Object oriented languages

- Start of year mentioned three paradigms: imperative, functional, and OO
- Covered functional in some detail, trust you've got reasonable background in use of imperative and OO
- Want to spend some time looking at relevant aspects of OO

OO Paradigm

- Based on collections of interacting objects, each with its own fields and methods, where an object's methods have access to that objects fields
- Interaction between objects handled by some form of communication (e.g. method calls)
- In pure OO, everything is an object – there are no stand-alone functions or primitive data types
- Most "OO" languages are actually hybrids (e.g. C++, Java, etc), there are a few pure OO languages (e.g. Smalltalk, Ruby)

OO features

- Begins with abstract data types, i.e. encapsulating data and the operations on it, with public interface and hidden implementation
- Adds inheritance, in which objects inherit fields/methods from ancestor classes, possibly over-riding with local replacements
- Adds dynamic dispatch, in which method calls are bound to actual code at run time, rather than statically

Inheritance

- Allows top-down design of object heirarchy, with successively greater specialization at each level
- Improves modularity, abstraction, reuse, maintainability
- Possible marginal loss of time/space efficiency
- May include features for access/permission restriction of inherited fields/methods
- May include provisions to override inherited fields/methods with local versions
- Might allow multiple inheritance (multiple immediate parent classes)

Dynamic dispatch

- Called method is determined at run time, not compile time
- Suppose we have ParentClass and ChildClass, each with their own version of method foo, and the following method/call somewhere,

  When f runs, which version of foo should be executed? Static dispatch would use ParentClass, dynamic dispatch would use ChildClass

```
void f(ParentClass x) { x.foo(); }

....

ChildClass c;

f(c);
```