CS 4120/5120
Introduction to Compilers

Ross Tate
Cornell University

Lecture 24: Miscellaneous Features

Exceptions

- Many languages allow exceptions: alternate return paths from a function
  - null pointer, arithmetic overflow, out of stack/heap space, ...
- Function either terminates normally or with an exception
  - total functions ⇒ robust software
  - normal-case code separated from unusual cases
  - no ignorable encoding of error conditions in result (e.g., null)
- Exception propagates dynamically to nearest enclosing try/catch statement (up call tree)
  - Tricky to implement dynamic exceptions efficiently

Exceptions: goals

1. normal return adds little/no overhead
2. try/catch free if no exception
3. catching exception ~ cheap as checking for error value

- **Static exception tables** (CLU):
  - insight: can map pc to handler in each function.
  - on exception: climb stack using return pc, look up exception handler at each stack frame (binary search on pc)

Example

Source Code

```
int x = 1;
int y = 1;
try {
        int z = foo(x, y);
        y = z*5;
} catch (DivideByZeroException e) {
        y = 0;
}
x = y + 1;
return x;
```

Compiled

```
int x = 1;
int y = 1;
 int z = foo(x, y);
 y = z * 5;
 label: x = y+1;
 return x;
handler: (assumes x, y, and e on stack)
if (e instanceof DivideByZeroException)
        y = 0;
        goto label;
throw e;
```

exceptionless code

Coroutine iterators

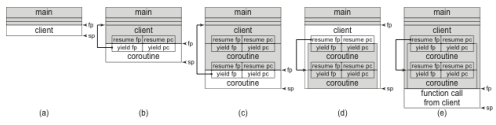
- iteration via coroutines
- Now in C#, Python, Ruby, our JMatch language:

C#

```
public static IEnumerable<T> append<T>(
        IEnumerable<T> left,
        IEnumerable<T> right) {
    if (left != null)
        foreach (T x in left)
            yield return x;
    if (right != null)
        foreach (T x in right)
            yield return x;
}
```


Stack-allocating coroutines

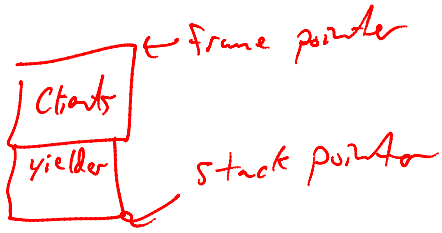
- Client and coroutine share same stack
  - Frame pointer and stack pointer in different stack frames!
  - Can't do this in JVM



Coroutines for Iteration

```
for (x in coll)
    ...
```

7



8

Type Classes

Eq Integer

```
(==) : ∀a. Eq a => a -> a -> Bool
```

9

```
equals<T>(Equator<T>, T, T) : Bool
interface Equator<T> {
    bool eq(T, T);
}
```

10