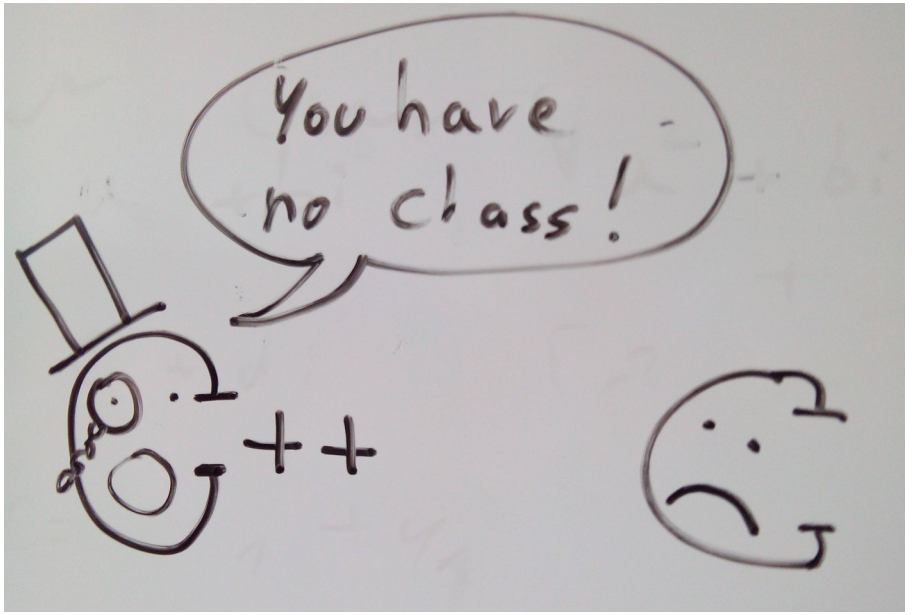
CS 430
Spring 2015

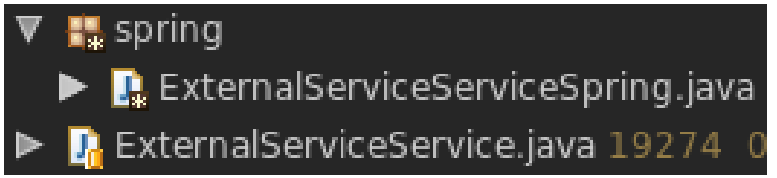
Mike Lam, Professor



Abstraction and Object-Oriented Programming

Abstraction

- Fundamental concept in CS
- Textbook definition: "*a view or representation of an entity that includes only the most significant attributes*"
- Mathematical notion: "*equivalence classes*"
- Practical reality: the first line of defense against software complexity!
- Key: finding the correct level of abstraction



org.apache.xmlrpc.server

Interface RequestProcessorFactoryFactory

All Known Implementing Classes:

RequestProcessorFactoryFactory.RequestSpecificProcessorFactoryFactory, RequestProcessorFactoryFactory.StatelessProcessorFactoryFactory

Types of abstraction

- Process abstraction
  - Structured (block) syntax
  - Subprograms and modules
- Data abstraction
  - Abstract data types and interfaces
    - Polymorphism and generics
  - Encapsulation and information hiding
    - Classes and objects
    - Inheritance

Abstract data types

- An ADT is basically an interface
  - Type specifier for the general category
  - List of supported operations
    - Common operations: constructor, accessors, iterators, destructors
  - Not specified: underlying representation
- Examples
  - List: `append(value), get(index), remove(index)`
  - Stack: `push(value), pop`
  - Set: `add(value), isMember(value), union(otherSet)`
  - Map: `store(key, value), lookup(key)`

Design issues

- Information hiding: should underlying data be exposed?
  - Levels: public, private, protected
  - Public fields vs. getters and setters
  - Convenience/writability vs. safety and extensibility
- Polymorphism: is parameterization possible?
  - Specifying parameters
  - Specifying restrictions on the parameters
  - Power/expressivity vs. readability
- Encapsulation: how is related code and data collected?
  - Header files, namespaces, packages, modules, etc.
  - Modularity and readability
  - Extensibility and inheritance

History of OOP

- Simula: data abstractions for simulation and modeling
- Smalltalk: objects and messages
- C++: originally “C with classes”
- Most modern languages have some form of OOP
  - Abstract data types
  - Inheritance
  - Dynamic binding

Object-oriented programming

- Inheritance
  - Original motivation: code re-use
  - Parent/superclass vs. child/derived/subclass
  - Overriding methods
  - Single vs. multiple inheritance (simplicity vs. power)
  - Abstract methods and classes
  - Non-overridable methods: "final" methods in Java
- Dispatch
  - Static dispatch: all method calls can be resolved at compile time
  - Dynamic dispatch: polymorphic method calls
    - "virtual" methods in C++
- Non-object types in OOP languages
  - "Primitive" or "intrinsic" types

Object-oriented implementation

- Pure vs. hybrid (is everything an object?)
- Class instance record
  - List of member variables for classes
  - Subclass CIR is a copy of the parents' with (potentially) added fields
- Virtual method table
  - List of dynamically-dispatched methods w/ pointers to implementations
  - Often implemented directly (no CIR) with a single VPTR member field in objects

Design issues

- Languages:
  - C++
  - Java
  - Ruby

Abstraction in C++

- Classes and structs
- Header file and implementation file
- Visibility: public (default for structs) or private (default for classes)
  - "Friend" functions for private access outside class
- Stack or heap allocation
- Manual memory management: constructors and destructors
- All forms of polymorphism (parametric via templates)
- Multiple inheritance
- Namespaces for naming and encapsulation

Abstraction in Java

- Classes similar to C++
- Single inheritance tree (rooted at Object)
- No stack allocation (everything on heap)
- Automatic memory management
- Access modifiers required
  - Public, private, protected, package
- No separate header file
- All forms of polymorphism (parametric via generics)
- Packages for naming and encapsulation
- Interfaces for pseudo-multiple inheritance

Abstraction in Ruby

- Dynamic classes
- Members can be added/removed at run time
- Multiple definitions of a single class allowed
- Keywords for function visibility (public by default)
- All data is private
  - "@" symbol for instance variables
  - Attributes accessed through methods
- Polymorphism via dynamic types; no overloading
- Modules for encapsulation and multiple inheritance (mixins)

Announcements

- Talk Wed April 15 (12:15pm, nTelos Room)
  - Graphics and high-performance computing
  - Dr. Amitabh Varshney
- No office hours tomorrow (4/15) or next Mon-Tue (4/20-4/21)
- Mark your calendars and plan to attend class 4/28 (final presentations) and 4/30 (review)