Introduction to Object Oriented Programming

CMSC 331

Concept of Abstraction

• “An abstraction is a view or representation of an entity that includes only the attributes of significance in a particular context.” ~Sebesta, pg. 434

  – Process abstraction
  – Data abstraction

Encapsulation

• Beyond a certain size, process-abstraction is no longer sufficient to organize large programs
• Modularization – the organization of code into logically related processes and data
• Encapsulation – organization of … into a separately or independently compilable unit

Data Abstraction

• Abstract Data Type (ADT) is an encapsulation containing the data representation and methods for one specific data type.
Data Abstraction: Formally

• An abstract data type is a data type that satisfies the following two conditions:
  – The representation, or definition, of the type and the operations on objects of the type are contained in a single syntactic unit. Also, other program units may be allowed to create variables of the defined type.
  – The representation of objects of the type is hidden from the program units that use the type, so the only direct operations possible on those objects are those provided in the type's definition.

Advantages

• Helps to organize program into logical units.
• Reduces code interdependencies by creating interface 'boundaries' – enhancing modification and reuse.
• Improves reliability by keeping details 'hidden'.

Example

• Simple Data Types
  – Double
• More Complex
  – Stack, Queue, Hashtable
• Very Complex
  – Tax Return

Design Issues

• Facilities must be present for encapsulation of type definition and methods
• Basic objects should have minimal definitions
• Some entities will be common to most objects
### ADT in C++
- Both imperative and object type models
- ADTs implemented as Classes, based on Classes of Simula67
  - Data elements are *data members*
  - Functions are *member functions*
- Can be static, stack- or heap-dynamic, and have heap-dynamic members
- Member functions can be defined in place or by header only
- Members are private or public

### ADT in Java
- ADTs implemented as Classes
- All Class instances are heap-dynamic
- Methods are *only* defined in the context of Classes
- Private and public modifiers are applied directly to members
- Java adds the package, and package scope (like `friend`), as an encapsulation construct beyond class

### Object Oriented Programming
- Support for:
  - Abstract Data Types
  - Inheritance
  - Dynamic Binding
- Precursors:
  - Procedure oriented programming
  - Data oriented programming

### Inheritance
- ADTs can be derived from existing ADTs
- Properties of parent class *(superclass)* are inherited *(by the subclass)*
- Child Classes can add new entities, or override entities from the parent
Inheritance: Benefits

• Facilitates reuse
• Allows structuring of Classes into logical hierarchies

Polymorphism

• Polymorphic variables can hold a value of more than one type
  – e.g. if B is a derived Class of Class A, and variable x has type A, x can hold a value of type B as well

Polymorphism: Benefits

• Allows for more flexible code
• Supports operations on generic objects
• Provides for use of virtual classes

• Drawback: Does make type checking more difficult; requires more caution at runtime

Application of OO Language

• “An executing program in an object-oriented language can be described as a simulation of a collection of computers that communicate with each other through messages.” (Sebesta p. 463)
• Invoking the method of a class is ‘sending a message’
• Collection of methods define a ‘message protocol’
• Identify the objects and interactions in your problem, and simulate them
Design Issues

- Exclusivity of Objects
- Are Subclasses Subtypes?
- Implementation and Interface Inheritance
- Type Checking and Polymorphism
- Single and Multiple Inheritance
- Allocation and Deallocation of Objects
- Dynamic and Static Binding

Issues: Exclusivity of Objects

- In purest form, 'everything' is an object
  - Benefit of elegance in simplicity
  - Drawback in that message interface is slower
- Compromises:
  - Retain a complete, imperative typing model, add object model
  - Retain imperative model only for primitive, scalar types

Issues: Subclass = Subtype?

- Generally, this reduces to the question of whether a subclass inherits all entities of the parent class
  - Where entities are overridden, they should be compatible

Issues: Inheritance

- Interface Inheritance
  - Only the interface of the superclass is seen by the subclass
- Implementation Inheritance
  - The implementation details of the superclass are available to the subclass
Issues: Type Checking

- Strong (static) typing implies
  - Checking message parameter types against formal parameters
  - Checking return types against expected return type
- Dynamic type checking delays check until method call; more expensive, and delays error detection

Issues: Single/Multi-Inheritance

- Multiple inheritance is of clear value when derived classes should have properties of more than one parent
- Drawbacks:
  - Complexity (e.g. name collisions)
  - Perceived problem of efficiency

Issues: Allocation

- Allocation
  - Should objects be static, stack-dynamic, or heap-dynamic
- Should deallocation be implicit or explicit?

Issues: Dynamic/Static Binding

- Dynamic binding an essential part of an OO language
- Alternative; allow user to specify whether dynamic or (faster) static binding is to be used
Summary

• OO = ADT + Inheritance + Dynamic Binding
• OO Languages support this with Classes, methods, objects, and message passing
• Some languages, like C++, are hybrid
• Others, like Java support only OO

A Few Words on Java…

CMSC 331

Java is…

• Object oriented!
• Portable!
• The language of the WWW!

OO

• The Java programming language provides support for:
  – Abstract Data Types (classes)
  – Inheritance (single, with some extensions)
  – Polymorphism
Portability

• Java is a hybrid language
  – Source is compiled to byte code
  – Byte code is interpreted by a Java Virtual Machine (JVM)
• JVM has been ported to many platforms
• Byte code for a Java program should run on any JVM
• Programs are not 100% portable

Memory Management

• JVM implements garbage collection
  – No explicit deallocation of objects
  – Unreferenced objects are periodically recycled automatically

Self-Documenting Code

• Java supports the writing of self-documenting code:
  – A simple markup language for use in source code, which allows embedded html
  – A documentation compiler, ‘javadoc’, which creates easy to read html Class documentation.

Everything is a Class

• From basic types, to programs
• Language contains certain primitive types
  – int, double
  – Arrays
  – …
• Primitive data types have analogs in the Class hierarchy
  – Integer, Double, …
Class Hierarchy

• Everything is an Object

Single Inheritance

• A class may be 'derived' from only one parent class
  – Class A extends Class B
• Java improves on this by allowing interfaces (partially defined classes); several interfaces can be used to constrain a new Class
  – Class A implements Classes C, D and E

Java Standard Libraries

• Many useful Classes for:
  – Graphics/Windowing (AWT, Swing)
  – Internet Programming (Applet)
  – Math
  – Basic Data Structures (Vector, HashTable)

Packages

• Classes can belong to named collections, called packages
• Source/Classes must be in a directory on your classpath
• Named subdirectories correspond to named packages:
  – Classpath = C:\Java\myfiles
  – Subdirectory C:\Java\myfiles\examples\new contains Classes of package examples.new
Exception Handling

• Error conditions generate *exceptions*
• Like everything else in Java, exceptions are Classes
  – Predefined
  – User defined
• Exceptions can be *thrown* by a method, or *caught* and *handled*

Security

• Execution inside a virtual machine makes it possible to tightly control access to machine resources
  – Disk
  – Network
  – …
• There are facilities for digitally signing code

Jar

• Classes can be bundled together in jar files
• Jar works just like tar (Unix)
• Jar files can be used as libraries

Threading

• Java provides good support for writing multi-threaded programs, and managing concurrency.
  – Thread Class
  – Low-level synchronization primitives
Memory? What Memory?

- No pointers
- No pointer arithmetic
- No memory hassles

- Only references to Class instances