Solutions - SE exam 22-06-2013

I. Please describe shortly the concepts of model and view. Enumerate the models and the views that you know. Please note that it is about the concepts and not about the MVC pattern. 1.5p

A **model** is an abstract representation of a system that enables us to answer questions about the system. Models are useful when dealing with systems that are too large, too small, too complicated, or too expensive to experience firsthand. Models also allow us to visualize and understand systems that either no longer exist or that are only claimed to exist.

Unfortunately, even a model may become so complex that it is not easily understandable. We can continue to use the divide-and-conquer method to refine a complex model into simpler models. A **view** focuses on a subset of a model to make it understandable

Models are realized in different life cycle phases of a system development. Corresponding to these phases we may have: analysis models, design models, implementation models. Analysis and design models may include: functional views, structural views, behavioral views, component views, deployment views a.s.o.

Another correct answer about view examples can be:

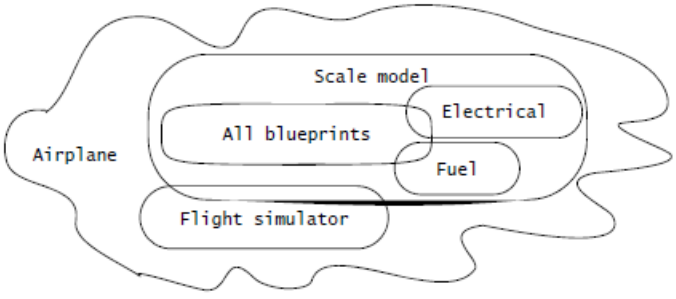
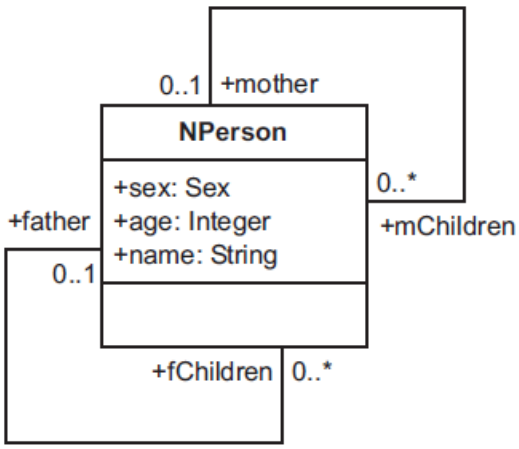


Figure 2-6 A model is an abstraction describing a subset of a system. A view depicts selected aspects of a model. Views and models of a single system may overlap each other.

II. A person has 2 parents: a mother and a father and is characterized by sex, age and name. Using the UML please represent the architecture of a model that supports the management of the above information. Please notice that in some cases, one parent or both may be unknown. 1.5 pt

Using the OCL please specify an invariant requiring that the sex of the mother has to be female and the sex of father, male. Also, please specify another invariant requiring that each parent is at least 16 years older than his children. 1.5 pt



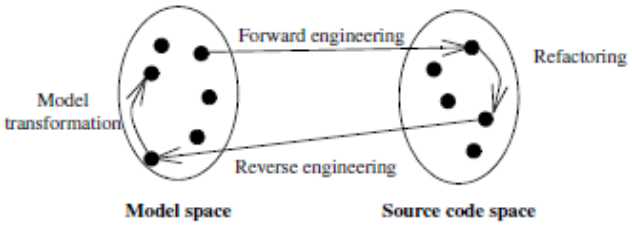

```
context NPerson
   inv parentsSexP2 :
      (self.mother ->size() = 1 implies
      (let ms:Sex = self.mother.sex in
      if not ms. oclIsUndefined ()
         then ms = Sex::Female
         else false endif))
      and
      (self.father ->size() = 1 implies
      (let fs:Sex = self.father.sex in
      if not fs.oclIsUndefined()
         then fs = Sex::Male
         else false endif))
```

With respect to the second constraint, we propose the following specification:

```
context NPerson
   inv parentsAge:
     self.mChildren ->reject(p | self.age - p.age >= 16) - >isEmpty() and
     self.fChildren ->reject(p | self.age - p.age >= 16) - > isEmpty ()
```

III. Please describe shortly which is the role of design patterns and mention the name of the activity by means of which design patterns are used to improve the model. What's the name of the similar activity realized in the code? 1 pt

A **design pattern** is a repeatable solution to a software engineering problem. Unlike most program-specific solutions, design patterns are used in many programs. Design patterns are not considered finished product; rather, they are templates that can be applied to multiple situations and can be improved over time, making a very robust software engineering tool. Because development speed is increased when using a proven prototype, developers using design pattern templates can improve coding efficiency and final product readability. Design patterns support constructing open architectures enabling an efficient management of changes in software development.



Te activity of improving models by using design patterns is named model transformation. At the code level, the similar activity is named refactoring.

IV. Which is the motivation of using qualified associations in UML. Using the UML please describe in a class diagram, both by using or not a qualified association, the relationship that exists between a directory and the contained files. In our case, the qualified association enable getting a file by means of its name. Please specify the Java code that can be automatically generated in the class Directory for the method addFile(String fileName, File arg). (1.5+1) = 2.5 pt

Qualification is a technique for reducing multiplicity by using keys. Associations with a 0..1 or 1 multiplicity are easier to understand than associations with a 0..n or 1..n multiplicity. Often in the case of a one-to-many association, objects on the "many" side can be distinguished from one another using a name. For example, in a hierarchical file system, each file belongs to exactly one directory. Each file is uniquely identified by a name in the context of a directory. Many files can have the same name in the context of the file system; however, two files cannot share the same name within the same directory.

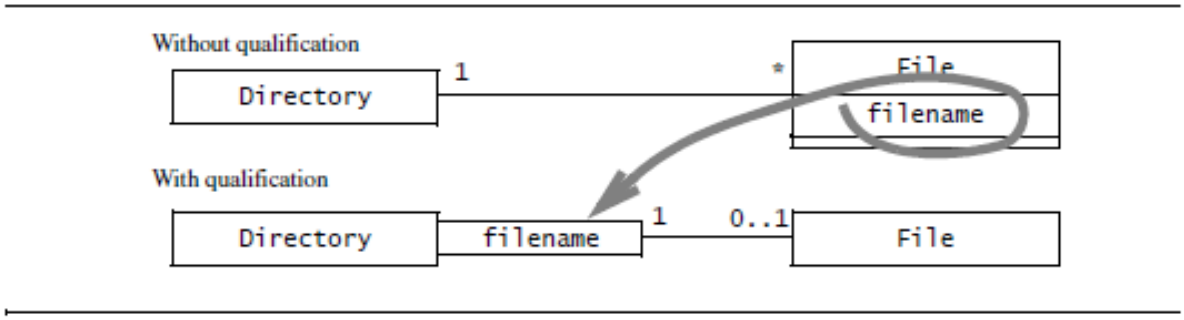


Figure 2-31 Example of how a qualified association reduces multiplicity (UML class diagram). Adding a qualifier clarifies the class diagram and increases the conveyed information. In this case, the model including the qualification denotes that the name of a file is unique within a directory.

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.LinkedHashSet;
import java.util.Map;
import java.util.Set;

public class Directory {
...
public final void addFile(String fileName, File arg) {

        if (arg != null) {
            ArrayList key = new ArrayList();
            key.add(fileName);

            if (file == null) file = new HashMap();
            File temp = (File)file.put(key, arg);//the previous value,
if any
            if (temp != arg) {
                arg.setDirectory(this);
                if (temp != null) {
                    temp.setDirectory(null);
                }
            }
        }

    }
```

```
...
  public Map file;
}
```

V. Please describe testing in a concise manner. Which is the goal of testing? 1 pt

Testing is a fault detection technique that tries to create failures or erroneous states in a planned way. This allows the developer to detect failures in the system before it is released to the customer. Note that this definition of testing implies that a successful test is a test that identifies faults. We will use this definition throughout the development phases. Another often-used definition of testing is that "it demonstrates that faults are not present." We will use this definition only after the development of the system when we try to demonstrate that the delivered system fulfills the functional and nonfunctional requirements.

If we used this second definition all the time, we would tend to select test data that have a low probability of causing the program to fail. If, on the other hand, the goal is to demonstrate that a program has faults, we tend to look for test data with a higher probability of finding faults. The characteristic of a good test model is that it contains test cases that identify faults. Tests should include a broad range of input values, including invalid inputs and boundary cases, otherwise, faults may not be detected. Unfortunately, such an approach requires extremely lengthy testing times for even small systems.