Brad Bensch

Kevin Mulligan

Jenny Zhen

William Spaw

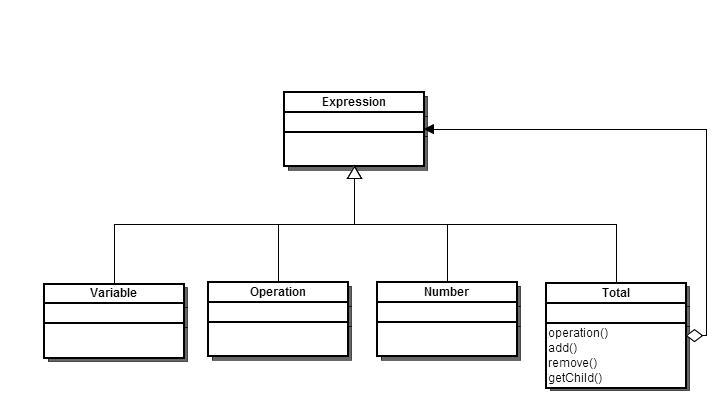
Chad Koppes Shared w/Prof R. 9:43 on Thurs 3/21

1. **Explain the statement "The command pattern supports separation of execution and invocation in both time and space."  
   Jenny -** The command pattern supports separation of execution and invocation in terms of time because the final execution of the requests generated may not occur immediately. There may be a queue of commands that have been invoked and are waiting to be removed from the waitlist and processed. Thus, the command pattern ensures that the lifespan of the request becomes independent of the component that invoked it. The command pattern separates in terms of space because the request is not fulfilled by the invoker. Instead, the request is delegated to a pool of components that the request may become assigned to. Thus, the execution and invocation are two separate actions.
2. **Argue both for and against including the functions to handle the Composite's children in the Component interface. What are the implications for implementation of the pattern and on the intention of the pattern?**

**Brad -** By including the functions to handle the children in the component interface the pattern is that the leaf and composite classes can inherit the reference and the operations that manage it. If it is implemented once in the add and remove operations, then it can be inherited by the subclasses automatically.

By doing it externally you can treat all components uniformly which may not be the safest of options, or you can declare it in the composite class where any errors will be caught at compile time.

1. **Describe a class structure you could use to represent a four-function (add, subtract, multiple, divide) math expression with parenthetical expressions using the Composite pattern. Provide the class structure diagram. How do you "execute" the expression? For the expression a = 3.0 \* b + 2 \* ( 6 + c ) / ( d + 4.0 ) show an object structure representation using your class structure assuming standard precedence rules.**



**Chad, Jenny -** When operation() is called on total, it is supposed to build a recursive tree structure with Total at the root and the operations called in the correct order with Number’s and Variable’s as leafs.

1. **A broad definition of entropy is "the degree of disorder or uncertainty in a system". Why will the entropy of a design tend to increase over time? Describe how a Facade can be put to good use in a system with high entropy.**

**Kevin -** The entropy of a software system increases over time because as more and more code is added, especially with multiple authors, it becomes more and more complex. This complexity makes it harder to understand, and less ordered. Even in systems where there was a strong initial design, after a long enough time code will have been added which doesn’t follow the design perfectly. A facade can be used to improve the use of such a system by providing a way of using the system that hides the underlying processes. The facade serves to make the system more usable by simplifying access to the same functions that the system already performs.

1. **Why may the use of a Facade make the building, i.e. compilation, of a system run quicker.  
   Jenny -** Since a facade hides a complicated system with an interface that simplifies the structure, the facade can make the building of the system quicker in many ways. One would be the testing aspect. The facade would allow software testers to test the overall system easier, and in a much higher level to ensure the system works as a whole. Another way would be when new developers need to work on a complicated system. Adapting to what has already been done would be facilitated by the simpler interface. Additionally, since there is one simple interface, less components need to be linked together at build time.
2. **One method of classifying iterators does so on two dimensions. One which indicates the location of control of the iteration (*internal* to the iterator or *external* client control) and a second which indicates the location of the definition of the iteration (*embedded* as part of the collection or *separate* from the collection). Considering each dimension separately, what are the positive and/or negative aspects of iterators of each type?  
   Bill -** External iterators are much more flexible than internal iterators, but are generally more complex than internal iterators. Internal iterators are easier to use and implement by the client, but are more limited than external iterators in that you may not be able customize in some manner the way you are iterating through the collection.
3. **What language constructs would you use to give iterators privileged access in Java and in C++? How will your answer depend on the classifications for iterators given above?**

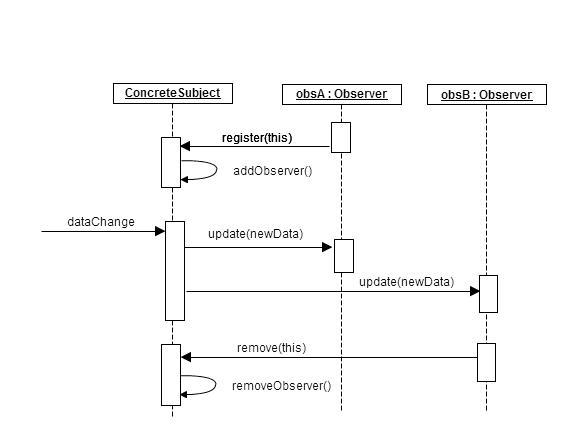
**Chad, Jenny -** In C++ make the iterator a ‘friend’ of its aggregate. For Java, you could use the protected keyword for privileged access. The principal difference between internal and external iterators is that the internal iterator does not need privileged access since the iterator is embedded with the object.

1. **Iteration over a recursive data structure such as a binary tree can be tricky using an external iterator. What are the problems with this? How can you accomplish this using an external iterator? What makes it easier with an internal iterator?  
   Jenny -** In order for the binary tree to be iterated using an external iterator, the user/client would have to know how to iterate through that data structure. Since iterating through different data structures varies based on many aspects of the specific data structure, an external iterator is not ideal. To accomplish iterating through an external iterator, all of the hidden internal structure of the data structure would have to be known to the client/user. Using an internal iterator would facilitate the iterating process because the iterator is contained within the data structure, so it obviously knows how to return its own data without exposing the internal structure.
2. **Explain how the Observer pattern effectively isolates the observed from the observer.**

**Kevin -** The observer pattern isolates the observed from the observer by creating an intermediary object whose sole function is to monitor for state changes. Rather than directly linking an object with the thing that it needs to observe, an observer object is created which watches for state changes and notifies the necessary objects when a change occurs. This way the various observers need not know anything about one another and can be added or removed to the subject class more flexibly.

1. **Draw sequence diagrams for the registration and update cycles of an Observer pattern implemented using appropriate Java classes. Be sure to label the object with the Java class and its role in the Observer pattern.**

Push Version. Assumes obsB is already registered.



1. **The nature of the relationship between the Proxy and RealSubject is different for each of the four types of proxies described in the textbook. For each proxy type: What are the characteristics of that relationship for each proxy type? What interactions occur between the two classes? What capabilities must the relationship support?**

**Brad** - A Remote Proxy has a representative for the RealSubject in another location, therefore the remote proxy must know where the RealSubject is and have access

- A Virtual Proxy creates the RealSubject objects on demand, rather than all at once must be able to generate the objects in the RealSubject when required

- A Protection Proxy controls the access into the RealSubject, giving certain objects certain access rights. This requires that the proxy have access to the objects that require access in order to assign the access rights.

- A Smart Reference locks the RealSubject so that nothing can change it while accessed, and counts the number of references to the RealSubject

1. **The intention for the State pattern includes the statement "The object will appear to change its class." Explain this. Explain how the structure of the pattern reinforces this.  
   Bill -** What the statement means is the object, or the state base class, will change its behavior based on the context. The structure of the pattern reinforces this with the base object being able to change into any number of inherited classes, based on the context.
2. **What issues must be considered if state objects are to be shared by multiple instances of the same Context class?  
   Bill -** One issue that must be considered is the issue of the multiple context classes firing more or less at the same time at the same state object, causing the state object to change states at an inconvenient time for a part of the program.