function name)))
    (let ((new-lambda (traced-lambda old-lambda)))
      (setcdr old-lambda (cdr new-lambda))
      name)))
```

Emacs Lisp

Original

```
(lambda (n)
  (if (= n 0)
    1
    (* n (fact (- n 1)))))
```

Emacs Lisp

Original

```
(lambda (n)
  (if (= n 0)
    1
    (* n (fact (- n 1)))))
```

Traced

```
(lambda (n)
  (princ (list 'fact n))
  (princ '->)
  (if (= n 0)
    1
    (* n (fact (- n 1)))))
```

Emacs Lisp

Original

```
(lambda (n)
  (if (= n 0)
    1
    (* n (fact (- n 1)))))
```

Traced

```
(lambda (n)
  (princ (list 'fact n))
  (princ '->)
  (if (= n 0)
    1
    (* n (fact (- n 1)))))
```

Tracing

```
(defun traced-lambda (name lambda-form)
  (cons 'lambda
        (cons (cadr lambda-form)
              (cons (list 'princ
                          (cons 'list
                                (cons (list 'quote name)
                                      (cadr lambda-form))))
                    (cons '(princ '->)
                          (cddr lambda-form))))))
```

Backquote in Lisp (1978)

Definition

- To simplify meta-programming:
- `` `expr `` → returns *expr* unevaluated *except* for subexpressions preceded by comma.
- `,subexpr` → evaluates *subexpr* and *inserts* the value in the containing expression.
- `,@subexpr` → evaluates *subexpr* and *splices* the value (a list) in the containing expression.

Example

```
> '(5 (list (+ 1 3) 3) 2 1)
(5 (list (+ 1 3) 3) 2 1)
> `(5 ,(list (+ 1 3) 3) 2 1)
(5 (4 3) 2 1)
> `(5 ,@(list (+ 1 3) 3) 2 1)
(5 4 3 2 1)
```

Backquote in Julia (2012)

Definition

- To simplify meta-programming:
- `:(`*expr*`)` → returns *expr* unevaluated *except* for subexpressions preceded by $.
- `$(`*subexpr*`)` → evaluates *subexpr* and *inserts* the value in the containing expression.
- `$(`*subexpr*`...)` → evaluates *subexpr* and *splices* the value (a list) in the containing expression.

Example

```
> :((5, tuple(1 + 3, 3), 2, 1))
:((5, tuple(1 + 3, 3), 2, 1))
> :((5, $(tuple(1 + 3, 3)), 2, 1))
:((5, (4, 3), 2, 1))
> :((5, $(tuple(1 + 3, 3)...), 2, 1))
:((5, 4, 3, 2, 1))
```

Backquote

From

```
(lambda (n)
  (if ...))
```

To

```
(lambda (n)
  (princ (list 'fact n))
  (princ '->)
  (if ...))
```

Backquote

From

```
(lambda (n)
  (if ...))
```

To

```
(lambda (n)
  (princ (list 'fact n))
  (princ '->)
  (if ...))
```

Without *backquote*

```
(cons 'lambda
      (cons (cadr lambda-form)
            (cons (list 'princ
                        (cons 'list
                              (cons (list 'quote name)
                                    (cadr lambda-form))))
                  (cons '(princ '->)
                        (cddr lambda-form)))))
```

Backquote

From

```
(lambda (n)
  (if ...))
```

To

```
(lambda (n)
  (princ (list 'fact n))
  (princ '->)
  (if ...))
```

With *backquote*

```
`(lambda ,(cadr lambda-form)
   (princ (list ',name ,@(cadr lambda-form)))
   (princ '->)
   ,@(cddr lambda-form))
```

Backquote

Definition

- If the backquote syntax is nested, the innermost backquoted form should be expanded first. This means that if several commas occur in a row, the leftmost one belongs to the innermost backquote.
- ``` ``*expr* ``` → returns ``` `*expr* ``` unevaluated *except* for subexpressions preceded by comma.
- ```,*subexpr*``` → evaluates *subexpr when* the inner *backquote* is evaluated and *inserts* the value in the containing expression.
- ```,,*subexpr*``` → evaluates *subexpr* twice and *inserts* the value in the containing expression.
- ```,',*subexpr*``` → evaluates *subexpr when* the outer *backquote* is evaluated and *inserts* the value in the containing expression.

Self Modification

Exponential Grow

```
> (defun fib (n)
    (if (< n 2)
      n
      (+ (fib (- n 1))
         (fib (- n 2)))))
```

Self Modification

Exponential Grow

```
> (defun fib (n)
    (if (< n 2)
      n
      (+ (fib (- n 1))
         (fib (- n 2)))))
fib
```

Self Modification

Exponential Grow

```
> (defun fib (n)
    (if (< n 2)
      n
      (+ (fib (- n 1))
         (fib (- n 2)))))
fib
> (fib 10)
```

Self Modification

Exponential Grow

```
> (defun fib (n)
    (if (< n 2)
      n
      (+ (fib (- n 1))
         (fib (- n 2)))))
fib
> (fib 10)
..55
```

Self Modification

Exponential Grow

```
> (defun fib (n)
    (if (< n 2)
      n
      (+ (fib (- n 1))
         (fib (- n 2)))))
fib
> (fib 10)
..55
> (fib 20)
```

Self Modification

Exponential Grow

```
> (defun fib (n)
    (if (< n 2)
      n
      (+ (fib (- n 1))
         (fib (- n 2)))))
fib
> (fib 10)
..55
> (fib 20)
........6765
```

Self Modification

Exponential Grow

```
> (defun fib (n)
    (if (< n 2)
      n
      (+ (fib (- n 1))
         (fib (- n 2)))))
fib
> (fib 10)
..55
> (fib 20)
........6765
> (fib 30)
```

Self Modification

Exponential Grow

```
> (defun fib (n)
    (if (< n 2)
      n
      (+ (fib (- n 1))
         (fib (- n 2)))))
fib
> (fib 10)
..55
> (fib 20)
........6765
> (fib 30)
................................832040
```

Self Modification

Exponential Grow

```
> (defun fib (n)
    (if (< n 2)
      n
      (+ (fib (- n 1))
         (fib (- n 2)))))
fib
> (fib 10)
..55
> (fib 20)
........6765
> (fib 30)
..............................832040
> (fib 40)
```

Self Modification

Exponential Grow

```
> (defun fib (n)
    (if (< n 2)
      n
      (+ (fib (- n 1))
         (fib (- n 2)))))
fib
> (fib 10)
..55
> (fib 20)
........6765
> (fib 30)
..............................832040
> (fib 40)
........................................................................
```

Self Modification

Exponential Grow

```
> (defun fib (n)
    (if (< n 2)
      n
      (+ (fib (- n 1))
         (fib (- n 2)))))
fib
> (fib 10)
..55
> (fib 20)
........6765
> (fib 30)
................................832040
> (fib 40)
........................................................................
> (memoize (symbol-function 'fib))
```

Self Modification

Exponential Grow

```
> (defun fib (n)
    (if (< n 2)
      n
      (+ (fib (- n 1))
         (fib (- n 2)))))
fib
> (fib 10)
..55
> (fib 20)
........6765
> (fib 30)
................................832040
> (fib 40)
........................................................................
> (memoize (symbol-function 'fib))
(...)
```

Self Modification

Exponential Grow

```
> (defun fib (n)
    (if (< n 2)
      n
      (+ (fib (- n 1))
         (fib (- n 2)))))
fib
> (fib 10)
..55
> (fib 20)
........6765
> (fib 30)
..............................832040
> (fib 40)
........................................................................
> (memoize (symbol-function 'fib))
(...)
> (fib 10)
```

Self Modification

Exponential Grow

```
> (defun fib (n)
    (if (< n 2)
      n
      (+ (fib (- n 1))
         (fib (- n 2)))))
fib
> (fib 10)
..55
> (fib 20)
........6765
> (fib 30)
................................832040
> (fib 40)
........................................................................
> (memoize (symbol-function 'fib))
(...)
> (fib 10)
55
```

Self Modification

Exponential Grow

```
> (defun fib (n)
    (if (< n 2)
      n
      (+ (fib (- n 1))
         (fib (- n 2)))))
fib
> (fib 10)
..55
> (fib 20)
........6765
> (fib 30)
..............................832040
> (fib 40)
........................................................................
> (memoize (symbol-function 'fib))
(...)
> (fib 10)
55
> (fib 20)
```

Self Modification

Exponential Grow

```
> (defun fib (n)
    (if (< n 2)
      n
      (+ (fib (- n 1))
         (fib (- n 2)))))
fib
> (fib 10)
..55
> (fib 20)
........6765
> (fib 30)
..............................832040
> (fib 40)
........................................................................
> (memoize (symbol-function 'fib))
(...)
> (fib 10)
55
> (fib 20)
6765
```

Self Modification

Exponential Grow

```
> (defun fib (n)
    (if (< n 2)
      n
      (+ (fib (- n 1))
         (fib (- n 2)))))
fib
> (fib 10)
..55
> (fib 20)
........6765
> (fib 30)
................................832040
> (fib 40)
........................................................................
> (memoize (symbol-function 'fib))
(...)
> (fib 10)
55
> (fib 20)
6765
> (fib 40)
```

Self Modification

Exponential Grow

```
> (defun fib (n)
    (if (< n 2)
      n
      (+ (fib (- n 1))
         (fib (- n 2)))))
fib
> (fib 10)
..55
> (fib 20)
........6765
> (fib 30)
................................832040
> (fib 40)
........................................................................
> (memoize (symbol-function 'fib))
(...)
> (fib 10)
55
> (fib 20)
6765
> (fib 40)
102334155
```

Self Modification

Memoization

```
(defun memoize (lambda-form)
  (setcar (cddr lambda-form)
          `(let ((result ,(caddr lambda-form)))
             (setcar (cddr ',lambda-form)
                     `(if (eql ,',(caadr lambda-form)
                               ',,(caadr lambda-form))
                          ',result
                          ,(caddr ',lambda-form)))
             result)))
```

Evolution of Function `fib`

After definition

```
(lambda (n)
  (if (< n 2)
    n
    (+ (fib (- n 1))
       (fib (- n 2)))))
```

Evolution of Function `fib`

After *memoization*

```
(lambda (n)
  (let ((result
         (if (< n 2)
           n
           (+ (fib (- n 1))
              (fib (- n 2))))))
    (setcar (cddr '(lambda (n) …))
            (list 'if
                  (list 'eql 'n (list 'quote n))
                  (list 'quote result)
                  (caddr '(lambda (n) …))))
    result))
```

Evolution of Function `fib`

After (fib 0)

```
(lambda (n)
  (if (eql n '0)
    '0
    (let ((result
           (if (< n 2)
             n
             (+ (fib (- n 1))
                (fib (- n 2))))))
      (setcar (cddr '(lambda (n) ...))
              (list 'if
                    (list 'eql 'n (list 'quote n))
                    (list 'quote result)
                    (caddr '(lambda (n) ...))))
      result)))
```

Evolution of Function fib

After (fib 1)

```
(lambda (n)
  (if (eql n '1)
    '1
    (if (eql n '0)
      '0
      (let ((result
              (if (< n 2)
                n
                (+ (fib (- n 1))
                   (fib (- n 2))))))
        (setcar (cddr '(lambda (n) …))
                (list 'if
                      (list 'eql 'n (list 'quote n))
                      (list 'quote result)
                      (caddr '(lambda (n) …))))
        result))))
```

Evolution of Function fib

After (fib 40)

```
(lambda (n)
  (if (eql n '40)
    '102334155
    (if (eql n '39)
      '63245986
           …
                (if (eql n '2)
                  '1
                  (if (eql n '1)
                    '1
                    (if (eql n '0)
                      '0
                      (let ((result
                              (if (< n 2)
                                n
                                (+ (fib (- n 1))
                                   (fib (- n 2))))))
                        (setcar (cddr '(lambda (n) …))
                                (list 'if
                                      (list 'eql 'n (list 'quote n))
                                      (list 'quote result)
                                      (caddr '(lambda (n) …))))
                        result)))))))))))))))))))))))))))))))))))))))))))
```

Languages with Reflective Architecture

Power

- Unrestricted intercession provides great power:
  - We can write programs that dynamically modify other programs.
  - We can write programs that dynamically modify themselves.

Responsibility

- With great power comes great responsibility:
  - What is the semantics of a program that modifies itself?
  - How to debug a program whose source code was self-modified?
  - How to compile a program that does not have a stable form?
- Modern Lisps restrict intercession to provide better compilation.
- The power is still there (but harder to use “by accident”).

Intercession with Javassist

Javassist

- Load-time intercession for Java.
- Does not modify the runtime or compiler.
- Modifies class bytecodes at class load-time.

Operation Sequence

Reification Creating a `CtClass` (Compile time Class) object representing the bytecodes of a class.

Modification Introspecting and altering the class definition.

Translation Computing the bytecodes of the modified class.

Reflection Loading the obtained bytecodes into the JVM or rewriting them to class files

Intercession with Javassist

Fibonacci

```
public class Fib {
    public static Long fib (Long n) {
        if (n < 2) {
            return n;
        } else {
            return fib(n - 1) + fib(n - 2);
        }
    }
    public static void main(String[] args) {
        System.out.println(fib(Long.parseLong(args[0])));
    }
}
```

Intercession with Javassist

Fibonacci

```
public class Fib {
    public static Long fib (Long n) {
        if (n < 2) {
            return n;
        } else {
            return fib(n - 1) + fib(n - 2);
        }
    }
    public static void main(String[] args) {
        System.out.println(fib(Long.parseLong(args[0])));
    }
}

$ java Fib 10
```

Intercession with Javassist

Fibonacci

```
public class Fib {
    public static Long fib (Long n) {
        if (n < 2) {
            return n;
        } else {
            return fib(n - 1) + fib(n - 2);
        }
    }
    public static void main(String[] args) {
        System.out.println(fib(Long.parseLong(args[0])));
    }
}

$ java Fib 10
..55
```

Intercession with Javassist

Fibonacci

```
public class Fib {
    public static Long fib (Long n) {
        if (n < 2) {
            return n;
        } else {
            return fib(n - 1) + fib(n - 2);
        }
    }
    public static void main(String[] args) {
        System.out.println(fib(Long.parseLong(args[0])));
    }
}

$ java Fib 10
..55
$ java Fib 20
```

Intercession with Javassist

Fibonacci

```
public class Fib {
    public static Long fib (Long n) {
        if (n < 2) {
            return n;
        } else {
            return fib(n - 1) + fib(n - 2);
        }
    }
    public static void main(String[] args) {
        System.out.println(fib(Long.parseLong(args[0])));
    }
}

$ java Fib 10
..55
$ java Fib 20
.........6765
```

Intercession with Javassist

Fibonacci

```
public class Fib {
    public static Long fib (Long n) {
        if (n < 2) {
            return n;
        } else {
            return fib(n - 1) + fib(n - 2);
        }
    }
    public static void main(String[] args) {
        System.out.println(fib(Long.parseLong(args[0])));
    }
}

$ java Fib 10
..55
$ java Fib 20
........6765
$ java Fib 30
```

Intercession with Javassist

Fibonacci

```
public class Fib {
    public static Long fib (Long n) {
        if (n < 2) {
            return n;
        } else {
            return fib(n - 1) + fib(n - 2);
        }
    }
    public static void main(String[] args) {
        System.out.println(fib(Long.parseLong(args[0])));
    }
}

$ java Fib 10
..55
$ java Fib 20
........6765
$ java Fib 30
..............................832040
```

Intercession with Javassist

Fibonacci

```
public class Fib {
    public static Long fib (Long n) {
        if (n < 2) {
            return n;
        } else {
            return fib(n - 1) + fib(n - 2);
        }
    }
    public static void main(String[] args) {
        System.out.println(fib(Long.parseLong(args[0])));
    }
}

$ java Fib 10
..55
$ java Fib 20
........6765
$ java Fib 30
..............................832040
$ java Fib 40
```

Intercession with Javassist

Fibonacci

```
public class Fib {
    public static Long fib (Long n) {
        if (n < 2) {
            return n;
        } else {
            return fib(n - 1) + fib(n - 2);
        }
    }
    public static void main(String[] args) {
        System.out.println(fib(Long.parseLong(args[0])));
    }
}

$ java Fib 10
..55
$ java Fib 20
........6765
$ java Fib 30
..............................832040
$ java Fib 40
........................................................................
```

Evolution of the `fib` method

Pre-*Memoization*

```
public class Fib {
    public static Long fib (Long n) {
        if (n < 2) {
            return n;
        } else {
            return fib(n - 1) + fib(n - 2);
        }
    }
}
```

Evolution of the `fib` method

Post-*Memoization*

```
public class Fib {
    public static Long fib$original (Long n) {
        if (n < 2) {
            return n;
        } else {
            return fib(n - 1) + fib(n - 2);
        }
    }

    static Hashtable cachedResults = new Hashtable();

    public static Long fib (Long n) {
        Object result = cachedResults.get(n);
        if (result == null) {
            result = fib$original(n);
            cachedResults.put(n, result);
        }
        return (Long)result;
    }
}
```

Intercession with Javassist

Memoization

```
import javassist.*;
import java.io.*;

public class Memoize {

  public static void main(String[] args)
    throws NotFoundException, CannotCompileException, IOException {
    if (args.length != 2) {
        System.err.println("Usage: java Memoize <class> <method>");
        System.exit(1);
    } else {
        ClassPool pool = ClassPool.getDefault();
        CtClass ctClass = pool.get(args[0]);
        memoize(ctClass, ctClass.getDeclaredMethod(args[1]));
        ctClass.writeFile();
    }
  }

  static void memoize(CtClass ctClass, CtMethod ctMethod) { … }
}
```

Intercession with Javassist

Memoization

```
static void memoize(CtClass ctClass, CtMethod ctMethod)
    throws NotFoundException, CannotCompileException {
    CtField ctField =
        CtField.make("static java.util.Hashtable cachedResults = " +
                     "    new java.util.Hashtable();",
                     ctClass);
    ctClass.addField(ctField);
    String name = ctMethod.getName();
    ctMethod.setName(name + "$original");
    ctMethod = CtNewMethod.copy(ctMethod, name, ctClass, null);
    ctMethod.setBody("{" +
                     "  Object result = cachedResults.get($1);" +
                     "  if (result == null) {" +
                     "    result = " + name + "$original($$);" +
                     "    cachedResults.put($1, result);" +
                     "  }" +
                     "  return ($r)result;" +
                     "}");
    ctClass.addMethod(ctMethod);
}
```

Intercession with Javassist

Example

Results

```
$ time java Fib 40
102334155

real    0m13.784s
user    0m12.521s
sys     0m0.056s

$ java -classpath ".:javassist.jar" Memoize Fib fib

$ time java Fib 40
102334155

real    0m0.093s
user    0m0.036s
sys     0m0.012s
```

Meta-Variables in Javassist

Definition

- Injected code is described using a *template* (in a `String`).
- The *template* represents either a *statement* (when it ends with `;`) or a block (when it is contained within `{}`).
- The *template* might contain *meta-variables*:
- `$0` is the receptor (nonexistent for static methods).
- `$1`,`$2`,`$3`, etc, are the method parameters (parameter names are not accessible). It is possible to read or write them.
- `$$` represent the parameters of the method, i.e., `$1,$2,...`.
- `$r` is the method return type (useful for *casts*).
- `$w` is the *wrapper* type (useful for *casts* of primitive types).

Intercession with Javassist

Problems

- *Templates* based on `String` concatenation are error prone.
- Javassist's compiler is fragile and does not deal with all features of Java (e.g., inner classes, anonymous classes, enums, and generics).
- It is possible to violate Java semantics (wrong return type, lack of *type casts*, wrong dispatch, etc).
- JVM *byte-code* verifier might catch some violations (at *run-time*) but it is not guaranteed.
- Manual recompilation of *class files* is not practical.

Intercession with Javassist

Problems

- *Templates* based on `String` concatenation are error prone.
- Javassist's compiler is fragile and does not deal with all features of Java (e.g., inner classes, anonymous classes, enums, and generics).
- It is possible to violate Java semantics (wrong return type, lack of *type casts*, wrong dispatch, etc).
- JVM *byte-code* verifier might catch some violations (at *run-time*) but it is not guaranteed.
- Manual recompilation of *class files* is not practical.

Solution (for the last problem)

Intercession at *load time*.

Intercession with Javassist at *Compile Time*

Writes on disk the modified class

```
import javassist.*;
import java.io.*;


public class Memoize {

    public static void main(String[] args) throws … {
        if (args.length < 2) {
            …
        } else {
            ClassPool pool = ClassPool.getDefault();
            CtClass ctClass = pool.get(args[0]);
            memoize(ctClass, ctClass.getDeclaredMethod(args[1]));
            ctClass.writeFile();
        }
    }
}
```

Intercession with Javassist at *Load Time*

Transfers control to the modified class

```
import javassist.*;
import java.io.*;


public class MemoizeAndRun extends Memoize {

    public static void main(String[] args) throws … {
        if (args.length < 2) {
            …
        } else {
            ClassPool pool = ClassPool.getDefault();
            CtClass ctClass = pool.get(args[0]);
            memoize(ctClass, ctClass.getDeclaredMethod(args[1]));









        }
    }
}
```

Intercession with Javassist at *Load Time*

Transfers control to the modified class

```
import javassist.*;
import java.io.*;


public class MemoizeAndRun extends Memoize {

    public static void main(String[] args) throws … {
        if (args.length < 2) {
            …
        } else {
            ClassPool pool = ClassPool.getDefault();
            CtClass ctClass = pool.get(args[0]);
            memoize(ctClass, ctClass.getDeclaredMethod(args[1]));
            Class<?> rtClass = ctClass.toClass();




        }
    }
}
```

Intercession with Javassist at *Load Time*

Transfers control to the modified class

```
import javassist.*;
import java.io.*;
import java.lang.reflect.*;

public class MemoizeAndRun extends Memoize {

    public static void main(String[] args) throws … {
        if (args.length < 2) {
            …
        } else {
            ClassPool pool = ClassPool.getDefault();
            CtClass ctClass = pool.get(args[0]);
            memoize(ctClass, ctClass.getDeclaredMethod(args[1]));
            Class<?> rtClass = ctClass.toClass();
            Method main = rtClass.getMethod("main", args.getClass());



        }
    }
}
```

Intercession with Javassist at *Load Time*

Transfers control to the modified class

```
import javassist.*;
import java.io.*;
import java.lang.reflect.*;

public class MemoizeAndRun extends Memoize {

    public static void main(String[] args) throws … {
        if (args.length < 2) {
            …
        } else {
            ClassPool pool = ClassPool.getDefault();
            CtClass ctClass = pool.get(args[0]);
            memoize(ctClass, ctClass.getDeclaredMethod(args[1]));
            Class<?> rtClass = ctClass.toClass();
            Method main = rtClass.getMethod("main", args.getClass());
            String[] restArgs = new String[args.length - 2];


        }
    }
}
```

Intercession with Javassist at *Load Time*

Transfers control to the modified class

```
import javassist.*;
import java.io.*;
import java.lang.reflect.*;

public class MemoizeAndRun extends Memoize {

    public static void main(String[] args) throws … {
        if (args.length < 2) {
            …
        } else {
            ClassPool pool = ClassPool.getDefault();
            CtClass ctClass = pool.get(args[0]);
            memoize(ctClass, ctClass.getDeclaredMethod(args[1]));
            Class<?> rtClass = ctClass.toClass();
            Method main = rtClass.getMethod("main", args.getClass());
            String[] restArgs = new String[args.length - 2];
            System.arraycopy(args, 2, restArgs, 0, restArgs.length);

        }
    }
}
```

Intercession with Javassist at *Load Time*

Transfers control to the modified class

```
import javassist.*;
import java.io.*;
import java.lang.reflect.*;

public class MemoizeAndRun extends Memoize {

    public static void main(String[] args) throws … {
        if (args.length < 2) {
            …
        } else {
            ClassPool pool = ClassPool.getDefault();
            CtClass ctClass = pool.get(args[0]);
            memoize(ctClass, ctClass.getDeclaredMethod(args[1]));
            Class<?> rtClass = ctClass.toClass();
            Method main = rtClass.getMethod("main", args.getClass());
            String[] restArgs = new String[args.length - 2];
            System.arraycopy(args, 2, restArgs, 0, restArgs.length);
            main.invoke(null, new Object[] { restArgs });
        }
    }
}
```

Intercession with Javassist at *Load Time*

Example

Results

```
$ time java Fib 40
102334155

real    0m0.093s
user    0m0.036s
sys     0m0.012s
$ javac Fib.java
$ time java Fib 40
102334155

real    0m13.501s
user    0m12.509s
sys     0m0.032s
$ time java -classpath ".:javassist.jar" MemoizeAndRun Fib fib 40
102334155

real    0m0.381s
user    0m0.268s
sys     0m0.032s
```

Intercession with Javassist at *Load Time*

Problems

- The *memoization* program is difficult to use.
- It is hard to *memoize* several methods at the same time.

Intercession with Javassist at *Load Time*

Problems

- The *memoization* program is difficult to use.
- It is hard to *memoize* several methods at the same time.

Solution

```
public class Fib {

    @Memoized
    public static Long fib (Long n) {
        if (n < 2) {
            return n;
        } else {
            return fib(n - 1) + fib(n - 2);
        }
    }

    …
}
```

Annotations

Definition

- Information about a program.
- Not part of the program.
- Do not affect program semantics.
- Allow annotations over packages, classes, methods, fields, parameters, and variables.
- Might have parameters.
- Can survive the compilation process (unlike Javadoc).
- Can be processed at *compile time*, *load time* or *run time*.
- The annotation definition specifies (via meta-markers) its *target* and retention policy.

Three Annotation Types - Multi Value

Annotation Definition

```
import java.lang.annotation.*;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Foo {
    String bar();
    long baz();
}
```

Annotation Use

```
public class C1 {


    @Foo(bar="Hello World", baz=100)
    public void m1(int a, long b) {
        …
    }

}
```

Three Annotation Types - Single Value

Annotation Definition

```
import java.lang.annotation.*;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Foo {
    String bar();
}
```

Annotation Use

```
public class C1 {


    @Foo(bar="Hello World")
    public void m1(int a, long b) {
        …
    }

}
```

Three Annotation Types - Single Value

Annotation Definition

```
import java.lang.annotation.*;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Foo {
    String value();
}
```

Annotation Use

```
public class C1 {


    @Foo("Hello World")
    public void m1(int a, long b) {
        …
    }

}
```

Three Annotation Types - Marker

Annotation Definition

```
import java.lang.annotation.*;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Foo {
}
```

Annotation Use

```
public class C1 {


    @Foo
    public void m1(int a, long b) {
        …
    }

}
```

Annotations

Syntax

- Interfaces preceded by `@`.
- Methods with empty parameter list and empty throws clause.
- Return type of the methods: primitive types, `String`, `Class`, *enums*, and *arrays* of the these types.

Annotations

Syntax

- Interfaces preceded by `@`.
- Methods with empty parameter list and empty throws clause.
- Return type of the methods: primitive types, `String`, `Class`, *enums*, and *arrays* of the these types.

Pre-defined annotations

- `@Override`
- `@Deprecated`
- `@Suppresswarnings({` $warning_0$, ..., $warning_n$ `})`

Annotations

Pre-defined annotations - Meta-annotations - @Target

- `ElementType.TYPE`
- `ElementType.FIELD`
- `ElementType.METHOD`
- `ElementType.PARAMETER`
- `ElementType.CONSTRUCTOR`
- `ElementType.LOCAL_VARIABLE`
- `ElementType.ANNOTATION_TYPE`

Pre-defined annotations - Meta-annotations - @Retention

- `RetentionPolicy.SOURCE`
- `RetentionPolicy.CLASS`
- `RetentionPolicy.RUNTIME`

Intercession with Javassist at *Load Time*

Current Version - Without Annotated Methods

```
import javassist.*;
import java.io.*;
import java.lang.reflect.*;

public class MemoizeAndRun extends Memoize {

    public static void main(String[] args) throws … {
        if (args.length < 2) {
            …
        } else {
            ClassPool pool = ClassPool.getDefault();
            CtClass ctClass = pool.get(args[0]);
            memoize(ctClass, ctClass.getDeclaredMethod(args[1]));
            Class<?> rtClass = ctClass.toClass();
            Method main = rtClass.getMethod("main", args.getClass());
            String[] restArgs = new String[args.length - 2];
            System.arraycopy(args, 2, restArgs, 0, restArgs.length);
            main.invoke(null, new Object[] { restArgs });
        }
    }
}
```

Intercession with Javassist at *Load Time*

Current Version - With Annotated Methods

```
import javassist.*;
import java.io.*;
import java.lang.reflect.*;

public class MemoizeAndRun extends Memoize {

    public static void main(String[] args) throws … {
        if (args.length < 2) {
            …
        } else {
            ClassPool pool = ClassPool.getDefault();
            CtClass ctClass = pool.get(args[0]);
            memoizeMethods(ctClass);
            Class<?> rtClass = ctClass.toClass();
            Method main = rtClass.getMethod("main", args.getClass());
            String[] restArgs = new String[args.length - 1];
            System.arraycopy(args, 1, restArgs, 0, restArgs.length);
            main.invoke(null, new Object[] { restArgs });
        }
    }
}
```

Intercession with Javassist at *Load Time*

Current Version - With Annotated Methods

```
static void memoizeMethods(CtClass ctClass) throws … {
    for (CtMethod ctMethod : ctClass.getDeclaredMethods()) {
        Object[] annotations = ctMethod.getAnnotations();
        if ((annotations.length == 1) &&
            (annotations[0] instanceof Memoized)) {
            memoize(ctClass, ctMethod);
        }
    }
}
```

Evolution of the `fib` method

Pre-*Memoization*

```
public class Fib {
    public static Long fib (Long n) {
        if (n < 2) {
            return n;
        } else {
            return fib(n - 1) + fib(n - 2);
        }
    }
}
```

Evolution of the `fib` method

Post-*Memoization* for one method

```
public class Fib {
    public static Long fib$original (Long n) {
        if (n < 2) {
            return n;
        } else {
            return fib(n - 1) + fib(n - 2);
        }
    }

    static Hashtable cachedResults = new Hashtable();

    public static Long fib (Long n) {
        Object result = cachedResults.get(n);
        if (result == null) {
            result = fib$original(n);
            cachedResults.put(n, result);
        }
        return (Long)result;
    }
}
```

Evolution of the `fib` method

Post-*Memoization* for more than one method

```
public class Fib {
    public static Long fib$original (Long n) {
        if (n < 2) {
            return n;
        } else {
            return fib(n - 1) + fib(n - 2);
        }
    }

    static Hashtable fibResults = new Hashtable();

    public static Long fib (Long n) {
        Object result = fibResults.get(n);
        if (result == null) {
            result = fib$original(n);
            fibResults.put(n, result);
        }
        return (Long)result;
    }
}
```

Intercession with Javassist at *Load Time*

Previous Version - Just One Method

```
static void memoize(CtClass ctClass, CtMethod ctMethod)
    throws NotFoundException, CannotCompileException {
    String name = ctMethod.getName();
    CtField ctField =
        CtField.make(
            "static java.util.Hashtable cachedResults = " +
            "    new java.util.Hashtable();",
            ctClass);
    ctClass.addField(ctField);
    ctMethod.setName(name + "$original");
    ctMethod = CtNewMethod.copy(ctMethod, name, ctClass, null);
    ctMethod.setBody(
        "{" +
        "  Object result = cachedResults.get($1);" +
        "  if (result == null) {" +
        "    result = " + name + "$original($$);" +
        "    cachedResults.put($1, result);" +
        "  }" +
        "  return ($r)result;" +
        "}");
    ctClass.addMethod(ctMethod);
}
```

Intercession with Javassist at *Load Time*

Current Version - With Annotated Methods

```
static void memoize(CtClass ctClass, CtMethod ctMethod)
    throws NotFoundException, CannotCompileException {
    String name = ctMethod.getName();
    CtField ctField =
        CtField.make(
            "static java.util.Hashtable " + name + "Results = " +
            "    new java.util.Hashtable();",
            ctClass);
    ctClass.addField(ctField);
    ctMethod.setName(name + "$original");
    ctMethod = CtNewMethod.copy(ctMethod, name, ctClass, null);
    ctMethod.setBody(
        "{" +
        "  Object result = " + name + "Results.get($1);" +
        "  if (result == null) {" +
        "    result = " + name + "$original($$);" +
        "    " + name + "Results.put($1, result);" +
        "  }" +
        "  return ($r)result;" +
        "}");
    ctClass.addMethod(ctMethod);
}
```

Intercession with Javassist at *Load Time*

Problems

- Only one class can be *memoized*.
- The class to *memoize* must be specified.
- The *memoizer* cannot automatically process classes.

Intercession with Javassist at *Load Time*

Problems

- Only one class can be *memoized*.
- The class to *memoize* must be specified.
- The *memoizer* cannot automatically process classes.

Solution

Specialize the *Class Loader*.

Java Type Life Cycle

Phases

1. *Loading*: Locates the binary form of a type with a particular name and creates a `Class` object to represent the type.
2. *Linking*: Combines the binary representation of a type into the runtime state of the Java virtual machine, so that it can be executed.
3. *Initialization*: Executes the class static initializers and the initializers for static fields of classes and interfaces.
4. *Unloading*: When a type is *unreachable* (it is not referenced) it might be *garbage collected*.

Loading

Phases

1. Creates a binary *stream* that represents the type.
2. *Parses* the *stream* to produce internal structures stored in the JVM.
3. Creates one instance of `java.lang.Class` to represent the type.

Loading Time

- Not specified.
- Must happen before *Linking Time* that must happen before *Initialization Time* that must happen before use.
- Typically, it is done as late as possible.
- But it can be done as soon as possible.

Loading

Class Loaders

- Responsible for loading types.
- Organized in a delegation hierarchy: one *Class Loader* on each node whose *parent* is the *Class Loader* that loaded it.
- Each *Class Loader* can (should) delegate on its parent the loading of a type.
- When the *parent Class Loader* cannot load a type, the delegating *Class Loader* can load it.
- The *Class Loader* that loads a type becomes associated with that type as the *Defining Class Loader* of that type.
- Each *Class Loader* that delegates loading a type becomes associated with that type as *Initiating Class Loader* of that type.

Loading

Class Loaders starting from Java 1.2

Starting from Java 1.2, by default, the *Class Loader* delegating hierarchy includes:

1. *Bootstrap Class Loader* (also known as *Primordial Class Loader*) for fundamental classes `java.*`, `javax.*`, etc.
2. *Extension Class Loader* (`ExtClassLoader`) for classes contained in the *runtime* extension directories (`java.ext.dirs`).
3. *System Class Loader* (`AppClassLoader`) for classes contained in the *classpath* (`java.class.path`).

Loading

java.lang.ClassLoader

```
protected synchronized Class loadClass (String name, boolean resolve)
    throws ClassNotFoundException{
    // First check if the class is already loaded
    Class c = findLoadedClass(name);
    if (c == null) {
        try {
            if (parent != null) {
                c = parent.loadClass(name, false);
            } else {
                c = findBootstrapClass0(name);
            }
        } catch (ClassNotFoundException e) {
            // Still not found
            c = findClass(name); // Must be implemented
        }
    }
    if (resolve) {
        resolveClass(c); //linking
    }
    return c;
}
```

Loading

Type Identification

- Each type is identified by its *fully qualified name*.
- Each loaded type is identified by its *fully qualified name* and *class loader*.
- Each *class loader* becomes a different *namespace* for types.
- The same type loaded by two *class loader*s has two occurrences.
- Each occurrence of a class associated with a *class loader* is incompatible with all the other occurrences of the same class associated with other *class loaders*.
- The incompatibility has nothing to do with the **type** of *class loader* but, instead, with the **instance** of *class loader*.

Loading

Bug

```
MyClassLoader myClassLoader = new MyClassLoader();
Class boxClass = myClassLoader.loadClass("Box");
Object obj = boxClass.newInstance();
Box box = (Box)obj;
```

Loading

Bug

```
MyClassLoader myClassLoader = new MyClassLoader();
Class boxClass = myClassLoader.loadClass("Box");
Object obj = boxClass.newInstance();
Box box = (Box)obj;
```

Explanation

- The code is loaded by one *class loader*.
- That *class loader* loads (and becomes associated with) the class `Box` because there is one variable with that type.
- Running the code creates a second *class loader* that also loads the class `Box`.
- The second loading of class `Box` becomes associated with the second *class loader*.
- The *typecast* fails because the classes are considered different.

Loading

Javassist's *Class Loader*

```
import javassist.*;
import Foo;

public class Main {

  public static void main(String[] args) throws Throwable {
     ClassPool pool = ClassPool.getDefault();
     //Create Javassist class loader
     Loader classLoader = new Loader(pool);
     //Obtain the compile time class Foo
     CtClass ctFoo = pool.get("Foo");

     //Modify class Foo
     …

     //Obtain the run time class Foo
     Class rtFoo = classLoader.loadClass("Foo");
     //Instantiate Foo
     Object foo = rtFoo.newInstance();
     …
  }
}
```

Loading

Listeners

- It is possible to associate *listeners* to Javassist's *class loader*.
- *Listeners* are notified:
  - Whenever they are added to the *class loader* (method `start`).
  - Whenever a class is about to be loaded (method `onLoad`).
- *Listeners* implement interface `javassist.Translator`:

`javassist.Translator`

```
public interface Translator {

    public void start(ClassPool pool)
        throws NotFoundException, CannotCompileException;

    public void onLoad(ClassPool pool, String classname)
        throws NotFoundException, CannotCompileException;
}
```

Loading

Javassist's *Class Loader*

```
public class MyTranslator implements Translator {

    void start(ClassPool pool)
        throws NotFoundException, CannotCompileException {
        // Do nothing
    }

    void onLoad(ClassPool pool, String className)
        throws NotFoundException, CannotCompileException {
        // Obtain the compile time class
        CtClass ctClass = pool.get(className);

         // Modify the class
        …

        // That's all.  The class will now be automatically
        // loaded from the modified byte code
    }
}
```

Intercession with Javassist at *Load Time*

Previous Version - Without Javassist's *Class Loader*

```
public class MemoizeAndRun extends Memoize {

    public static void main(String[] args) throws … {
        if (args.length < 2) {
            …
        } else {
            ClassPool pool = ClassPool.getDefault();
            CtClass ctClass = pool.get(args[0]);
            memoizeMethods(ctClass);
            Class<?> rtClass = ctClass.toClass();
            Method main = rtClass.getMethod("main", args.getClass());
            String[] restArgs = new String[args.length - 1];
            System.arraycopy(args, 1, restArgs, 0, restArgs.length);
            main.invoke(null, new Object[] { restArgs });
        }
    }
}
```

Intercession with Javassist at *Load Time*

Current Version - With Javassist's *Class Loader*

```
public class MemoizeAndRun {

    public static void main(String[] args) throws … {
        if (args.length < 1) {
            …
        } else {
            Translator translator = new MemoizeTranslator();
            ClassPool pool = ClassPool.getDefault();
            Loader classLoader = new Loader();
            classLoader.addTranslator(pool, translator);
            String[] restArgs = new String[args.length - 1];
            System.arraycopy(args, 1, restArgs, 0, restArgs.length);
            classLoader.run(args[0], restArgs);
        }
    }
}
```

Intercession with Javassist at *Load Time*

Current Version - With Javassist's *Class Loader*

```
class MemoizeTranslator implements Translator {

    public void start(ClassPool pool)
        throws NotFoundException, CannotCompileException {
    }

    public void onLoad(ClassPool pool, String className)
        throws NotFoundException, CannotCompileException {
        CtClass ctClass = pool.get(className);
        try {
            memoizeMethods(ctClass);
        } catch (ClassNotFoundException e) {
            throw new RuntimeException(e);
        }
    }

    …
}
```

Intercession with Javassist at *Load Time*

Current Version - With Javassist's *Class Loader*

```
class MemoizeTranslator implements Translator {

    …

    void memoizeMethods(CtClass ctClass)
        throws NotFoundException, CannotCompileException,
               ClassNotFoundException {
        for (CtMethod ctMethod : ctClass.getDeclaredMethods()) {
            Object[] annotations = ctMethod.getAnnotations();
            if ((annotations.length == 1) &&
                (annotations[0] instanceof Memoized)) {
                memoize(ctClass, ctMethod);
            }
        }
    }

    …
}
```

Undoable Programs

Problem

- We want to be able to *undo* the execution of Java programs.
- We want to be able to create *checkpoints* representing the execution state of a Java program.
- We want to be able to force a program to go back in time until it reaches a given *checkpoint*.

Undoable Programs

A person has a name, an age, and a friend

```
class Person {
    String name;
    int age;
    Person friend;

    public String toString() {
        return "[" + name + "," + age +
            ((friend == null) ? "" : " with friend " + friend) +
            "]";
    }
}
```

Yes, I know:

- Missing constructor.
- Missing *getters* and *setters*.
- They are not relevant for the example.

Paul, John and Mary

```
Person p0 = new Person() {{ name = "John"; age = 21; }};
Person p1 = new Person() {{ name = "Paul"; age = 23; }};
//Paul has friend named John
p1.friend = p0;
println(p1);//[Paul,23 with friend [John,21]]
```

Paul, John and Mary

```
Person p0 = new Person() {{ name = "John"; age = 21; }};
Person p1 = new Person() {{ name = "Paul"; age = 23; }};
//Paul has friend named John
p1.friend = p0;
println(p1);//[Paul,23 with friend [John,21]]
int state0 = History.currentState();
```

Paul, John and Mary

```
Person p0 = new Person() {{ name = "John"; age = 21; }};
Person p1 = new Person() {{ name = "Paul"; age = 23; }};
//Paul has friend named John
p1.friend = p0;
println(p1);//[Paul,23 with friend [John,21]]
int state0 = History.currentState();
//32 years later, John changed his name to 'Louis' and got a friend
```

Paul, John and Mary

```
Person p0 = new Person() {{ name = "John"; age = 21; }};
Person p1 = new Person() {{ name = "Paul"; age = 23; }};
//Paul has friend named John
p1.friend = p0;
println(p1);//[Paul,23 with friend [John,21]]
int state0 = History.currentState();
//32 years later, John changed his name to 'Louis' and got a friend
p0.age = 53;
p1.age = 55;
p0.name = "Louis";
p0.friend = new Person() {{ name = "Mary"; age = 19; }};
```

Paul, John and Mary

```
Person p0 = new Person() {{ name = "John"; age = 21; }};
Person p1 = new Person() {{ name = "Paul"; age = 23; }};
//Paul has friend named John
p1.friend = p0;
println(p1);//[Paul,23 with friend [John,21]]
int state0 = History.currentState();
//32 years later, John changed his name to 'Louis' and got a friend
p0.age = 53;
p1.age = 55;
p0.name = "Louis";
p0.friend = new Person() {{ name = "Mary"; age = 19; }};
println(p1);//[Paul,55 with friend [Louis,53 with friend [Mary,19]]]
```

Paul, John and Mary

```
Person p0 = new Person() {{ name = "John"; age = 21; }};
Person p1 = new Person() {{ name = "Paul"; age = 23; }};
//Paul has friend named John
p1.friend = p0;
println(p1);//[Paul,23 with friend [John,21]]
int state0 = History.currentState();
//32 years later, John changed his name to 'Louis' and got a friend
p0.age = 53;
p1.age = 55;
p0.name = "Louis";
p0.friend = new Person() {{ name = "Mary"; age = 19; }};
println(p1);//[Paul,55 with friend [Louis,53 with friend [Mary,19]]]
int state1 = History.currentState();
```

Paul, John and Mary

```
Person p0 = new Person() {{ name = "John"; age = 21; }};
Person p1 = new Person() {{ name = "Paul"; age = 23; }};
//Paul has friend named John
p1.friend = p0;
println(p1);//[Paul,23 with friend [John,21]]
int state0 = History.currentState();
//32 years later, John changed his name to 'Louis' and got a friend
p0.age = 53;
p1.age = 55;
p0.name = "Louis";
p0.friend = new Person() {{ name = "Mary"; age = 19; }};
println(p1);//[Paul,55 with friend [Louis,53 with friend [Mary,19]]]
int state1 = History.currentState();
//15 years later, John (hum, I mean 'Louis') died
```

Paul, John and Mary

```
Person p0 = new Person() {{ name = "John"; age = 21; }};
Person p1 = new Person() {{ name = "Paul"; age = 23; }};
//Paul has friend named John
p1.friend = p0;
println(p1);//[Paul,23 with friend [John,21]]
int state0 = History.currentState();
//32 years later, John changed his name to 'Louis' and got a friend
p0.age = 53;
p1.age = 55;
p0.name = "Louis";
p0.friend = new Person() {{ name = "Mary"; age = 19; }};
println(p1);//[Paul,55 with friend [Louis,53 with friend [Mary,19]]]
int state1 = History.currentState();
//15 years later, John (hum, I mean 'Louis') died
p1.age = 70;
p1.friend = null;
```

Paul, John and Mary

```
Person p0 = new Person() {{ name = "John"; age = 21; }};
Person p1 = new Person() {{ name = "Paul"; age = 23; }};
//Paul has friend named John
p1.friend = p0;
println(p1);//[Paul,23 with friend [John,21]]
int state0 = History.currentState();
//32 years later, John changed his name to 'Louis' and got a friend
p0.age = 53;
p1.age = 55;
p0.name = "Louis";
p0.friend = new Person() {{ name = "Mary"; age = 19; }};
println(p1);//[Paul,55 with friend [Louis,53 with friend [Mary,19]]]
int state1 = History.currentState();
//15 years later, John (hum, I mean 'Louis') died
p1.age = 70;
p1.friend = null;
println(p1);//[Paul,70]
```

Paul, John and Mary

```
Person p0 = new Person() {{ name = "John"; age = 21; }};
Person p1 = new Person() {{ name = "Paul"; age = 23; }};
//Paul has friend named John
p1.friend = p0;
println(p1);//[Paul,23 with friend [John,21]]
int state0 = History.currentState();
//32 years later, John changed his name to 'Louis' and got a friend
p0.age = 53;
p1.age = 55;
p0.name = "Louis";
p0.friend = new Person() {{ name = "Mary"; age = 19; }};
println(p1);//[Paul,55 with friend [Louis,53 with friend [Mary,19]]]
int state1 = History.currentState();
//15 years later, John (hum, I mean 'Louis') died
p1.age = 70;
p1.friend = null;
println(p1);//[Paul,70]
//Let's go back in time
History.restoreState(state1);
```

Paul, John and Mary

```
Person p0 = new Person() {{ name = "John"; age = 21; }};
Person p1 = new Person() {{ name = "Paul"; age = 23; }};
//Paul has friend named John
p1.friend = p0;
println(p1);//[Paul,23 with friend [John,21]]
int state0 = History.currentState();
//32 years later, John changed his name to 'Louis' and got a friend
p0.age = 53;
p1.age = 55;
p0.name = "Louis";
p0.friend = new Person() {{ name = "Mary"; age = 19; }};
println(p1);//[Paul,55 with friend [Louis,53 with friend [Mary,19]]]
int state1 = History.currentState();
//15 years later, John (hum, I mean 'Louis') died
p1.age = 70;
p1.friend = null;
println(p1);//[Paul,70]
//Let's go back in time
History.restoreState(state1);
println(p1);//[Paul,55 with friend [Louis,53 with friend [Mary,19]]]
```

Paul, John and Mary

```
Person p0 = new Person() {{ name = "John"; age = 21; }};
Person p1 = new Person() {{ name = "Paul"; age = 23; }};
//Paul has friend named John
p1.friend = p0;
println(p1);//[Paul,23 with friend [John,21]]
int state0 = History.currentState();
//32 years later, John changed his name to 'Louis' and got a friend
p0.age = 53;
p1.age = 55;
p0.name = "Louis";
p0.friend = new Person() {{ name = "Mary"; age = 19; }};
println(p1);//[Paul,55 with friend [Louis,53 with friend [Mary,19]]]
int state1 = History.currentState();
//15 years later, John (hum, I mean 'Louis') died
p1.age = 70;
p1.friend = null;
println(p1);//[Paul,70]
//Let's go back in time
History.restoreState(state1);
println(p1);//[Paul,55 with friend [Louis,53 with friend [Mary,19]]]
//and even earlier
History.restoreState(state0);
```

Paul, John and Mary

```
Person p0 = new Person() {{ name = "John"; age = 21; }};
Person p1 = new Person() {{ name = "Paul"; age = 23; }};
//Paul has friend named John
p1.friend = p0;
println(p1);//[Paul,23 with friend [John,21]]
int state0 = History.currentState();
//32 years later, John changed his name to 'Louis' and got a friend
p0.age = 53;
p1.age = 55;
p0.name = "Louis";
p0.friend = new Person() {{ name = "Mary"; age = 19; }};
println(p1);//[Paul,55 with friend [Louis,53 with friend [Mary,19]]]
int state1 = History.currentState();
//15 years later, John (hum, I mean 'Louis') died
p1.age = 70;
p1.friend = null;
println(p1);//[Paul,70]
//Let's go back in time
History.restoreState(state1);
println(p1);//[Paul,55 with friend [Louis,53 with friend [Mary,19]]]
//and even earlier
History.restoreState(state0);
println(p1);//[Paul,23 with friend [John,21]]
```

Undoable Programs

Save Program State

```
import java.util.Stack;
import java.lang.reflect.*;

public class History {

    static Stack<ObjectFieldValue> undoTrail =
        new Stack<ObjectFieldValue>();

    public static void storePrevious(Object object,
                                     String className,
                                     String fieldName,
                                     Object value) {
        undoTrail.push(new ObjectFieldValue(object,
                                            className,
                                            fieldName,
                                            value));
    }

    …
}
```

Undoable Programs

Save Program State

```
import java.util.Stack;
import java.lang.reflect.*;

public class History {

    …

    public static int currentState() {
        return undoTrail.size();
    }

    public static void restoreState(int state) {
        //undo all actions until size == state
        while (undoTrail.size() != state) {
            undoTrail.pop().restore();
        }
    }
}
```

Undoable Programs

Save Program State

```
class ObjectFieldValue {
    Object object;
    String className;
    String fieldName;
    Object value;

    ObjectFieldValue(Object object,
                     String className,
                     String fieldName,
                     Object value) {
        this.object = object;
        this.className = className;
        this.fieldName = fieldName;
        this.value = value;
    }

    …
}
```

Undoable Programs

Save Program State

```
class ObjectFieldValue {

    …

    void restore() {
        try {
            Field field =
                Class.forName(className).
                getDeclaredField(fieldName);
            field.setAccessible(true);
            field.set(object, value);
        } catch (ClassNotFoundException e) {
            throw new RuntimeException(e);
        } catch (NoSuchFieldException e) {
            throw new RuntimeException(e);
        } catch (IllegalAccessException e) {
            throw new RuntimeException(e);
        }
    }
}
```

Undoable Programs

Javassist

```
import javassist.*;
import javassist.expr.*;
import java.io.*;
import java.lang.reflect.*;

public class Undoable {

    public static void main(String[] args) throws … {
        if (args.length < 1) {
            …
        } else {
            Translator translator = new UndoableTranslator();
            ClassPool pool = ClassPool.getDefault();
            Loader classLoader = new Loader();
            classLoader.addTranslator(pool, translator);
            String[] restArgs = new String[args.length - 1];
            System.arraycopy(args, 1, restArgs, 0, restArgs.length);
            classLoader.run(args[0], restArgs);
        }
    }
}
```

Undoable Programs

Javassist

```
class UndoableTranslator implements Translator {

    public void start(ClassPool pool)
        throws NotFoundException, CannotCompileException {
    }

    public void onLoad(ClassPool pool, String className)
        throws NotFoundException, CannotCompileException {
        CtClass ctClass = pool.get(className);
        makeUndoable(ctClass);
    }

    void makeUndoable(CtClass ctClass) {
        …
    }
}
```

Undoable Programs

Javassist

```
void makeUndoable(CtClass ctClass)
    throws NotFoundException, CannotCompileException {
    final String template =
        "{" +
        "  History.storePrevious($0, \"%s\",\"%s\", ($w)$0.%s);" +
        "  $0.%s = $1;" +
        "}";
    for (CtMethod ctMethod : ctClass.getDeclaredMethods()) {
        ctMethod.instrument(new ExprEditor() {
                public void edit(FieldAccess fa)
                    throws CannotCompileException {
                    if (fa.isWriter()) {
                        String name = fa.getFieldName();
                        fa.replace(String.format(template,
                                                 fa.getClassName(),
                                                 name, name, name));
                    }
                }
            });
    }
}
```

Intercession at Run Time

Class Generation at Run Time

- Besides intercession at load time, Javassist can create new classes at run time.
- By careful use of these new classes, it is possible to do limited forms of intercession.

Class Generation at Run Time

```
ClassPool pool = ClassPool.getDefault();
```

Intercession at Run Time

Class Generation at Run Time

- Besides intercession at load time, Javassist can create new classes at run time.
- By careful use of these new classes, it is possible to do limited forms of intercession.

Class Generation at Run Time

```
ClassPool pool = ClassPool.getDefault();
CtClass ctFoo = pool.makeClass("Foo");
```

Intercession at Run Time

Class Generation at Run Time

- Besides intercession at load time, Javassist can create new classes at run time.
- By careful use of these new classes, it is possible to do limited forms of intercession.

Class Generation at Run Time

```
ClassPool pool = ClassPool.getDefault();
CtClass ctFoo = pool.makeClass("Foo");
ctFoo.setSuperclass(...);
```

Intercession at Run Time

Class Generation at Run Time

- Besides intercession at load time, Javassist can create new classes at run time.
- By careful use of these new classes, it is possible to do limited forms of intercession.

Class Generation at Run Time

```
ClassPool pool = ClassPool.getDefault();
CtClass ctFoo = pool.makeClass("Foo");
ctFoo.setSuperclass(...);
...
ctFoo.addField(...);
...
ctFoo.addMethod(...);
...
```

Intercession at Run Time

Class Generation at Run Time

- Besides intercession at load time, Javassist can create new classes at run time.
- By careful use of these new classes, it is possible to do limited forms of intercession.

Class Generation at Run Time

```
ClassPool pool = ClassPool.getDefault();
CtClass ctFoo = pool.makeClass("Foo");
ctFoo.setSuperclass(...);
...
ctFoo.addField(...);
...
ctFoo.addMethod(...);
...
Class rtFoo = ctFoo.toClass();
```

Java Expression Compiler/Interpreter

Simple Evaluator

```
import javassist.*;
import java.lang.reflect.*;

public class Evaluator {

    public static void main (String[] args) throws Exception {
      ClassPool pool = ClassPool.getDefault();
      CtClass ctEvaluator = pool.makeClass("Eval");
      String expression = args[0];
      String template =
          "public static double eval () { " +
          "  return ("+ expression +");" +
          "},";
      CtMethod ctMethod = CtNewMethod.make(template, ctEvaluator);
      ctEvaluator.addMethod(ctMethod);
      Class evaluator = ctEvaluator.toClass();
      Method meth = evaluator.getDeclaredMethod("eval");
      System.out.println(meth.invoke(null));
    }
}
```

Java Expression Compiler/Interpreter

Simple Evaluator

```
import javassist.*;
import java.lang.reflect.*;

public class Evaluator {

    public static void main (String[] args) throws Exception {
      ClassPool pool = ClassPool.getDefault();
      CtClass ctEvaluator = pool.makeClass("Eval");
      String expression = args[0];
      String template =
          "public static double eval () { " +
          "  return ("+ expression +");" +
          "},";
      CtMethod ctMethod = CtNewMethod.make(template, ctEvaluator);
      ctEvaluator.addMethod(ctMethod);
      Class evaluator = ctEvaluator.toClass();
      Method meth = evaluator.getDeclaredMethod("eval");
      System.out.println(meth.invoke(null));
    }
}
```

Java Expression Compiler/Interpreter

Simple Evaluator

```
import javassist.*;
import java.lang.reflect.*;

public class Evaluator {

    public static void main (String[] args) throws Exception {
      ClassPool pool = ClassPool.getDefault();
      CtClass ctEvaluator = pool.makeClass("Eval");
      String expression = args[0];
      String template =
          "public static double eval () { " +
          "  return ("+ expression +");" +
          "},";
      CtMethod ctMethod = CtNewMethod.make(template, ctEvaluator);
      ctEvaluator.addMethod(ctMethod);
      Class evaluator = ctEvaluator.toClass();
      Method meth = evaluator.getDeclaredMethod("eval");
      System.out.println(meth.invoke(null));
    }
}
```

Java Expression Compiler/Interpreter

Simple Evaluator

```
import javassist.*;
import java.lang.reflect.*;

public class Evaluator {

    public static void main (String[] args) throws Exception {
      ClassPool pool = ClassPool.getDefault();
      CtClass ctEvaluator = pool.makeClass("Eval");
      String expression = args[0];
      String template =
          "public static double eval () { " +
          "  return ("+ expression +");" +
          "},";
      CtMethod ctMethod = CtNewMethod.make(template, ctEvaluator);
      ctEvaluator.addMethod(ctMethod);
      Class evaluator = ctEvaluator.toClass();
      Method meth = evaluator.getDeclaredMethod("eval");
      System.out.println(meth.invoke(null));
    }
}
```

Java Expression Compiler/Interpreter

Simple Evaluator

```
import javassist.*;
import java.lang.reflect.*;

public class Evaluator {

    public static void main (String[] args) throws Exception {
      ClassPool pool = ClassPool.getDefault();
      CtClass ctEvaluator = pool.makeClass("Eval");
      String expression = args[0];
      String template =
          "public static double eval () { " +
          "  return ("+ expression +");" +
          "},";
      CtMethod ctMethod = CtNewMethod.make(template, ctEvaluator);
      ctEvaluator.addMethod(ctMethod);
      Class evaluator = ctEvaluator.toClass();
      Method meth = evaluator.getDeclaredMethod("eval");
      System.out.println(meth.invoke(null));
    }
}
```

Java Expression Compiler/Interpreter

Simple Evaluator

```
import javassist.*;
import java.lang.reflect.*;

public class Evaluator {

    public static void main (String[] args) throws Exception {
      ClassPool pool = ClassPool.getDefault();
      CtClass ctEvaluator = pool.makeClass("Eval");
      String expression = args[0];
      String template =
          "public static double eval () { " +
          "  return ("+ expression +");" +
          "},";
      CtMethod ctMethod = CtNewMethod.make(template, ctEvaluator);
      ctEvaluator.addMethod(ctMethod);
      Class evaluator = ctEvaluator.toClass();
      Method meth = evaluator.getDeclaredMethod("eval");
      System.out.println(meth.invoke(null));
    }
}
```

Java Expression Compiler/Interpreter

Simple Evaluator

```
import javassist.*;
import java.lang.reflect.*;

public class Evaluator {

    public static void main (String[] args) throws Exception {
      ClassPool pool = ClassPool.getDefault();
      CtClass ctEvaluator = pool.makeClass("Eval");
      String expression = args[0];
      String template =
          "public static double eval () { " +
          "  return ("+ expression +");" +
          "},";
      CtMethod ctMethod = CtNewMethod.make(template, ctEvaluator);
      ctEvaluator.addMethod(ctMethod);
      Class evaluator = ctEvaluator.toClass();
      Method meth = evaluator.getDeclaredMethod("eval");
      System.out.println(meth.invoke(null));
    }
}
```

Java Expression Compiler/Interpreter

Simple Evaluator

```
import javassist.*;
import java.lang.reflect.*;

public class Evaluator {

    public static void main (String[] args) throws Exception {
      ClassPool pool = ClassPool.getDefault();
      CtClass ctEvaluator = pool.makeClass("Eval");
      String expression = args[0];
      String template =
          "public static double eval () { " +
          "  return ("+ expression +");" +
          "},";
      CtMethod ctMethod = CtNewMethod.make(template, ctEvaluator);
      ctEvaluator.addMethod(ctMethod);
      Class evaluator = ctEvaluator.toClass();
      Method meth = evaluator.getDeclaredMethod("eval");
      System.out.println(meth.invoke(null));
    }
}
```

Java Expression Compiler/Interpreter

Example of Use

```
$ javac -cp .:./javassist.jar Evaluator.java
```

Java Expression Compiler/Interpreter

Example of Use

```
$ javac -cp .:./javassist.jar Evaluator.java
$ java -cp .:./javassist.jar Evaluator 1.0
```

Java Expression Compiler/Interpreter

Example of Use

```
$ javac -cp .:./javassist.jar Evaluator.java
$ java -cp .:./javassist.jar Evaluator 1.0
1.0
```

Java Expression Compiler/Interpreter

Example of Use

```
$ javac -cp .:./javassist.jar Evaluator.java
$ java -cp .:./javassist.jar Evaluator 1.0
1.0
$ java -cp .:./javassist.jar Evaluator 1.0+2
```

Java Expression Compiler/Interpreter

Example of Use

```
$ javac -cp .:./javassist.jar Evaluator.java
$ java -cp .:./javassist.jar Evaluator 1.0
1.0
$ java -cp .:./javassist.jar Evaluator 1.0+2
3.0
```

Java Expression Compiler/Interpreter

Example of Use

```
$ javac -cp .:./javassist.jar Evaluator.java
$ java -cp .:./javassist.jar Evaluator 1.0
1.0
$ java -cp .:./javassist.jar Evaluator 1.0+2
3.0
$ java -cp .:./javassist.jar Evaluator 1.0+2*3
```

Java Expression Compiler/Interpreter

Example of Use

```
$ javac -cp .:./javassist.jar Evaluator.java
$ java -cp .:./javassist.jar Evaluator 1.0
1.0
$ java -cp .:./javassist.jar Evaluator 1.0+2
3.0
$ java -cp .:./javassist.jar Evaluator 1.0+2*3
7.0
```

Java Expression Compiler/Interpreter

Example of Use

```
$ javac -cp .:./javassist.jar Evaluator.java
$ java -cp .:./javassist.jar Evaluator 1.0
1.0
$ java -cp .:./javassist.jar Evaluator 1.0+2
3.0
$ java -cp .:./javassist.jar Evaluator 1.0+2*3
7.0
```

Problems

- *Fragile*: expressions must not have spaces (or must be quoted).
- *One shot*: only one expression can be evaluated at a time.

Java Expression Compiler/Interpreter

Simple Evaluator

```
public static void main (String[] args) throws Exception {
    ClassPool pool = ClassPool.getDefault();
        String template =
            "public static double eval () { " +
            "  return ("+ expression +");" +
            "},";
        CtMethod ctMethod = CtNewMethod.make(template, ctEvaluator);
        ctEvaluator.addMethod(ctMethod);
        Class evaluator = ctEvaluator.toClass();
        Method meth = evaluator.getDeclaredMethod("eval");
        System.out.println(meth.invoke(null));
    }
}
```

Java Expression Compiler/Interpreter

Simple Evaluator

```
public static void main (String[] args) throws Exception {
    ClassPool pool = ClassPool.getDefault();
    BufferedReader input =
      new BufferedReader(new InputStreamReader(System.in));




        String template =
            "public static double eval () { " +
            "  return ("+ expression +");" +
            "},";
        CtMethod ctMethod = CtNewMethod.make(template, ctEvaluator);
        ctEvaluator.addMethod(ctMethod);
        Class evaluator = ctEvaluator.toClass();
        Method meth = evaluator.getDeclaredMethod("eval");
        System.out.println(meth.invoke(null));
    }
}
```

Java Expression Compiler/Interpreter

Simple Evaluator

```
public static void main (String[] args) throws Exception {
    ClassPool pool = ClassPool.getDefault();
    BufferedReader input =
      new BufferedReader(new InputStreamReader(System.in));
    for(int i = 0; true; i++) {



        String template =
            "public static double eval () { " +
            "  return ("+ expression +");" +
            "},";
        CtMethod ctMethod = CtNewMethod.make(template, ctEvaluator);
        ctEvaluator.addMethod(ctMethod);
        Class evaluator = ctEvaluator.toClass();
        Method meth = evaluator.getDeclaredMethod("eval");
        System.out.println(meth.invoke(null));
    }
}
```

Java Expression Compiler/Interpreter

Simple Evaluator

```
public static void main (String[] args) throws Exception {
    ClassPool pool = ClassPool.getDefault();
    BufferedReader input =
      new BufferedReader(new InputStreamReader(System.in));
    for(int i = 0; true; i++) {
        System.out.print("> ");


        String template =
            "public static double eval () { " +
            "  return ("+ expression +");" +
            "},";
        CtMethod ctMethod = CtNewMethod.make(template, ctEvaluator);
        ctEvaluator.addMethod(ctMethod);
        Class evaluator = ctEvaluator.toClass();
        Method meth = evaluator.getDeclaredMethod("eval");
        System.out.println(meth.invoke(null));
    }
}
```

Java Expression Compiler/Interpreter

Simple Evaluator

```
public static void main (String[] args) throws Exception {
    ClassPool pool = ClassPool.getDefault();
    BufferedReader input =
      new BufferedReader(new InputStreamReader(System.in));
    for(int i = 0; true; i++) {
        System.out.print("> ");
        String expression = input.readLine();

        String template =
            "public static double eval () { " +
            "  return ("+ expression +");" +
            "},";
        CtMethod ctMethod = CtNewMethod.make(template, ctEvaluator);
        ctEvaluator.addMethod(ctMethod);
        Class evaluator = ctEvaluator.toClass();
        Method meth = evaluator.getDeclaredMethod("eval");
        System.out.println(meth.invoke(null));
    }
}
```

Java Expression Compiler/Interpreter

Simple Evaluator

```
public static void main (String[] args) throws Exception {
    ClassPool pool = ClassPool.getDefault();
    BufferedReader input =
      new BufferedReader(new InputStreamReader(System.in));
    for(int i = 0; true; i++) {
        System.out.print("> ");
        String expression = input.readLine();
        CtClass ctEvaluator = pool.makeClass("Eval" + i);
        String template =
            "public static double eval () { " +
            "  return ("+ expression +");" +
            "},";
        CtMethod ctMethod = CtNewMethod.make(template, ctEvaluator);
        ctEvaluator.addMethod(ctMethod);
        Class evaluator = ctEvaluator.toClass();
        Method meth = evaluator.getDeclaredMethod("eval");
        System.out.println(meth.invoke(null));
    }
}
```

Java Expression Compiler/Interpreter

Example of Use

```
$ java -cp .:./javassist.jar Evaluator
```

Java Expression Compiler/Interpreter

Example of Use

```
$ java -cp .:./javassist.jar Evaluator
> 1.0
```

Java Expression Compiler/Interpreter

Example of Use

```
$ java -cp .:./javassist.jar Evaluator
> 1.0
1.0
```

Java Expression Compiler/Interpreter

Example of Use

```
$ java -cp .:./javassist.jar Evaluator
> 1.0
1.0
> 1.0 + 2
3.0
> 1.0 + 2 * 3
7.0
```

Java Expression Compiler/Interpreter

Example of Use

```
$ java -cp .:./javassist.jar Evaluator
> 1.0
1.0
> 1.0 + 2
3.0
> 1.0 + 2 * 3
7.0
> 1.0 + 2 * (3 + 4)
15.0
```

Java Expression Compiler/Interpreter

Example of Use

```
$ java -cp .:./javassist.jar Evaluator
> 1.0
1.0
> 1.0 + 2
3.0
> 1.0 + 2 * 3
7.0
> 1.0 + 2 * (3 + 4)
15.0
> Math.sin(Math.PI/2)
1.0
```

Java Expression Compiler/Interpreter

Example of Use

```
$ java -cp .:./javassist.jar Evaluator
> 1.0
1.0
> 1.0 + 2
3.0
> 1.0 + 2 * 3
7.0
> 1.0 + 2 * (3 + 4)
15.0
> Math.sin(Math.PI/2)
1.0
```

Problems

- Mathematical functions should be easier to use.

Java Expression Compiler/Interpreter

Simple Evaluator

```
public static void main (String[] args) throws Exception {
    ClassPool pool = ClassPool.getDefault();
    BufferedReader input =
      new BufferedReader(new InputStreamReader(System.in));

    for(int i = 0; true; i++) {
        System.out.print("> ");
        String expression = input.readLine();
        CtClass ctEvaluator = pool.makeClass("Eval" + i);

        String template =
            "public static double eval () { " +
            "  return ("+ expression +");" +
            "},";
        CtMethod ctMethod = CtNewMethod.make(template, ctEvaluator);
        ctEvaluator.addMethod(ctMethod);
        Class evaluator = ctEvaluator.toClass();
        Method meth = evaluator.getDeclaredMethod("eval");
        System.out.println(meth.invoke(null));
    }
}
```

Java Expression Compiler/Interpreter

Simple Evaluator

```
public static void main (String[] args) throws Exception {
    ClassPool pool = ClassPool.getDefault();
    BufferedReader input =
      new BufferedReader(new InputStreamReader(System.in));
    CtClass ctEval = pool.get("Eval");
    for(int i = 0; true; i++) {
        System.out.print("> ");
        String expression = input.readLine();
        CtClass ctEvaluator = pool.makeClass("Eval" + i);

        String template =
            "public static double eval () { " +
            "  return ("+ expression +");" +
            "},";
        CtMethod ctMethod = CtNewMethod.make(template, ctEvaluator);
        ctEvaluator.addMethod(ctMethod);
        Class evaluator = ctEvaluator.toClass();
        Method meth = evaluator.getDeclaredMethod("eval");
        System.out.println(meth.invoke(null));
    }
}
```

Java Expression Compiler/Interpreter

Simple Evaluator

```
public static void main (String[] args) throws Exception {
    ClassPool pool = ClassPool.getDefault();
    BufferedReader input =
      new BufferedReader(new InputStreamReader(System.in));
    CtClass ctEval = pool.get("Eval");
    for(int i = 0; true; i++) {
        System.out.print("> ");
        String expression = input.readLine();
        CtClass ctEvaluator = pool.makeClass("Eval" + i);
        ctEvaluator.setSuperclass(ctEval);
        String template =
            "public static double eval () { " +
            "  return ("+ expression +");" +
            "},";
        CtMethod ctMethod = CtNewMethod.make(template, ctEvaluator);
        ctEvaluator.addMethod(ctMethod);
        Class evaluator = ctEvaluator.toClass();
        Method meth = evaluator.getDeclaredMethod("eval");
        System.out.println(meth.invoke(null));
    }
}
```

Java Expression Compiler/Interpreter

Base Class

```
class Eval {

    public static double pi = Math.PI;

    public static double sin (double arg) { return Math.sin(arg); }

    public Eval() {}
}
```

Example of Use

Java Expression Compiler/Interpreter

Base Class

```
class Eval {

    public static double pi = Math.PI;

    public static double sin (double arg) { return Math.sin(arg); }

    public Eval() {}
}
```

Example of Use

```
$ java -cp .:./javassist.jar Evaluator
> 1.0
1.0
> 1.0 + 2 * (3 + 4)
15.0
> Math.sin(Math.PI/2)
1.0
```

Java Expression Compiler/Interpreter

Base Class

```
class Eval {

    public static double pi = Math.PI;

    public static double sin (double arg) { return Math.sin(arg); }

    public Eval() {}
}
```

Example of Use

```
$ java -cp .:./javassist.jar Evaluator
> 1.0
1.0
> 1.0 + 2 * (3 + 4)
15.0
> Math.sin(Math.PI/2)
1.0
> sin(pi/2)
1.0
```


Jason Baker and Wilson C. Hsieh.
Maya: Multiple-dispatch syntax extension in java.
In *PLDI*, pages 270–281, 2002.


Robert J. Chassell.
An Introduction to Programming in Emacs Lisp.
GNU Press, pub-GNU-PRESS:adr, 2001.


S. Chiba.
Javassist – A reflection-based programming wizard for java.
In *Proceedings of the Workshop on Reflective Programming in C++ at the 13th ACM Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'98)*, Vancouver, Canada, October 1998.
http://www.csg.is.titech.ac.jp/~chiba/oopsla98/proc/chiba.pdf.


Shigeru Chiba.
Load-time structural reflection in Java.
Lecture Notes in Computer Science, 1850:313–??, 2000.


Curtis Clifton, Todd Millstein, Gary T. Leavens, and Craig Chambers.
MultiJava: Design rationale, compiler implementation, and applications.
ACM Transactions on Programming Languages and Systems, 28(3), May 2006.


Sheng Liang and Gilad Bracha.
Dynamic class loading in the java tm virtual machine.
In *In Proc. 13th ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'98), volume 33, number 10 of ACM SIGPLAN Notices*, pages 36–44. ACM Press, 1998.


Sheng Liang and Gilad Bracha.
Dynamics class loading in the java virtual machine.
In *OOPSLA*, pages 36–44, 1998.


Pattie Maes.

Concepts and experiments in computational reflection.
In Norman Meyrowitz, editor, *Proceedings of the 2nd Annual Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA '87)*, pages 147–155, Orlando, FL, USA, October 1987. ACM Press.

Radu Muschevici, Alex Potanin, Ewan Tempero, and James Noble.

Multiple dispatch in practice.
In *OOPSLA '08: Proceedings of the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications*, pages 563–582, New York, NY, USA, 2008. ACM.