Discussion of interfaces*

March 12, 2018

1 What are they?

An interface is a type that you can't create instances of.
Eg:

- `String s=new String();` is fine because `String` is a class.
- `Comparable l=new Comparable();` is not legal because `Comparable` is an interface so you can't actually make instances of it.

In order to get objects which "conform to" the interface, you need a class which implements the interface. Eg: `Comparable l=new String();`

2 What use are they?

We'll discuss two uses for them here.

- Forming groups from disparate classes.
- Implementation substitution and "principle of least priviledge".

2.1 Forming groups

Suppose you have a hierachy of classes with ZooAnimal at the top.

```
ZooAnimal
 |
 + Carnivore
 |   + Crocodillian
 |   + Falcon
 |   + Hawk
 |   + BigCat
 |
 + Herbivore
     + SeedEater
     + Hippo
     + Elephant
     + Butterfly
```

*A lot of this also applies to abstract classes, but they come later

The inheritance hierachy has one set of relationships between the classes, but might not be the only one. Perhaps we also want to be able talk about:

- "ZooAnimals which can be used in performances" — Elephant, Crocodillian, BigCat but not Butterfly and Hippo ?
- "Airborne" — Hawk, Falcon, SeedEater, Butterfly — not Hippo ?
- "Australian Native" — Crocodillian, SeedEater — not Hippo
- ...

We can't always rearrange our inheritance hierachy to naturally support all the groupings we might want (especially in a single inheritance language like Java).

We'll focus on airborne animals. We want to be able to call the following methods on airborne animals:

```
int getMaxHeight();
int getAirSpeed();
...
```

So we'll just make sure that Hawk, Falcon, SeedEater, Butterfly all have those methods. That's important but only solves part of the problem. Let's say we have a method `handleAirborne(...)` which we want to work with all of our airborne animals. We have two options at the moment:

1. ```
   handleAirborne(ZooAnimal z) throws NotActuallyAirborneException {
     if (z instanceof Hawk) {
        ...
     } else if (z instanceof Falcon) {
        ...
     }
     ...
   }
   ```

2. ```
   handleAirborne(Falcon a) {...}
   handleAirborne(Hawk a) {...}
   handleAirborne(Butterfly a) {...}
   handleAirborne(SeedEater a) {...}
   ```

   Option #2 is the better of the two but still is not great.

   - If a new flying animal is added, a new version of the routine needs to be written.
   - There is also the problem that it is not clear whether all of these routines are supposed to be the same or if there are subtle differences. (If a change is made in one does it need to be made in the others as well?)[1]

[1] Code duplication problem

If we define:

```
public interface Airborne {
  public int getMaxHeight();
  public int getAirSpeed();
  ...
}
```

and

```
public class Hawk extends Carnivore implements Airborne ...
public class Falcon extends Carnivore implements Airborne ...
public class SeedEater extends Herbivore implements Airborne ...
...
```

Then we can have a single implementation: `handleAirborne(Airborne a)`. We can also do things like making arrays and collections to store any airborne objects. `Stack<Airborne> sas=new Stack<Airborne>();`

2.2 Implementation substitution

`java.util.List` is an interface not a class — it describes what a list needs to be able to do, but not how. Now Java actually has at least three classes which can do the job of a list (and they all implement the `List` interface). They have different strengths and weaknesses depending on exactly how they will be used but any of them can "do the job". eg:
`List<ZooAnimal> zl=new LinkedList<ZooAnimal>();` or
`List<ZooAnimal> zl=new ArrayList<ZooAnimal>();`.

Giving the variable the type `List<ZooAnimal>` as opposed to something more specific, means that if a better list is implemented (or if testing shows that another type would perform better), it can be substituted without needing to change everything.

The principle of least priviledge says that you should implement something with the most general / last powerful items which will do the job[2]. If all you need is a list, don't force it to be a specific type. For one thing someone reading the code later will have no idea whether the code really *requires* a `LinkedList` or if the author just liked them.

[2]actually least priviledge is more about the level of access users need in order perform certain system operations but it's a convient term.