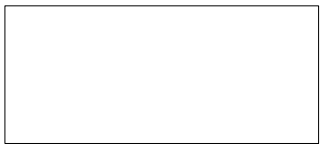
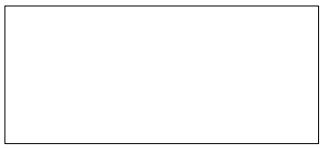
1. Draw a class diagram of the *Factory Method Pattern* with these elements:

Factory (interface), Product (interface), ConcreteFactory, ConcreteProduct, makeProduct( )

Show relationships between elements.





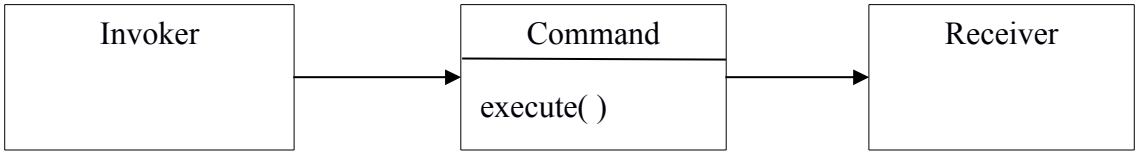

2. *Iterable* and *Iterator* use the Factory Method pattern. For example:

```
ArrayList list = new ArrayList();
list.add( "apple" );  list.add( "banana" ); . . .
Iterator iter = list.iterator( );
```

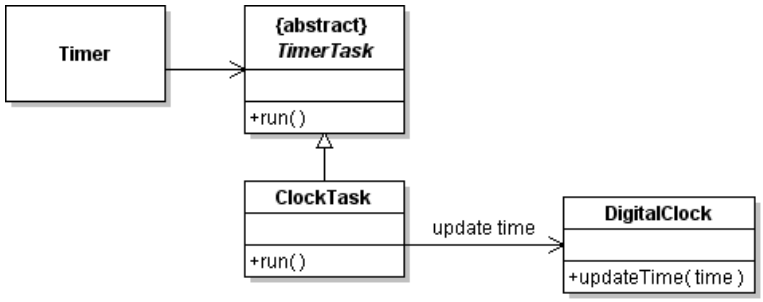
Complete the table to show how this example uses the Factory Method pattern

Name in Pattern	Name in This Example
Factory	
Product	
ConcreteFactory	
ConcreteProduct	
makeProduct( )	

3. The purpose of the Command Pattern is to encapsulate actions so that we can have a family of interchangeable actions, like "turn on", "turn off", or "play sound".



The digital clock uses as Timer and TimerTask to update the time and another TimerTask to sound the alarm (at least, you *should* program it that way... don't check the alarm time every second!).



a) Fill in the table:

Name in Pattern	Name in this example
Command	
Invoker	
execute( )	
Receiver	

b) How does the ClockTask "know" which DigitalClock object it should update?

4. Which pattern should we consider using? Why?

4.1 Java has an old interface named *Enumeration* with these methods:

StringTokenizer (a useful class for parsing strings), HashMap, and Vector create an *Enumeration*.

But our code requires an *Iterator*, not an *Enumeration*. What should we do?

Enumeration

hasMoreElements()
nextElement()

4.2 In a Calculator, the user enters a number by pressing keys. Such as "1", "7", ".", "5" for the number 17.5. The user may also correct mistakes by pressing a DEL or "←" key before finishing the number.

The Calculator itself accepts *numbers* as operands. We don't want to complicate the calculator by having it handle each key the user types. Instead we'd like the user to finish typing the number, then give it to the calculator. *What pattern should we use?*

4.3 In the Calculator again, after performing a calculation the display shows the result (a number).

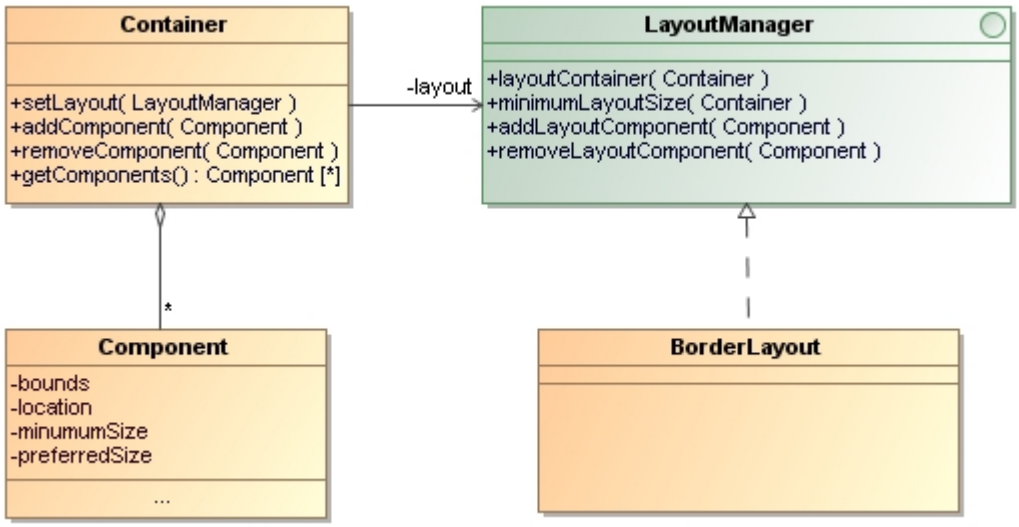
The *first digit key* the user presses will replace the old value on the display.

Additional digit keys the user presses will be appended to the display value.

Somehow, we need to treat the first digit press differently from other digit presses. What pattern should we use?

5. Java Layout Managers are an example of the Strategy pattern.
a. Complete the following table:

Name in Pattern	Name in This Example
`Context`	
`Strategy`	
`ConcreteStrategy`	
`setStrategy( Strategy )`	
`doStrategy( Context )`	



5b. Why does the Strategy (usually) need a reference to the Context?

5d. What other design pattern has a structure like Strategy?

6. The above diagram also illustrates the Composite Pattern.

6a. What class is the *composite*?

6b. What UML relation must you add to the above diagram to complete the composite pattern?