

CPSC 410 — Advanced Software Engineering  
Midterm Examination (Term I 2011)  
Instructor: Eric Wohlstadter

You have **50 minutes** to complete this exam. You must sign your name in the space provided below.

Candidates guilty of any of the following, or similar, dishonest practices shall be immediately dismissed from the examination and shall be liable to disciplinary action:

- Speaking or communicating with other candidates, or anyone outside of the exam area.
- Purposely exposing written papers to the view of other candidates. The plea of accident or forgetfulness shall not be received.

If you have a question during the exam, raise your hand and one of the invigilators will come and answer your question. We will only answer questions about errors in the exam. If you have to make an assumption about a question, write down the assumptions you make.

Partial credit for a question awarded on scales of 25%, 50%, or 75% out of the question total.

The exam is out graded out of 100 marks. The value of each question is indicated. Use your time wisely. If you do not know the answer to a question, move on to the next question and come back to the question at the end.

The exam is double-sided. There are questions on **both** sides of the page.

Good luck!

Print Name: \_\_\_\_\_

Signature: \_\_\_\_\_

1.	/4
2.	/4
3.	/4
4.	/14
5.	/20
6.	/10
7.	/10
8.	/14
9.	/20
	/100

**1. (4 pts)**

The *development view* of software architecture captures the structure of classes and their relationships.

T or F. Justification:

The development view captures the dependencies and distribution of deployable packaged components such as jar files or dll files.

**2. (4 pts)**

In contrast to the *Publish-Subscribe* architecture, the *Client-Server* architecture provides for distributed systems deployment.

T or F. Justification:

Both of them provide for distributed systems deployment.

**3. (4 pts)**

In a *Virtual Machine* architecture, data can propagate both from higher to lower layers and also from lower to higher layers.

T or F. Justification:

Higher layers use function calls to communicate with lower layers, so data propagates from high to low as function calls and low to high as return values.

**4. a.** (4 pts)

According to the connector taxonomy, what is one potential use for an *Arbitrator* connector?

Enforce security controls on communication, manage concurrency, etc...

**4.b.** (4 pts)

Compared to *Decorator* and *Adapter*, why does *RemoteProxy* have no `delegate` object?

It delegates over a network and does not extend from *Delegator*.

**4.c.** (6 pts)

Write an implementation of *InvocationHandler* which filters out all *String*-type-arguments that contain inappropriate text. You can assume the existence of a helper method *isInappropriate*. Filtered strings should be replaced with " ".

```
class Filter implements InvocationHandler extends Delegator<Object> {  
  
    public Object invoke(Object proxy, Method m, Object[] args) throws  
        Throwable  
    {  
        for(int i=0;i < args.length; i++) {  
            Object current = args[i];  
            if((current instanceof String) && isInappropriate(current)) {  
                args[i] = " ";  
            }  
        }  
        return m.invoke(delegate, args);  
    }  
}
```

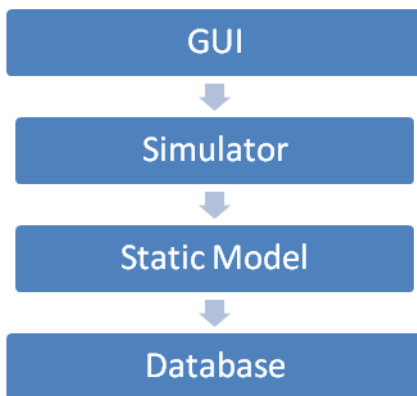
For the following problem, describe the system architecture in the following form.

- a. (4 pts) Name one architectural style that you will use.
- b. (4 pts) Draw a diagram that describes your system.
- c. (4 pts) Explain briefly how the system works.
- d. (8 pts) State two important advantages of using this architectural style for the problem.

5. You need to create the initial design to organize a project for implementing a CAD tool (Computer-Aided Design). The CAD tool will be used by mechanical engineers to design and simulate 3D models of physical materials, e.g. gears, wheels, axles, etc... Engineers should be able to interact with the design using the mouse, which will cause the program to simulate the effect of different forces being applied to the materials in real-time. Each design should be saved in a database. However, the information in the database only captures the static structure of the design (how materials are connected), but not the dynamics (how they interact in a simulation). This simulation effect would be added by another part of the program.

- a. Any architecture except for batch sequential, pipe-and-filter, and publish-subscribe, given a reasonable description. I'll use VM architecture here.

b.



- c. User interacts with UI which triggers event handlers in the GUI layer. GUI layer responds by telling simulator which elements are manipulated and requesting feedback about model changes. Simulator uses basic structural information read from the static model which is saved to and loaded from the database. Optionally, the GUI could bypass the simulator and communicate directly to the Static Model for simple changes to the model structure.
- d. Easy to replace storage layer without affecting the rest of the program, e.g. move from database to Web service. Simulation layer could be reused in a context where an interactive GUI was not used, e.g. to provide information to an artificial intelligence layer rather than an end-user.

6. 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. (5 pts) One book where the title is "Essential Java". If more than one book has that title, still only one of them should be matched.

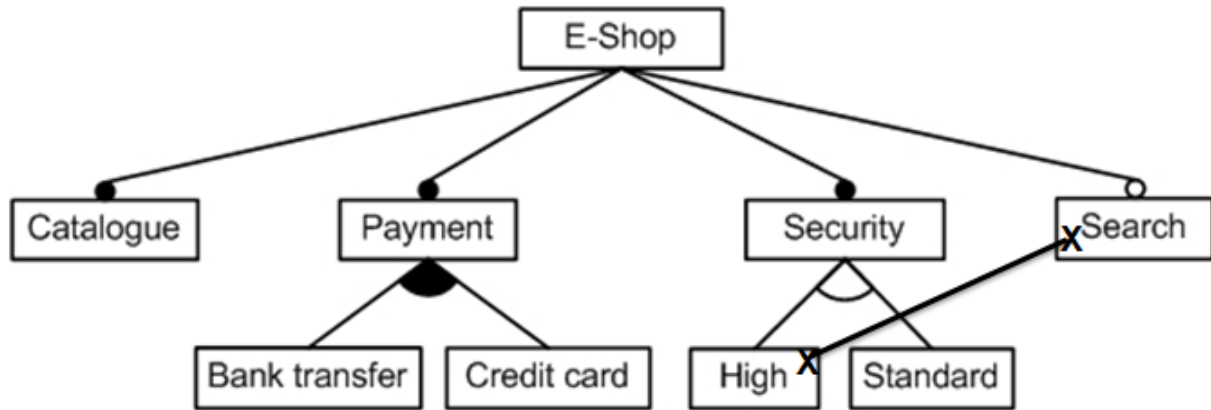
```
//book[title/text() = ' Essential Java '][1]
```

- b. (5 pts) The highest currentBid among all books in an <items> XML.

```
//currentBid[not(text() < //currentBid/text())]
```

7. (10 pts)

How many *product instances* can be made from this *product line*?



Payment:: 3

Security and Search::  $(2 * 2) - 1 = 3$

$3 * 3 = 9$

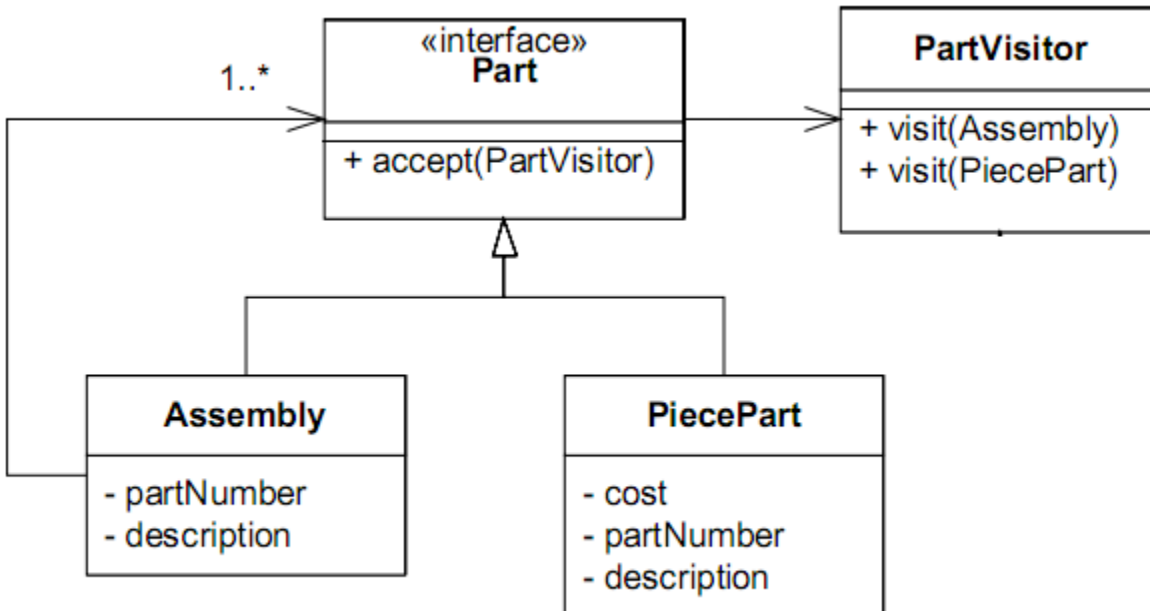
**8. (14 pts)**

Using *Linda* syntax, implement a blackboard-style FIFO queue data-structure that provides the enqueue and dequeue operations.

```
Object dequeue(String qID);  
void enqueue(Object value, String qID);
```

```
Object dequeue(String qID) {  
  
    int first, size;  
    Object retVal = null;  
    in(qID, "size", ?size);  
    if(size == 0) {  
        out(qID, size);  
        return retVal;  
    }  
    in(qID, "first", ?first);  
    in(qID, first, ?retVal);  
    out(qID, "size", size-1);  
    out(qID, "first", first+1);  
    return retVal;  
}  
  
void enqueue(Object value, String qID) {  
  
    int first, size;  
    in(qID, "size", ?size);  
    in(qID, "first", ?first);  
    out(qID, first+size, value);  
    out(qID, "size", size+1);  
    out(qID, "first", first);  
}
```

9. Consider the following UML diagram of a *Visitor* pattern.



a. (12 pts)

Each `Assembly` object is a composite data-structure that possibly contains `PiecePart` objects and other `Assembly` objects (recursively). You can assume that the objects in the data-structure form a tree (i.e. an `Assembly` cannot contain an `Assembly` which contains it, and also each `PiecePart` is directly contained by only one `Assembly`).

Implement an `accept` method for the `Assembly` class. You will need to assume some implementation for the 1-to-many relationship. Make any sensible choice and document it.

```
List<Part> children; //1-to-many relationship
```

```
public void accept(PartVisitor pv) {
    pv.visit(this);
    for(Part part : children) {
        //Assume 1-to-many as
        part.accept(pv);
    }
}
```

**b. (8 pts)**

Now assume that `PartVisitor` has been extended with an `endVisit` method for both the `Assembly` and `PiecePart` types.

Notice that each `PiecePart` has a cost associated with it, but its *effective cost* depends on its position in the composite data-structure. *The effective cost* of a `PiecePart` is equal to its cost multiplied by its number of `Assembly` ancestors.

Write a `Visitor` implementation to compute the aggregate *effective cost* for all `PiecePart` objects in an `Assembly` (i.e. for all `PiecePart` descendents).

Announced at exam: You can assume a `PiecePart.getCost()` method which returns `PiecePart.cost`

```
public class CostVisitor extends PartVisitor {
    int depth = 0;
    double cost = 0.0;

    public void visit(PiecePart p) {
        cost += p.getCost() * depth;
    }

    public void endVisit(PiecePart p) { }

    public void visit(Assembly p) {
        depth++;
    }

    public void endVisit(PiecePart p) {
        depth--;
    }
}
```