

Programming Languages and Techniques (CIS120)

Lecture 21

March 17, 2014

Connecting OCaml to Java

Announcements

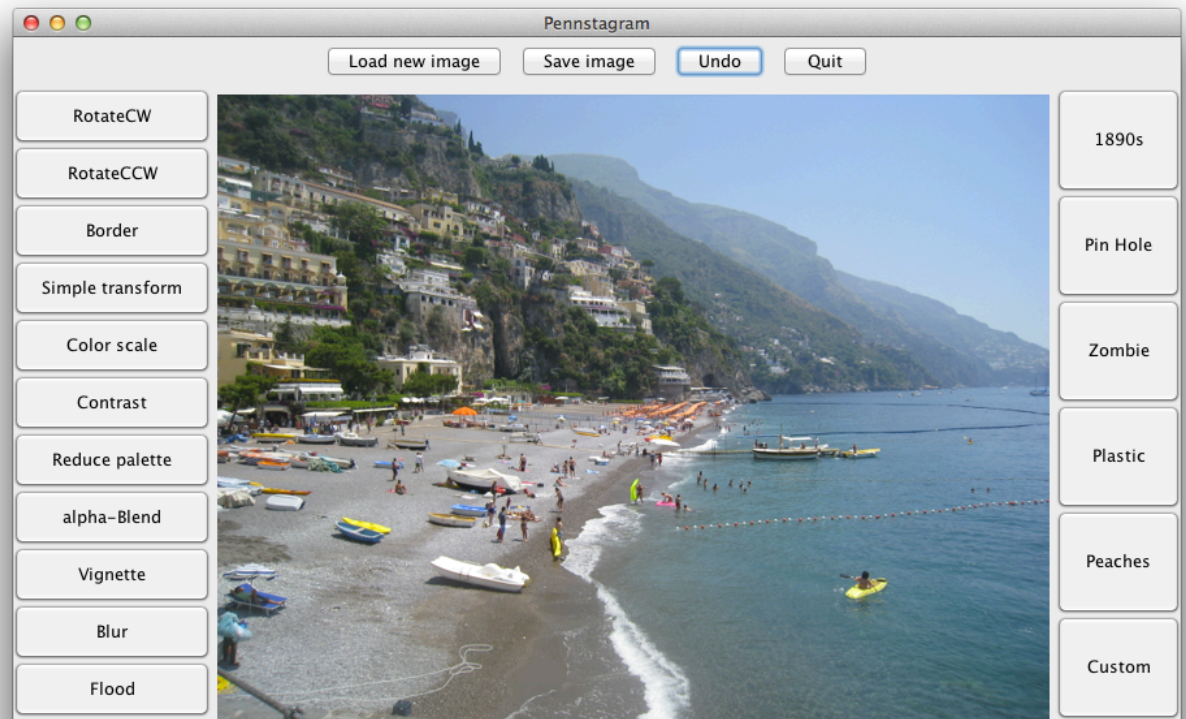
No Weirich OH today -> Wed 1-3PM instead

Read Chapters 19-22 of the lecture notes

HW07 available

- Image processing in Java
- Due Tuesday, March 25th at 11:59:59pm

Pennstagram



What is the value of ans at the end of this program?

```
Counter x = new Counter();  
x.inc();  
int ans = x.inc();
```

1. 1
2. 2
3. 3
4. NullPointerException

Answer: 2

```
public class Counter {  
  
    private int r;  
  
    public Counter () {  
        r = 0;  
    }  
  
    public int inc () {  
        r = r + 1;  
        return r;  
    }  
  
}
```

What is the value of ans at the end of this program?

```
Counter x;  
x.inc();  
int ans = x.inc();
```

1. 1
2. 2
3. 3
4. NullPointerException

Answer: NPE

```
public class Counter {  
  
    private int r;  
  
    public Counter () {  
        r = 0;  
    }  
  
    public int inc () {  
        r = r + 1;  
        return r;  
    }  
  
}
```

What is the value of ans at the end of this program?

```
Counter x = new Counter();  
x.inc();  
Counter y = x;  
y.inc();  
int ans = x.inc();
```

1. 1
2. 2
3. 3
4. NullPointerException

```
public class Counter {  
  
    private int r;  
  
    public Counter () {  
        r = 0;  
    }  
  
    public int inc () {  
        r = r + 1;  
        return r;  
    }  
  
}
```

Answer: 3

Java Core Language

differences between OCaml and Java

Expressions vs. Statements

- OCaml is an *expression language*
 - Every program phrase is an expression (and returns a value)
 - The special value () of type `unit` is used as the result of expressions that are evaluated only for their side effects
 - Semicolon is an *operator* that combines two expressions (where the left-hand one returns type `unit`)
- Java is a *statement language*
 - Two-sorts of program phrases: expressions (which compute values) and statements (which don't)
 - Statements are *terminated* by semicolons
 - Any expression can be used as a statement (but not vice-versa)

Types

- As in OCaml, every Java *expression* has a type
- The type describes the value that an expression computes

Expression form	Example	Type
Variable reference	x	Declared type of variable
Object creation	new Counter ()	Class of the object
Method call	c.inc()	Return type of method
Equality test	x == y, x.equals(y)	boolean
Assignment	x = 5	<i>don't use as an expression!!</i>

Type System Organization

	OCaml	Java
<i>primitive types</i> (values stored “directly” in the stack)	int, float, char, bool, ...	int, float, double, char, boolean, ...
structured types (a.k.a. <i>reference</i> <i>types</i> — values stored in the heap)	tuples, datatypes, records, functions, arrays <i>(objects encoded as records of functions)</i>	objects, arrays <i>(records, tuples, datatypes, strings, first-class functions are a special case of objects)</i>
<i>generics</i>	'a list	List<A>
<i>abstract types</i>	module types (signatures)	interfaces (flexibility) public/private modifiers (encapsulation)

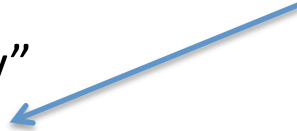
Arithmetic & Logical Operators

OCaml	Java	
=, ==	==	equality test
<>, !=	!=	inequality
>, >=, <, <=	>, >=, <, <=	comparisons
+	+	addition (string concatenation in Java)
-	-	subtraction (and unary minus)
*	*	multiplication
/	/	division
mod	%	remainder (modulus)
not	!	logical "not"
&&	&&	logical "and" (short-circuiting)
		logical "or" (short-circuiting)

Equality

- like OCaml, Java has two ways of testing reference types for equality:
 - “pointer equality”
`o1 == o2`
 - “deep equality”
`o1.equals(o2)`
- Normally, you should use `==` to compare primitive types and “`.equals`” to compare objects

every object provides an “equals” method that “does the right thing” depending on the type of object



Strings

- `String` is a *built in* Java class
- Strings are sequences of characters
"`"` "`Java`" "`3 Stooges`" "`富士山`"
- `+` means String concatenation (overloaded)
"`3`" + "`"`" + "`Stooges`" \Rightarrow "`3 Stooges`"
- Text in a String is immutable (like OCaml)
 - but variables that store strings are not
 - `String x = "OCaml";`
 - `String y = x;`
 - Can't do anything to `x` so that `y` changes
- **Always use `.equals` to compare Strings**

New: Operator Overloading

- The meaning of an operator is determined by the *types* of the values it operates on
 - Integer division
 $4/3 \Rightarrow 1$
 - Floating point division
 $4.0/3.0 \Rightarrow 1.3333333333333333$
 - Automatic conversion
 $4/3.0 \Rightarrow 1.3333333333333333$
- Overloading is a general mechanism in Java
 - we'll see more of it later

Style

```
public class Turtle {  
    private Turtle Turtle;  
    public Turtle() { }  
  
    public Turtle Turtle (Turtle Turtle) {  
        return Turtle;  
    }  
}
```

<http://www.cis.upenn.edu/~cis1xx/resources/codingStyleGuidelines.html>

Static Methods

aka “functions”

Static methods: by example

```
public class Max {
```

```
    public static int max (int x, int y) {  
        if (x > y) {  
            return x;  
        } else {  
            return y;  
        }  
    }
```

closest analogue to
functions in OCaml

```
    public static int max3(int x, int y, int z) {  
        return max( max (x,y), z);  
    }  
}
```

Internally, call with just
the method name

if then and else cases must
be statements

return statement
terminates a method call

```
public class Main {
```

```
    public static void  
        main (String[] args) {  
  
        System.out.println(Max.max(3,4));  
        return;  
    }  
}
```

Externally, call with
name of the class

mantra

Static == Decided at *Compile* Time

Dynamic == Decided at *Run* Time

Static vs. Dynamic Methods

- Static Methods are *independent* of object values
 - Cannot refer to the local state of objects (fields or dynamic methods)
- Use static methods for:
 - Non-OO programming
 - Primitive types: `Math.sin(60)`, `Integer.toString(3)`, `Boolean.valueOf("true")`
 - “public static void main”
- “Normal” methods are *dynamic*
 - Need access to the local state of the object on which they are invoked
 - We only know at *runtime* which method will get called

```
void isTheTrueString (Object o) {  
    o.equals ("The True String");  
}
```

Method call examples

- Calling a (dynamic) method of another object that returns a number:

```
x = o.m() + 5;
```

- Calling a static method of another object that returns a number:

```
x = C.m() + 5;
```

- Calling a method of another class that returns void:

Static

```
C.m();
```

Dynamic

```
o.m();
```

- Calling a static or dynamic method of the same class:

```
m(); x = m() + 5;
```

- Calling (dynamic) methods that return objects:

Watch for null!

```
x = o.m().n();  
x = o.m().n().x().y().z().a().b().c().d().e();
```