2805ICT System and Software Design
3815ICT Software Engineering
7805ICT Principles of Software Engineering
This is an assessed workshop

WORKSHOP 7 **UML**

Student name: Yasin Cakar Student ID: S2921450 Enrolled Course Code: 3815ICT

Student name: David Todorovic Student ID: S5167246 Enrolled Course Code: 3815ICT

Student name: Jerry Li Student ID: S2928643 Enrolled Course Code: 3815ICT

Student name: Mohammad Mari Student ID: S5185052 Enrolled Course Code: 7805ICT

1. Identify suitable design patterns

For each of the design purpose, provide the name of a suitable design pattern. (20pts)

1. Arrange for a set of objects to be affected by a single object

   Answer: Observer Design Pattern

2. Provide an interface to a package of classes

   Answer: Façade Design Pattern

3. Increase flexibility in calling for a service e.g., allow undo-able operations

   Answer: Command Design Pattern

4. Cause an object to behave in a manner determined by its state

   Answer: State Design Pattern

5. Allow a set of objects to service a request and Present clients with a simple interface

   Answer: Chain of Responsibility Design Pattern

6. Create a set of almost identical objects whose type is determined at runtime

   Answer: Prototype Design Pattern

7. Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation

   Answer: Iterator Design Pattern

8. Ensure that there is exactly one instance of a class

   Answer: Singleton Design Pattern

9. Allow runtime variants on an algorithm

Answer: Template Design Pattern

10. Create individual objects in situations where the constructor alone is inadequate

    Answer: Factory Design Pattern

11. Interpret expressions written in a formal grammar

    Answer: Interpreter Design Pattern

12. Add responsibilities to an object at runtime

    Answer: Decorator Design Pattern

13. Represent a tree of objects

    Answer: Composite Design Pattern

14. Avoid references between dependent objects

    Answer: Mediator Design Pattern

15. Allow an application to use external functionality in a re-targetable manner

    Answer: Adaptor Design Pattern

16. Provide an interface for creating families of related or dependent objects without specifying their concrete classes

    Answer: Abstract Factory Pattern

17. Manage a large number of objects without construction them all

    Answer: Flyweight Design Pattern

18. Avoid the unnecessary execution of expensive functionality in a manner transparent to clients

    Answer: Proxy Design Pattern

2. Command design pattern

a) Study the entry in Wikipedia for the “Command pattern” (https://en.wikipedia.org/wiki/Command_pattern). Review the following piece of code. Explain why this represents an anti-pattern:), What are the disadvantages of the template suggested by the anti-pattern above?(10pt)

```
if (op==1)
{
        User u = new User( request.getAttribute("user"), request.getAttribute("pass"));
}
else if (op==2)
{ ... }
...
```

Answer:

This code does not conform to the command design principles where the intended behaviour is encapsulated with all information, objects, methods and its parameters needed to perform an action at a later time in a command object

For options 1 and 2 can be represented as states in the command object to be placed in slots that can be called to be implemented using the execute() method of the Command interface. Rather than creating the object “u”, it should perform an action that delegates this process.

b) Examine the code below that defines an abstract class. What part of the Command pattern does this achieve? What does the constructor achieve? (10pt)

```
//Action.java
public abstract class Action
{
        protected Model model;
        public Action(Model model)
        {
                this.model = model;
        }
        public abstract String getName();
        public abstract Object perform(HttpServletRequest req);
}
```

Answer:

This shows the code that represents the command interface, that will have the command interface’s execute() method overwritten. The constructor passes the receiver “model” to the command.

c) Examine the code below that defines a concrete class. What part of the Command pattern, does this achieve? Explain the method getName().(10pt)

```
// CreateUserAction.java:
public class CreateUserAction extends Action
```

```
{
    public CreateUserAction(Model model)
    {
        super(model);
    }
    public String getName()
    {
        return "createUser";
    }
    public Object perform(HttpServletRequest req)
    {
        return
    model.createUser(req.getAttribute("user"),req.getAttribute("pass"));
    }
}
```

Answer:

This code represents a **concrete command**, the *getName()* function returns the name of the command.

d) Examine the code below that defines a concrete class. What part of the Command pattern does this achieve? If this code is part of MVC, which part does it belong to? (10pts)

```
public class ControllerServlet extends HttpServlet {
    private HashMap actions;
    public void init(Model model) throws ServletException {
        actions = new HashMap();
        CreateUserAction que = new CreateUserAction(model);
        actions.put(que.getName(), que);
        //... create and add more actions
    }
    ...
}
```

Answer:

This code represents the **invoker** in the command pattern. In a model view controller paradigm this code serves as the **controller** aspect of the MVC design Pattern

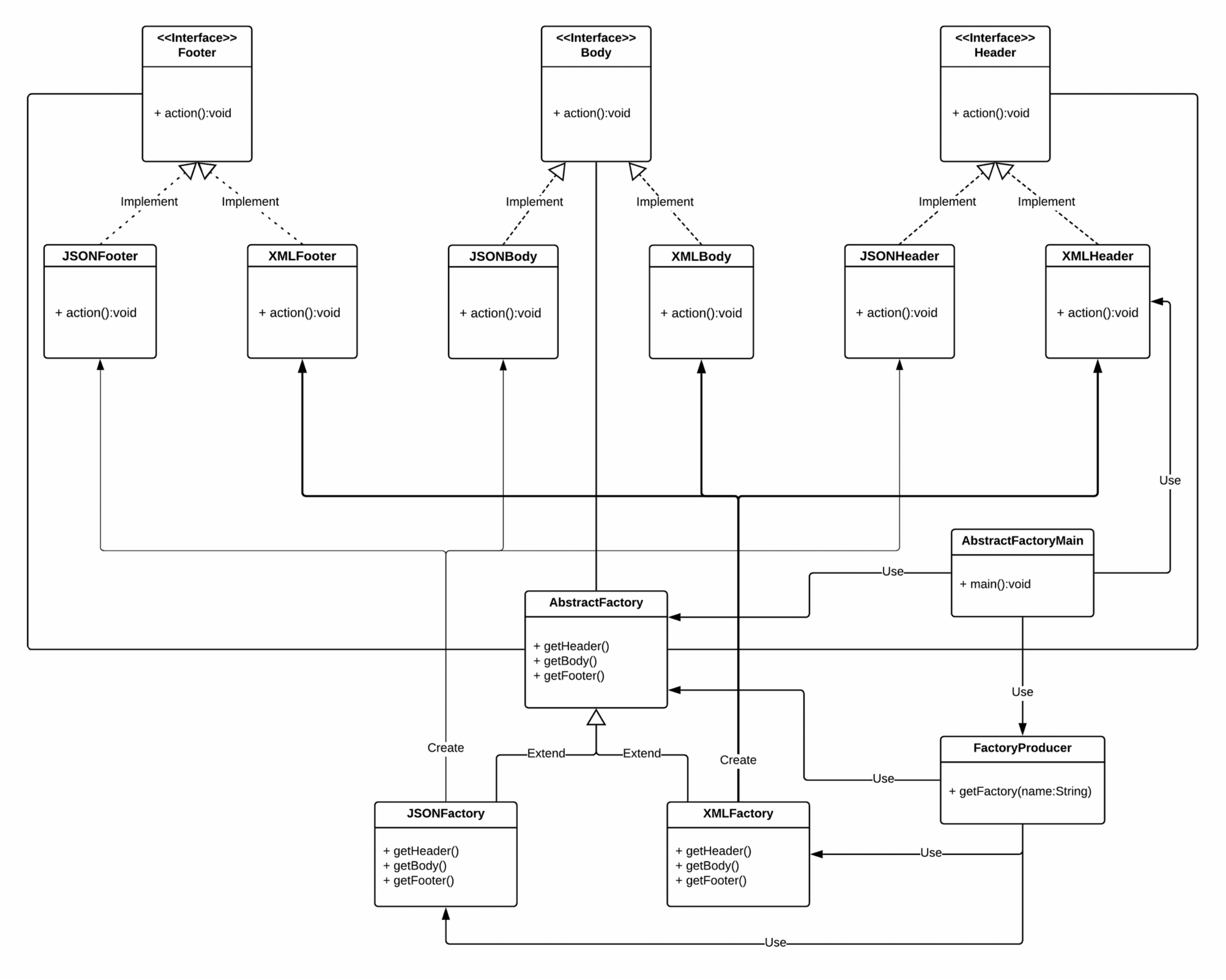
3. Abstract factory pattern

Study the code below of an abstract factory pattern,

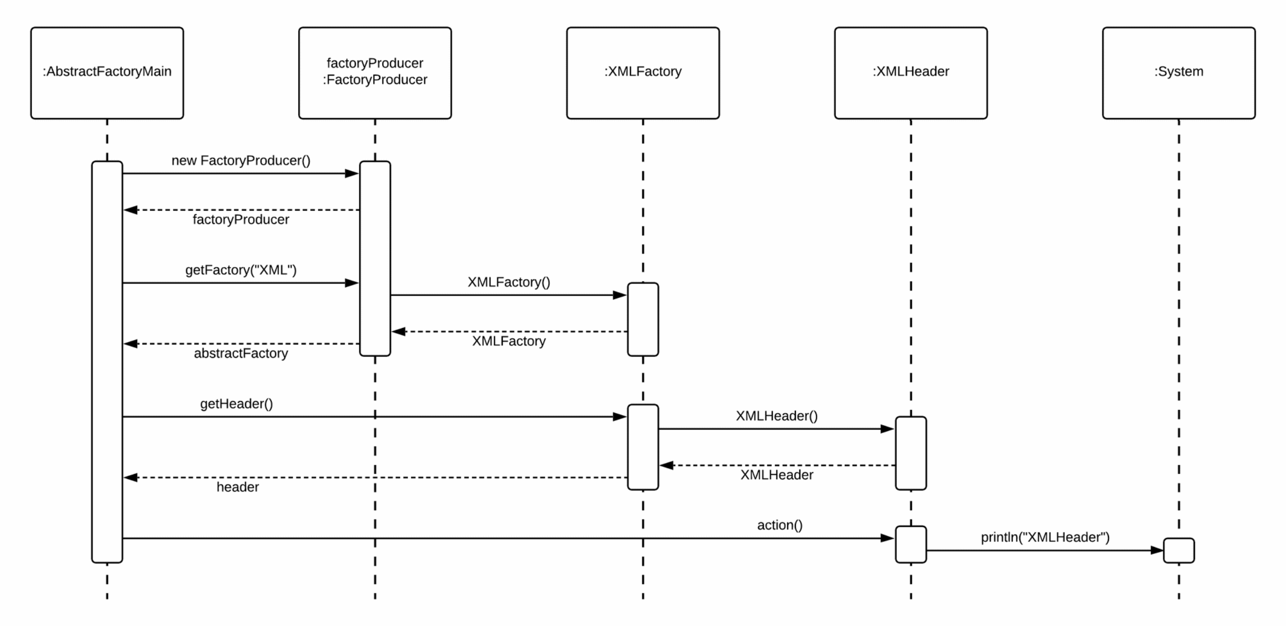
```
interface Header  {
   void action();
}

interface Body {
   void action ();
}

interface Footer {
   void action();
}

class JSONHeader implements Header {
   public void action() {
      System.out.println("JSONHeader");
   }
}

class JSONBody implements Body {
   public void action() {
      System.out.println("JSONBody");
   }
}

class JSONFooter implements Footer {
   public void action() {
      System.out.println("JSONFooter");
   }
}

class XMLHeader implements Header {
   public void action() {
      System.out.println("XMLHeader");
   }
}

class XMLBody implements Body {
   public void action() {
      System.out.println("XMLBody");
   }
}

class XMLFooter implements Footer {
   public void action() {
      System.out.println("XMLFooter");
   }
}

abstract class AbstractFactory {
   abstract Header getHeader();
   abstract Body getBody();
```

```
    abstract Footer getFooter();
}

class JSONFactory extends AbstractFactory{
    Header getHeader() {
        return new JSONHeader();
    }

    Body getBody() {
        return new JSONBody();
    }

    Footer getFooter() {
        return new JSONFooter();
    }
}

class XMLFactory extends AbstractFactory {
    Header getHeader() {
        return new XMLHeader();
    }
    Body getBody() {
        return new XMLBody();
    }
    Footer getFooter() {
        return new XMLFooter();
    }
}

class FactoryProducer {
    AbstractFactory getFactory(String name) {
        switch (name) {
            case "XML": return new XMLFactory();
            case "JSON": return new JSONFactory();
        }
        return null;
    }
}

public class AbstractFactoryMain {
    public static void main(String[] args) {
        FactoryProducer factoryProducer = new FactoryProducer();
        AbstractFactory abstractFactory = factoryProducer.getFactory("XML");
        Header header = abstractFactory.getHeader();
        header.action();
    }
}
```

a) draw a class diagram that contains all the classes and methods (you may ignore multiplicity in the diagram)(20pts).



b) Draw a sequence diagram to show scenario (20pts).



4. Additional exercises for 3815ICT and 7805ICT

This question is not assessed, however, if you don't complete this question, your other questions will not be marked (the question is only for students who enrol in 3815ICT and 7805ICT).

Write 15 lines of a reflective report on the previous activities. Analyse and evaluate their relevancy to your future work.

Answer:

This week's topic was concerned with the importance in employing the relevant design patterns based that is determined by the design scenario, which is consequently determined by the requirements of the application context.

The design concepts introduced for this topic include state-logic-display, Model view Controller pattern, Module view of Integrare, Presentation-Abstraction-Controller, Sense-Compute Control and various other well-known design patterns.

The design pattern is not intended to directly provide a complete designed code solution, but to solve a variety of different problems in various situations. It does not involve solving a specific category of objects or completing the application. Design patterns can rely on relative certainty, as it is a guiding framework and not a concrete design template, its purpose, specifically is relative avoidance of common design problems, that aims to avoid tight coupling of models, controllers and views, so as to enhance the of software design quality by making it adaptable to changes, by means of providing modularity and testability of code products.

Design patterns specifically refer to problems at the software "design" level. There are other non-design patterns, such as structural patterns. At the same time, the algorithm cannot be a design mode, because the algorithm is mainly to solve the problem of calculation, not the problem of design.

As the software development community's interest in design patterns grows, some related expertise has been published, special lectures are held from time to time, and Ward Cunningham himself invented the experience of Wikipedia to communicate design patterns.

The design literature that deals with design patterns also focus on common design mistakes that are initially carried out in order to solve common problems. This phenomena is known as anti-patterns. Anti-patterns are common mistake of design approaches that employ a seemingly correct solution that proves to be highly ineffective, inefficient or even causing further problems in the long term.

5. Additional exercises for 7805ICT

This question is not assessed, however, if you don't complete this question, your other questions will not be marked (the question is only for students who enrol in 7805ICT).

Design an open-ended question (that means there may be several correct answers) that could be suitable for

1. A final exam
2. A job interview for software engineer.

Answer:

Are design patterns language specific? Do they differ from language to another? Since that most of the examples we learnt in this course pertains to OOP please explain how can you use them in non-OO languages such as c or haskell?