Java

Lecture 18

Lecture Outline

- Java: COOL on steroids
  - History
- Arrays
- Exceptions
- Interfaces
- Coercions
- Threads
- Dynamic Loading & Initialization
- Summary

Java History

- Began as Oak at SUN
  - Originally targeted at set-top devices
  - Initial development took several years ('91-'94)
- Retargeted as the Internet language ('94-95)
  - Every new language needs a "killer app"
  - Beat out TCL, Python
  - ActiveX came later

The People

- James Gosling
  - Principal designer
  - CMU Ph.D.
- Bill Joy
  - ABD from Berkeley (Unix)
- Guy Steele
  - MIT PhD
  - Famous languages researcher

Influences

- Modula-3
  - types
- Eiffel, Objective C, C++
  - Object orientation, interfaces
- Lisp
  - Java's dynamic flavor (lots of features)

Java Design

- From our perspective, COOL plus
  - Exceptions
  - Interfaces
  - Threads
  - Dynamic Loading
  - Other less important ones . . .
- Java is a BIG language
  - Lots of features
  - Lots of feature interactions

Arrays

Assume B < A. What happens in the following?

```
B[] b = new B[10];
A[] a = b;

a[0] = new A();
b[0].aMethodNotDeclaredInA();
```


Subtyping In Java

B < A	if B inherits from A	as in Cool
C < A	if C < B and B < A	as in Cool
B[] < A[]	if B < A	not as in Cool

This last rule is unsound!

What's Going On?

```
B[] b = new B[10];
A[] a = b;
a[0] = new A();
b[0].aMethodNotDeclaredInA();
```

Having multiple aliases to updateable locations with different types is unsound

The Right Solution

- Disallow subtyping through arrrays
- Standard solution in several languages

B < A	if B inherits from A
C < A	if C < B and B < A
B[] < A[]	if B = A

The Java Solution

- Java fixes the problem by checking each array assignment at runtime for type correctness
  - Is the type of the object being assigned compatible with the type of the array?
- Huge overhead on array computations!
- But note: arrays of primitive types unaffected
  - Primitive types are not classes

A Common Problem

- Deep in a section of code, you encounter an unexpected error
  - Out of memory
  - A list that is supposed to be sorted is not
  - etc.
- What do you do?

Exceptions

- Add a new type (class) of *exceptions*
- Add new forms
  try { something } catch(x) { cleanup }
  throw exception

Example

```
class Foo {
   public static void main(String[] args) {
        try { X(); } catch (Exception e) {
        System.out.println("Error!") } }

   public void X() throws MyException {
        throw new MyException();
   }
}
```


Semantics (pseudo-Java)

T(o) = an exception that has been thrown

o = an ordinary object

$$\frac{E \vdash e_1 \rightarrow o}{E \vdash \text{try } \{ e_1 \} \text{ catch(x) } \{ e_2 \} \rightarrow o}$$

$$\frac{E \vdash e_1 \rightarrow T(o_1) \quad E[x \leftarrow o_1] \vdash e_2 \rightarrow o_2}{E \vdash \text{try } \{ e_1 \} \text{ catch(x) } \{ e_2 \} \rightarrow o_2}$$

Semantics (Cont.)

$$\frac{E \vdash e \rightarrow o}{E \vdash \text{throw } e \rightarrow T(o)}$$

$$\frac{E \vdash e_1 \rightarrow T(o)}{E \vdash e_1 + e_2 \rightarrow T(o)}$$

All forms except catch propagate thrown exceptions

Simple Implementation

- When we encounter a try
  - Mark current location in the stack
- When we throw an exception
  - Unwind the stack to the first try
  - Execute corresponding catch
- More complex techniques reduce the cost of try and throw

Trivia Question

What happens to an uncaught exception thrown during object finalization?

Type Checking

- Methods must declare types of exceptions they may raise

  public void X() throws MyException

  - Checked at compile time
  - Some exceptions need not be part of the method signature
    - e.g., dereferencing null
- Other mundane type rules
  - throw must be applied to an object of type Exception

Interfaces

- Specify relationships between classes without inheritance

```
interface PointInterface { void move(int dx, int dy); }

class Point implements PointInterface {
   void move(int dx, int dy) { … }
}
```


Interfaces

"Java programs can use interfaces to make it unnecessary for related classes to share a common abstract superclass or to add methods to Object."

In other words, interfaces play the same role as multiple inheritance in C++, because classes can implement multiple interfaces

```
class X implements A, B, C { … }
```


Why is this Useful?

- A graduate student may be both an University employee and a student

  class GraduateStudent implements Employee, Student { … }

- No good way to incorporate Employee, Student methods for grad students with single inheritance

Implementing Interfaces

- Methods in classes implementing interfaces need not be at fixed offsets.

```
interface PointInterface { void move(int dx, int dy); }

class Point implements PointInterface {
   void move(int dx, int dy) { … } }
class Point2 implements PointInterface {
  void dummy() { … }
   void move(int dx, int dy) { … } }
```


Implementing Interfaces (Cont.)

- Dispatches e.f(…) where e has an interface type are more complex than usual
  - Because methods don't live at fixed offsets
- One approach:
  - Each class implementing an interface has a lookup table method names → methods
  - Hash method names for faster lookup
    - hashes computed at compile time

Coercions

- Java allows primitive types to be *coerced* in certain contexts.
- In 1 + 2.0, the int 1 is widened to a float 1.0
- A coercion is really just a primitive function the compiler inserts for you
  - Most languages have extensive coercions between base numeric types

Coercions & Casts

- Java distinguishes two kinds of coercions & casts:
  - *Widening* always succeed (int → float)
  - *Narrowing* may fail if data can't be converted to desired type (float → int, downcasts)
- Narrowing casts must be explicit
- Widening casts/coercions can be implicit

Trivia Question

What is the only type in Java for which there are no coercions/casts defined?

Coercions in PL/I

- Let A,B,C be strings of 3 characters.

B = '123'
C = '456'
A = B + C

- What is A?

Threads

- Java has concurrency built in through *threads*
- Thread objects have class Thread
  - Start and stop methods
- Synchronization obtains a lock on the object:
  synchronized { e }
- In synchronized methods, this is locked

Example (from the Java Spec)

```
class Simple {
  int a = 1, b = 2;
  void to() { a = 3; b = 4; }
  void fro() {println("a= " + a + ", b=" + b); }
}
```

Two threads call to() and fro(). What is printed?

Example (Cont.)

```
class Simple {
   int a = 1, b = 2;
   void synchronized to() { a = 3; b = 4; }
   void fro() {println("a= " + a + ", b=" + b); }
}
```

Two threads call to() and fro(). What is printed?

Example (Cont.)

```
class Simple {
   int a = 1, b = 2;
   void synchronized to() { a = 3; b = 4; }
   void synchronized fro() {println("a= " + a + ",
   b=" + b); }
}
```

Two threads call to() and fro(). What is printed?

Semantics

- Even without synchronization, a variable should only hold values written by some thread
  - Writes of values are atomic
  - Violated for doubles, though
- Java concurrency semantics are difficult to understand in detail, particularly as to how they might be implemented on certain machines

Dynamic Loading

- Java allows classes to be loaded at run time
  - Type checking source takes place at compile time
  - Bytecode *verification* takes place at run time
- Loading policies handle by a ClassLoader
- Classes may also be unloaded
  - But poorly specified in the definition

Initialization

- Initialization in Java is baroque
  - Everything in COOL plus much more
  - Greatly complicated by concurrency
- A class is initialized when a symbol in the class is first used
  - Not when the class is loaded
  - Delays initialization errors to a predictable point (when something in the class is referenced)

Class Initialization Procedure (Partial)

1. Lock the class object for the class
   - Wait on the lock if another thread has locked it
2. If the same thread is already initializing this class, release lock and return
3. If class already initialized, return normally
4. Otherwise, mark initialization as in progress by this thread and unlock class

Class Initialization (Cont.)

5. Initialize superclass, fields (in textual order)
   - But initialize static, final fields first
   - Give every field a default value before initialization
6. Any errors result in an incorrectly initialized class, mark class as erroneous
7. If no errors, lock class, label class as initialized, notify threads waiting on class object, unlock class

Features and Feature Interactions

- In any system with N features, there are potentially N^2 feature interactions.
- Big, featureful systems are hard to understand!
  - Including programming languages

Summary

- Java is pretty well done
  - By production language standards, very well done
- Java brings many important ideas into the mainstream
  - Strong static typing
  - Garbage collection
- But Java also
  - Includes many features we don't understand
  - Has a lot of features