Decorator Design Pattern

A Lecture for KMP's COMP 401 Class

Aaron Smith

October 23, 2018

Adapted from Ketan Mayer-Patel's Lecture Slides

What are design patterns?

Design patterns are techniques

for **organizing your code**

that are often used in the real world

What are design patterns?

Writing code **without** design patterns

Writing code **with** design patterns

Common Design Patterns

Iterator	Factory	Singleton	Decorator
Observer / Observable	Model-View	Model-View-Controller	And more…

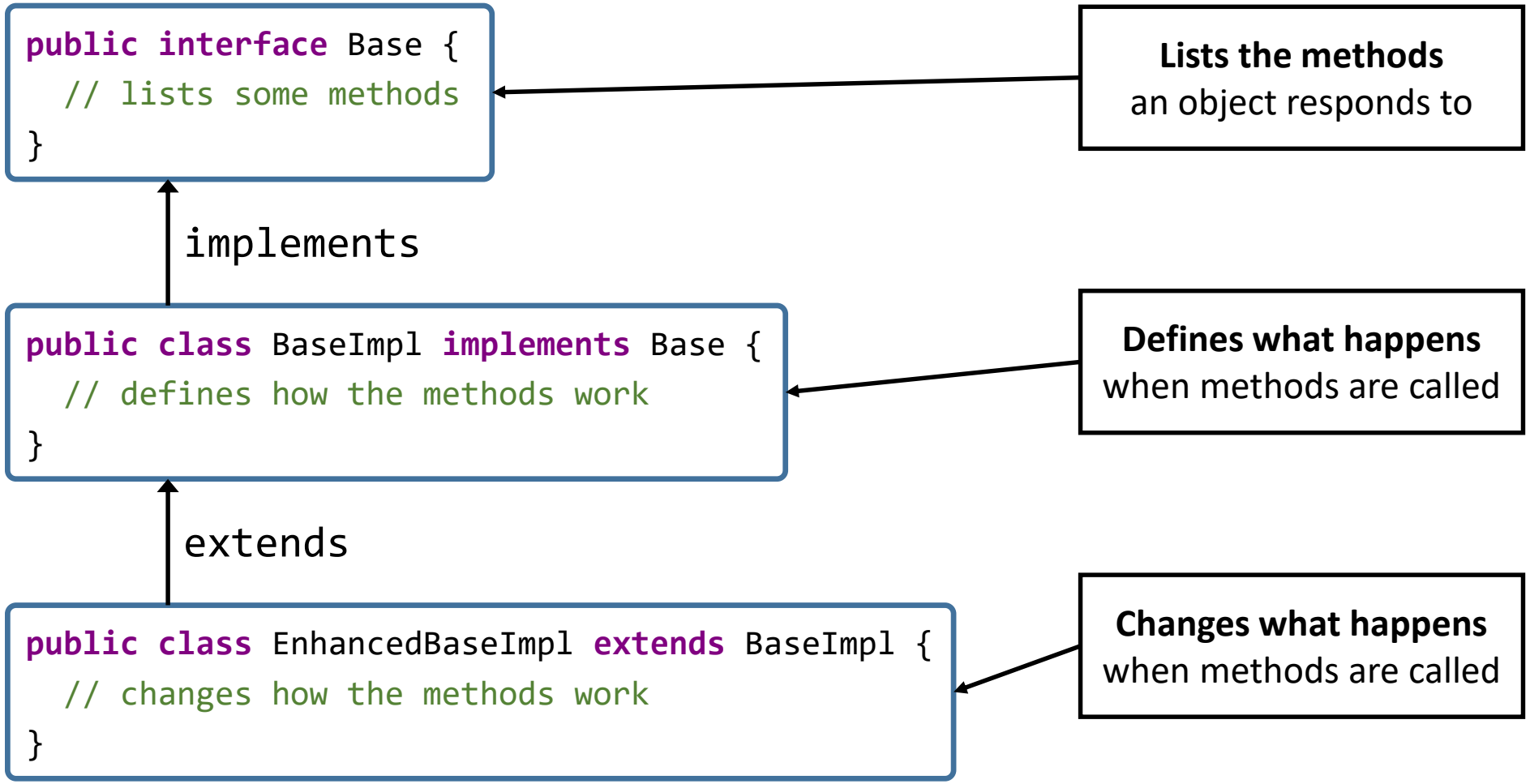
The decorator design pattern is a way to

change the behavior of an object

of a specific interface.

Object behavior is defined by methods.

What defines object behavior?



Question: Don't **subclasses** "change the behavior of an object?"

Answer: **Yes**, but they are **limited**.

Limitations of Subclasses for Changing Object Behavior

An object's class cannot be changed after it is initialized.

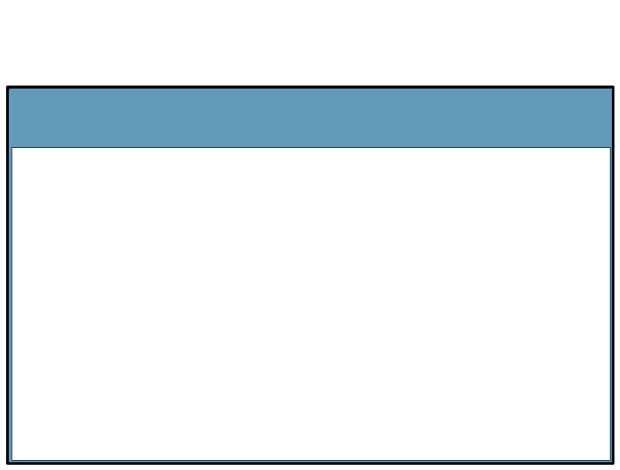
- Impossible to dynamically change the object's subclass at runtime.

A class may only extend from a single parent class.

- Behavior defined in a subclass is forever linked to its parent class.

https://en.wikipedia.org/wiki/Decorator_pattern

Subclass Limitation Illustration



https://en.wikipedia.org/wiki/Decorator_pattern

Subclass Limitation Illustration

https://en.wikipedia.org/wiki/Decorator_pattern

Subclass Limitation Illustration

https://en.wikipedia.org/wiki/Decorator_pattern

Subclass Limitation Illustration

https://en.wikipedia.org/wiki/Decorator_pattern

Subclass Limitation Illustration

https://en.wikipedia.org/wiki/Decorator_pattern

Subclass Limitation Illustration

If the functionality being added

is independent of the underlying

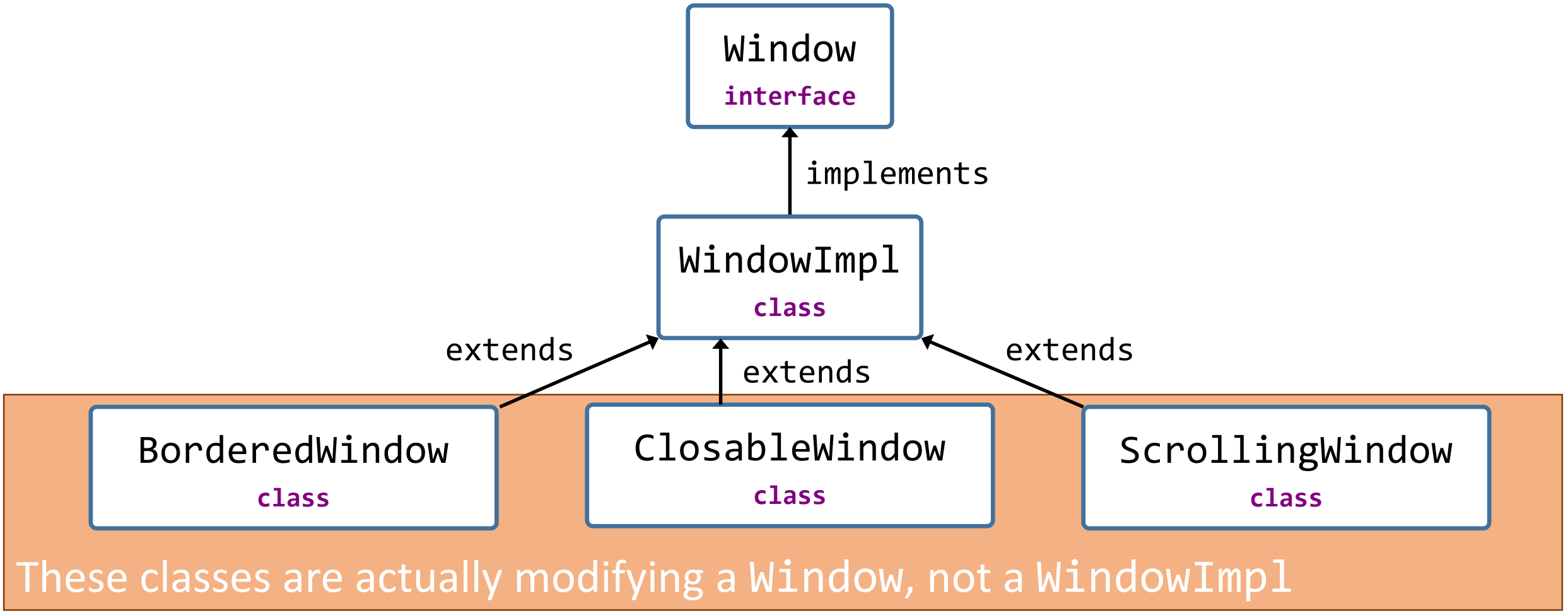
implementation …

…then a decorator is a better

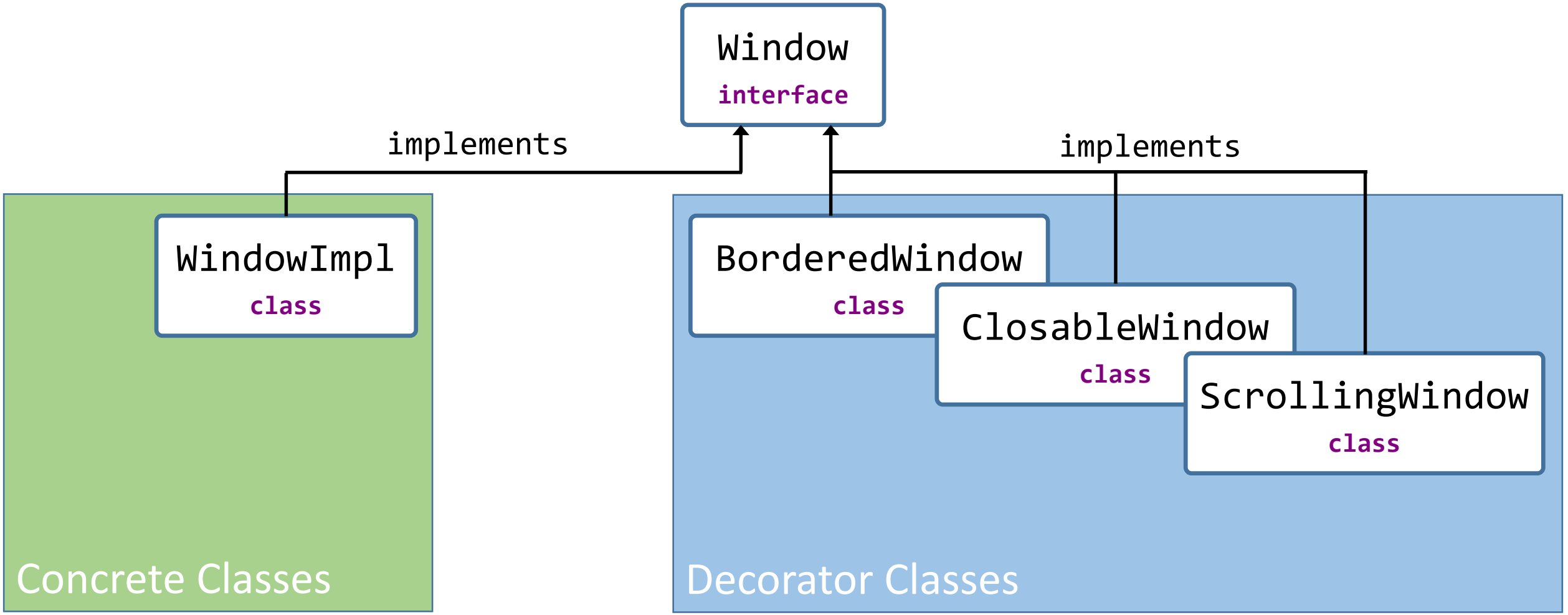
choice than a subclass

https://en.wikipedia.org/wiki/Decorator_pattern

Sometimes, subclasses are implementation-independent



Sometimes, subclasses are implementation-independent



Decorator Pattern Recipe

Implement	• Make a new class that implements the interface
Encapsulate	• Wrap (encapsulate) another instance of the interface inside the new class
Delegate	• Forward (delegate) all methods to the base interface
Modify	• Selectively add or change method functionality as desired

Anatomy of a Decorator Class

Decorators *always* **encapsulate** an instance of the same interface they **implement**

The constructor *always* takes as input an instance of the interface

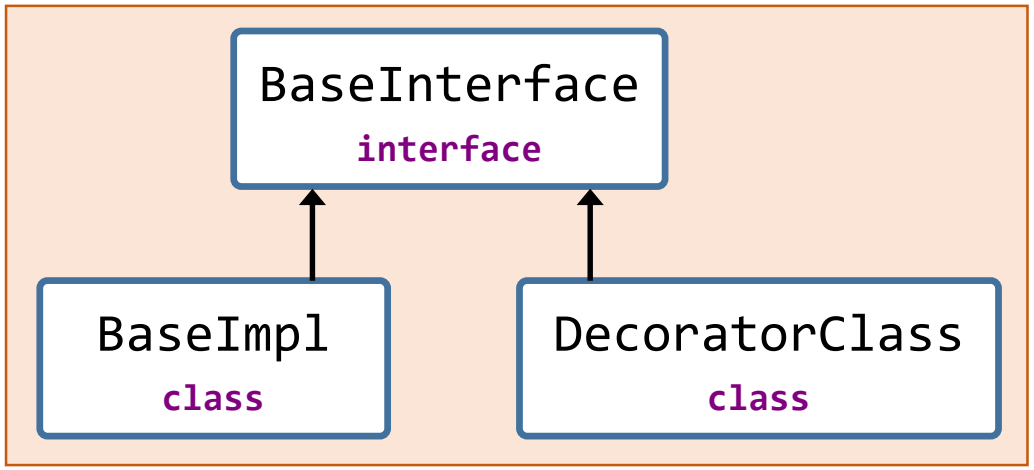
Instead of implementing functionality directly, methods **delegate** to the encapsulated instance

```
public class DecoratorClass implements BaseInterface {
  private BaseInterface wrapped_obj;

  DecoratorClass(BaseInterface obj) {
    this.wrapped_obj = obj;
  }

  public double process() {
    // Decoration behavior before delegating
    double val = wrapped_obj.process();
    // Decoration behavior after delegating
    return val;
  }
}
```

Creating and Using a Decorator Object



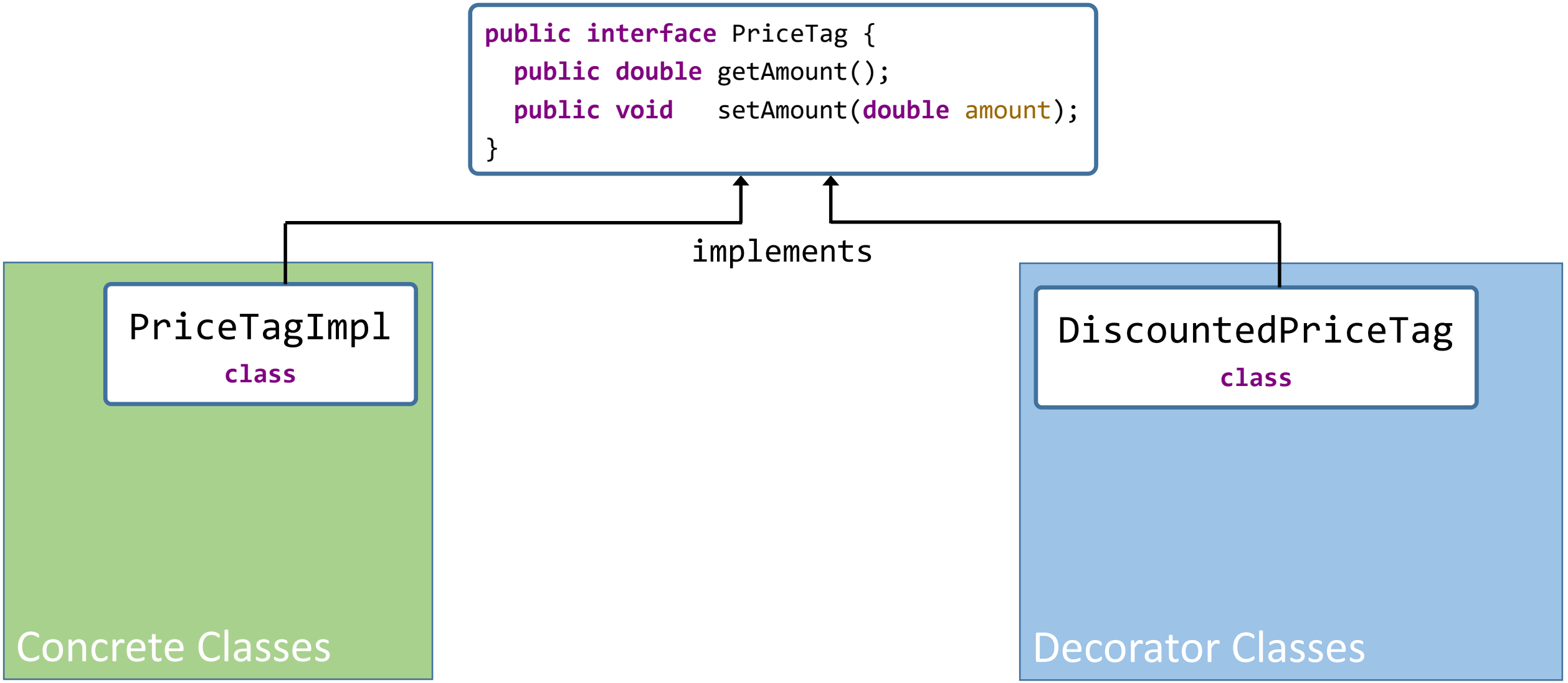
The base object is wrapped in a decorator object

```
BaseInterface baseObject      = new BaseImpl();
BaseInterface decoratorObject = new DecoratorClass(baseObject);

double value = decoratorObject.process();
```

The decorator object is used instead of the base object

Example: Price Tag



Code Version 1: Basic Decorator

PriceTag.java

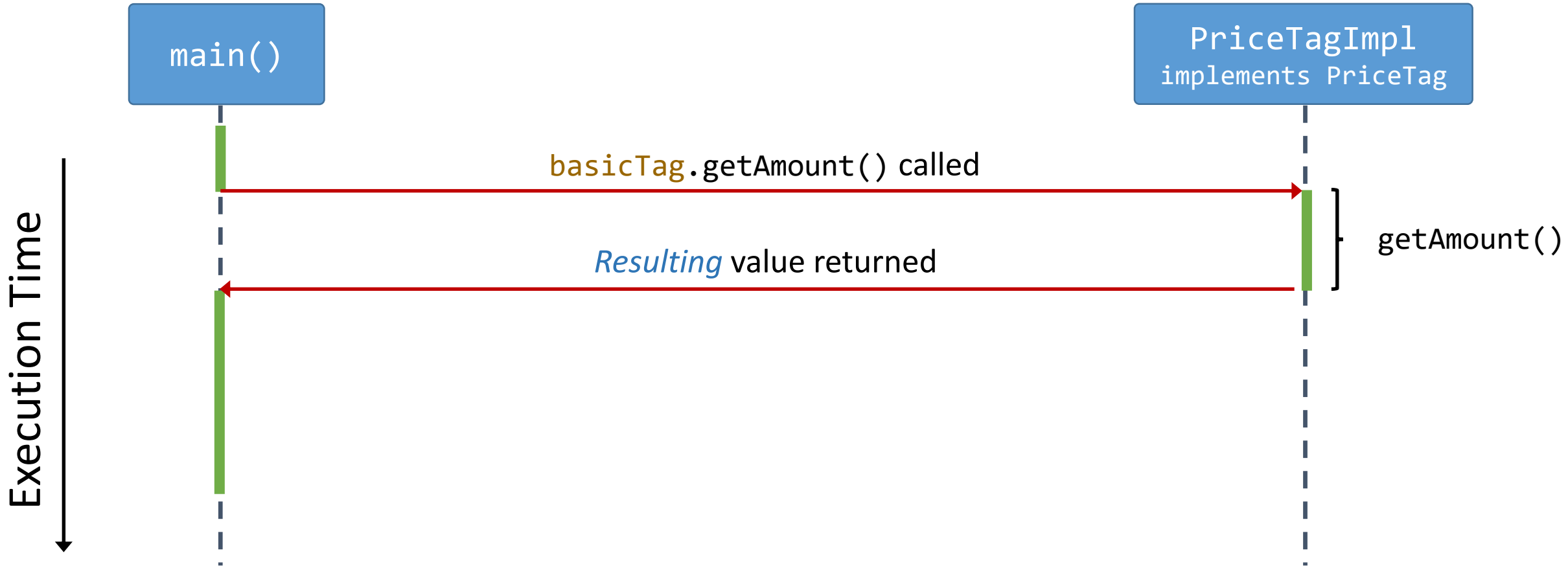
PriceTagImpl.java

DiscountedPriceTag.java

Main.java

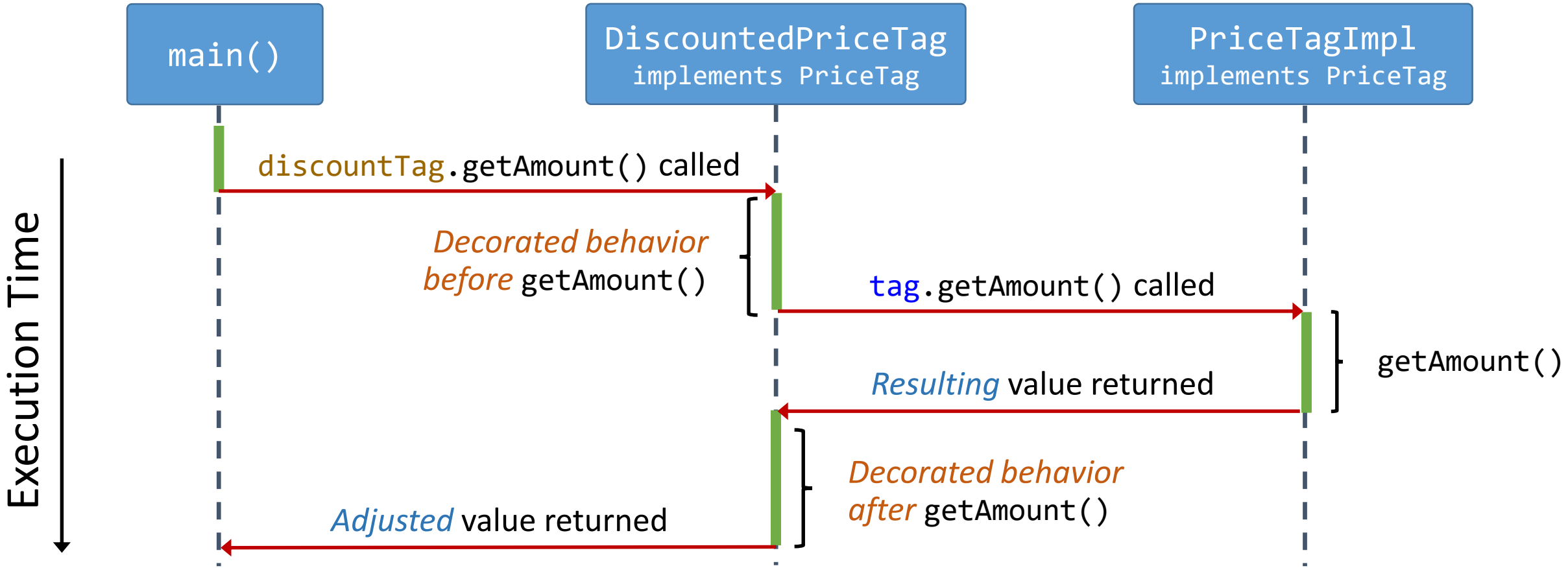
Tracing Method Execution

```
PriceTag basicTag    = new PriceTagImpl(100);
PriceTag discountTag = new DiscountedPriceTag(basicTag, 20);
```



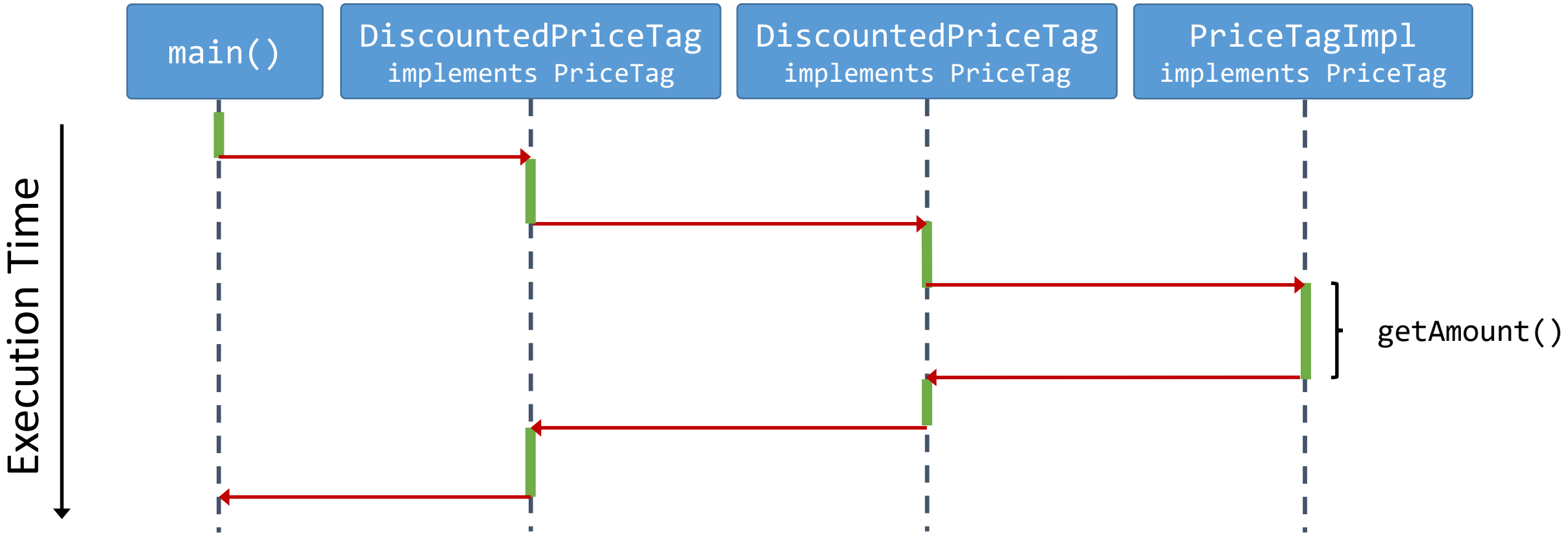
Tracing Method Execution

```
PriceTag basicTag    = new PriceTagImpl(100);
PriceTag discountTag = new DiscountedPriceTag(basicTag, 20);
```



Chaining Multiple Decorators

```
PriceTag basicTag     = new PriceTagImpl(100);
PriceTag discountTag1 = new DiscountedPriceTag(basicTag, 20);
PriceTag discountTag2 = new DiscountedPriceTag(discountTag1, 20);
```



Code Version 2: Decorator Chaining

Main.java

Unwrapping Decorators

- “Decorating” an object *creates a new object*
  - The original object is encapsulated (wrapped) inside

- Sometimes, you might need to access the original object
  - This is called “undecorating” or “unwrapping”

Code Version 3:
Decorator Unwrapping

DiscountedPriceTag.java

Main.java

Limitations of Decorator Pattern

1. No access to parent class's protected variables
2. A decorator can be added to the same object more than once
3. With multiple decorators, order of decorator application may affect behavior
4. Two different incompatible decorations may be wrapped around the same object

Decorator Pattern in the Java Core

- java.io.InputStream
- java.io.OutputStream
- java.io.Reader
- java.io.Writer

Java.io.InputStream