Lecture 24: Classes, Traits & Inheritance

CSC 131
Spring, 2019

Kim Bruce

Dialects in Grace

- Can add features as well as remove.
  - Add library
- Checker can enforce restrictions on code
  - Grace code running on AST rep of program
  - Beginning Programmer (use only simple constructs)
  - RequiredTypes (all id's must be associated with type)
  - Static type checker.

Grace Details

- No parens needed w/parameterless methods
- No parens if parameter bounded by "" or {}
- Must insert parens for most precedence
- Use blocks for code evaluated variable number of times

Traits

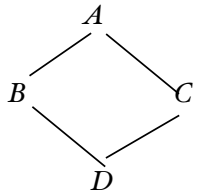
Inheritance

- Subclass "inherits" all methods and features from superclass.
  - Can override inherited methods
  - Can add new methods & features
- Java interfaces can extend multiple interfaces
  - Classes can't

Multiple Inheritance

- Appealing idea, but no good designs/ implementations (Eiffel's is nicest)
  - "Multiple inheritance is good, but there is no good way to do it." [Alan Snyder paraphrase]
- Java used interfaces to get some of benefits
- Traits are good intermediate approach.
  - Related to "mixins" from Flavors (& hence CLOS)

Diamond Problem



- For methods
  - m defined in A, overridden in B,C
- For instance variables
- Methods easier — select one, explicitly or implicitly, call other with extra syntax
- Instance variables harder: Need to keep state around for methods. Duplicate or share?
  - Initialization still problem! What order?

Traits in Grace

- Prime use case: Combine dialects
- Avoid problems with variables
- Traits like classes, but no explicit state
- Adapt solution proposed for Smalltalk
  - https://rmod.inria.fr/archives/papers/Blac03a-OOSPLA03-TraitsHierarchy.pdf

What is a trait?

- Traits are objects that directly contain no field declarations, inherit statements, or executable code.
- Traits can contain methods, types, traits, and classes
- Defined by using keyword trait:
  - trait emptiness {method isEmpty {size == 0}; ... }
  - objects w/certain restrictions

Traits in Grace

- If any ambiguity, programmer must select which method to inherit
  - Typically by excluding redundant ones
- Can gain access to original inherited methods
  - alias newName(_) is oldMeth(_)

Inheritance in Grace

- Class can inherit from one superclass (if none, then by default graceObject)
- Class can use any number of traits
- See examples in traits.grace
- Traits can be used as “mixins”
  - Require clause states what is needed to complete them.

Conflicts!

- What if there are conflicts?
  - See Conflicts.grace
  - trait overrides subclass
  - if traits conflict then must exclude all but one or override in new subclass

History of Traits

- Originally in Self (1986): object-based language using delegation (rather than inheritance)
- Theoretically nicer versions proposed for Smalltalk and implemented in Scala

Attaining Grace

- Seems to work well with novices
  - Grades generally higher than with Java
  - Higher percentage persist in follow-up courses.
- Nearly impossible to convince novices it is a good idea!!

Contributions of Grace

- Simple w/ minimal "accidental" complexity
- Support for blocks as lexical closures (anonymous function)
  - Define new control constructs
- Dialects very helpful
- Optional typing not as useful as expected.

Evaluating OOLs

Evaluation of OOL's

- Pro's (e.g., with Eiffel and Java)
  - Good use of information hiding. Objects can hide their state.
  - Good support for reusability. Supports generics.
  - Support for inheritance and subtyping provides for reusability of code.
  - Great support for use of frameworks
    - Plug in code to provide behavior
      - Subject-Observer, etc.

Evaluation of OOL's

- Con's
  - Loss of locality.
  - Type-checking too rigid, unsafe, or requires link time global analysis. Others require run-time checks.
  - Semantics of inheritance is very complex.
    - Small changes in methods may make major changes in semantics of subclass.
    - Must know definition of methods in superclass in order to predict impact on changes in subclass. Makes provision of libraries more complex.
  - Weak or non-existent support of modules.

Concurrent & Parallel Programming Constructs

Parallelism vs Concurrency

- Parallel programming is about using additional computational resources to produce an answer faster.
- Concurrent programming is about correctly and efficiently controlling access by multiple threads to shared resources.
  - Includes providing reasonable response times.

Definitions by Dan Grossman

Why Important

- Speed-ups limited w/single processors
  - dual/quad/oct processors now standard
- Required for distributed processing
- Concurrency required for event-driven programming

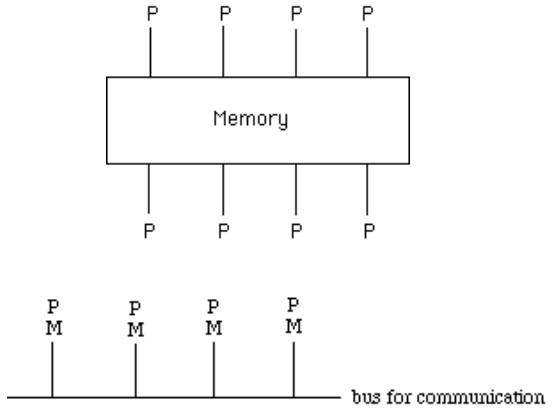
Processes vs Threads

- Processes are independent, process may be composed of multiple threads
- Processes contain separate state info, while threads w/in process share same state/memory
- Context switching between threads much cheaper than between processes.

Flavors of Concurrency

- Multiprogramming -- interleaving on 1 computer
- Multiprocessing -- parallel computation
- Codes:
  - M - Multiple
  - S - Single
  - I - Instruction (*now P for Program*)
  - D - Data
- MIMD most interesting from CS point of view

Shared Memory vs Distributed Models



Problems

- Threads/processes need to
  - Synchronize with other threads
  - Communicate data
- Shared Memory:
  - Synchronization of memory accesses
    - *See ATM1/2 programs*
  - Mutual Exclusion: Reader-Writer problem
- Distributed
  - Asynchronously send and receive messages

OS Responsibilities

- Create and destroy processes
- Schedule processes on one or more processors
- Implement mutual exclusion
  - *for shared memory*
- Create & maintain communication channels
  - *for distributed*

Key Concepts in Conflicts

- Critical Section
  - *where two processes can access shared resource*
- Race condition
  - *answer depends on order of execution of other events*
- Mutual exclusion
  - *allow only one process in critical section*
- Deadlock
  - *no process can proceed because cannot obtain needed locks*

Dining Philosophers

demo