1.00 Lecture 15

Interfaces, or
Wimpy Classes

Reading for next time: Big Java: sections 12.1-12.5

Interfaces

- **Interface is an abstract base class with only:**
  - **Abstract methods**
  - **Constant (static final) data members**
- **Interface is thus a "wimpy superclass": just a set of abstract methods a subclass must implement**
  - **They are a to-do list for a subclass that implements them**
- **A subclass that inherits from the interface must implement all of its (abstract) methods, just as any (concrete) subclass inheriting an abstract method must implement it**
  - **You will use interfaces frequently in Swing (GUI), sensors, numerical methods and data structures**

Interfaces, p.2

- **Interface (wimpy class) is like an abstract class but:**
  - **If Java had only abstract classes, a subclass could only inherit from one superclass**
  - **Multiple interfaces (wimpy classes) can be implemented (inherited) in your class**
  - **Interfaces, such as `Rotatable`, cannot be instantiated**

```
Rotatable shape1= new Rotatable();      // Error
```

  - **You can declare objects to be of type interface**

```
Rotatable shape1;                       // OK
```

  - **They can be names for objects of a class that implements the interface. If `Rectangle` implements `Rotatable`:**

```
Rotatable shape1= new Rectangle();      // OK
```

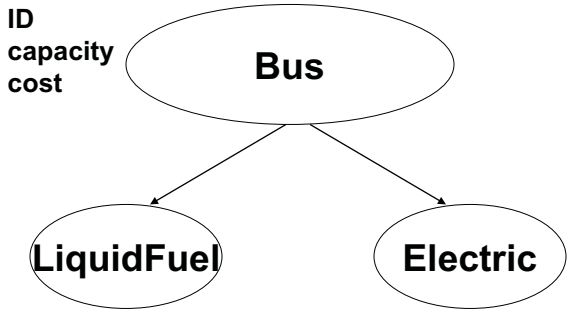
  - **Interfaces contain only abstract methods and constants**

```
public interface Rotatable {
    void rotate(double theta);  // Implicitly public
    double MAX_ROTATE= 360; }   // Implicitly
                                // public static final
```

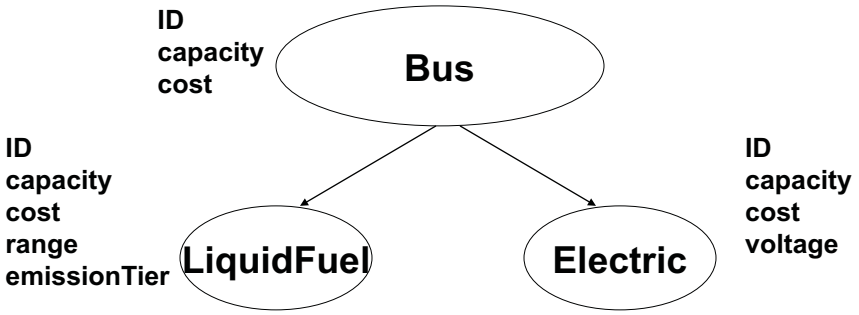
Abstract Classes vs. Interfaces

- **Abstract class has**
  - **Static and instance data members**
  - **Concrete and/or abstract methods**
  - **Single inheritance (via `extends`)**
  - **Constructor**
- **Interface has**
  - **Only static final data members (constant)**
  - **Only abstract methods**
  - **Multiple inheritance (via `implements`)**
  - **No constructor**

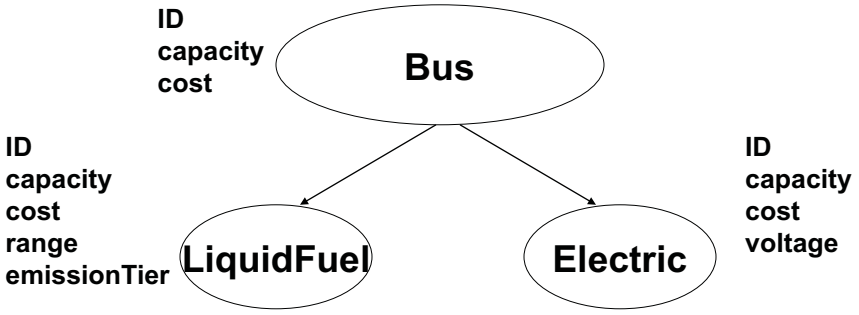
Interfaces and multiple inheritance



Interfaces and multiple inheritance

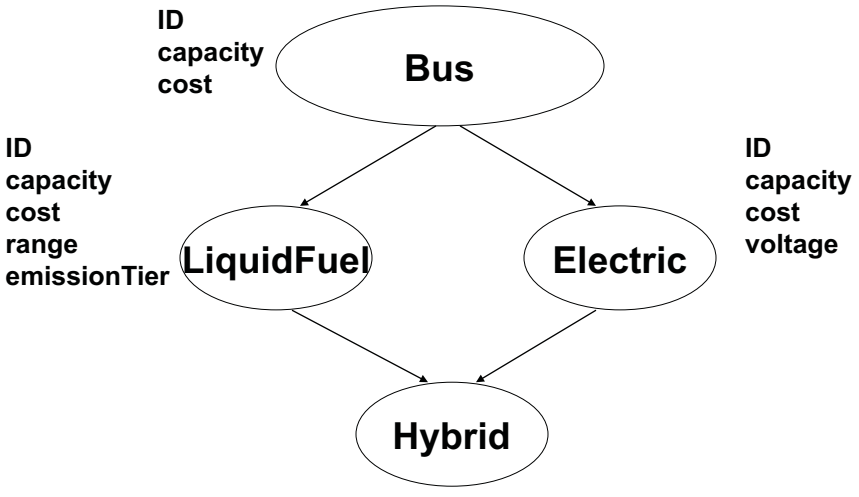


Interfaces and multiple inheritance



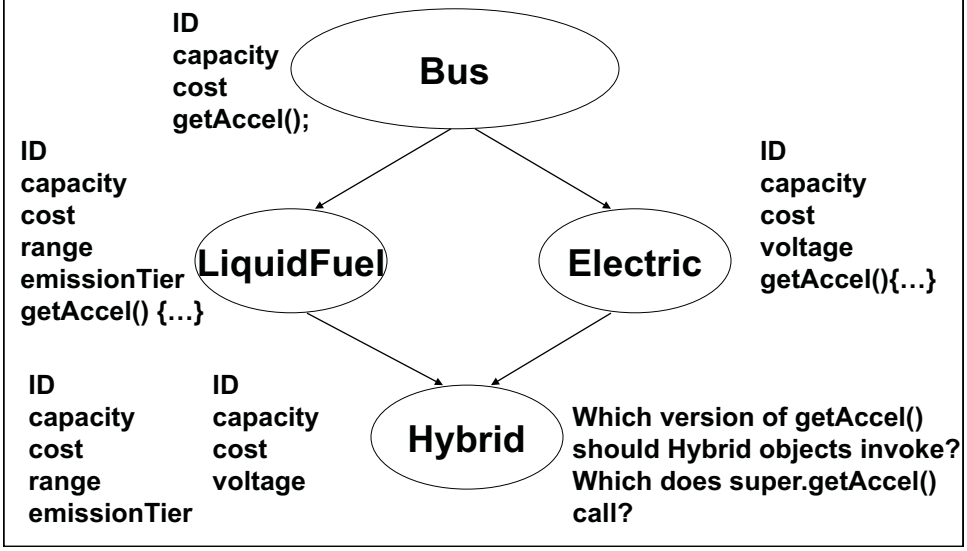
The MBTA Silver Line to the airport is now built, and it uses dual mode buses: electric in the South Station tunnel, and CNG powered the rest of the way to the airport.

Interfaces and multiple inheritance



What member data fields will Hybrid have *(in C++)*?

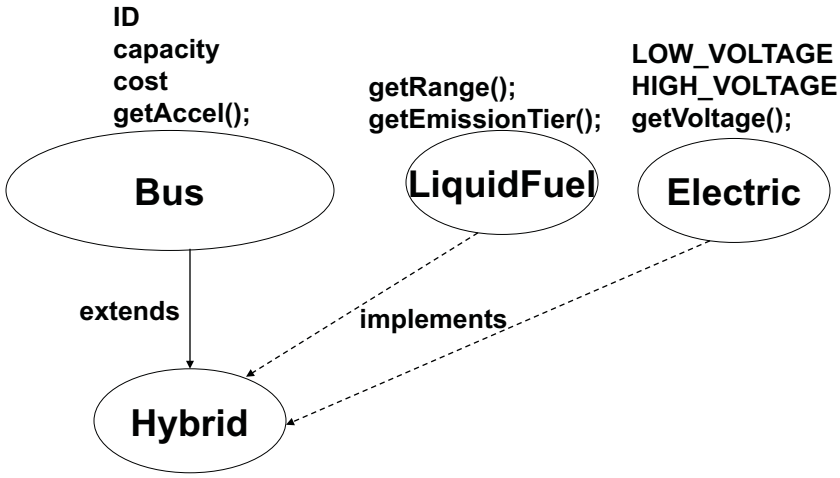
Interfaces and multiple inheritance: methods



Interfaces and multiple inheritance

- Data members in classes with (C++) multiple inheritance can be duplicated in many cases
  - To prevent this, Java does not allow a class to extend more than one class
  - A class may implement one or more interfaces
  - Java allows no data members in interfaces, only `public static final` (constant) fields that don't have this difficulty
- Which method to call in classes with multiple inheritance can be ambiguous
  - Java allows no concrete method bodies in interfaces, only abstract methods that don't allow ambiguity
- Both `extend` and `implements` inherit from a superclass
  - `extend` from a single "full fledged" class
  - `implements` from one or many "wimpy" classes

Java interfaces



Hybrid inherits all fields, methods from super class, interfaces
It must declare range, emissionTier, voltage data members itself
It must have getRange(), getEmissionTier(), getVoltage(), getAccel() bodies
Question: What happens if there is a getRange(); in Bus as well?

Interface exercise

```
// This abstract base class is in your Lecture 15 download

public abstract class Bus {
   private int ID;
   private int capacity;
   private double cost;
   private static int nextID= 1;

   public Bus(int capacity, double cost) {
        ID= nextID++;
        this.capacity = capacity;
        this.cost = cost;
   }
   public abstract double getAccel();
   public final int getID() {return ID;}
   public int getCapacity() {return capacity;}
   public double getCost() {return cost;}
}
```

Interface exercise, p.1

- Download abstract base class Bus (previous slide)
  - Data members: ID, capacity, cost
  - Constructor
  - Abstract method double getAccel(), other getXXX() methods
- Write interface LiquidFuel
  - In Eclipse: File->New->Interface
  - Write two method signatures:
    - double getRange(), int getEmissionTier()
- Write interface Electric
  - In Eclipse: File->New->Interface
  - Write method signature double getVoltage()
  - Define constants HIGH_VOLTAGE=600, LOW_VOLTAGE=480
    - Both are doubles

Interface exercise, p.2

- Write a Hybrid class (File->New->Class, as usual)
  - "extends _____ implements _____, _____"
  - Data members voltage, range, emissionTier (plus inherited)
  - Write a constructor
  - Write getRange(), getEmissionTier(), getVoltage(), getAccel()
    - getAccel() always returns 4.0
- Use Eclipse to help you:
  - After writing the data members, use Source-> Generate Constructor Using Fields
  - Click on the wavy red line under Hybrid and select 'Add unimplemented methods'
    - Eclipse will add the method signatures to your class
    - It will also add an @Override annotation, which checks that the method signature matches the inherited signature

Interface exercise, p.3

- **Write a CNGBus class (a liquid fueled bus)**
  - **Extend, implement appropriately**
  - **Data members: range, emissionTier (plus inherited)**
  - **Write constructor**
  - **Implement inherited abstract methods; getAccel() returns 3.0**
- **Use the same Eclipse features to help you:**
  - **Generate constructor**
  - **Add unimplemented methods**

Interface exercise, p.4

- **Write a BusTest class, with just a main method:**
  - `import java.util.*;` **at line 1 to be able to use ArrayList**
  - **Create one Hybrid and one CNGBus**
    - **CNG bus range 200 miles, emission tier 2, capacity 50, cost $1 million**
    - **Hybrid range 150 miles, emission tier 1, high voltage, capacity 45, cost $1.2 million (for each bus)**
  - **Create an ArrayList**
    - `ArrayList<what type?> arr= new ArrayList<what type? >();`
  - **Add the Hybrid and the CNGBus to the ArrayList**
    - `arr.add(h);`
  - **Loop through the ArrayList and invoke** `getEmissionTier()` **and** `getID()` **on each element, and print out the value**
    - **You must cast object types depending on the approach you use: either**
    - `((LiquidFuel) b).getEmissionTier();  // Or`
    - `((Bus) liq).getID();`

Interface exercise p.5

- Create new class ElectricBus that implements Electric
  - Use extends, implements appropriately
  - Data member voltage
  - Write constructor
  - Implement methods needed
    - getAccel() returns 5.0
- Use Eclipse features to help you:
  - Generate constructor
  - Add unimplemented methods

Interface exercise conclusion

- Last, we create an ElectricBus object in BusTest's main()
  - Low voltage, capacity 55, cost $0.9 million
- We add the ElectricBus object to the BusTest ArrayList
  - We modify the ArrayList in BusTest, if necessary, so it can hold the ElectricBus object as well as the Hybrid and CNGBus.
    - We'll need to have an ArrayList<Bus>
  - Java has a keyword `instanceof`

```
if (b instanceof Electric)
  double v= ((Electric) b).getVoltage();
```

  - Print the voltage and/or emissionTier within the loop over ArrayList in BusTes, as appropriate for each Bus object
    - Hybrid will have both, Electric just voltage, CNGBus just tier

Inheritance- key points

- **Super classes or base classes**
  - **Abstract or concrete**
- **Sub classes or derived classes**
  - **Abstract or concrete**
  - **Inherit all data members and methods from superclass**
- **Method types**
  - **Abstract method: no method body**
  - **Non-abstract method: use superclass version or override**
    - **Use super.<methodName>() to call superclass version of method**
  - **Final method: cannot be overridden**
  - **Constructor: use super() to call superclass constructor**
- **Inheritance mechanisms**
  - **Extends: inherits data members, methods with bodies**
  - **Implements: multiple inheritance using interfaces**
    - **Inherits only method signatures, constants**
- **Access: protected (or private, package or public)**

1.00 / 1.001 / 1.002 Introduction to Computers and Engineering Problem Solving
Spring 2012