UBC CPSC 410

Solutions to Sample Midterm

2010W

1. The process view in the 4+1 view model for documenting architectures describes the development process used in implementing the system.

F: Process view presents the execution processes and threads of a running program and deals with tne concurrency and synchronization aspects of software execution.

2. In an MVC architecture, the user input device is separated from the user output device, allowing the two to vary independently.

T: The Controller handles input and View handles output.

3. The pipes-and-filters architectural style uses a central repository to store information.

F: Data are passed from one filter (data producer) to the next (data consumer) via pipes.

4. A single architectural style should always be selected for building a software system, in order to avoid confusion between developers.

F: The purpose of having a architectural style is to provide helpful guidance for developers to build scalable and reusable software components. In reality, any large-scale software is often a composition/integration of different architectural styles to fit a client's specific needs.

5. The direction of dependency in a layered architecture is from the more abstract layers to less abstract layers (i.e. lower layers depend on higher layers but not vice versa).

F: It's vice-versa. Higher layers call functions implemented by lower layers.

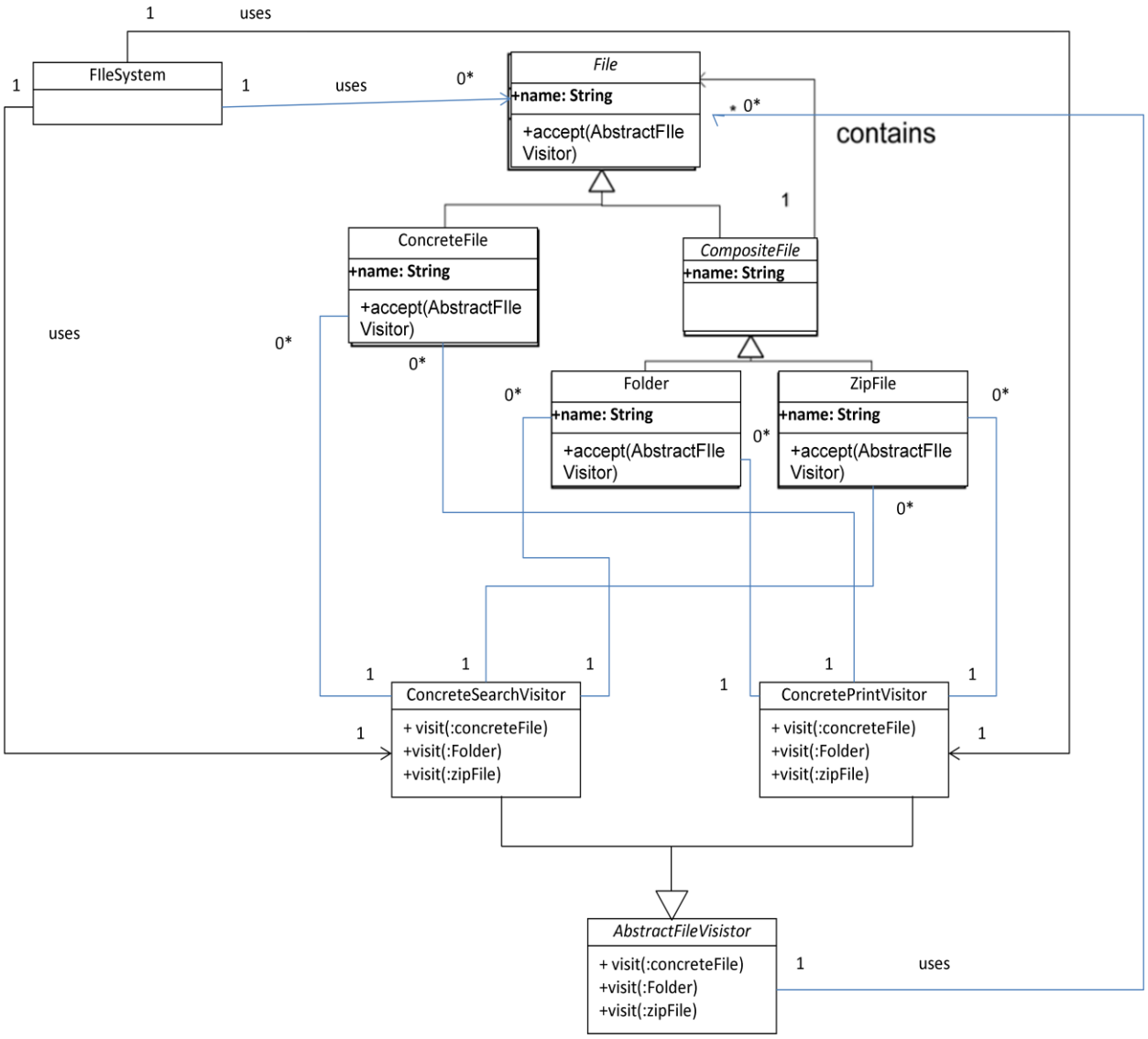
6. Concurrency makes systems easier to program because components do not need to synchronize with each other.

F: Often concurrency makes implementation more difficult because issues such as race-conditions and dead-blocks may arise.

7. *Visitor Pattern*

Below is the object-oriented decomposition for a file system consisting of regular (concrete) files, folders and compressed folders (zipfiles). The design supports two features: search and print, as indicated by the methods shown. The print feature simply prints the hierarchical structure of the files. The search feature traverses the files recursively, adding any file with a name that matches the String argument to the List of returned results.

Draw an URL diagram illustrating a functional decomposition of the same system using the Visitor pattern. (Hint: You will need a Visitor interface and two concrete Visitors, which may contain both field and method declarations; you will also need to make changes to the given design below, you can simply mark any changes to the diagram or redraw the diagram). Additional space is provided on the next page.



8. The keyword-in-context system accepts an ordered set of lines, each line is an ordered set of words, and each word is an ordered set of characters. Any line may be circularly shifted by repeatedly removing the first word and appending it at the end of the line. The keyword-in-context index system outputs a listing of all circular shifts of all lines in alphabetical order.

Sketch a software architecture for the keyword-in-context index system using any architectural style that was mentioned in class or in the reading. The diagram will contain components and connectors specifically for this example. Name the components according to their purpose in this example. Explain in two or more sentences (each), three software qualities which benefit from your choice of architectural style.

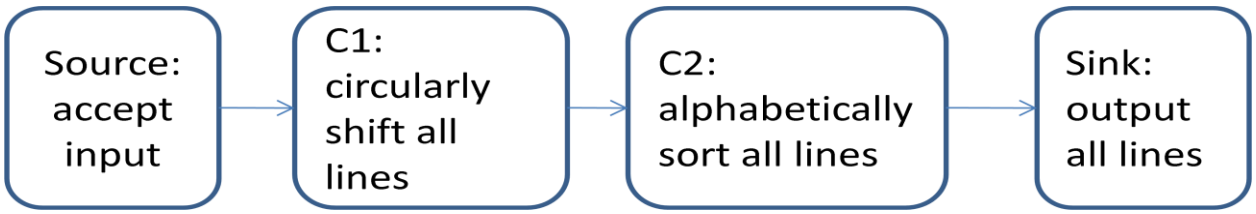
Sample Input	(1)	(2) & (3): Sample Output
aa bb cc az yz	aa bb cc bb cc aa cc aa bb az yz yz az	aa bb cc az yz bb cc aa cc aa bb yz az

Architectural style does not take into account data-structures, so we'll ignore relationship between lines and words: e.g. "each line is an order set of words."

The system

1. Generate permutations of all circular shifts of each line (1), then
2. In alphabetical order, sorts all lines (2), then
3. Outputs all sorted lines (Sink).

- (1) executes before (2), (2) before (Sink), implying dataflow style
- (1) must finish execution entirely before (2); (2) must finish execution entirely before (Sink), implying batch sequential, not pipe-and-filter (pipe-and-filter enables streams)



1. Maintainability
    a. Components operate independently and are self-contained
    b. Each component has no static knowledge of neighboring components
2. Extensibility/Elasticity
    a. Easy to add additional processing-components to system
    b. Also applies to modifying/removing components
3. Reliability/Stability/Security: Only 1 component is executing at any given time
    a. Easy to identify and fix source of failure
    b. No concurrency issues such as deadlocks/race conditions

9. What is the output of executing the following program after weaving the following aspect?

```
public class Test {
  public static void main(String[] args) {
    Integer i = new Integer(5); // 1.
    Integer j = new Integer(6); // 2.
              // 5.              3.              4.
    Integer k = new Integer(i.intValue() + j.intValue());
                         // 6.
    System.out.println(k.toString());
  }
}

public aspect Tracer {
  pointcut traced() :
    (call(* Integer.*()) || call(Integer.new(..)) ) &&
    !call(* *.toString());

  after() : traced() {
    System.out.println("Called Trace Method");
  }
}
```

Output

```
1.System.out.println("Called Trace Method");
2.System.out.println("Called Trace Method");
3.System.out.println("Called Trace Method");
4.System.out.println("Called Trace Method");
5.System.out.println("Called Trace Method");
6.11
```

10. A three-tier system consists of a client browser connected to a web server, and the web server is connected to a database. Does the control-flow and data-flow of request processing in a three-tier system behave closer to that of a layered architecture or a pipe-and-filter architecture?
Explain why in two to four sentences. The answer must explain why it is similar to your choice and also not similar to the other.

Data-flow: more similar to Layered Abstract Machine.

- In 3-tier, both input data and output data occur at the Front-end / Top / Application layer.
- Data-flow in both 3-tier and LAM is bi-directional from one layer to the next and then back.
- Data-flow in pipe-and-filter is only 1-way: input at the producer-end and output at the consumer-end.

Control-flow: more similar to Layered Abstract Machine. In 3-tier:

- Front-layer can depend on multiple back-end servers, like lower-level layers in LAM
- Web-services / XMLHttpRequest are like remote procedure-calls.
- In pipe-and-filter, control flow is 1 filter at a time, almost no multiple dependencies.

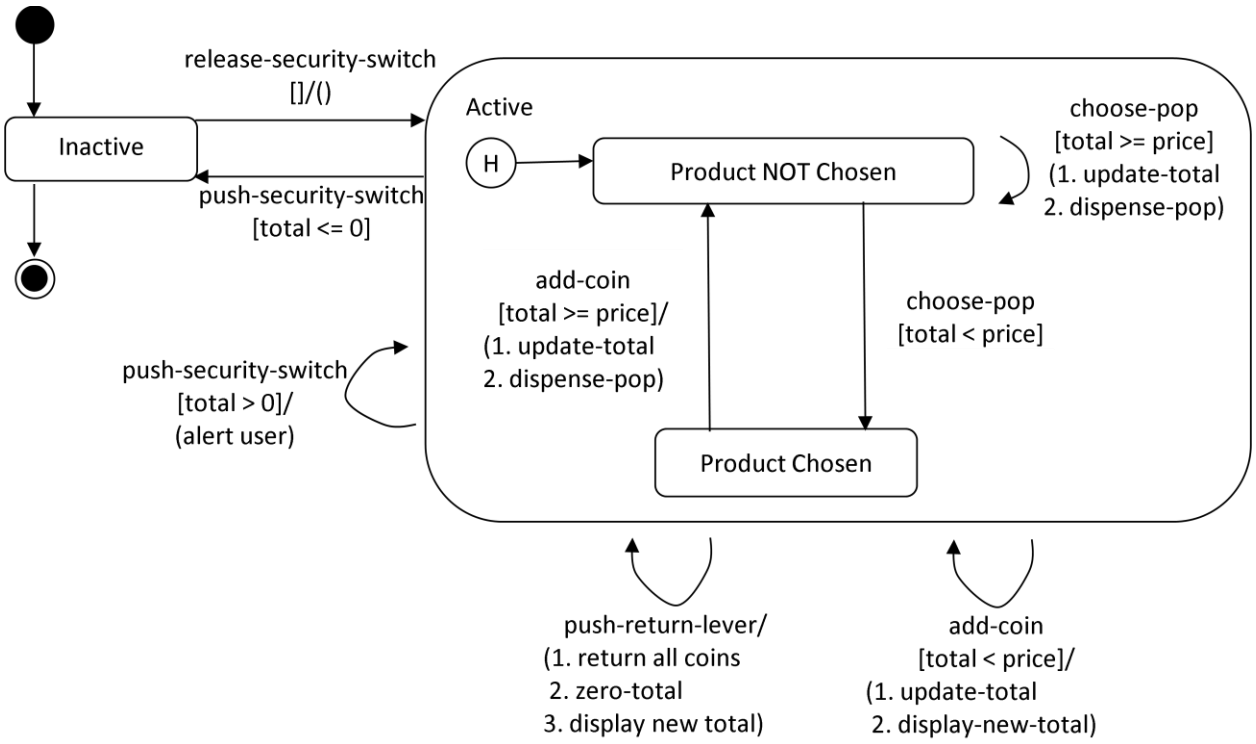
11. Statecharts

You have been contracted by PopStar Beverage Corporation to design and test a line of pop machines. Below are the requirements specified by the customer.

- The machine is in inactive mode until a security switch is released.
- Users can choose pop products by pressing buttons on the machine.
- When the machine is not off, coins can be added, and a coin total is displayed to the user.
- After a coin has been added, the machine detects the value of the coin and adds it to the total.
- If a product is chosen and the current total is exactly equal to the product price, a product is dispensed.
- If a product is chosen and the current total is greater than the product price, a pop is dispensed, and the difference becomes the new coin total.
- If the machine is put back to inactive mode and the coin total is not zero, an error message is displayed to the user.
- A coin return lever can be pressed by the user when the machine is active. An amount equal to the current total will be dispensed to the user and the total will be set to zero.

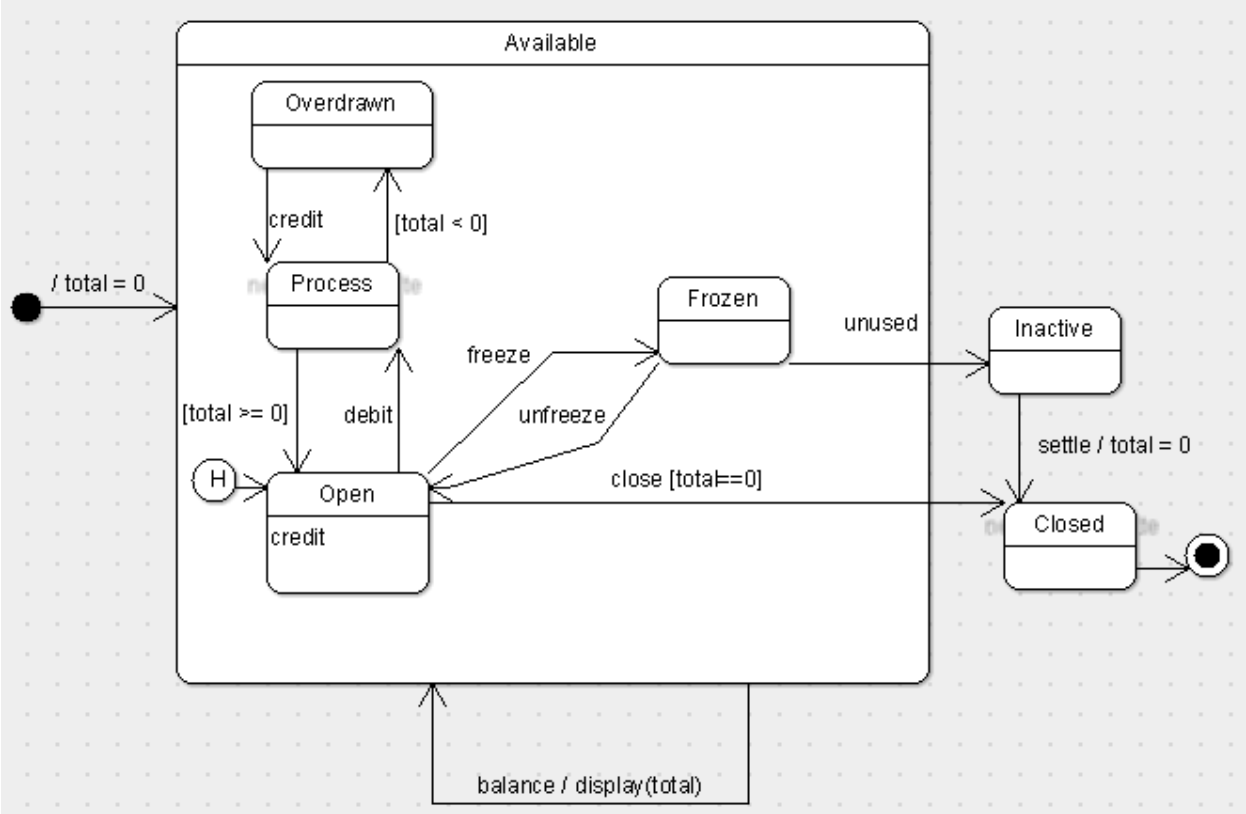
Draw a state-machine which represents the above requirements. Use transition guards where applicable but only when necessary.

Syntax: **Event[condition]/(action)**



CPSC 410: Statechart Exercise

For which states, *t*, is the following true, on *at least one path* (i.e. state sequence) starting at *t*:



1. eventually(*t*, Frozen) Open/Process/Overdrawn/Frozen
2. until(*t*, total < 0, Process) Process/Overdrawn
3. globally(*t* , total ==0) Open/Frozen/Inactive/Overdrawn/Process
4. next(*t*, total == 3.14) Open/Process/Frozen/Overdrawn
5. There exists a state, *s*, such that: next(*t*, *s*) && globally(*s*, total == 0)

   s => Open/Frozen/Inactive/Overdrawn/Process

therefore t => Open/Process/Frozen/Overdrawn/Process