ISS exam Monday, June 11, 2012 solutions

1. **Please explain why modeling supports using abstraction on a wider scale compared with simply programming in software development. Also please make clear which are the advantages obtained by supporting a higher abstraction level and which is the price to pay. 1.5 p**

2. Modeling uses different views specified by means of different modeling languages. Functional, architectural, behavioral, component and distribution views are the views supported in UML. Therefore, almost all modeling specialist agree that UML is a family of modeling languages. Each language has a restricted number of concepts used for specifying the associated view (model). Managing a simpler language is easier - this is the main advantage obtained by a higher abstraction. The price to pay is due to the fact effort needed for checking the consistency and non-contradiction of the information specified in different views.

3. **Please describe shortly which are in your opinion the main features of the Unified Modeling Language. In this context, please describe shortly the concepts of concrete and abstract syntax in UML. Which is the relationship between UML and OCL. 1.5p**

4. UML is a general modeling language supporting model specification irrespective of the problem domain and the technology used. UML is a semiformal language characterized by a graphical concrete syntax, an abstract syntax described by means of a metamodel and a semantics described by means of a natural language. UML is a MOF based language because the abstract syntax of the language is specified by using the MOF language. Initially, OCL was conceived as an UML complementary language. After, OCL was adapted as a MOF based complementary language. OCL is a textual language meant to be used both for querying models and for specifying different kind of constraints required for an unambiguous model specification. In order to support model simulation, UML has an action language enabling to create and update model instances. Even if UML has an important number of concepts, the language need to be extended. Using appropriate extension mechanisms (stereotypes, constraints and tagged values) supports a lightweight metamodel extension. Each extension information is grouped in profiles meant to improve the efficiency of using the language in specifying different kind of applications.
5. **A passengers airways company offers flights toward different destinations. Each departure and destination have an associated airport characterized by a name and a unique ID. Flights are planned for a date, a departure time and an arrival time (estimated). Each flight has a**

maximum number of passengers. Passengers of a flight are transported by a single plane. Each plane has a capacity of transport. The departure and arrival of each flight have to be different, and the arrival time has to be at least half an hour after the departure time. Also, the maximum number of passengers of each plane have to be less or equal to the capacity of the associated plane. By means of a UML class diagram, please describe the structure of a model complying with the above mentioned requirements. Using the OCL language, please specify the above mentioned constraints. 3p

In the diagram represented in Figure 1, the AirwaysCompany was not represented for simplicity reasons.

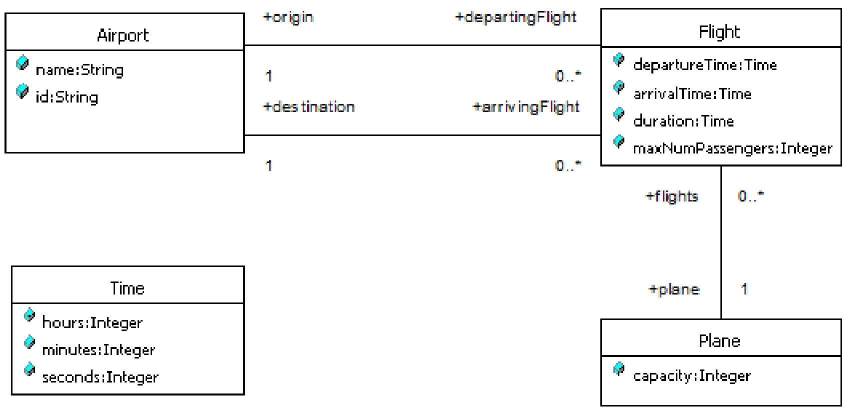


Figure 1 - An architecture complying with the requirements

```
context Flight
    inv originAndDestinationAreDifferent:
        let aOrigin = self.origin in
        aOrigin <> self.destination

    inv originAndDestinationAreDifferent_:
    self.origin.id <> self.destination.id

    inv flightCapacity:
        self.maxNumPassengers <= self.plane.capacity

    inv flightTime:

        let durationInHours=self.arrivalTime.hours - self.departureTime.hours
    let difMin: Integer=self.arrivalTime.minutes - self.departureTime.minutes

        in

        durationInHours*60 +difMin <= 30
```

6. **Please describe shortly the concept of Design by Contract. 1p**

Design by contract, also known as **programming by contract** and is an approach for designing software. It prescribes that software designers should define formal, precise and verifiable interface specifications for software components, which extend the ordinary definition of abstract data types with preconditions, postconditions and invariants. These specifications are referred to as "contracts", in accordance with a conceptual metaphor with the conditions and obligations of business contracts.

7. **For the model below, please construct two snapshots, one complying with the model, and the other not. In order to construct easier the snapshot you may represent firstly the equivalent construction. 2p**

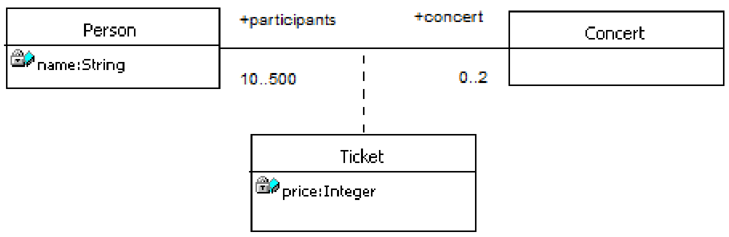


Figure 2 Using associatedClasses in UML

As you can see, figure 3 represents the class diagram associated to the equivalent design.

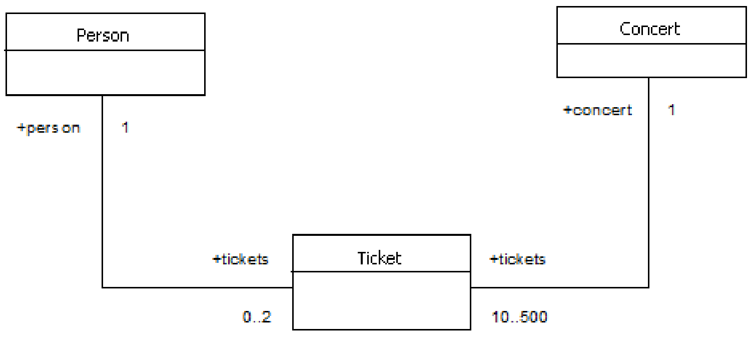


Figure 3 - Equivalent Design for the assocClass

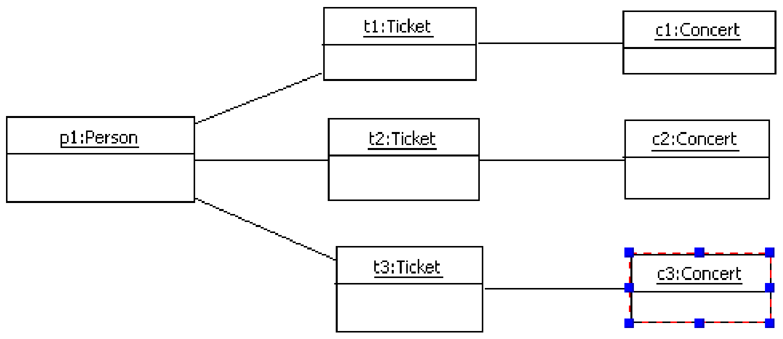


Figure 4 - Wrong snapshot

The snapshot represented in Figure 4 does not comply with the architecture because: as seen in the equivalent design, a person may buy at most 2 tickets (and in the above snapshot p1 bought 3 tickets), each concert have to sell at least 10 tickets. In our snapshot, each concert sell only one ticket.

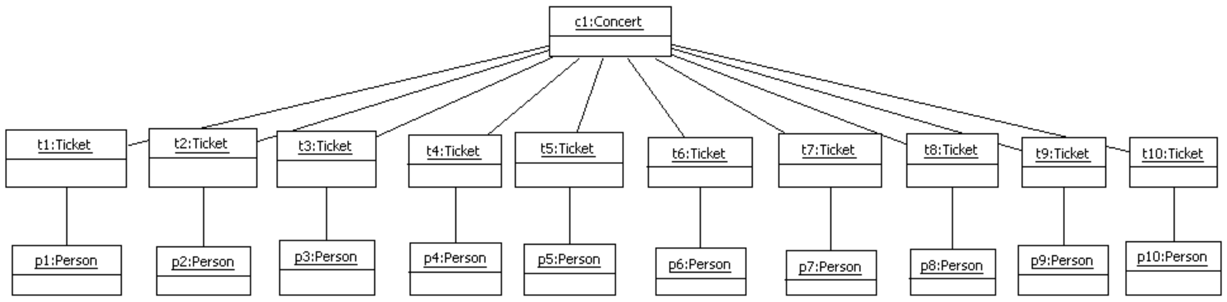


Figure 5 - Complying snapshot

The snapshot presented in Figure 5 comply with the architecture represented in Figure 3 because both the concert c1 and the persons p1, ..., p10 have associated a correct number of tickets.

1. **Please describe the concepts of coupling and coherence of subsystems. 2p**

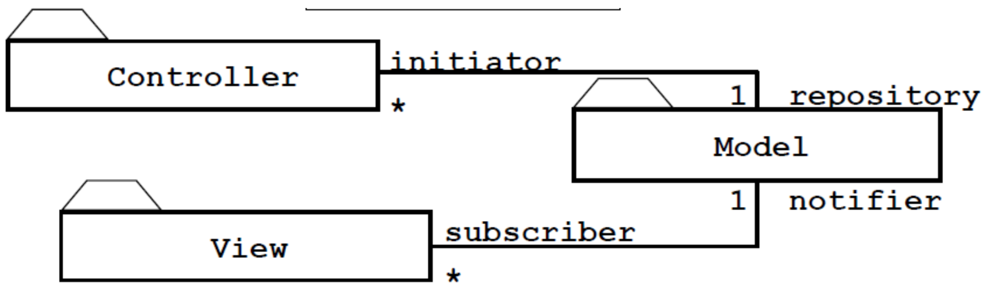
Layered systems are hierarchical. This is a desirable design, because hierarchy reduces complexity - low coupling. **Coupling and Coherence of Subsystems**

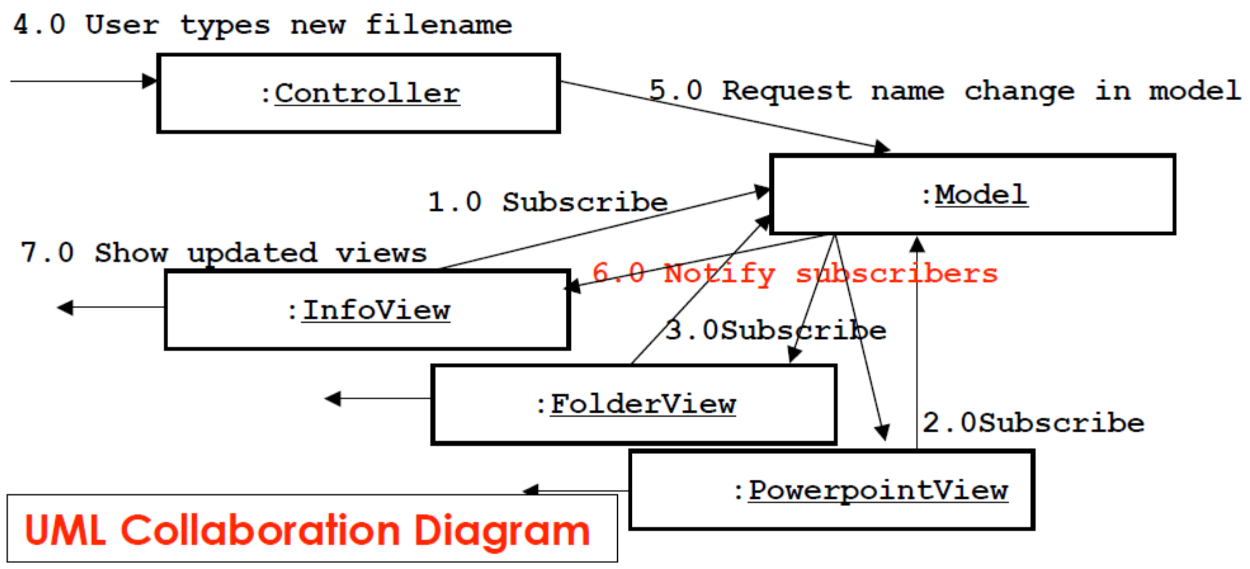
- **Goal**: Reduce system complexity while allowing change
- **Coherence** measures dependency among classes of the same subsystem:
- **High coherence**: The classes in the subsystem perform similar tasks and are related to each other via many associations**. High coherence** can be achieved if most of the interaction is within subsystems, rather than across subsystem boundaries.
    - **Low coherence**: Lots of miscellaneous and auxiliary classes, almost no associations
- **Coupling** measures dependency among subsystems:
    - **High coupling**: Changes to one subsystem will have high impact on the other subsystem
    - **Low coupling**: A change in one subsystem does not affect any other subsystem. **Low coupling** can be achieved if a calling class does not need to know anything about the internals of the called class (Principle of information hiding, Parnas)

2. **Please characterize shortly the Model View Controller Architectural Style. The corresponding class diagram is required. 2p**

Problem: Assume a system with high coupling. Then changes to the boundary objects (user interface) often force changes to the entity objects (data). The user interface cannot be re-implemented without changing the representation of the entity objects. The entity objects cannot be reorganized without changing the user interface.
Solution: The model-view-controller architectural style, which decouples data access (entity objects) and data presentation (boundary objects):

- The **Data Presentation subsystem** also called the **View subsystem** is responsible for displaying application domain objects to the user.
- The **Data Access** subsystem is called the **Model.** The **Model subsystem** is responsible for the application domain knowledge
- The **Controller** is a new subsystem that mediates between **View (data presentation)** and **Model (data access)**. The **Controller subsystem** is responsible for sequence of interactions with the user and notifying views of changes in the model.





3. Please describe the Java code corresponding to the following class diagram. 2p

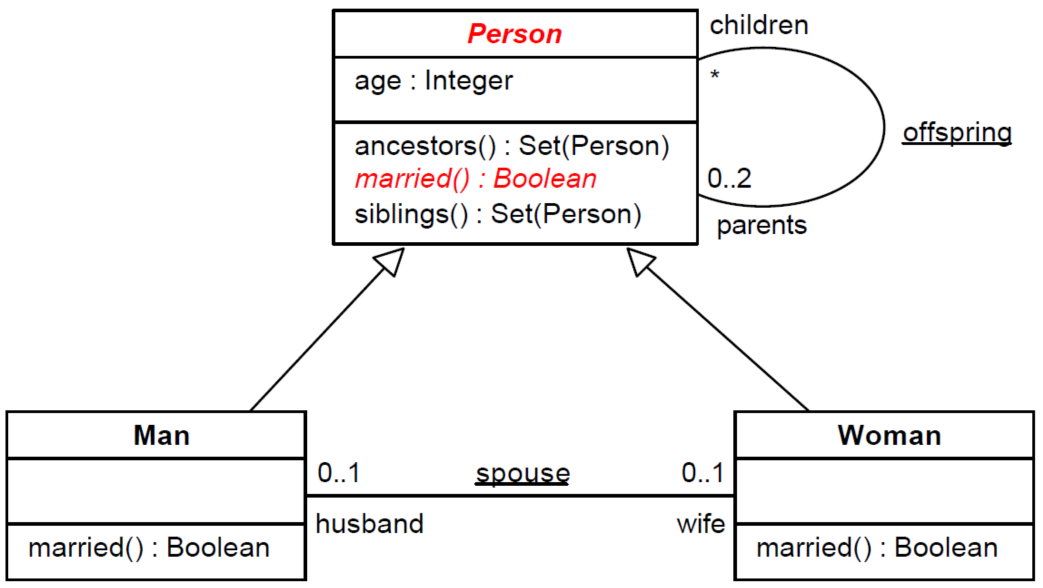


Figure 6 A family model

The main target was to see how the code corresponding to bidirectional associations is managed by means of add and remove methods and not to focuses on dataTypes used for collections.

```
private abstract class Person {


    public abstract boolean married();
    public final Set getParents() {
        if (parents == null) {
            return java.util.Collections.EMPTY_SET;
        }
        return java.util.Collections.unmodifiableSet(parents);
    }

    public final void addParents(Person arg) {

        if (arg != null) {
            if (parents == null) parents = new LinkedHashSet();
            if (parents.add(arg)) {
                arg.addChildren(this);
            }
        }

    }

    public final void removeParents(Person arg) {

        if (parents != null && arg != null) {
            if (parents.remove(arg)) {
                arg.removeChildren(this);
            }
        }

    }

    public final Set getChildren() {

        if (children == null) {
            return java.util.Collections.EMPTY_SET;
        }
        return java.util.Collections.unmodifiableSet(children);
    }

    public final void addChildren(Person arg) {

        if (arg != null) {
            if (children == null) children = new LinkedHashSet();
            if (children.add(arg)) {
                arg.addParents(this);
            }
        }

    }

    public final void removeChildren(Person arg) {

        if (children != null && arg != null) {
            if (children.remove(arg)) {
                arg.removeParents(this);
            }
        }

    }

    public Person() {
    }

    public Set parents;

    public Set children;

    protected int age;

}


public class Man extends Person {


    public boolean married() {

        return false;

    }

    public final Woman getWife() {

        return wife;
    }

    public final void setWife(Woman arg) {

        if (wife != arg) {
            Woman temp = wife;
            wife = null;//to avoid infinite recursion
            if (temp != null) {
                temp.setHusband(null);
            }
            if (arg != null) {
                wife = arg;
                arg.setHusband(this);
            }
        }

    }

    public Man() {

    }

    public Woman wife;

}
```

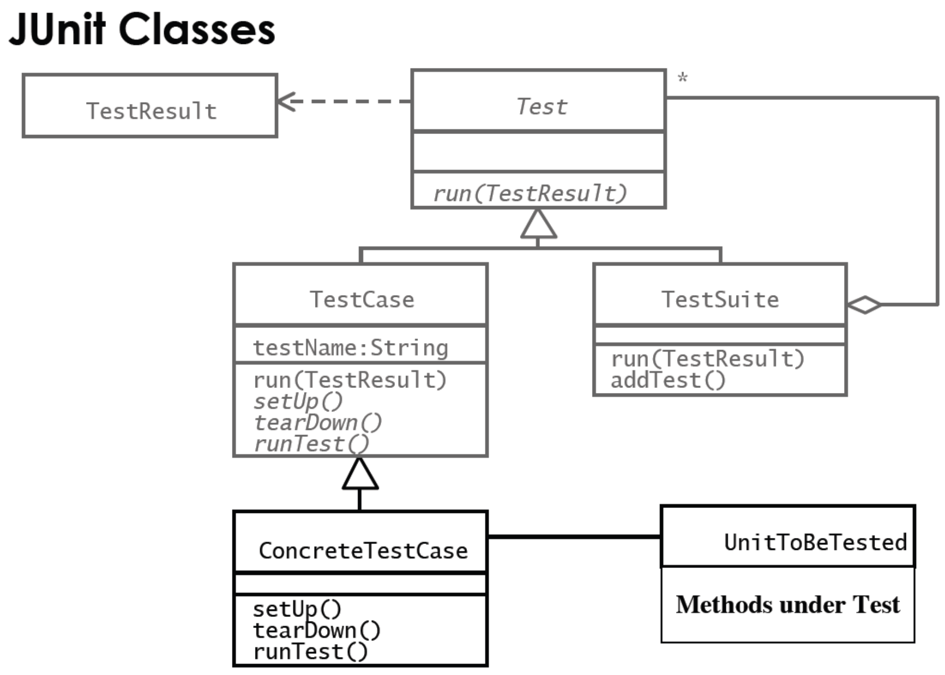
4. **Please describe the concept of unit testing and a class diagram to corresponding to the Junit testing framework conceived by Kent Beck and Erich Gamma. 2p**

Unit Testing concerns individual components (classes or subsystems), is carried out by developers having the goal to confirm that components or subsystems are correctly coded and carries out the intended functionality. **Unit Testing** is structured in: **Static Testing** done at compile time, and **Dynamic Testing** done at run time. **Static Testing (Analysis)** includes a **review**, an informal **Walk-through** and a formal **Walk-through**. **Dynamic Testing** may be a **Black-box testing** or a **White-box testing.**

In Static testing, **Compiler Warnings** and **Errors** enable to identify: **possibly uninitialized variable**, **undocumented empty blocks**, and **assignments having no effect**. Metrics support finding structural anomalies. There are **specialized applications for Static testing** like **Checkstyle** and **FindBugs**.

Black-box testing is focused on the I/O behavior. If for any given input, we can predict the output, then the component passes the test. That's why test oracle are required. Due to the large amount of date, an important challenge is to reduce the number of test cases. This can be realize by equivalence partitioning: input conditions are divided into equivalence classes. Test cases are choosed for each equivalence class.

In **White-box testing**, test cases are derived from the internal structure of the tested unit (e.g. source code).
JUnit is a framework for writing and automating the execution of unit tests for Java classes. Developers write new test cases by subclassing the **TestCase** class. The **setUp()** and **tearDown()** methods of the concrete test case initialize and clean up the testing environment. The **runTest()** method includes the actual test code that exercise the class under test and compare the results with an expected



5. **Please explain what do you mean by Software Project Management 1p**

Software project management concerns activities realized in the context of a software project - activities to develop a software system within a given time frame and with limited resources. These activities are specified in a **Software Project Management Plan (SPMP)**. The documents included in the (SPMP) specify the technical and managerial approaches to develop the software product.