Australian
National
University

The Decorator Design Pattern

Eric McCreath

Overview

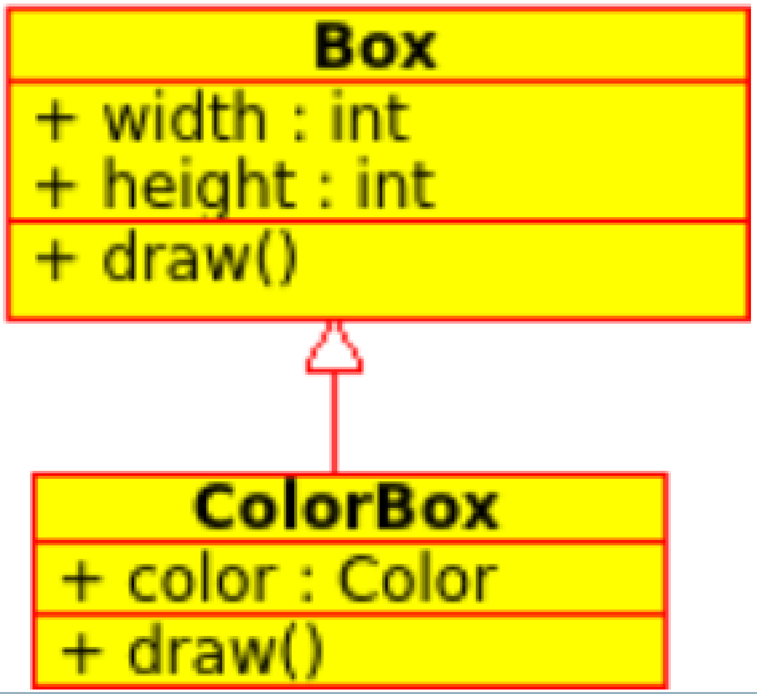
In this lecture we will:

- motivate and define the decorator design pattern,
- give some examples of where this design pattern is used,
- provide a template for implementing your own classes that use this pattern, and
- in the following lecture we will have a "live code" example.

Australian National University

The Context

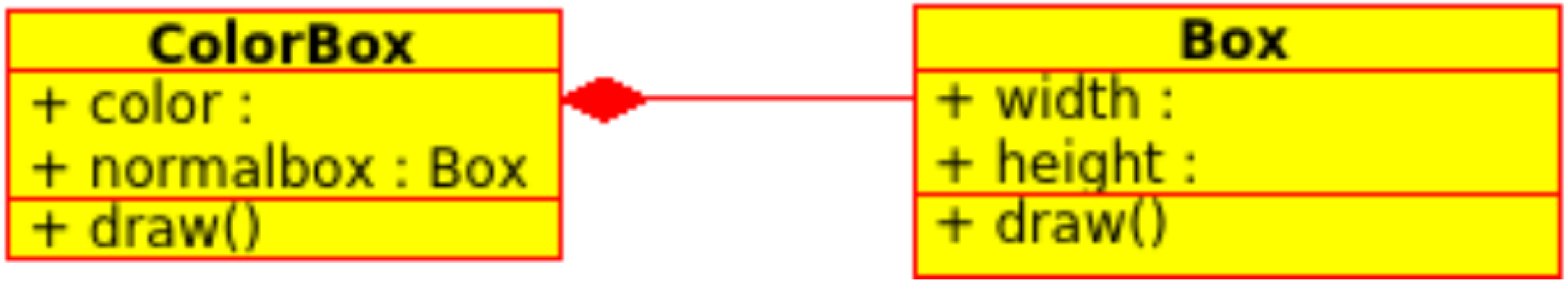
- Object oriented programming languages, such as Java, provide a way of "extending" a class. This is known as inheritance which lets a programmer create a new class that builds upon an old class. This may involve:
  - adding new fields,
  - adding new methods, or
  - modifying existing methods (these modified methods may call the super class' method).



Australian National University

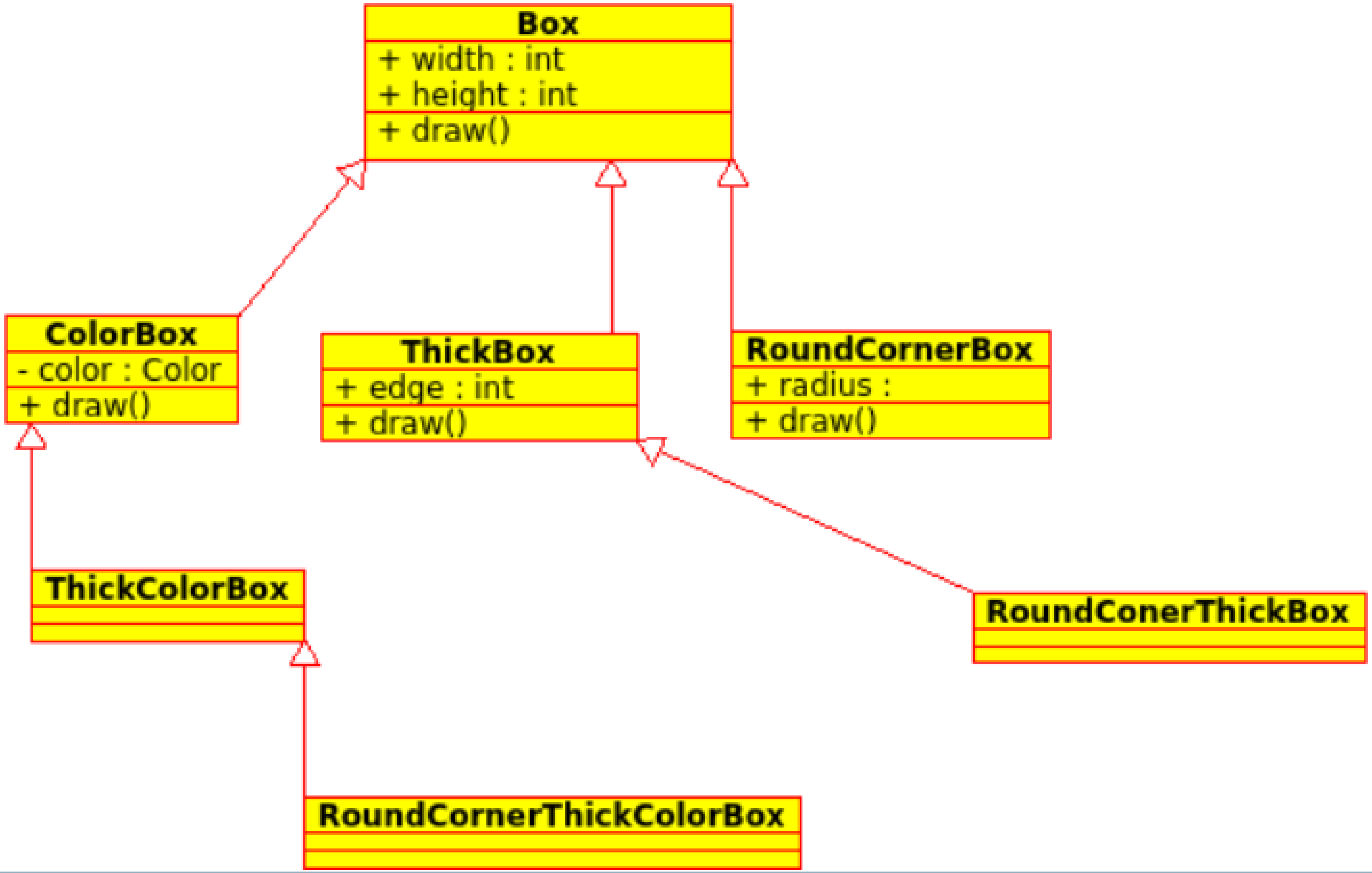
The Context

Note that using inheritance is not the only way we can take an existing class and modify or add to it. We can also use aggregation.



The Context

We may also wish to make various other additions to a base class. However this quickly becomes rather messy and complex, since we end up creating many classes and duplicating code. Also, all such combinations are fixed when we compile the code.

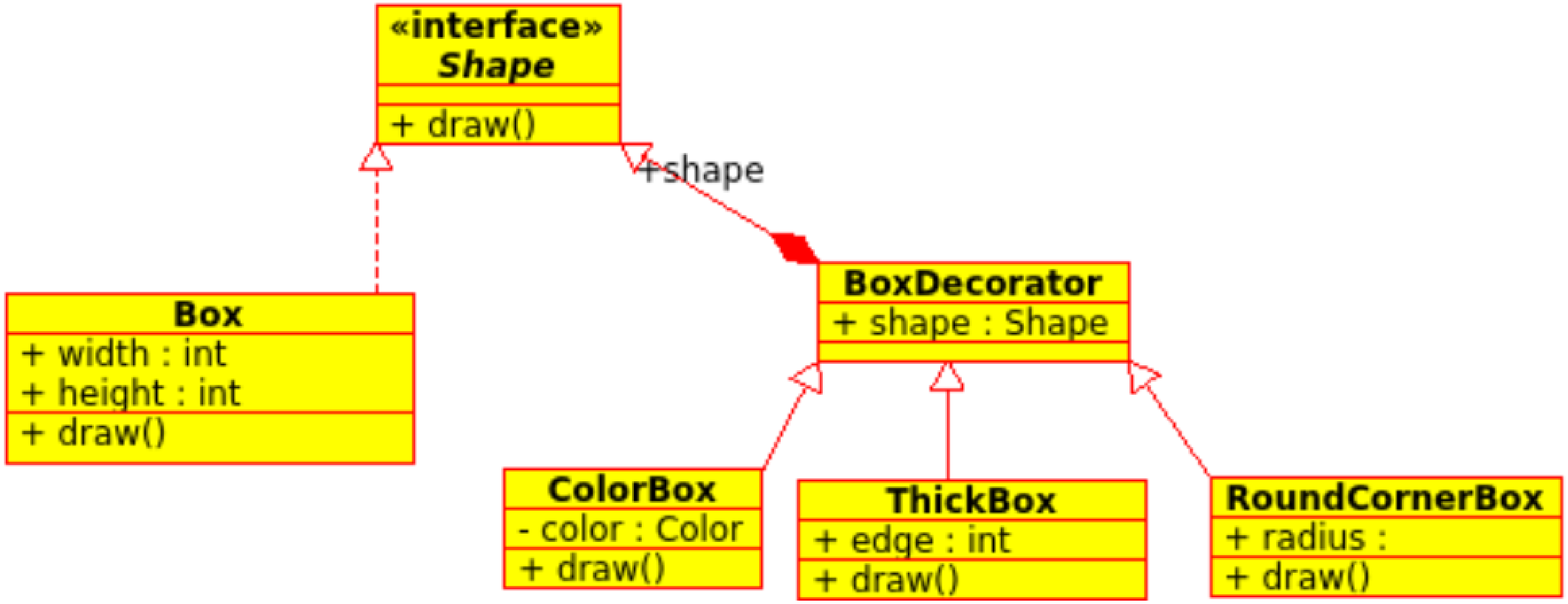


Australian National University

The Decorator Design Pattern

- The decorator design pattern combines inheritance with aggregation enabling us to "decorate" a class with a combination of different features or attributes.
- We need an abstract class (or interface) of the class we wish to decorate.   This abstract class (or interface) should have the key methods that you expect of the base class and of the decorated cases of the base class.
- Now the trick! We create a decorator class that both inherits (or implements) the abstract class (or interface) AND has a field which is of this type.
- This decorator class can be sub-classed with many different concrete decorator classes.
- Now we can "wrap" instances of the base class with different "layers" of decorations.

The Decorator Design Pattern



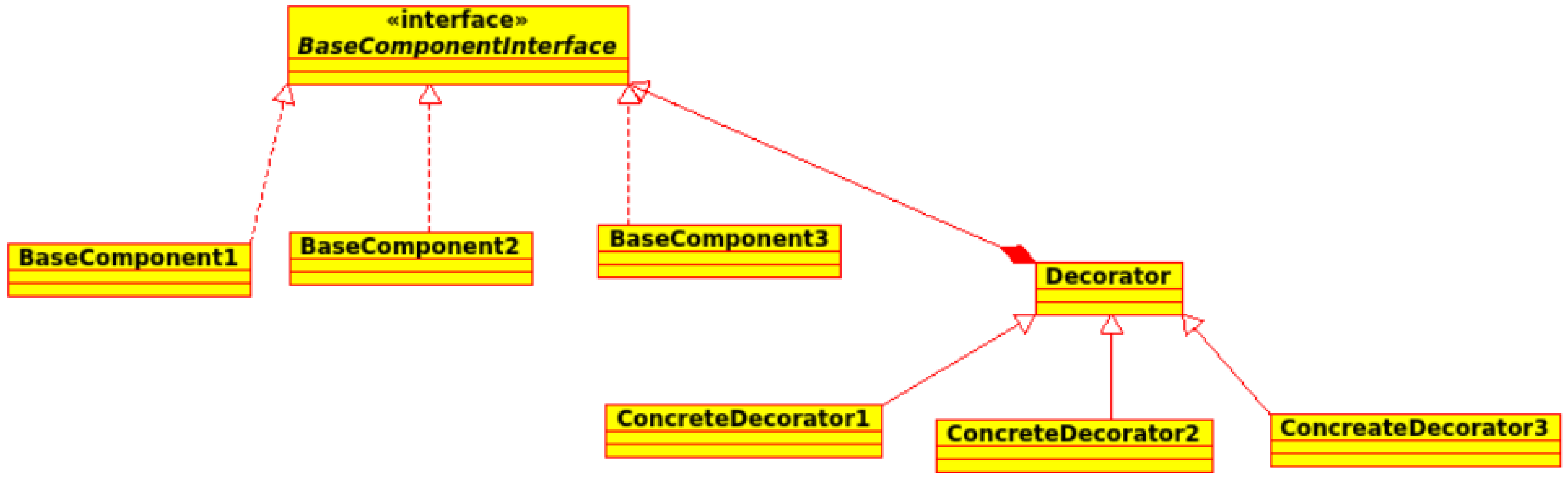
Assuming we have appropriate constructors we could make a box that is 10X20 with edges of thickness 3 and that is red by using:

```
Shape myshape = new ThickBox(3, new ColorBox(Color.red,new Box(10,20)));
```

The decoration of classes is all done at run-time (not fixed at compile time).

The Decorator Design Pattern

More generally we have:



Use of the Decorator Pattern

- The Java file IO class makes extensive use of the decorator design pattern.

For example, an AudioInputStream extends an InputStream and when you construct a new AudioInputStream you give it an existing InputStream. Thus the AudioInputStream decorates the InputStream.

- The JScrollPane is also a decorator class, as it is a JComponent and it "wraps" an existing JComponent.

Limitations

- If using inheritance is viable then it is often a simpler design approach.
- If the methods within base classes check the type of objects (i.e. they use the "instanceof" operator) then your code may come to the wrong conclusion about the type of a particular object.
- The methods of the BaseComponentInterface must be amenable to be composed and combined using the decorator pattern. So the implementations of the methods within the concrete decorator classes need to be able to work from calling the methods of the referenced object's concrete decorator class.

Australian National University

Example Exam Questions

- What are the similarities and differences between using inheritance and the decorator design pattern to add extra features to a class?
- Suppose you have a base class that only has fields and you wish to create a new class that has extra fields added (leaving the original class unchanged). Is the decorator design pattern a possible approach to use? If so how would you do it? If not why not?
- Suppose you are implementing a car racing game. There are three types of cars: the SUV, the sedan, and the wagon. All of these can be given different "abilities" inlcuding: double speed, stopped, slippery tyres, and no steering control. At any one time a car can have multiple extra abilities. Using the decorator design pattern create a UML class diagram for this situation.

Example Exam Questions

- The code below uses inheritance to extend the Name class adding the surname to the end of a persons name.

```
class Name {
   String name;
   public Name(String name) {
       this.name = name;
   }
   String showname() {
       return name;
   }
}
class FullName extends Name {
   String surname;
   public FullName(String firstname, String surname) {
      this.name = firstname;
      this.surname = surname;
   }
   String showname() {
       return super.showname() + " " + surname;
   }
}
```

Re-implement this using the decorator design pattern rather than inheritance.