Today

1. Brief Discussion of Inheritance Issues
2. Abstract classes
3. Template Method Design Pattern
4. Guided Tour of Some Class Hierarchies
5. Synthesis Design Problem

Code Demonstrating Compositional Inheritance

```
public class CrashProperties
{
    public static void main(String[] args)
    {
//      Stack<String> stack = new Stack<>();
//      stack.push("1");
//      stack.push("2");
//      stack.push("3");
//      stack.insertElementAt("squeezed in", 1);
//      while(!stack.empty()) System.out.println(stack.pop());

        Properties properties = new Properties(System.getProperties());
        properties.list(System.out);
        properties.setProperty("user.name", "fred");
        properties.list(System.out);
        properties.put("user.name", new UserName());
        properties.list(System.out);
    }
}

class UserName
{}
```

11

Effective Java 2nd Edition, Item 16: Favor composition over inheritance:

> Inheritance is appropriate only in circumstances where the subclass really is a *subtype* of the superclass. In other words, a class *B* should only extend a class *A* only if an "is-a" relationship exists between the two classes. If you are tempted to have a class *B* extend a class *A*, ask yourself this question: Is every *B* really an *A*? If you cannot truthfully answer yes to this question, *B* should not extend *A*. If the answer is no, it is often the case that *B* should contain a private instance of *A* and expose a smaller and simpler API; *A* is not an essential part of *B*, merely a detail of its implementation.
>
> There are a number of obvious violations of this principle in the Java platform libraries. For example, a stack is not a vector, so `Stack` should not extend `Vector`. Similarly, a property list is not a hash table, so `Properties` should not extend `Hashtable`. In both cases, composition would have been preferrable.

The book goes in greater detail, and combined with *Item 17: Design and document for inheritance or else prohibit it*, advises against overuse and abuse of inheritance in your design.

Here's a simple example that shows the problem of `Stack` allowing un-`Stack`-like behavior:

```
Stack<String> stack = new Stack<String>();
stack.push("1");
stack.push("2");
stack.push("3");
stack.insertElementAt("squeeze me in!", 1);
while (!stack.isEmpty()) {
    System.out.println(stack.pop());
}
// prints "3", "2", "squeeze me in!", "1"
```

This is a gross violation of the stack abstract data type.

Java Properties

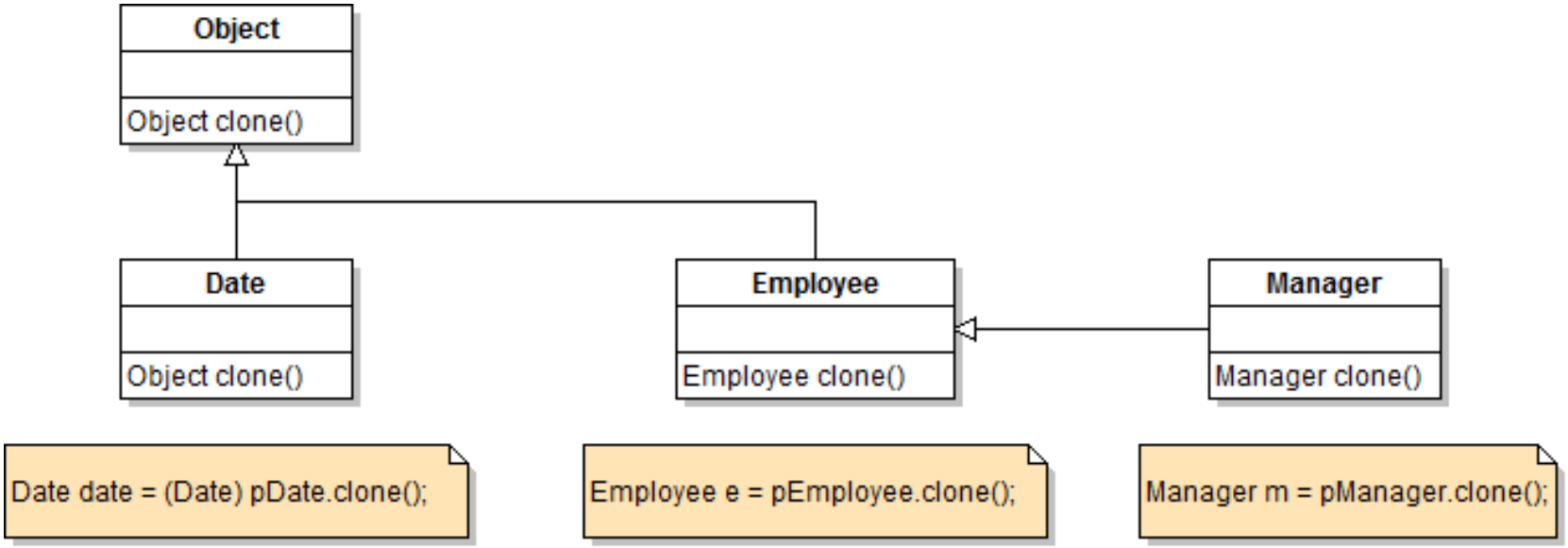
```
public class Properties
extends Hashtable<Object,Object>
```

The `Properties` class represents a persistent set of properties. The `Properties` can be saved to a stream or loaded from a stream. Each key and its corresponding value in the property list is a string.

A property list can contain another property list as its "defaults"; this second property list is searched if the property key is not found in the original property list.

Because `Properties` inherits from `Hashtable`, the `put` and `putAll` methods can be applied to a `Properties` object. Their use is strongly discouraged as they allow the caller to insert entries whose keys or values are not `Strings`. The `setProperty` method should be used instead. If the `store` or `save` method is called on a "compromised" `Properties` object that contains a non-`String` key or value, the call will fail. Similarly, the call to the `propertyNames` or `list` method will fail if it is called on a "compromised" `Properties` object that contains a non-`String` key.

Covariant Return Types



Do covariant return types break the Liskov Substitution Principle? –> No

Template Method Design Pattern

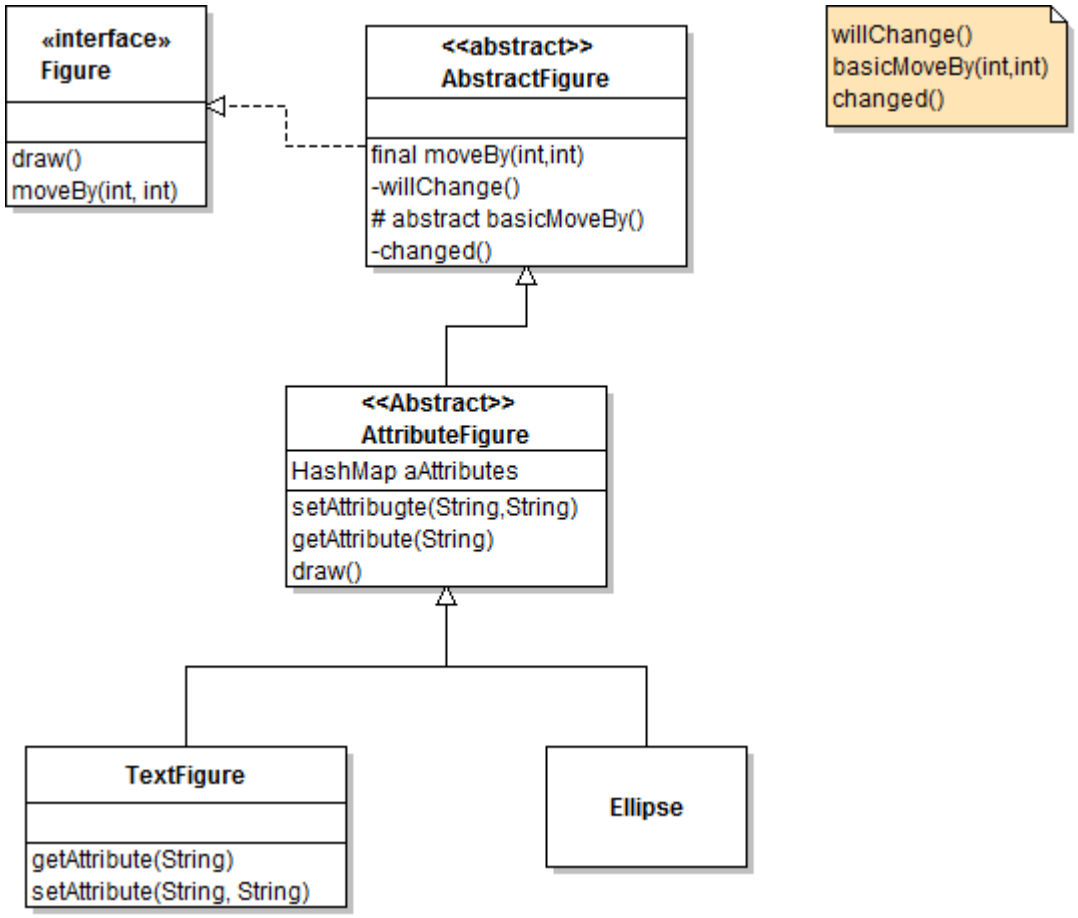
Context

- An overall algorithm is the same for all subclasses
- Some of the steps need to be specialized for different subclasses.

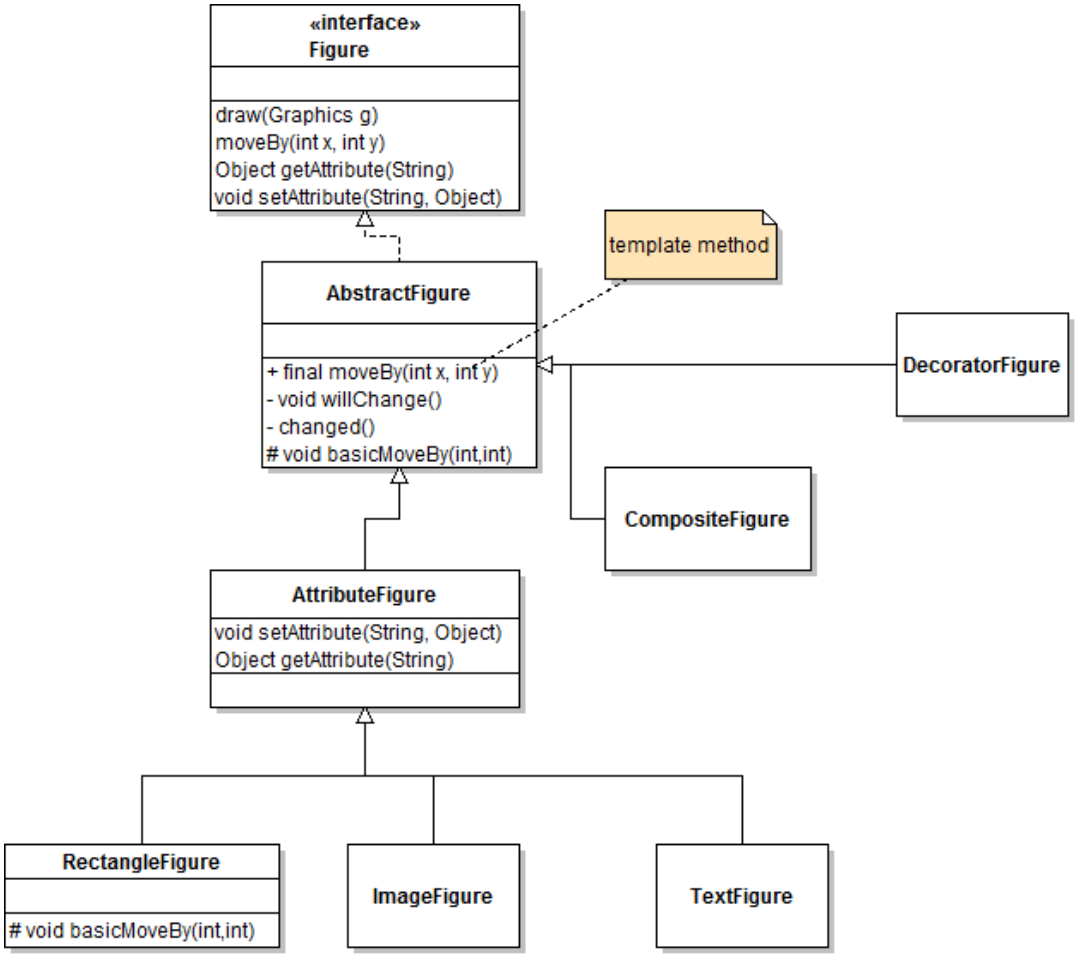
Solution

- Put the general algorithm in the method of an abstract class.
- Define the variable steps as abstract methods.

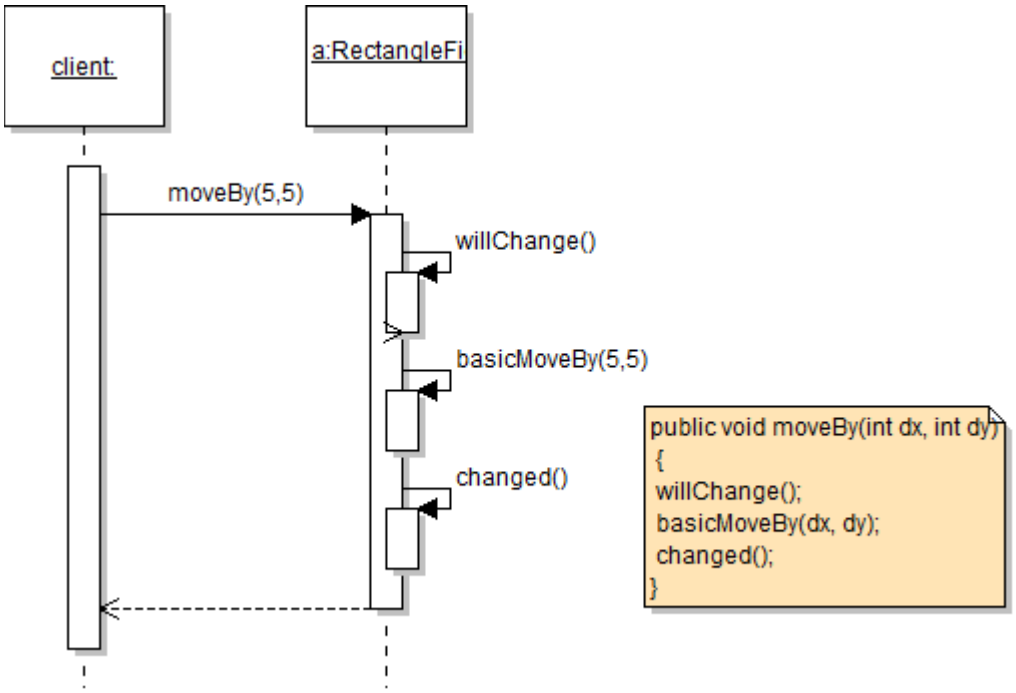
Template Method Design Pattern in JHotDraw



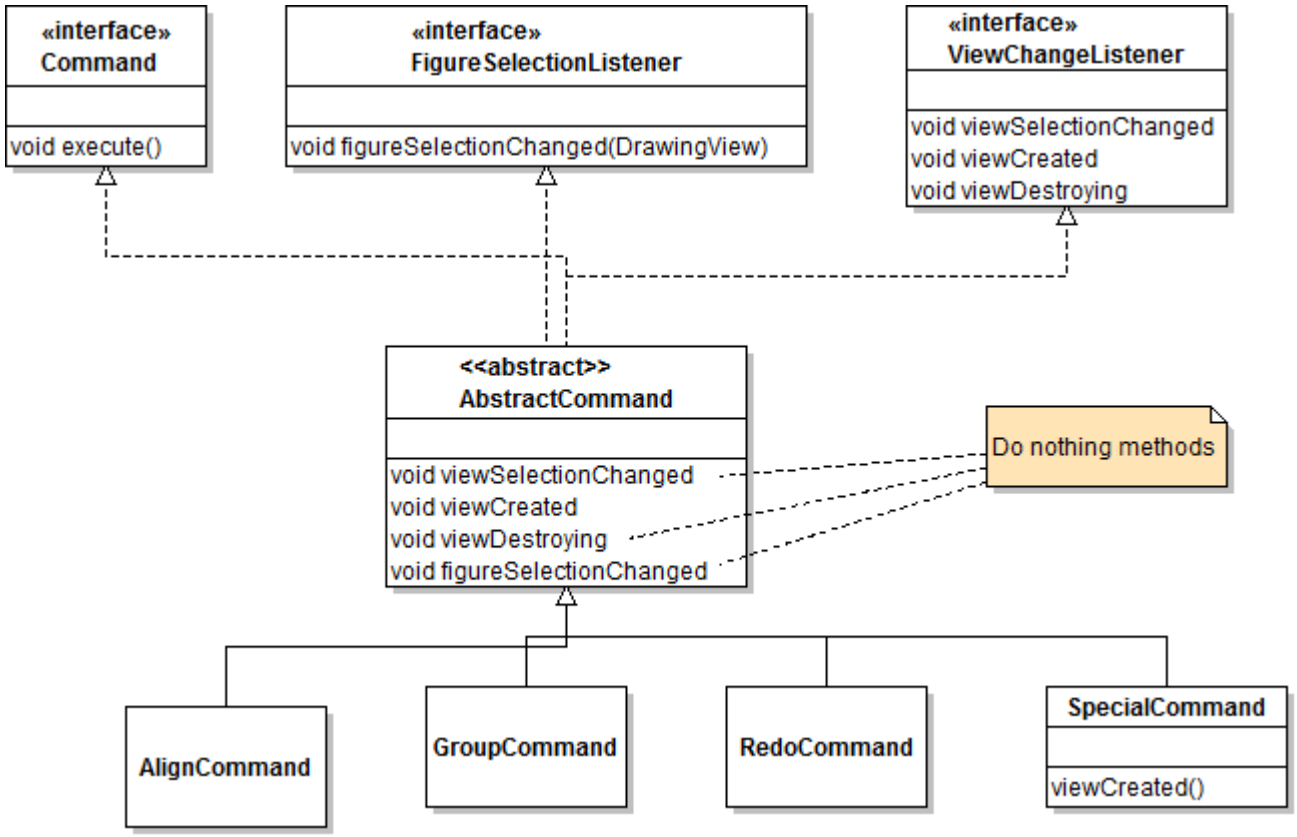
Additional Information on the Figure Class Hierarchy in JHotDraw



Call to the template method Figure.moveBy in JHotDraw



Part of the Command Hierarchy in JHotDraw



Skeleton of the Practice Question