1 Administrivia

- No lab tomorrow
- Project 2B due next Monday
- Problem Set 3 out next Monday

2 Inheritance

You want to use inheritance to:

- Express a genuine subtype relationship ("is-a")
- Implement an abstract base class
- Add additional functionality to an existing class

You want to avoid inheritance because:

- Superclasses can call overridden public methods without warning
- Subclasses that are not proper subtypes can be used incorrectly
- You can only inherit from one superclass (single inheritance)

3 Composition

The key idea behind composition is to encapsulate common behavior in its own object. Now, instead of sharing code by inheriting from a common superclass, composition allows you to share code by simply passing around an object that contains that behavior. A specific example of composition is the strategy pattern, where classes can define different strategies for accomplishing the same goal through a common interface.

4 Delegation

Delegation is a special type of composition that can be a useful alternative to inheritance when you want to override or change certain aspects of the behavior of an existing class. The main ideas are:

- Create an instance of the existing class as a member field
- Mirror the existing class’ methods
- For methods that don’t change, simply forward them along

Delegation is far safer, although more tedious, than inheritance, since the class doing the delegating is in complete control of which methods it calls.
Listing 1: Example of bad inheritance

```java
class Stack<T> extends Vector<E> {
    T peek() { ... }
    T pop() { ... }
    T push(T item) { ... }
    int search(Object o) { ... }
}
```

Listing 2: Parsing and lexing example

```java
public interface Lexer {
    List<String> lex(InputStream inputStream);
}

public interface Parser<T> {
    T parse(InputStream inputStream);
}

public class MusicParser implements Parser<Music> {
    private final Lexer lexer;

    public Parser(Lexer lexer) {
        this.lexer = lexer;
    }

    public Music parse(InputStream inputStream) {
        List<String> tokens = this.lexer.lex(inputStream);
        for (String token : tokens) {
            ...
        }
        return ...;
    }
}
```

Listing 3: Read only list example

```java
public class ReadOnlyList<T> implements List<T> {
    private final List<T> elements;

    public ReadOnlyList(List<T> elements) {
        this.elements = elements;
    }
    ....

    public T get(int index) {
        // simply delegate supported operations to underlying list
        return this.elements.get(index);
    }
    ....

    public void add(int index, T element) {
        // forbid operations that modify the list with a runtime exception
        throw new UnsupportedOperationException();
    }
}
```