Dipartimento di Elettronica e Informazione
Politecnico di Milano
prof. Carlo Ghezzi -prof. Elisabetta Di Nitto
20133 Milano (Italia)
Piazza Leonardo da Vinci, 32
Tel. (39) 02-2399.3400
Fax (39) 02-2399.3411

Software Engineering II
Part I

27 January 2009

Last name
First name
Identification number (Matricola)
Section (specify the professor you are associated with, either Prof. Ghezzi or Prof. Di Nitto)

Notes

1. Missing identification data invalidate the exam.
2. Return **only** these pages. Extra sheets will be ignored. You may use a pencil.
3. The exam is in 2 parts. For part 1 you will not be allowed to use books or class notes. For part 2 you will be allowed to use books and class notes. The first part must be in 30 min., the second part in 55 min. The final result is the sum of the scores obtained in both parts.
4. Any use of electronic devices (computers, calculators, cell phones, …) is forbidden.
5. You cannot keep a copy of the exam when you leave the room.

Question 1 (2.5 points)

1. What is software maintenance?

2. How do maintenance costs compare to development costs of an application?

3. Which specific kinds of software maintenance do you know?

SOLUTION

1. Any post-delivery software development activity on a given application.
2. It is known that maintenance costs normally exceed development costs. They may be up 80% of total costs.
3. Corrective maintenance**.** Adaptive maintenance**.** Perfective maintenance

Question 2 (2.5 points)

Briefly list the definitions we gave in class for

- error (or defect)
- fault
- failure

Provide also simple examples for each concept.

SOLUTION

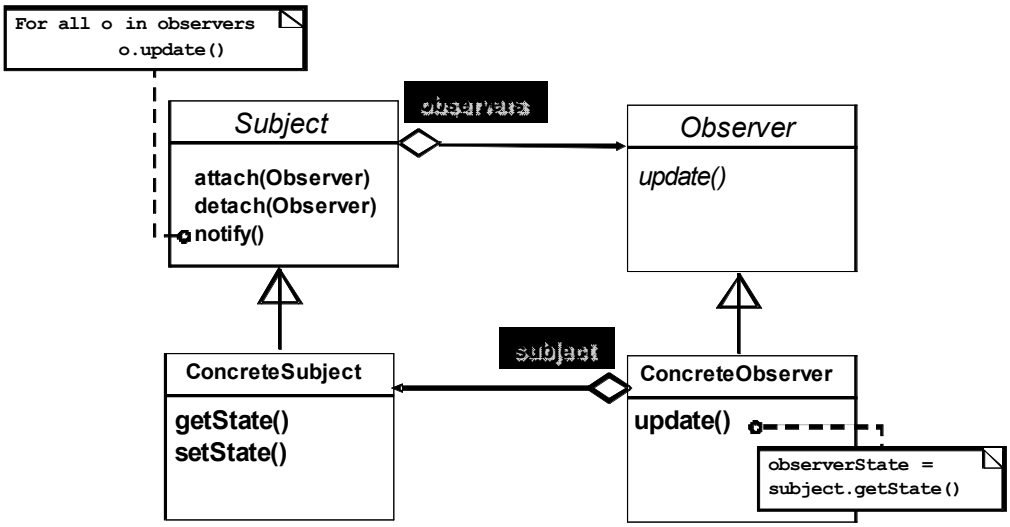
1. Error is any mistake in the code that causes a failure. Examples are a division by 0, the omission of some test on the value of some variable, etc.
2. Fault is an incorrect intermediate state entered by a program. An example is the case when a program enters in a commit state before all updates are actually performed on its persistent data.
3. Failure is an incorrect output that has been produced by a program for a certain input set of values. Incorrect in this case means different than what expected. An example is the case when after a withdraw operation of 100 Euros the program reports no change in the account balance.

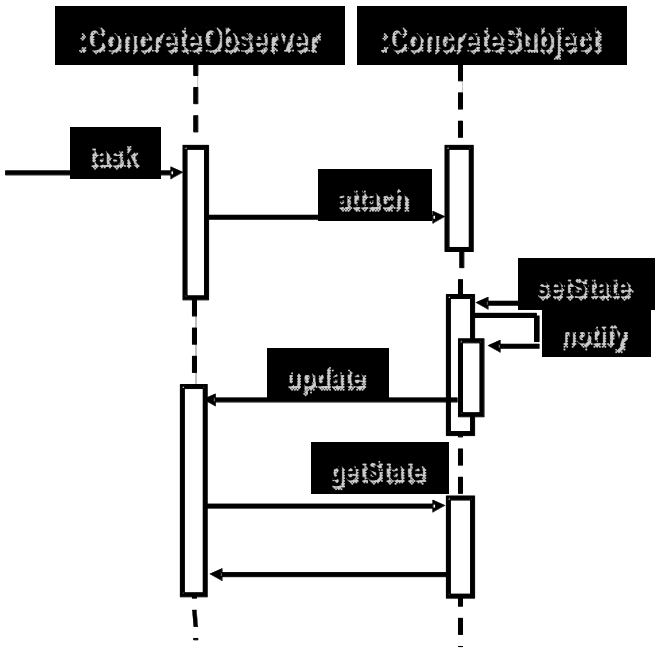
Question 3 (2.5 points)

Briefly describe the purpose of the **observer** pattern and the roles involved. Please provide UML diagrams to describe it.

SOLUTION

The observer pattern can be used in all cases when we have some objects (the observers) that need to perform some action in response to some change of state in some other object (the subject). The pattern defines a proper structure for the involved classes that allow observers to be attached/detached from the subject at runtime in a fully dynamic way. The following diagrams describe the structure and the dynamics of the pattern





Question 4 (2.5 points)
Briefly describe a publish-subscribe software architecture. Which is the main advantage you see in it? Briefly illustrate one relevant property that a publish-subscribe middleware may not guarantee to an application built according to the style.

SOLUTION
In a publish-subscribe software architecture three main roles are defined: the subscribers are those who declare their interest to receiving some events. The publishers are those who publish events. The dispatcher acts as an intermediary between the two making sure that published events reach those subscribers that are interested to them. Each component of the system can take any of the three roles. Usually, the role of the dispatcher is taken by a middleware component while the other components can act both as subscribers and publishers.
The main advantage of the publish-subscribe architecture is the high level of decoupling between the publishers of a certain event and its subscribers. None of them, in fact, has to know neither the identity nor the location of the counterpart. This allows for a high adaptability of the architecture, where components can enter and leave without impacting on the other components.
As a drawback, the high level of decoupling results in the fact that guaranteeing that events are actually delivered in the same order in which they are produced is very difficult to achieve. Usually, middleware offer some less stringent guarantee such as the fact that events are received according to their causal order or that events produced by the same publisher are received in the same relative order in which they have been generated.

Last name

First name

Identification number (Matricola)

Section (specify the professor you are associated with, either Prof. Ghezzi or Prof. Di Nitto)

Part II

Question 1 Alloy (7 points)

Specify in Alloy a world of countries, persons, and treaties, for which the following properties hold. Each *Person* belongs to one *Country* as a citizen and to one (possibly different) *Country* as a resident. A *Treaty* is subscribed by a set of at least two countries and defines the possibility for a citizen or a resident of a country to visit any other country that participates in the treaty. A person can be a resident of a country of which he or she is not a citizen only if that country participates in a treaty with the country he or she is a citizen of.

Specify an operation *move* for a person *p* and a country *c*, which means that *p* moves from his/her residence country (call it *c1*) to *c*. The precondition is that *c* must participate in a treaty with *c1* and must also participate in a treaty with *p*'s country of citizenship.

SOLUTION

```
sig Country {}
sig Treaty {
        members: set Country
}{
        #members>1
}

sig Person {
        residency : one Country,
        citizenship : one Country
}
{
        citizenship=residency ||
        some t: Treaty |
                residency in t.members && citizenship in t.members
}

pred move [c1, c: Country, p, p': Person] {
//Precondition
        c1 !=c
        some t : Treaty | c1 in t.members && c in t.members
        some t' : Treaty | p.citizenship in t'.members && c in t'.members

//Postcondition
        p'.citizenship = p.citizenship
        p'.residency = c
}

run move
```

Question 2 Testing (4 points)

Consider the following simple function.
1. Derive the path condition for the following paths:

- you start with x and y positive, you iterate the loop twice, the first time executing the *then* branch and the second time executing the *else* branch
- you start with x and y negative and you don't loop.

2. For each of the path conditions you derive, say whether it is satisfiable and derive a test case that satisfies it

```
int gcd (int x, y) {
        if (x<0) x= -x;
        if (y<0) y= -y;
        while (x != y) {
                if (x < y)
                        y= y-x;
                else
                        x= x-y;
        return x;
}
```

SOLUTION
We number the instructions of the program

```
1 int gcd (int x, y) {
2       if (x<0) x= -x;
3       if (y<0) y= -y;
4       while (x != y) {
5               if (x < y)
6                       y= y-x;
7               else
8                       x= x-y;
9       return x;
}
```

The path described in the first bullet is the following: <1 2 3 4 5 6 4 5 7 8 4 9>. The symbolic execution starts with x = X, y = Y and the following path condition X>=0 AND Y>=0.
At point 4 we should have X != Y
At point 5 we should have X < Y
At point 6 we have that y = Y-X
At point 4 we should have X != Y-X, that is, Y != 2X
At point 5 we should have X >=Y-X
At point 8 we have that x = X-y, that is x = X-Y+X, that is x = 2 X -Y
At point 4 we should have that 2 X -Y == Y-X, that means 3X == 2Y.
So, the whole condition for the path would be
X>=0 AND Y>=0 AND X < Y AND 2 X >Y AND 3X=2Y
This is a satisfiable condition: for example X =8 Y = 12

The path described in the second bullet is the following: <1 2 3 4 9>. The symbolic execution starts with x = X, y = Y and the following path condition X<0 AND Y<0 as required by the exercise. To go

through path 4 9 we need to have X == Y. So, the full condition is X<0 AND Y<0 AND X == Y. It is satisfiable and we could select as test cases any pair or negative and equal numbers (e.g., x=-3, y-3).