6. Data Abstraction and Object Orientation

Object-Oriented Programming, Encapsulation and Inheritance, Initialization and Finalization, Dynamic Binding, Multiple Inheritance
Generally agreed to have three components
1. Encapsulation
2. Inheritance
3. Dynamic binding / polymorphism

Languages using encapsulation alone are called object-based
Encapsulation

- Covered earlier – modularization
- Quick review
  - Module as manager
  - Module as type – class
- Inheritance can be added to either (Ada, Java)
- Inheritance can exist *without* encapsulation (Simula 67, Python)
  - Encapsulation is generally considered too important to give up
- Most common way is to add inheritance to class-based approach (Module as type)
Define a new class that inherits from (extends, is derived from, is based on) the original

New class automatically inherits data and methods from the original

In the new class, you only need to add
  - Any new instance variables (fields, member data) and methods (member functions)
  - Any changed methods
Both C++ and Java use constructors, which have the same name as the class.

Python uses a specially named constructor.

Eiffel uses functions with ordinary names, but specified as creation functions.

C++ also has destructors, named with ~ before class name.

Java calls “destruction” method finalize, but role is different, because of garbage collection.

- Java’s finalize method cannot be counted on to run at all, much less at a special time.
- Sometimes being able to dispose of resources yourself is a good thing.
- Every instance of a subclass is also an instance of the superclass
  - You can use a subclass object wherever a superclass object is expected
  - Example:

```java
void transfer(Account first, Account second, double amount)
{
    CheckingAccount c;
    SavingsAccount s;
    transfer(s, c, 200);
}
```

**Polymorphic variables**

Method expects two arguments of type account, but works on any kinds of account
If a method has been overridden, the definition used is determined by the actual runtime type of the object

- Example:

```java
SavingsAccount s = new BonusAccount(100);
s.deposit(200);
```

- Above, the BonusAccount version will be used, because the current value of s is a BonusAccount (the variable is polymorphic)
Issues

- Inheritance works poorly with value model
  - Disallow values – references only (Java, Python)
  - Allow both values and – through pointers or reference variables – references, but slicing occurs if values used (C++)

- Late binding
  - Smalltalk: the only mode
  - Java: the default, but can declare `final` to avoid it
  - C++: must declare `virtual` to get late binding
Account
<<interface>>

- **BankAccount**
  - <<abstract>>
  - Handles basics common to all implementations, including data validation
  - Gives interest

- **SavingsAccount**
- **CheckingAccount**
- **BonusAccount**

- Anything that has a **balance**, to which you can **deposit**, from which you can **withdraw**

- Other kinds of accounts

- Applies **fees** for withdrawal

- Applies **bonus rate** for high balance
Implementations

- **Java**
  - Use both interfaces and abstract classes – both are part of language
  - Methods automatically use dynamic binding

- **C++**
  - Use an abstract class with *pure virtual functions* for the interface
  - Use an abstract class with some but not all functions implemented for the abstract account
  - Methods must be declared virtual to get dynamic binding, and must be applied to pointers or references to objects

- **Python**
  - No true encapsulation and no interfaces or abstract classes in language – otherwise similar to Java (Just be good!)
Some languages allow *multiple inheritance*

- A class can inherit from more than one superclass
  - *Example*: checking with interest from both savings and checking
  - But both savings and checking inherit from plain bank account

Raises problems

- *Example*: each account has a balance. Does checking with interest have *two* balances? *How many times is the bank account class inherited?*

Possibilities

- Use common subclass only *once* – Eiffel
- Allow user to decide – C++
  - *Virtual* inheritance uses subclass only once
- Punt: Disallow data in multiple inheritance – Java interfaces
  - But note that the above example not included
Class D inherits from both class B and class C, and both of the latter inherit from class A.