Inheritance (language independent)

- First view: exploit common interfaces in programming
  - Iterator and Comparable in Java, see List/ArrayList/Vector
  - Iterators in STL/C++ share interface by convention/templates
  - Implementation varies while interface stays the same
- Second view: share code, factor code into parent class
  - Default toString in Java
- Polymorphism/late(runtime) binding (compare: static)
  - Actual function called determined when program runs, not when program is compiled

Inheritance guidelines in C++

- Inherit from Abstract Base Classes (ABC)
  - one pure virtual function needed (=0)
  - Subclasses must implement, or they’re abstract too
  - must have virtual destructor implemented
  - can have pure virtual destructor implemented, but not normally needed
- Avoid protected data, but sometimes this isn’t possible
  - data is private, subclasses have it, can’t access it
  - keep protected data to a minimum
- Single inheritance, assume most functions are virtual
  - multiple inheritance ok when using ABC, problem with data in super classes
  - virtual: some overhead, but open/closed principle intact

Inheritance Heuristics

- A base/parent class is an interface
  - Subclasses implement the interface
    - Behavior changes in subclasses, but there’s commonality
  - The base/parent class can supply some default behavior
    - Derived classes can use, override, both
- The base/parent class can have state
  - Protected: inherited and directly accessible
  - Private: inherited but not accessible directly
- Abstract base classes are a good thing
- Push common behavior as high up as possible in an inheritance hierarchy
- If the subclasses aren’t used polymorphically (e.g., through a pointer to the base class) then the inheritance hierarchy is probably flawed

Inheritance Heuristics in C++

- One pure virtual (aka abstract) function makes a class abstract
  - Cannot be instantiated, but can be constructed (why?)
  - Default in C++ is non-virtual or monomorphic
  - Unreasonable emphasis on efficiency, sacrifices generality
  - If you think subclassing will occur, all methods are virtual
- Must have virtual destructor, the base class destructor (and constructor) will be called
- We use public inheritance, models is-a relationship
  - Private inheritance means is-implemented-in-terms-of
    - Implementation technique, not design technique
    - Derived class methods call base-class methods, but no “usable-as-a” via polymorphism
    - Access to protected methods, and can redefine virtual funcs
### Inheritance and Layering/Aggregation

- Layering (or aggregation) means “uses via instance variable”
  - Use layering/attributes if differences aren’t behavioral
  - Use inheritance when differences are behavioral

- Consider Student class: name, age, gender, sleeping habits
  - Which are attributes, which might be virtual methods

- Lots of classes can lead to lots of problems
  - It’s hard to manage lots of classes in your head
  - Tools help, use speedbar in emacs, other class browsers in IDEs or in comments (e.g., javadoc)

- Inheritance hierarchies cannot be too deep (understandable?)

### Inheritance guidelines (see Riel)

- Watch out for derived classes with only one instance/object
  - For the CarMaker class is GeneralMotors a subclass or an object?

- Watch out for derived classes that override behavior with a no-op
  - Mammal class from which platypus derives, live-birth?

- Too much subclassing? Base class House
  - Derived: ElectricallyCooledHouse, SolarHeatedHouse?

- What to do with a list of fruit that must support apple-coring?
  - Fruit list is polymorphic (in theory), not everything corable

### Java inheritance

- By default every class can be a base/parent class, every method is polymorphic. To inherit use `extends` keyword
  - Can change with `final` keyword (similar to const, but not)
  - A class can extend only one baseclass (but see interfaces)
  - Public, protected, private similar to C++, what’s not?

- A class can be an abstract class, `public abstract class Foo`
  - Can’t instantiate (no new Foo()), but can extend
  - A method can be abstract, like pure virtual in C++

- A class implements any number of interfaces
  - Like ABC, but function prototypes only, no state
  - Subclass must implement all methods of interface

### Interfaces, Comparator, Inner classes

- The `java.util.Comparator` interface is used in sorting
  - Different from the `java.lang.Comparable` interface?
  - What must be implemented?

- Suppose we want to change sort in `WordLinesHashProxy`
  - If we change `keySet` to `entrySet` what’s in `ArrayList`?
  - Program compiles/don’t run sorting `Map.Entry` objects
    - How is this different from C++ behavior?

- How can we sort by size of set while still sorting strings?
  - Use anonymous inner class that implements `Comparable`
  - Syntax is strange: create new interface
  - Access local variables, but some rules on parameters