CPSC 410 Midterm Study Questions: T1 2011

1. The process view of software architecture describes the development process used in implementing a software system.
T or **F**. Justification:

It describes the active runtime processes and threads.

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

T or **F**. Justification:

Data flows sequentially through filters and is not available globally.

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

T or **F**. Justification:

Different styles are useful at different layers of abstraction or in different sub-systems.

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).

T or F. Justification:

The more abstract layers build on top of the services of lower layers.

However the (i.e.) is a typo and is contradictory, because “higher” layers depend on “lower” layers! So, considering that, this sentence could also be considered false.

6. a. What is one difference between the Decorator and the Adapter pattern, in terms of the situations where each on is applicable (i.e. useful)?

Decorator wraps (i.e. delegates to) the same interface type that it implements. The Adapter is used to mediate between different interface types.

6.b. What is one advantage of the DynamicProxy over a Decorator?

It can be applied to any interface type at runtime so different implementations are not needed for different interfaces.

6.c. Write an implementation of InvocationHandler which transforms the return value of all String-type-methods to lowercase. A String-typed-method is any method that has a String return-type. Note: You may want to use the "instanceof" operator, and the String.toLowerCase() method.

```
class LCP implements InvocationHandler extends Delegator<Object> {

public Object invoke(Object p, Method m, Object[] a) throws Throwable
     {
          Object retVal = m.invoke(delegate, a);
          if(retVal instanceof String) {
               retVal = ((String) retVal).toLowerCase();
          }
          return retVal;
     }

}
```

7. Describe one difference between each pair of the following architectural styles. Don't repeat the same answer for any of the pairs (even if the same answer could apply twice). Note: There are 15 pairs.

Virtual-Machine style
Client-Server style
Pipe-and-Filters style
Batch Sequential style
Blackboard style
Publish-subscribe style

a. VM vs. CS
   CS provides guidelines for distributed system network connectivity, but not VM
b. VM vs PF
   PF helps to support concurrency between filters but concurrency is not addressed by VM
c. VM vs BSeq
   In BSeq data-flow is uni-directional
d. VM vs BB
   BB provides notification of datastorage updates but not VM
e. VM vs PS
   PS distinguishes publishers and subscribers but in VM all components are just layers
f. CS vs PF
   In CS, clients don't service requests, but in PF, all filters take some input.
g. CS vs BSeq
   BSeq helps to organize a workflow consisting of several tasks and CS does not decompose components along task boundaries.
h. CS vs BB
   The "server" in BB is always a generic datastorage platform while server in CS may take on arbitrary application functions.
i. CS vs PS
   PS provides routing infrastructure
j. PF vs BSeq
   PF can work on data-streams rather than coarse-grained application data units.
k. PF vs BB
   BB has a central repository of data but PF does not.
l. PF vs PS
   PS systems form an arbitrary network overlay graph, while PF are generally simple DAGs
m. BSeq vs BB
   Agents are autonomous (may come and go) while BSeq must respect dependencies between components
n. BSeq vs PS
   BSeq does not deliver data based on content subscriptions
o. BB vs PS
   BB manages persistent data but PS delivers transient messages.

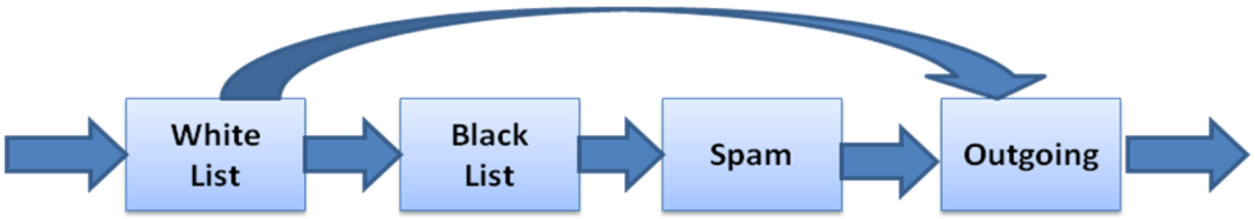
For the following problem, describe the system architecture in the following form.

a. Name one architectural style that you will use.
b. Draw a diagram that describes your system.
c. Explain briefly how the system works.
d. State two important advantages of using this architectural style for the problem.

8. E-mail Filter

An e-mail system filters incoming e-mails with a whitelist (e-mails from senders on the whitelist are accepted), a blacklist (e-mails from senders on the blacklist are deleted), and the Spamassassin tool (e-mails that do not pass this check are marked as spam). The system will run on a single-core server machine, but may be moved to a multi-core server if the load gets too high.

a. Pipe-and-filters
b.



c.
Input starts at WhiteList,
if matched, forwarded directly to Outgoing (for final delivery)
if not matched, forwarded through BlackList and Spam

d. (1) Easy to add additional filters
(2) Filters can be scheduled concurrently on different machine cores

9. Consider an auction Web site that allows developers to subscribe for auction item notifications using XPath.

The publication of book items on the site follows this example structure:

```
<items>
     <book isbn="11111" cat="fiction">
          <title lang="chn">Learning XPath</title>
          <retailPrice unit="us">79.99</retailPrice>
          <currentBid unit="us">32.00</currentBid>
     </book>
     …
</items>
```

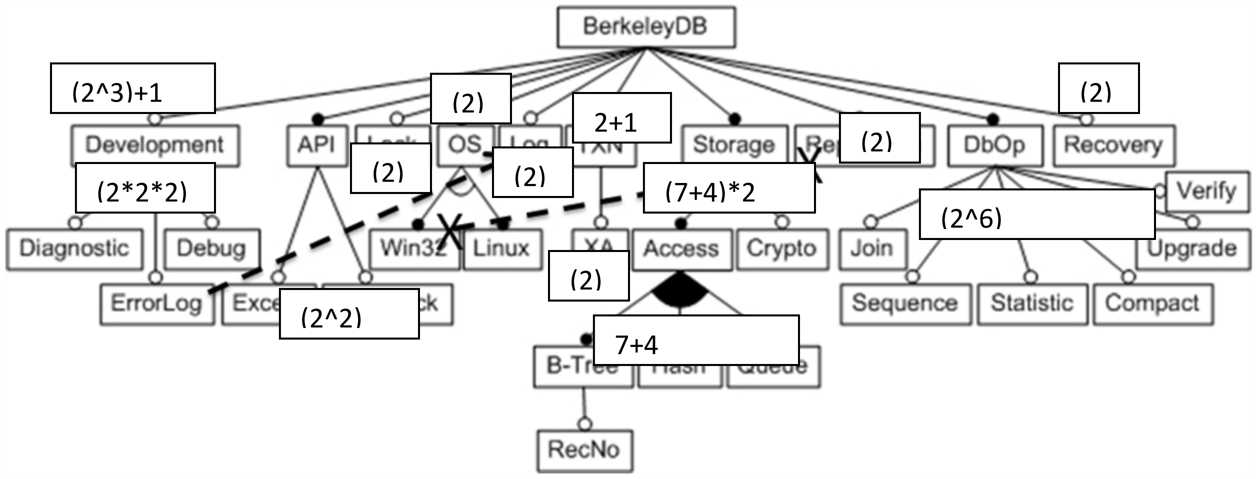
Write XPath to subscribe to the following:

a. The isbn of all book with a title in the language English (i.e. lang='en').
b. The title of all books with a price greater than 35.00 but less than 45.00.
c. All books where the currentBid is more than $40.00 less than the retailPrice.
d. The lowest currentBid among all books in an <items> XML. Note that inequality operators in XPath operate on node-sets (similar to the equality operators, discussed in the notes). So, x < y, is true if *any* element matched by x is less than *any* element matched by y, where x and y are paths.

```
a. //book[title/@lang='en']/@isbn

b. //book[retailPrice/text() > 35.00][retailPrice/text() < 45.00]/title

c. //book[(retailPrice/text() - currentBid/text()) > 40]

d. //currentBid[not(text() > //currentBid/text())]
```

10.

How many product instances can be made from this product line? Hint: Use a "divide-and-conquer" strategy starting with the leaves and working bottom-up. Make sure to account for conflicts and requires.



[(((2^2) * 2 * 2 * 3 * (11*2) * 2 * (2^6) * 2) * (3/4)] * [((2^3) + 1) * 2 – 4]

a. Multiply together the number of choices for each branch, except Development and Log
b. Multiply by the fraction of (a) which don't contain both Win32 and Replication (i.e. 3/4)
c. Determine the number of choices for Development and Log, which don't contain ErrorLog without Log
d. Multiply together (b) and (c)

11.a. Using Linda syntax, implement a blackboard-style Stack data-structure that provides the push and pop operations. This can simply be in the form of a push and pop functions that implement the signatures below (i.e. it is not necessary to consider classes or object fields). To get started, examine the example of the blackboard-style Array data-structure in the lecture notes.

Object pop(String stackID);
void push(Object value, String stackID);

```
Object pop(String stackID) {
     int size;
     in(stackID, “size”, ?size);
     Object retVal;
     in(stackID, ?retVal, size-1);
     out(stackID, “size”, size-1);
}
void push(Object value, String stackID) {
     int size;
     in(stackID, “size”, ?size);
     out(stackID, value, size);
     out(stackID, “size”, size+1);
}
```

(*b*) Examine the Client-Server Linda example provided in the lecture notes. Extend this example by providing a new function `ServerLoad()` which returns the current "load" on the server, defined as "the number of requests awaiting service"' You may also need to change the code inside the Client and Server to facilitate this new function.

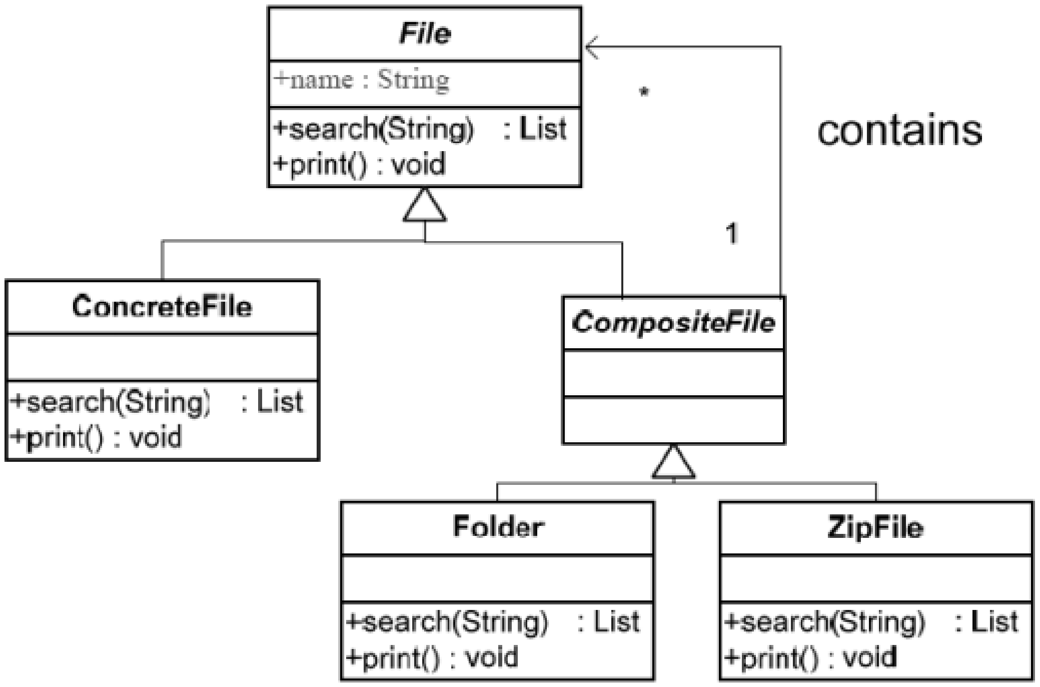
```
void client() //No changes required
{
 int index;
 …
  in (“server index”,? index);
  out (“server index”, index +1);
  …
  out (“server”, index, request);
  in (“response”, index, ? response);
  …
}

void server()
{
  int index = 1;
  int response_index = 1;
  out(“response index”, response_index);
  …
  while (true){                              //This was while(1) in the notes which is just C language for while(true)
      in (“request”, index, ?req);
      …
     out (“response”, index++, response);
     out(“reponse_index”, response_index++;);
  }
}

int serverLoad() {
  int server_index;
  read(“server index”, ?server_index);
  int response_index;
  read(“response index”, response_index);
  return server_index – response_index;
}
```

12.

Visitor Pattern

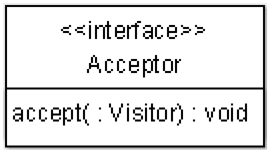
Below is the object-oriented decomposition for a file system consisting of regular (concrete) files, folders, and compressed folders (zipfiles). The design supports two features: search and print, as indicated by the methods shown. The print feature simply prints the hierachical structure of the files. 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 a UML 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.

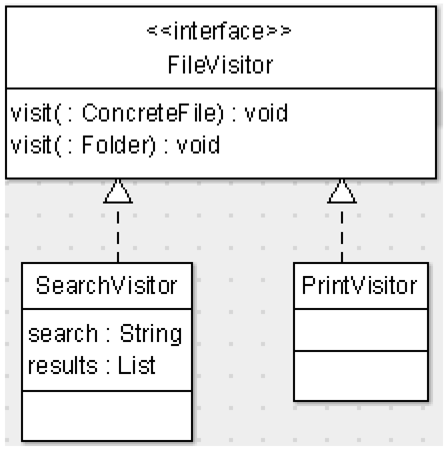


Changes to existing classes:

a. `search` and `print` methods can be removed.
b. All existing classes must implement accept method for Visitor pattern, this could be shown in the diagram by forcing the File abstract class to implement this interface below.



New classes:



<<interface>>
FileVisitor
visit(: ConcreteFile) : void
visit(: Folder) : void
SearchVisitor
search : String
results : List
PrintVisitor