Name:_____ **Registration-Nb.:**_____

Task 1 Software Development Processes

1.1 List three reasons why to use a Software Development Process. [/ 3 A]

- Expectation: Good processes result in good products
- Better planning and controlling
- To achieve easy maintenance and extensibility of the resulting software product
- To know the status of development
- To obtain better collaboration
- To get a software product which fulfills the requirements
- To get a software product in the most effective and efficient way

1.2 List three main activities in Software Development Processes. [/ 3 B]

- Analysis
- Design
- Implementation
- Test
- Project Management
- Documentation
- Maintenance
- Learning experience

1.3 Object Oriented Software Development [/ 15]

a. What is the goal of Object Oriented Analysis? [/ 2 A]

The goal of Object Oriented Analysis is an easy understandable model with objects and relationships

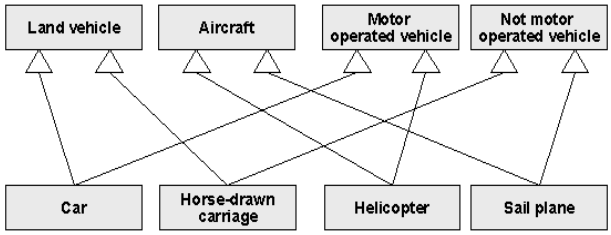
b. What is the goal of Object Oriented Design? [/ 2 B]

Design means to develop a solution for a given problem in consideration of given surrounding conditions. The goal of Object Oriented Design is to develop an object model of a system to implement the identified requirements

c. What are the main differences between an Object Oriented Analysis Model and an Object Oriented Design Model? [/ 2]

The Object Oriented Analysis model is the visualization of the requirement specification. The Object Oriented Design model is a blueprint of the system.

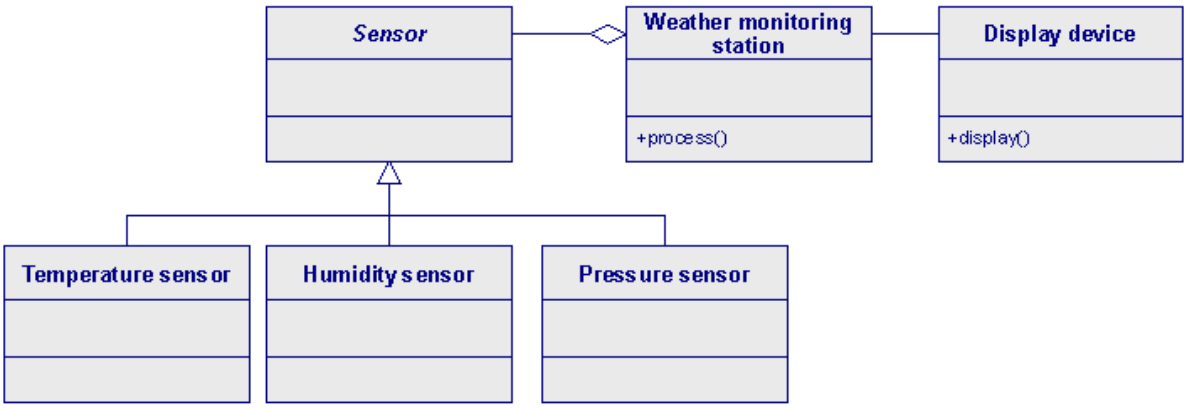
d. Following Object Oriented Analysis Model is given. Explain one possible reason or consideration, why the Object Oriented Design Model should look different. [/ 1]



Multiple inheritance should be resolved.
Compare with the design principle: Favour composition over inheritance.

e. Create an Object Oriented Analysis Model based on following description [/ 5]

A weather monitoring station contains a temperature sensor, an humidity sensor, and a pressure sensor. Additional sensors could be added in future.
The weather monitoring station processes the observed data from the sensors and displays the results on a display device.



f. What is a component in Object Oriented Design? [/ 2]

A component is part of a system or may serve as a part of a system. It is a reusable, logically cohesive, loosely coupled module that denotes a single abstraction.

g. List and explain three Object Oriented Design Principles [/ 3]

Extract of Class structure and relationships group

- Single Responsibility Principle (SRP)
  One class should have only one responsibility or cover only one functional unit
- Open/Closed Principle (OCP)
  Classes should be open to extension but closed to modification
- Liskov Substitution Principle (LSP)
  Subclasses should be able to substitute for their base classes
- Don't Repeat Yourself (DRY)
  Code should be written only once, duplication should be avoided
- Keep it simple, stupid (KISS)
  No including of needless abstraction levels / generalizations etc.

1.4 Agile Software Development [/ 6]

a. Describe 3 practices of Extreme Programming. [/ 3 **A**]

- Pair programming
  2 developers work together
- Testing – Test Driven Development
  Writing tests first before coding
- The Planning Game
  Release planning based on user stories, prioritized by customer – effort guessed by developers
- On-Site Customer
  Customer collaborates on the spot
- Continuous Integration
  If a user story is done, it gets integrated in the whole system
- Refactoring
  Continuous design improvement
- Small Releases
  to get early customer feedback
- Coding Standards
- Collective Code Ownership
- Simple Design
  Not needed code gets deleted immediately, implement only what is needed to fulfill an user story
- Metaphor
  Simple system story instead of a complex architecture description
- Sustainable Pace
  No overtime should not be done

b. What are the responsibilities and tasks of the participating roles during an iteration phase (“timebox”) in agile software development processes? [/ 3 **B**]

- Developers develop code and consult with customers as needed
- Customers define new requirements as desired
- Project manager plans, controls, and helps in process and meeting milestones

c. List the main differences between the Rational Unified Process and Agile Software Development Processes (Extreme Programming) [/ 3]

RUP

- Heavy process
- Document / Artifact concentrated
- Organization important
- Based on UML
- Roles
- Phases with Artifacts
- Specification

Agile software development

- Lean process
- User Stories / On-site Customer
- Organization minimized
- Based on User Stories / On-site Customer
- Collective Code Ownership
- Timeboxes
- User stories / storycards

Task 2 Requirements Analysis

2.1 Functional or non functional (required constraints) requirements? [/ 3]

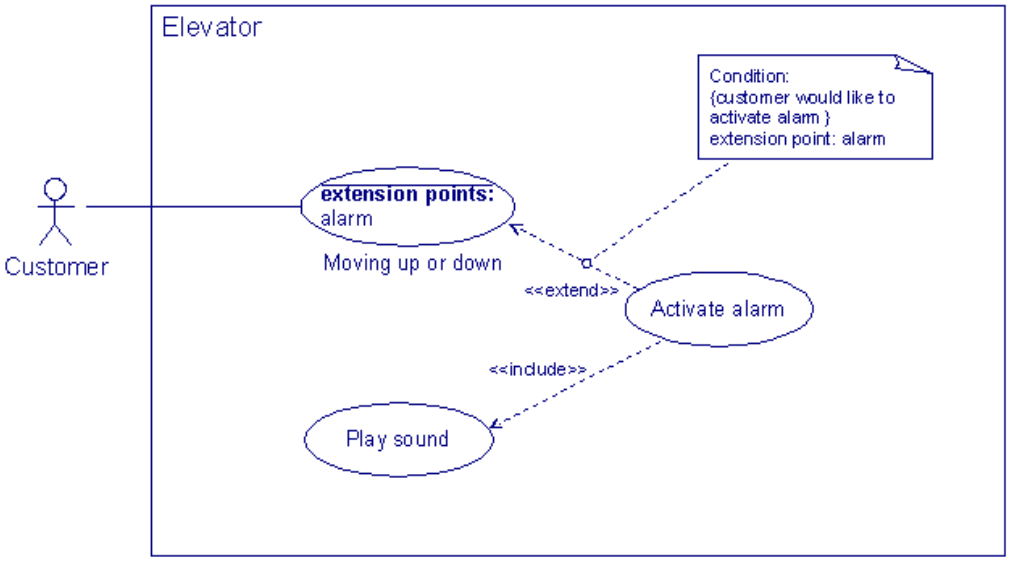
Requirement	Functional	Non functional
Mamegoma Mark 2 robot should clean sofa, bed, and bin	X	
The new cipher machine model should encrypt 1500 messages per hour		X
The Protoss Zealot is only able to attack unit on the ground	X	
The command “get ready” should be given to 1,000 soldiers in 5 minutes		X
The command “next” will change the state of the canvas object to the next color.	X	
We need to create a house in many styles with the steps - Create building framework - Create floor - Create wall - Create roof	X	

2.2 Use Case diagram [/ 3]

A

Create a Use Case diagram based on following description:

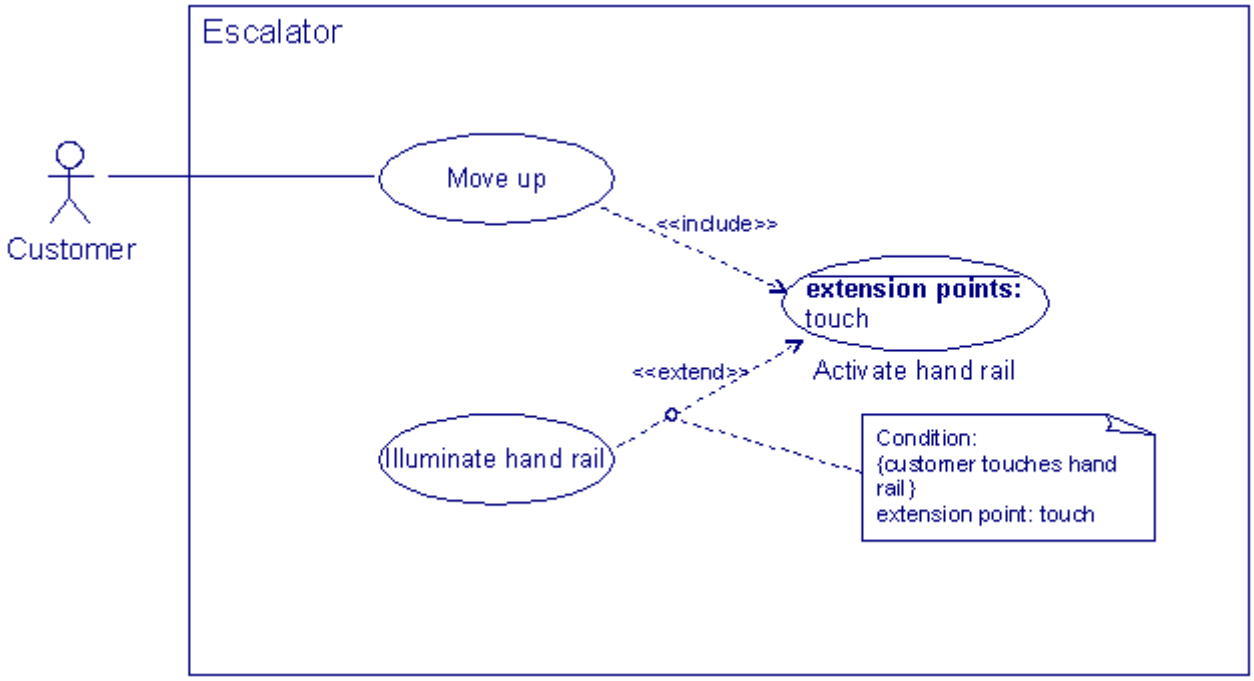
- A customer uses an elevator to move up or down
- If the customer is moving up or down, she optional could activate an alarm
- If the alarm is activated, the elevator sounds an alarm signal



B

Create a Use Case diagram based on following description:

- A customer uses an escalator to move up
- If the customer is moving up, the hand rail always gets activated
- If the customer touches the handrail, the handrail gets illuminated



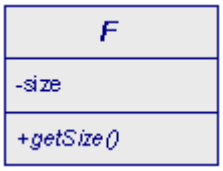
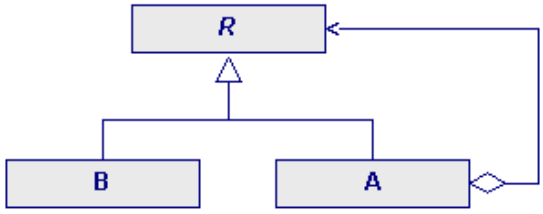
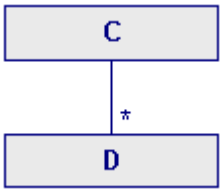
Task 3 UML diagrams

3.1 What does the abbreviation UML mean? [/ 1]

Unified Modelling Language

3.2 Class diagrams [/ 3]

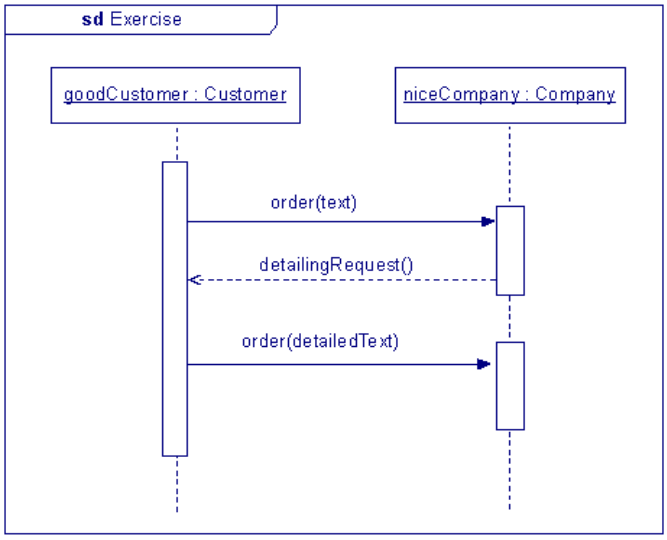
Please create class diagrams for following descriptions:

The abstract class `F` has a private attribute `size` and a public method `getsize()`	The classes `A` and `B` inherit from the abstract class `R`. `A` contains `R` as aggregate, so `A` has any `R`.	`C` is related to any number of `D`
		

3.3 Sequence diagram [/ 4]

Please create a sequence diagram representing the following facts:

> A customer `goodCustomer` sends to the company `niceCompany` an order containing the attribute `text`. The company `niceCompany` sends back a detailing request to the customer `goodCustomer`. The customer `goodCustomer` resends to the company `niceCompany` an order containing the attribute `detailedText`.



Task 4 Design Pattern – General

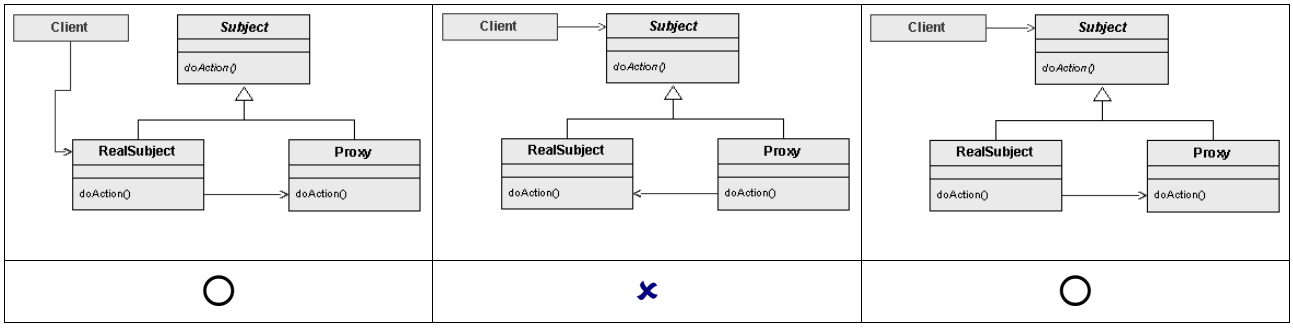
4.1 Characteristics of Design Pattern [/ 1]

Which of the following statements describe the characteristics of Design Pattern (more than one answer could be selected)?

1. Design Pattern offer a vocabulary for talking about design. ✘
2. Design Pattern identify key aspects of a common design structure. ✘
3. A Design Pattern is an implementation specific finished design ○
4. Design Pattern avoid reuse of design and code ○

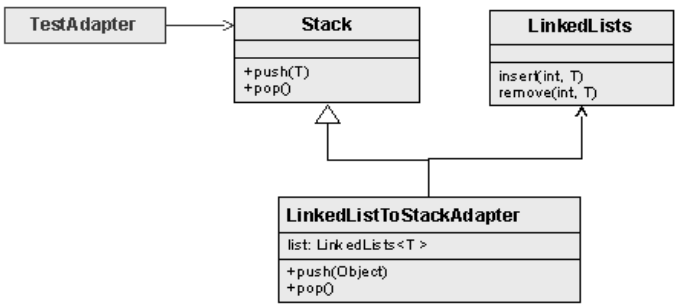
4.2 Structures of Design Pattern [/ 7]

a. Which diagram shows the Proxy Pattern? [/ 1]



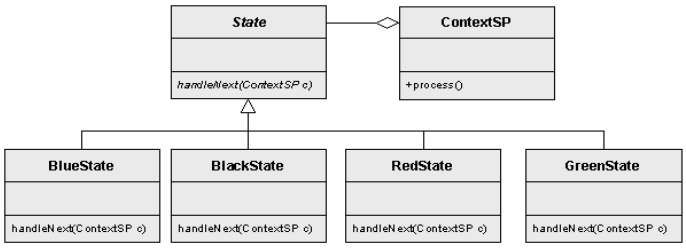
○ ✘ ○

b. Which special kind of which Design Pattern is described here? [/ 1]



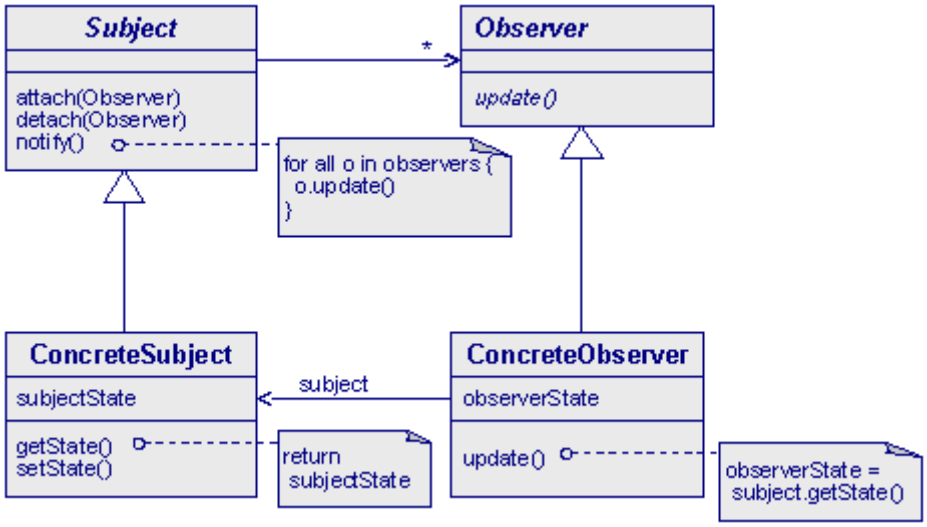
Object Adapter

c. Which Design Pattern is used here? [/ 1]

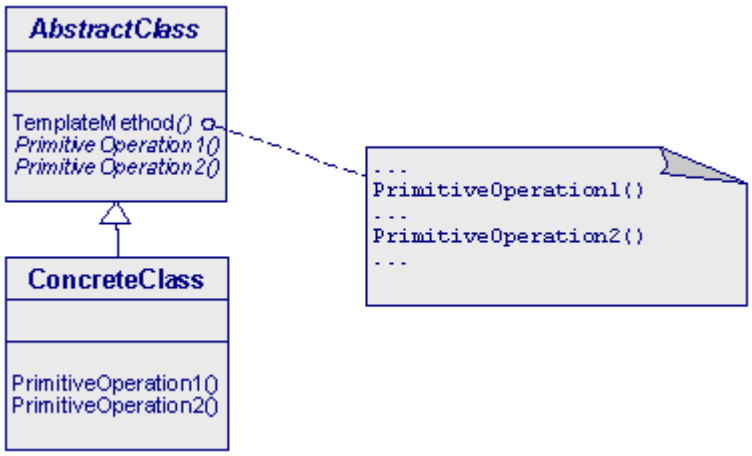


State Pattern

d. Design a UML diagram showing the structure of the Observer Pattern [/ 4 **A**]



e. Design a UML diagram showing the structure of the Template Method [/ 4 **B**]



4.3 Categories of Design Pattern [/ 4]

a. List 4 Structural Design Pattern [/ 2 **A**]
b. List 4 Behavorial Design Pattern [/ 2 **B**]

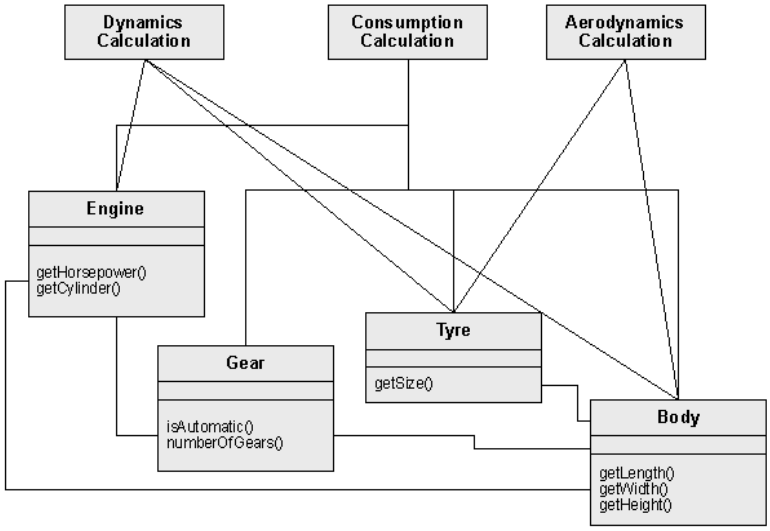
Structural	Behavorial	
Adapter	Chain of Responsibility	Observer
Bridge	Command	State
Composite	Interpreter	Strategy
Decorator	Iterator	Template Method
Façade	Mediator	Visitor
Flyweight	Memento	
Proxy		

c. Describe the intent of 1 Behavorial Design Pattern [/ 2 **A**]
d. Describe the intent of 1 Structural Design Pattern [/ 2 **B**]

see for example
http://en.wikipedia.org/wiki/Design_pattern_(computer_science)

Task 5 Façade Design Pattern

Following UML class diagram is given:



5.1 Design discussion [/ 2]

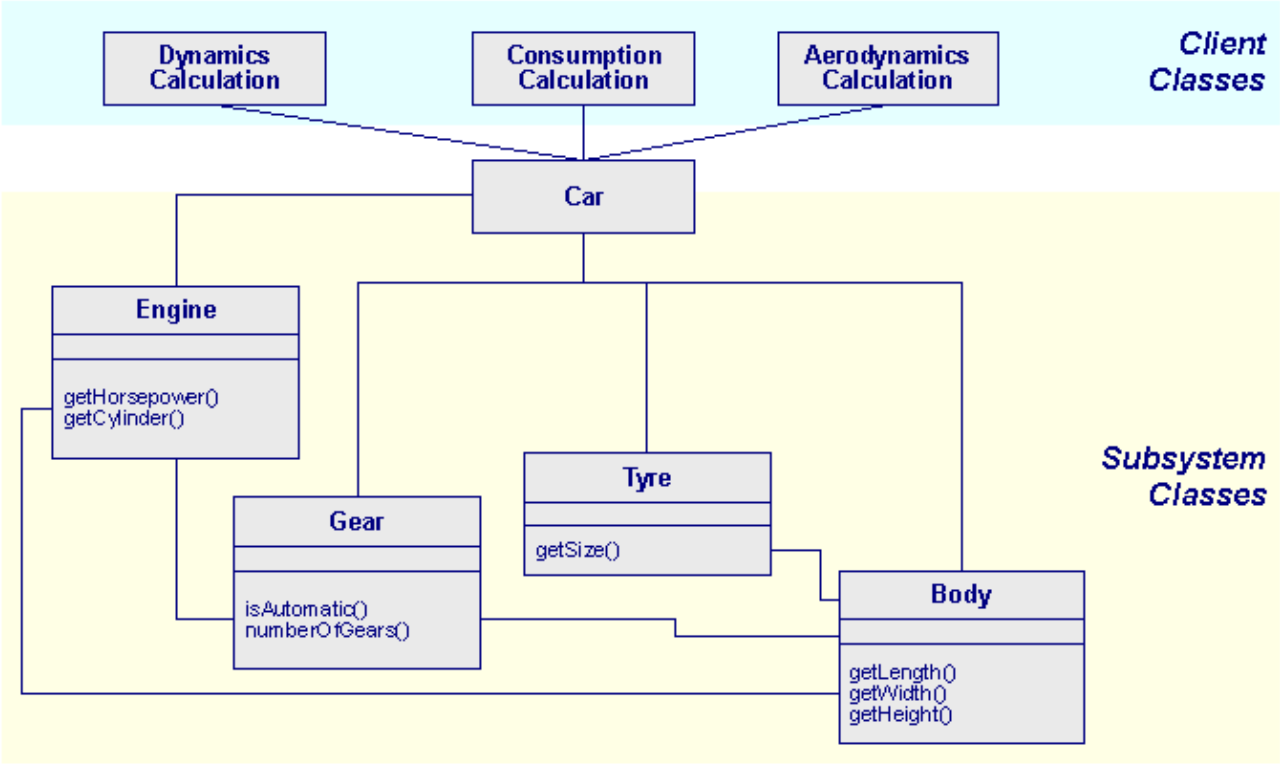
a. Why should you use the Façade Design Pattern here? [/ 1]

Many dependencies between clients and application classes.
If classes like Engine, Tyre, or Body get updated, client code has to be changed.
Portability is not easy possible.

b. If you use the Façade Design Pattern, what will be improved? [/ 1]

A simple interface makes communication to subsystem classes for the different calculation programs easier, as less objects and classes must be known
The calculation programs communicate with the subsystem by sending requests to the Façade, they don't access the subsystem / component objects directly any more. This reduces complex dependencies. Loose coupling allows independent development of the subsystem. Components of the subsystem could be changed without consequences on the client side.
Additionally lower compiling dependencies increase the portability.

5.2 Create a new UML class diagram using the Façade Design Pattern [/ 4]



5.3 Related Pattern [/ 2]

What is the difference between the Adapter pattern and the Façade pattern?

The adapter is changing an interface to one that a client can deal with. Without an adapter an access to the adapted class / classes would not be possible.
The façade encapsulates and offers a simplified access to a subsystem. A façade is not really necessary.

Task 6 Singleton Design Pattern

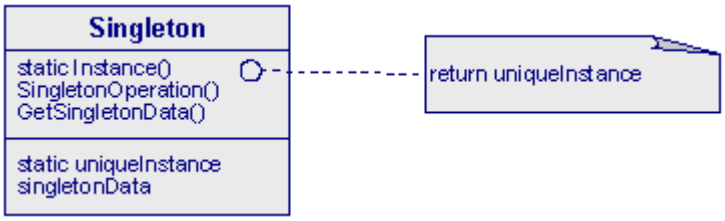
6.1 Describe the intent of the Singleton Design Pattern [/ 1]

A Singleton is the combination of two essential properties:

- Ensure a class only has one instance
- Provide a global point of access to it

6.2 Structure of the Singleton Pattern [/ 2]

Design and describe the structure of the Singleton Pattern in a UML class diagram



6.3 Which Design Pattern could be used with the Singleton Pattern? [/ 2]

Abstract factory – A concrete factory is often a Singleton. As it offers the possibility to create other objects it makes sense to have only one, e. g. to save memory.
Builder – To ensure that everyone is using the Builder properly.
Prototype – can use Singleton in its implementation
Adapter – If a shared resource is wrapped by an Adapter, an Adapter Singleton can ensure that this resource is used in the right way.
Façade – Façade objects are often Singletons because only one Façade object is required
State – State objects are often Singletons

6.4 Give an example, when the use of the Singleton Pattern is applicable [/ 1]

Wherever a device should be shared

- Print spooler
- File system
- Window manager
- Serial port
- Database connection

6.5 Singleton implementation [/ 2 A]

Attached you see a proposal for an implementation

```
public class Singleton {
   private final static Singleton instance = new Singleton();

   private Singleton() {}

   public static Singleton getInstance() {
     return instance;
   }
}
```

In another part of the code you see following line:

```
mySingleton = new Singleton();
```

a. What is your comment to this line? [/ 1]

Cannot compile, as the constructor Singleton() is private - not visible

b. How would you update this line? [/ 1]

```
Singleton mySingleton = Singleton.getInstance();
```

6.6 Singleton implementation [/ 2 B]

Attached you see a proposal for an implementation:

```
public class Singleton {
   private final static Singleton instance = new Singleton();

   public Singleton() {}

   public static Singleton getInstance() {
     return instance;
   }
}
```

a. What is your comment to this code? [/ 1]

As the constructor is public, many instances could be created. So the code does not guarantee, that only one instance is available.

b. What would you change in the code? [/ 1]

```
private Singleton() {}
```

Task 7 Abstract Factory Design Pattern

7.1 Describe the intent of the Abstract Factory Pattern [/ 2]

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

7.2 Implementation example [/ 20]

Following code sample is given:

```
public abstract class CarFactory {
    public static CarFactory getFactory(String model) {
        if (model == "BMVV") {
            return new BMVVFactory();
        } else if (model == "Morecedes") {
            return new MorecedesFactory();
        }
          else if (model == "OOOOFactory") {
            return new OOOOFactory();
        }
          else
            return null;
    }

    public abstract LuxuryClassCar createLuxuryClassCar();



}

public class MorecedesFactory extends CarFactory {
    public LuxuryClassCar createLuxuryClassCar() {
        return new SpecialClass();
    }
    public SmallClassCar createSmallClassCar() {
        return new AAClass();
    }
}

public class BMVVFactory extends CarFactory {
    public LuxuryClassCar createLuxuryClassCar() {
        return new SevenSeries();
    }
    public SmallClassCar createSmallClassCar() {
        return new OneSeries();
    }
}
```

```
public abstract class Car {
    public abstract void produce();
}

public abstract class LuxuryClassCar extends Car {
}

public class SevenSeries extends LuxuryClassCar {
    public void produce() {
        System.out.println("Producing a nice BMVV Seven Series Luxury Car!");
    }
}

public class SpecialClass extends LuxuryClassCar {
    public void produce() {
     System.out.println("Producing a nice Morecedes Special Class Luxury Car!");
    }
}

public class Client {
    public static void main(String[] args) {
        CarFactory factory = CarFactory.getFactory("BMVV");
        Car car = factory.createLuxuryClassCar();
        car.produce();
    }
}
```

a. Explain, which method in the code is a Factory Method. [/ 3]

For example the method
`public abstract LuxuryClassCar createLuxuryClassCar();`
is a **Factory Method** of the **Creator** `abstract class CarFactory`.
The **ConcreteCreator** `class BMVVFactory`
overrides the Factory Method and returns an instance of the **ConcreteProduct** `SevenSeries()`,
and the **ConcreteCreator** `class MorecedesFactory`
returns an instance of the **ConcreteProduct** `SpecialClass()`.

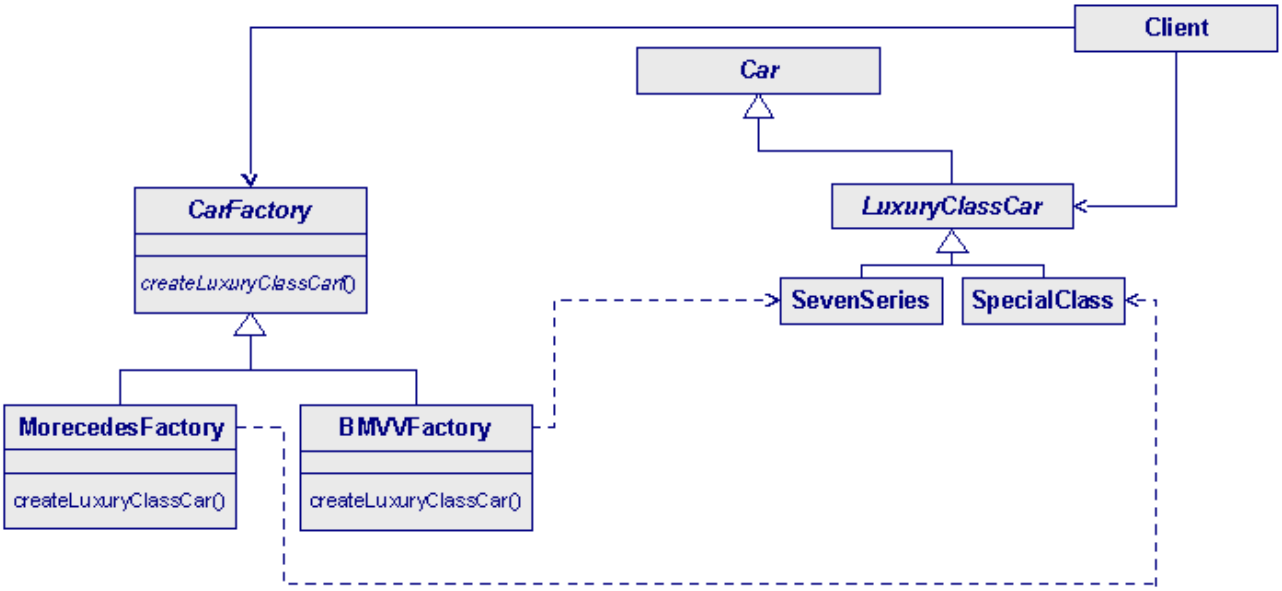
b. What will be the result of this code, if you compile it? [/ 2]

`"Producing a nice BMVV Seven Series Luxury Car!"`

c. What do you have to change in the code if you would like to get the output `"Producing a nice Morecedes Special Class Luxury Car!"`? [/ 2]

Update in `public class Client`
the line `CarFactory factory = CarFactory.getFactory("BMVV");`
to `CarFactory factory = CarFactory.getFactory("Morecedes");`

d. Design for the given code sample the corresponding UML class diagram [/ 4]



e. Which class is taking which role of the Abstract Factory Design Pattern? [/ 3]

Abstract Factory Pattern	Code sample
Abstract Factory	CarFactory
ConcreteFactory1	MorecedesFactory
ConcreteFactory2	BMVVFactory
AbstractProductA	LuxuryClassCar
ProductA1	SpecialClass
ProductA2	SevenSeries

f. Add another CarFactory "OOOOFactory", which could produce a luxury class car named "OOOOEight".

Write the corresponding code to be added into the code sample on pages **13 and 14**, add the necessary code for new classes here: [/ 6 **A**]

```
public class OOOOFactory extends CarFactory {
    public LuxuryClassCar createLuxuryClassCar() {
        return new OOOOEight();
    }
}

public class OOOOEight extends LuxuryClassCar {
    public void produce() {
        System.out.println("Producing a nice OOOO Eight Luxury Car!");
    }
}
```

g. Add additional car classes:

- For the "BMVVFactory" the small class car "OneSeries"
- For the "MorecedesFactory" the small class car "AAClass"

Write the corresponding code to be added into the code sample on pages **13 and 14**, add the necessary code for new classes here: [/ 6 **B**]

```
public abstract class SmallClassCar extends Car {
}

public class OneSeries extends SmallClassCar {
    public void produce() {
       System.out.println("Producing a nice BMVV One Series Small Car!");
    }
}

public class AAClass extends SmallClassCar {
    public void produce() {
    System.out.println("Producing a nice Morecedes AA Class Small Car!");
    }
}
```

7.3 Consequences of the Abstract Factory Design Pattern [/ 2]

Describe the consequences of the Abstract Factory Design Pattern.
What are the benefits and what are the disadvantages?

+ Concrete classes are isolated
  The names of the product classes do not appear in client code
+ Exchanging of product families easy
  The name of the concrete factory appears only once in the application – where it is instantiated
+ Consistency of products gets supported
  The creation of products with the factory avoids, that a client creates products from different families at the same time by accident
– Support of new products is complex
  The interface of the abstract factory has to be adapted.
  The abstract factory fixes the set of products which could be generated