

Announcements

- **Office hours are there to be used**
 - I'm at my desk waiting, but nobody has shown up yet.
 - You don't need a specific question. It's ok to come and say "I don't get lambda calculus."
 - The TA also has office hours.
- **Expectations for project**
 - You're expected to write a compiler.
 - How to get started
 - Homework 2 is almost completely about that.
 - If you still can't get started, talk to the TA or me.
 - How to get partial credit
 - Need a working compiler, but it can do only part of the work.
 - To simplify:
 - Remove arrays from the language
 - Remove recursion from the language
 - Remove functions and procedures from the language.
- **Expectations for midterm**
 - Written, closed-book exam.
 - Many problems will be of the style in Homework 1.
 - Expect to give some definitions and brief explanations.
 - Think before you write!
 - If you get questions about lambda calculus, the following will be given:
 - Substitution and conversions rules (brief, formal notation)
 - Definitions of Church Booleans, logical operations, cons/car/cdr, and the Y-combinator
 - You will get some questions on material that is in the book but not lectured on.
- **Homework 3 will be posted this afternoon**

Basic terms of OOP

- **Classes**
- **Interfaces**
- **Objects**
- **Member variables**
- **Messages and methods**
- **Inheritance**
- **Polymorphism**
- **Overriding and overloading**
- **Encapsulation and ADTs**
- **Abstract classes**
- **Virtual functions**

History

- **Simula**

- Ole-Johan Dahl and Kristen Nygaard
- Public/Protected/Private
- Virtual methods
- **Smalltalk**
 - Alan Kay, Adele Goldberg
 - Created on a bet that a language with message passing (inspired by Simula) could be implemented in a page of code.
 - All values are objects.
 - Even a class is an object => metaprogramming, reflection.
 - Unlike in Simula/C++
 - An object can hold state, and send and receive messages.
 - State is always private, so all you can do is send a message.
 - Methods are always public
 - Reflection/introspection
 - Metaprogramming: Classes are objects too, and can be manipulated programmatically.
 - Compare to Java Remote Method Invocation (RMI) and to aspect-oriented programming.
 - Very powerful when combined with reflection

Exploring the design space of object-oriented languages

- **Class-based vs. prototype-based**
 - Self and JavaScript are prototype based
 - No classes
 - Objects made from scratch by specifying member variables and methods, or by cloning an existing object.
 - Leads to more focus on the behavior of the program and less focus on the structure and taxonomy of the data.
- **Inheritance**
 - Single vs. multiple inheritance
 - Multiple inheritance can lead to ambiguity
 - What does a certain name refer to (from which ancestor)?
 - Many solutions: disallow conflicts, let child specify, use precedence rules, or allow renaming
 - The solution chosen affects whether a subclass is a subtype (see below)
 - Is a subclass also a subtype?
 - Remember inclusion polymorphism (from previous lecture): A member of a subtype is also considered a member of the supertype.
 - Can give a subtype wherever a supertype is expected without violating type safety.
 - If a subclass hides some of the parent's variables or modifies them in an incompatible way, then it does not create a subtype
 - **Contravariance**: input variables of the subclass method must be supertypes of the corresponding parameters of the overridden method.

- **Covariance:** input parameters of the subclass method must be subtypes of the corresponding parameters of the overridden method.
- Three natural choices:
 - Contravariance in input and covariance in result
 - Yields type safety
 - Emerald, Sather
 - Exact identity of the two methods
 - C++, Java, Object Pascal, Modula-3
 - Covariance in both input and result
 - Facilitates reuse
 - Eiffel, Ada-95
- Subclass: Code inheritance
 - Reuse of code
 - Relationship between implementations
- Subtype: Behavior inheritance
 - Polymorphic code
 - Organization of related concepts
 - Relationship between types/interfaces
- Many languages unify them
 - Single inheritance tree
- Why class hierarchy and interface hierarchy in Java?
- **To be continued.**