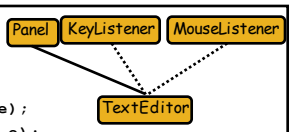
Java Part II

CSCI 334
Stephen Freund

Interfaces




```
interface KeyListener {
  void keyPressed(KeyEvent e);
  void keyReleased(KeyEvent e);
  void keyTyped(KeyEvent e);
}

interface MouseListener {
  void buttonClicked(MouseEvent e);
}

class TextEditor extends Panel
                 implements KeyListener,MouseListener {
  void keyPressed(KeyEvent e)  { /* code */ }
  void keyReleased(KeyEvent e) { /* code */ }
  void keyTyped(KeyEvent e)    { /* code */ }
  void buttonClicked(MouseEvent e) { /* code */ }
}
```

Scala Traits

- Completely Abstract

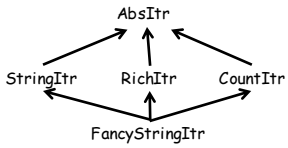
```
trait AbsIterator[T] {
  def hasNext(): boolean;
  def next(): T;
}
```

- Partially Implemented

```
trait RichIterator[T] extends AbsIterator[T]
  def foreach(f: T => Unit): Unit = {
    while (hasNext()) f(next())
  }
}
```

Scala Traits

```
trait CountingIterator[T] extends AbsIterator[T] {
  var count = 0;
  abstract override def next(): T = {
    count = count + 1;
    super.next();
  }
  def count() = count;
}
```

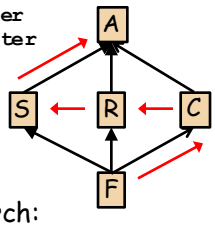



```
class FancyStringIterator(s: String)
       extends StringIterator(s)
       with RichIterator[Char]
       with CountingIterator[Char] { ... }
```

http://en.wikipedia.org/wiki/Mixi

Name Resolution via Linearization

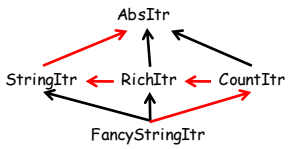
```
trait AbsIter
trait StringIter extends AbsIter
trait RichIter extends AbsIter
trait CountIter extends AbsIter
class FancyStringIter
       extends StringIter
       with RichIter
       with CountIter
```



Right-first depth-first search:
[F,C,A,R,A,S,A]
Eliminate all but last occurrence:
[F,C,R,S,A]

Scala Traits

```
trait CountingIterator[T] extends AbsIterator[T] {
  var count = 0;
  abstract override def next(): T = {
    count = count + 1;
    super.next();
  }
  def count() = count;
}
```



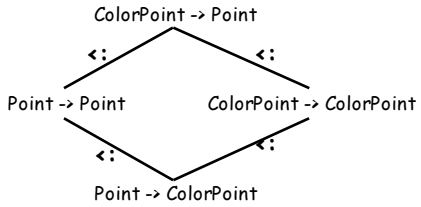

```
class FancyStringIterator(s: String)
       extends StringIterator(s)
       with RichIterator[Char]
       with CountingIterator[Char] { ... }
```

http://en.wikipedia.org/wiki/Mixi

Mixins at Allocation Time

```
val iter = new StringIterator("moo")
       with RichIterator[Char]
       with CountingIterator[Char] {

  def anotherMethod() = println("cow");

}

iter.foreach(println(_));
println(iter.count);
iter.anotherMethod();
```

Function Subtyping Revisited



- Return Types are *Covariant*
- Argument Types are *Contravariant*

Java Array Covariant Subtyping Rule

```
class Point { ... }
class ColorPoint extends Point { ... }

Point pts[] = new Point [100];


pts[0] = new Point (10,10);
pts[1] = new Point (20,20);
```

Java Array Covariant Subtyping Rule

```
class Point { ... }
class ColorPoint extends Point { ... }

ColorPoint[] cpts = new ColorPoint[100];
Point pts[] = cpts;

pts[0] = new Point (10,10);
pts[1] = new Point (20,20);

...
cpts[0].setColor(RED);
```

Why Did They Add It?

```
static void arrayCopy(Point src[], Point dst[]) {
  for (int i = 0; i < src.length; i++)
    dst[i] = src[i];
}


Point p[];
Point q[];
arrayCopy(p,q);
String s[];
String t[];
arrayCopy(s,t);
```

General arrayCopy Operation

```
static void arrayCopy(Object src[], Object dst[]) {
  for (int i = 0; i < src.length; i++)
    dst[i] = src[i];
}


Point p[];
Point q[];
arrayCopy(p,q);

String s[];
String t[];
arrayCopy(s,t);
```

From Bill Joy (Sun Cofounder)

Date: Fri, 09 Oct 1998 09:41:05 -0600
From: bill joy
Subject: …[discussion about java genericity]

actually, java array covariance was done for less noble reasons …: it made some generic "bcopy" (memory copy) and like operations much easier to write…
I proposed to take this out in 95, but it was too late (…).
i think it is unfortunate that it wasn't taken out…
it would have made adding genericity later much cleaner, and [array covariance] doesn't pay for its complexity today.
wnj

Variance in Scala

```
Array[ColorPoint]  <: Array[Point]  ?

 List[ColorPoint]  <: List[Point]?
```

Variance Annotations in Scala

- Class defined with covariant parameter T:

```
class List[+T] { ... }
```

- So `List[ColorPoint] <: List[Point]`
- Which of these is type-safe?

```
class List[+T] {
  def add(t : T) = ...
}
```

```
class List[+T] {
  def get() : T = ...
}
```

Variance Annotations in Scala

- Function Types:

```
class Function[-A,+R] {
  def apply(arg : A) : R = ...
}

Function[Int,ColorPoint] <: Function[Int,Point]
```

- Immutable Maps:

```
class Map[Key,+Value] extends ...

Map[String,ColorPoint] <: Map[String,Point]
```

- Mutable Maps:

```
class Map[Key,Value] extends ...

Map[String,ColorPoint] <: Map[String,Point]
```

✗

Java 1.0 (Subtype Poly.) vs Java 1.5 (Parametric Poly.)

```
class Stack {
  void push(Object o) {...}
  Object pop() {...}
  ...
}

String s = "Hello";
Stack st = new Stack();


st.push(s);

String t = (String)st.pop();
```

```
class Stack<T> {
  void push(T o) {...}
  T pop() {...}
  ...
}

String s = "Hello";
Stack<String> st =
  new Stack<String>();

st.push(s);

String t = st.pop();
```

Compilation

```
class Stack<T> {
  void push(T o) {...}
  T pop() {...}
  ...
}

String s = "Hello";
Stack<String> st =
  new Stack<String>();

st.push(s);

String t = st.pop();
```

```
class Stack {
  void push(Object o) {...}
  Object pop() {...}
  ...
}

String s = "Hello";
Stack st = new Stack();


st.push(s);

String t = (String)st.pop();
```

Type Checking

```
interface Printable {
  void print();
}

class PrintableStack<T> extends Printable {
  private Vector<T> elems;
  void push(T o) {...}
  T pop() {...}

  void print() {
    ...
    for (T t : elems) {
      t.print();
    }
  }
```

✗

```
PrintableStack<X> st = ..
st.push(x);
st.print();
```

Type Checking & Bounded Polymorphism

```
interface Printable {
  void print();
}

class PrintableStack<T extends Printable>
            implements Printable {
  private Vector<T> elems;
  void push(T o) {...}
  T pop() {...}

  void print() {
    ...
    for (T t : elems) {
      t.print();
    }
```

```
PrintableStack<X> st = ..
st.push(x);
st.print();
```

Compilation

```
class Stack<T extends
            Printable> {
  void push(T o) {...}
  T pop() {...}

  void print() {
    ...
    for (T t : elems) {
      t.print();
    }
  }
}
```

⇨

```
class Stack {
  void push(Printable o) {..
  Printable pop() {...}

  void print() {
    ...
    for (Printable t : elems)
      t.print();
    }
  }
}
```

F-Bounded Polymorphism

return neg. if this < other
return zero if this == other
return pos. if this > other

```
interface Comparable {
  int compareTo(Object other);
}

class Point implements Comparable {
  int compareTo(Object other) {
     Point otherAsPoint = (Point)other;
     return x - otherAsPoint.x;
  }
}

class PriorityQueue<T extends Comparable> {
  void add(T t) {
    T o = ...;
    ... t.compareTo(o) ...
```

F-Bounded Polymorphism

```
interface Comparable<T> {
  int compareTo(T other);
}

class Point implements Comparable<Point> {
  int compareTo(Point other) {
    return x - other.x;
  }
}

class PriorityQueue<T extends Comparable<T>> {
  void add(T t) {
    T o = ...;
    ... t.compareTo(o) ...
  }
```

In Scala...

```
trait Comparable[T] {
  def compareTo(other : T): Boolean;
}

class Point extends Comparable[Point] {
  def compareTo(other : Point) = {
    return x - other.x;
  }
}

class PriorityQueue[T <: Comparable[T]] {
  def add(t : T) = {
    val o : T = ...;
    ... t.compareTo(o) ...
  }
```

Java Wildcards

```
void printElements(Collection<Object> c) {
  for (Object e : c) {
    System.out.println(e);
  }
}

Collection<String> cs;
Collection<Integer> ci;
printElements(cs);
printElements(ci);
```

Java Wildcards

```
void printElements(Collection<?> c) {
  for (Object e : c) {
    System.out.println(e);
  }
}

Collection<String> cs;
Collection<Integer> ci;
printElements(cs);
printElements(ci);
```

Java Wildcards

```
void movePoints(Collection<?> c) {
  for (Point p : c) {
    p.move(10,10);
  }
}

Collection<Point> pts;
Collection<ColorPoint> cpts;
movePoints(pts);
movePoints(cpts);
```

Java Wildcards

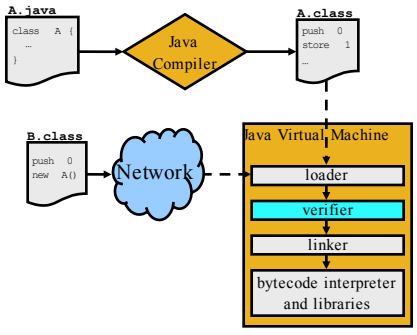
```
void movePoints(Collection<? extends Point> c) {
  for (Point p : c) {
    p.move(10,10);
  }
}

Collection<Point> pts;
Collection<ColorPoint> cpts;
movePoints(pts);
movePoints(cpts);
```

[Form of existential types.
Scala variant on HW…]

JITs

Stephen Freund
CS 334

JVM Architecture



Java Bytecodes

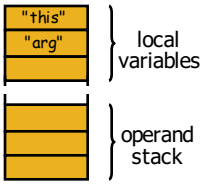
- Java:

```
class A {
   int x;
   void f(int val) { this.x = val+1; }
}
```

- Bytecode:

```
Method void f(int)
  0: aload 0      // push "this"
  1: iload_1      // push arg
  2: iconst_1     // push 1
  3: iadd
  4: putfield "int A.x"
  5: return
```

Activation Record



Java Bytecodes

- Java:

```
Point p;
...
p.getX();
...
```

- Bytecode:

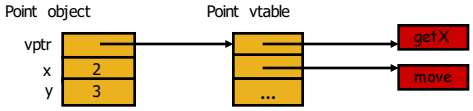
```
4: aload_1     // push p onto stack
5: invoke_virtual "int Point.getX()"
```

Java Run-Time Representation

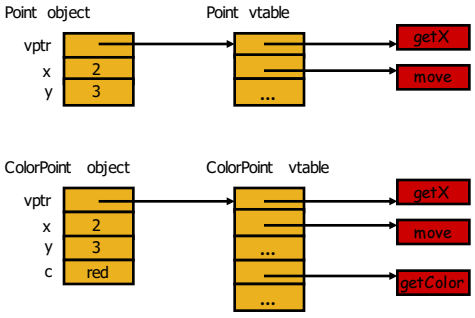
Map for symbolic names to offsets
(A lot like SmallTalk Class MetaObjects!)

Constant Pool for Point.class

field	"int x"	4
field	"int y"	8
method	"int getX()"	0
method	"move(int,int)"	1
...	"..."	...



Java Run-Time Representation



Method Invocation – Symbolic Linking

```
Point p;
...
p.getX()
```

```
...
aload 1    // push p
invoke_virtual "int Point.getX()"
```

Constant Pool for Point

field	"int x"	4
field	"int y"	8
method	"int getX()"	0
method	"move(int,int)"	1
...	"..."	...

```
...
aload_1
invoke_virtual_quick 0
```

1. Look up string description in constant pool
2. Find offset into vtable
3. Change instruction to "fast form"

Field Access

```
class Point {
  int getX() {
    return this.x;
  }
}
```

```
int getX() {
  aload 0
  getfield "int Point.x"
  ireturn
}
```

Constant Pool for Point

field	"int x"	4
field	"int y"	8
method	"int getX()"	0
method	"move(int,int)"	1
...	"..."	...

```
int getX() {
  aload 0
  getfield_quick 4
  ireturn
}
```

1. Look up string description in constant pool
2. Find offset into object record
3. Change instruction to "fast form"

Interface Method Invocation

```
MouseListener m;
m = ...;
m.buttonClicked(e)
```

```
aload 1   /* push m */
aload 2   /* push e */
invoke_interface
  "void MouseListener.
       buttonClicked(MouseEvent)"
```

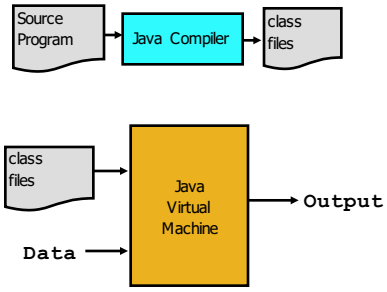
Constant Pool for TextEditor

field	...	...
field	...	...
method	...	...
method	"buttonClick..."	3
...	"..."	...

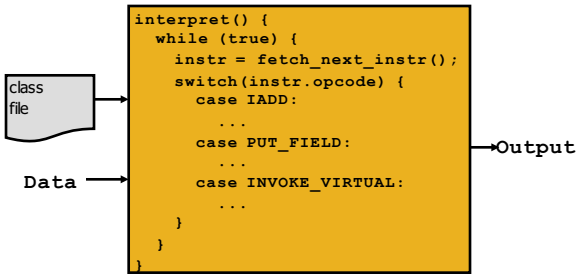
Search for method in constant pool

[Polymorphic Inline Caching, Holzle, Chambers, Ungar, *ECOOP '91*]

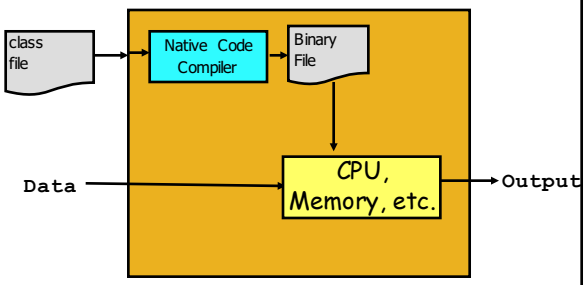
Java Compiler and Interpreter



Standard Interpreter



Interpreter with JIT Compiler



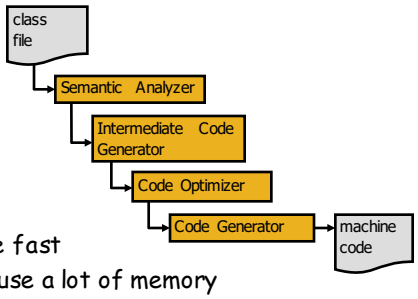
Example Translation

```
void f() {
  this.x = this.moo() + 1;
}
```

```
aload 0
invoke_virtual "Method moo"
aload 1
iadd
aload 0
putfield "Field int x"
```

```
mov (r0), r1
call 4(r1)
add 1, (r30)
mov (r30), 8(r0)
```

JIT



- Must be fast
- Cannot use a lot of memory
- Semantic analysis must consider dynamic loading

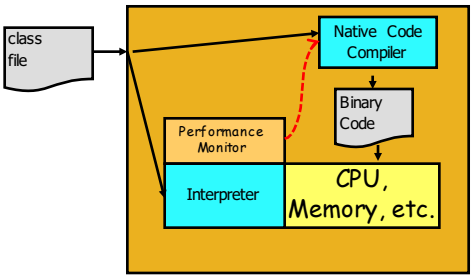
Example

```
class Cow {
  int x = 2;
  public void moo() { x++; }
}

Cow c = ...;
...
c.moo();
```

```
class Cow {
  int x = 2;
  public void moo() { x++;
}

Cow c = ...;
...
c.x++;
```

- If no subclasses of Cow are loaded:
  - No need to perform virtual method lookup.
  - (Can even inline method body…)
- Must undo optimization if a subclass of Cow is loaded after Cow is JIT-compiled

Interpreter with JIT Compiler



HotSpot, SpiderMonkey, …

JIT Example

```
> java -XX:+PrintCompilation RSG 20 < Haiku.g
  49    1             java.lang.String::hashCode  (55 bytes)
  61    2             sun.nio.cs.UTF_8$Encoder::encode  (361 bytes)
  67    3             java.lang.String::indexOf  (70 bytes)
  ...
Grammar is: ...
 rising far away/ Buddhist over clouds Kyushu/ purple in day tree

...

 rushing purple cow/ red yakitori falling/ drifting far away

  85     9           java.lang.String::equals  (88 bytes)
...

 rushing mountains/ red city to hell blue clouds/ faces sky falling

...
```