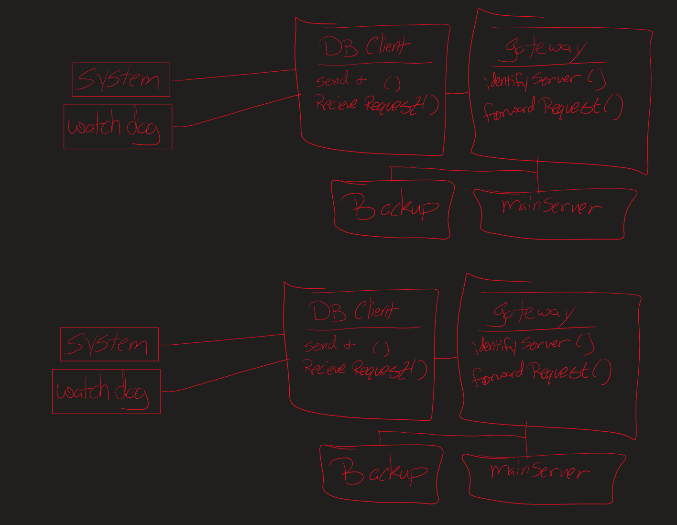
8-4 Consider a workflow system supporting software developers. The system enables managers to model the process the developers should follow in terms of activities and work products. The manager can assign specific processes to each developer and set deadlines for the delivery of each work product. The system supports several types of work products, including formatted text, picture, and URLs. The manager, while editing the workflow, can dynamically set the type of each work product at run time. Assuming one of your design goals is to design the system so that more work product types can be added in the future, which design pattern would you use to represent work products?

I’d probably go with the template method. This would allow for there to be pre-set skeleton design that is used for any future builds.

8-5 Consider a system that includes a database client and two redundant database servers. Both database servers are identical: the first acts as a main server, the second acts as a hot back-up in case the main server fails. The database client accesses the servers through a single component called a “gateway,” hence hiding from the client which server is currently being used. A separate policy object called a “watchdog” monitors the requests and responses of the main server and, depending on the responses, tells the gateway whether to switch over to the back-up server. What do you call this design pattern? Draw a UML class diagram to justify your choice.



This would be a strategy design pattern

9-1 Consider the List interface in the java.util package for ordered collections of

objects. Write preconditions and post conditions in OCL for the following operations:

* int size() returns the number of elements in the list.
  + Precondition: Can’t be null
  + Postcondition: Size will increase by 1
* void add(Object e) adds an object at the end of the list.
  + Precondition: can’t be null, and can’t add a null object
  + Postcondition: added object is added to the set
* void remove(Object e) removes an object from the end of the list.
  + Precondition: can’t be null and removed object can’t be null
  + Postcondition: set is removed
* boolean contains(Object e) returns true if the object is contained in the list.
  + Precondition: object can’t be null and check object can’t be null
  + Postcondition: object returns true/false11
* Object get(int idx) returns the object located at index idx, 0 being the index of the first object in the list.
  + Precondition: nothing can be null
  + Postcondition: index of object is returned

9-4 Consider a Rectangle class and a Square class that inherits from the Rectangle class:

* Write post conditions for the Rectangle.setWidth(w:int) and the Rectangle.setHeight(h:int) operations in terms of the Rectangle.getWidth():int and the Rectangle.getHeight():int operations.
  + Rectangle.setWidth(w:int):
    - After setting the width of the rectangle to w, the value returned by getWidth() should be equal to w.
  + Rectangle.setHeight(h:int):
    - After setting the height of the rectangle to h, the value returned by getHeight() should be equal to h.
* Write an invariant for the Square class stating that the width and height of a Square should always be the same.
  + The width and height of a Square should always be the same.
* Consider the rules for inheriting contracts described in Section 9.4.5 in the context of the Square.setWidth() and Square.setHeight() operations of the Square class. Are all rules met? Why not? What should change in the model?
  + Technically no, not all rules are met because you would not need to set the width and the height. A square has the same width and height, you would only need to set one which is lesser conditions from the rectangle therefor breaking the pre/post condition requirements.