CS345H: Programming Languages

Lecture 18: Java

Thomas Dillig

Overview

- Java: Our model OO language
- Arrays
- Exceptions
- Interfaces
- Coercions
- Threads
- Dynamic Loading and Initialization

Java History

- Began as Oak at SUN
  - Originally targeted at set-top devices
  - Initial development from 91-94
- Retargeted as the Internet Language 94-95
- Lesson: Every new language needs a "killer app"
- Java beat out TCL, Python

The People

- James Gosling, principal designer. CMU PhD
- Bill Joy. ABD from Berkeley (UNIX)
- Guy Steele, MIT PhD and famous language designer

Influences

- Modula 3:Types
- Eiffel, Objective C, C++: Object orientation, Interfaces
- LISP: Java's dynamic flavors (lots of them)

Java Design

- From our perspective, what we discussed last time plus
  - Exceptions
  - Interfaces
  - Threads
  - Dynamic Loading
  - Other less important ones ...
- Java is a big language
- Lots of features and feature interactions
- And language is still growing

Arrays

- Let's look at arrays in Java
- Assume B is a subtype of A ($B < A$)
- 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
- $C < A$ if $C < B$ and $B < A$
- $B[] < A[]$ if $B < A$
- Last rule is unsound!

The Right Solution

- Disallow subtyping through arrays
- $B < A$ if B inherits from A
- $C < A$ if $C < B$ and $B < A$
- $B[] < A[]$ if $B = A$

The Java Solution

- Java "fixes" this problem by checking each array assignment at run-time for type correctness
- This means that we can now get type errors at run-time!
- Also, huge overhead on array computations
- Note: Primitive types unaffected since they are not classes in Java

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
  - ...
- What do you do?

Exceptions

- Add a new type (class) of exceptions
- Add new syntactic forms:

```
try { something } catch(x) { cleanup }
trow exception
```

Exceptions Example

- Add a new type (class) of exceptions
- Add new syntactic forms:

```
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();
  }
}
```


Operational Semantics with Exceptions

- $T(o)$ an exception has been thrown
- o = an ordinary object
- Here is a (pseudo) rule for try:

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

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

Semantics (Cont.)

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

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

- All forms except catch propagate thrown exceptions

Simple Implementation

- When we encounter a try, mark current location in stack
- When we throw an exception, unwind the stack to the first try and execute corresponding catch
- More complex techniques reduce the cost of try and throw

Type Checking and Exceptions

- Methods must declare types of exceptions they may raise
  `public void X() throws MyException`
  - Checked at compile time
  - Some exception need not be declared , such as dereferencing null pointers
- Other expected rules for exception constructs, such as throw must be applied to something 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 super class or to add methods to it."
- In other words, interfaces play the same role as multiple inheritance in C++, because classes can implement multiple interfaces

Why is this Useful?

- A graduate student may be both a University employee and a student
  `class GraduateStudent implements Employee, Student {...}`
- No good way to incorporate Employee, Student methods for grad students with single inheritance

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 basic numeric types

Coercions & Casts

- Java distinguished two kinds of coercions & casts:
  1. Widening: Always succeeds (int -> float)
  2. Narrowing: May fail or loose information (float -> int, downcasts)
- Narrowing casts must be explicit
- Widening casts/coercion can be implicit

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
- Threads have class Thread and start and stop methods
- Synchronization obtains a lock on the object:
  `synchronized { e }`
- In synchronized methods, this is locked

Example

```
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's concurrency semantics are very difficult in details, but at least Java attempts to specify them!

Dynamic Loading

- Java allows classed to be loaded at run time
  - Type checking of source code takes place at compile time
  - Bytecode verification takes place at run-time
- This introduces many additional complications, but can be extremely flexible

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 brought many important ideas to the mainstream, such as strong static typing and garbage collection
- But Java also has many features that are hard to understand and lots of features