Class Design

To support information hiding

Minimize accessibility of

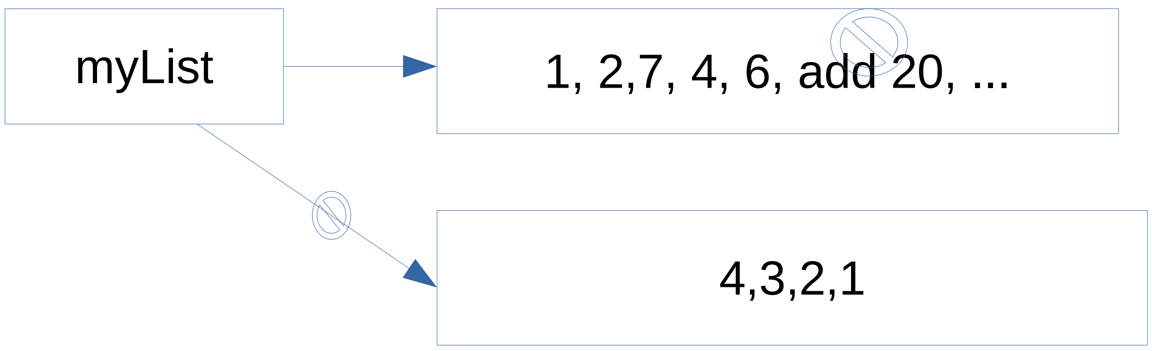
- Classes
    - public
    - package private
- Class members (e.g. routines)
    - public
    - protected
        - Big gap: once routine is protected, it has to be supported outside package
    - package-private
    - private (all data fields should always be private: use getters)

```
Package typeinference;

Public class Klass {
        Protected foo() {...
```

```
Package otherpackage;

Public myclass extends Klass {
        protected foo() { .. }
```

Public data fields (exception and caveats):

```
Class Year {
        public static final int DAYS = 365;
        public static final List<Integer> myList =
                Collection.unmodifiableList( ….);
```



Favor immutable objects / classes
    cannot be modified once it has been created
Always strive to make objects immutable
    (as immutable as possible)

E.g. TypeName in Programming Assignment 2

Benefits:
- Simple (to reason about, to test, assert correctness)
- Easy to share
- OS: thread-safe
- Failure atomicity for free

To make object immutable
- No mutators (e.g., no setters)
- class final (otherwise, subclass can introduce mutators)

```
    class Klass { // immutable class … }
    class SubKlass extends Klass {
        int foo() { … mutator … }

    Klass k; // is k immutable? If k is a SubKlass then
        it is not immutable, it if it is Klass it is
```

- all fields should be final
- all fields private

Law of Demeter
- Short (simplified) version: an expression can have at most one dot

🚫
```
dog.getLeg(FRONT, RIGHT).move();
dog.getLeg(…).move();
dog.getLeg(…).move();
dog.getLeg(…).move();
```

☺
```
dog.walk();
```

- Long (real) version: five articles and 3 amendments (not covered)