On to Java!

Corky Cartwright

Department of Computer Science

Rice University

From Racket to Java

- Racket and Java look completely different
- Don't be fooled. Java is very Racket-like underneath (perhaps excessively so).
  - Self-identifying data
  - Implicit sharing of objects (discouraging mutation); assignment does not copy!
- C++ vs. Java?
  - In the Rice curriculum, C# is little-used.
  - In industry, Java is still dominant. The flexibility of open source is more important than the first-class generics offered in C#.
- DrRacket → DrJava

Java Notation

Breezy Overview of Java

- Syntax is wordy and rather ugly. Lots of warts thanks to C/C++ heritage. I presume everyone in this class already knows how to program in Java (perhaps with bad taste).

A functional programmers view of Java:

- What is a Java program? A collection of classes.
- What is a class? Rough answer: a Racket struct on steroids. Instead of writing functions that manipulate structs, you add "methods" to a class.
- All Java code belongs to some class.

Guiding Vision

- Program design in Java is *data-directed*. Design the data abstractions first; they will determine the structure of the code. In OOP circles, this data design process is often called *object-modeling.*
- Software development is incremental and test-driven. Essentially the same design recipe.
- Key to OO approach: common data and programming abstractions are codified as *design patterns*.

Secondary Theme: DrJava

- DrJava, our lightweight, reactive environment for Java, was created specifically to foster learning to program in Java.
- DrJava facilitates *active learning*; with DrJava learning Java is a form of *exploration.*
- DrJava is not a toy; DrJava is developed using DrJava. It includes everything that we believe is important and nothing more.

Remainder of Lecture is Review

- Chapter 1 of my OO Design Notes presents an expository summary of core Java from a functional point of view. Skim it except for sections that cover aspects of Java program design that you have not seen in detail before, e.g., the visitor pattern.
- Since I suspect nearly all of you have seen essentially all of the Java material before, I am going to breeze through it in lecture.
- Important take-away. Note how I use familiar Java constructs in perhaps unfamiliar ways to support a functional programming perspective.

What Is an Object?

- Collection of *fields* representing the properties of a conceptual or physical object.
- Collection of operations called *methods* for observing and *changing* the fields of the object. Mutation is available, but should be used sparingly.

These fields and methods often called the *members* of the object.

How Are Objects Defined?

- All objects are created using templates (cookie cutters) just like Racket structs.
- Instead of writing **`define-struct`** statements, we write class definitions.
- Since all code is contained within a class, class definitions tend to be much richer (and more complex in real world examples) tha **`define-struct`** statements. After all, the code that would be written in function definitions in Racket must be written as methods of some class.

Example: a Phone Directory

- Task: maintain a directory containing the office address and phone number for each person in the Rice Computer Science Dept.
- Each entry in such a directory has a natural representation as an object with three fields containing a person's
    - name
    - address
    - phone number

  represented as character strings (no symbols in Java).

Summary of `Entry` Format

- Fields:
  - `String name`
  - `String address`
  - `String phone`
- Implicitly generated methods (in DrJava):
  - `String name()`
  - `String address()`
  - `String phone()`

Entry Demo in DrJava

- Create an object
- How do perform any computation with it?

Java Method Invocation

- A Java method `m` is executed by sending a *method invocation (method call)*

  ```
  o.m()
  ```

  to an object `o`, called the *receiver*. The method `m` must be a *member* of `o`. The
- The code defining the method `m` can refer to the receiver using the keyword `this`.

Method Invocation Demo

- Apply some auto-generated methods to an `Entry`
- How do we build up expressions from method invocations?
  - Apply operators (built-in to Java)
  - Invoke methods

Java Expressions

- Java supports essentially the same expressions over primitive types (`int`, `float`, `double`, `boolean`) as C/C++.
- Notable differences:
  - `boolean` is a distinct type from `int`
  - no unsigned version of integer types
  - explicit `long` type

Defining (Instance) Methods

- Recall our definition of the **Entry** class. How can we add methods to this class?
- Suppose we want **Entry** to support a method:

```
boolean match(String keyname)
```

  invoked by syntax like

```
  e.match("Corky")
```

  where **e** is an **Entry**.

Method Definition Demo

- Comment notation:
  - `//` opens a line comment
  - Block comments are enclosed in `/* … */`

Code for Entry with match

```
class Entry {
  /* fields */
  String name, address, phone;

  /** return true iff name matches keyName.*/
  boolean match(String keyName) {
    return keyName.equals(name);
  }
}
```

For Next Class

Reading: OO Design Notes, Ch 1.