Tentamen

DD2385 Software Engineering, spring 2014

Monday june 2nd 2014 10.00 – 13.00

The exam has two parts with 30 points each.

Mark FX:	22-23 points on part I
Mark E:	at least 24 points on part I
Mark C:	Passed part I and a total score of minimum 43
Mark A:	Passed part I and a total score of minimum 55

Maximum 3 bonus points from lab assignment will be added to the part I score.
All bonus points will be added to the total score for marks C and A.

part I

Please write several part I – solutions on the same sheet.

(5p) **1. Seven** design pattern names are specified followed by **five** short pattern descriptions. Choose the appropriate design pattern name for each short description. Two names will remain unused.

Iterator	**Mediator**	**State**	**Proxy**
Facade	**Template Method**	**Strategy**	

A) An object controls access to another object. Both implement the same interface.

B) Part of an algorithm is exchangeable at runtime. The exchangeable part is encapsulated in an object.

C) A class provides a simple interface to a complex system that may consist of many classes.

D) Objects in a collection are accessed in sequence without revealing the underlying structure.

E) The main features of an algorithm are defined in a superclass. Parts of the algorithm are defined (or redefined) in subclasses.

(8p) **2.** Draw a UML class diagram for the following classes and interface. Open and closed arrowheads must be used correctly. Multiplicity must be specified when relevant. Associations with filled diamonds are not required. Instance variables and methods are not required in the diagram but any associations defined by variables or methods must be included. *All classes mentioned in the code except ArrayList* must be specified in the diagram. ... means omitted code.

```
class Doer {
    ArrayList<Wit> wlist = new ArrayList<Wit>();

    void tie(Wit w) {wlist.add(w);}

    void untie(Wit w) {wlist.remove(w);}

    void tell() {
        for (Wit w:wlist)
            w.refresh();
    }
}

class Head extends Doer {
    ...
    void changestate(int newstate){
        mystate = newstate;
        tell();
    }
    int getstate() {...}
}

public interface Wit {
    public void refresh();
}

class Looker implements Wit {
    Head myDoer;
    ...
    Looker (Head d){ myDoer = d; }
    public void refresh() { ... }
    ...
}

class Gazer implements Wit {
    Head myDoer;
    ...
    Gazer (Head d) { myDoer = d; }
    public void refresh() { ... }
    ...
}
```

(1p+2p) **3.** The class names in question **2** are chosen to hint but not reveal in an obvoius way which design pattern is used. Which pattern is it? You'll get 1 point for the correct name and another 2 points if you can explain how you can see the pattern in the code fragment and/or in the UML diagram.

(2p) **4a.** State two properties or rules of the methodology *XP (eXtreme Programming)* that differ radically from the Waterfall methodology.

(2p) **4b.** State two additional rules of *XP*.

(2p) **5a.** What is the output from the following program? You will find alternatives after class `Athlete`.

```
class Question5 {
    public static void main(String[] u) {
        Athlete jenny = new Athlete();
        Person luke = new Person();
        Person ellie = new Athlete();

        jenny.activity();
        luke.activity();
        ellie.activity();
        System.out.println();
    }
}

class Person {
    void activity() {
        System.out.print("REST ");
    }
}

class Athlete extends Person {
    void activity() {
        System.out.print("RUN ");
    }
}
```

A) `REST RUN RUN`
B) `RUN REST RUN`
C) `RUN REST REST`
D) `REST RUN REST`

(1p) **5b.** Assume the class definitions from question **5a** and the following method head.

```
static void testm (Person p)
```

Which one of the statements below is NOT TRUE?

A) Objects of type `Object` may be passed as parameter to `testm()`
B) Objects of type `Person` may be passed as parameter to `testm()`
C) Objects of type `Athlete` may be passed as parameter to `testm()`

6. A class `Point` to represent points in two dimensions (x and y) is written.

(2p) **a.** Two points with identical x-coordinates and identical y-coordinates must be regarded as the *same* point. E.g. if two objects are created like this

```
p1 = new Point(7, 11); p2 = new Point(7, 11);
```

and both are entered into a set, e.g. a `HashSet`, the number of objects in the set will be *one* after inserting the *two* objects.

(2p) **b.** In addition, it must be possible to sort the points according to their x-coordinates. A list like

`ArrayList<Point> thePoints` will be sorted by

```
Collections.sort(thePoints)
```

Explain how to write the class `Point` with the properties above.

To get all the points you must show that you understand the principles used in Java for the properties. It is not required to give correct names on methods etc. Neither is Java code required but it may be easier to provide a clear answer if you write some code (or pseudocode).

(3p) **7.** Which *three* of the statements below are correct? Statements about programming relate to Java of course. Give exactly *three* answers. If you give more than three, only the first three will be considered.

A) You may declare instance variables in a Java `interface`.

B) `JUnit` is a Java development environment designed to teach small children how to program.

C) A class may implement two interfaces: `class A implements I1, I2 {...}`

D) A class may inherit from two classes
`class A{} class B{} class C extends A, B {...}`

E) A class may inherit from a class that in turn has inherited from a class:
`class A{} class B extends A {} class C extends B {}`

F) If you know that the `int` – variables `m` and `n` both are `17` you can be absolutely sure that the value of `(n == m)` is `true`.

G) If you know that the `String` – variables `p` and `q` both are `"PI"` you can be absolutely sure that the value of `(p == q)` is `true`.

H) Framework is an alternative name for the category of design patterns describing how objects are organized.

del II

Questions 8–10 may be answered on one paper but use a new one for question 11!

8. Look at class `Mystery` and the printout from the program.

```
class Mystery {
    static int spell;

    Mystery(int s) {
        spell = s;
    }

    public String toString() {
        return "" + spell; // "" + is used to enforce the String type,
    }                      // this is not part of the question

    public static void main(String[] u) {
        Mystery m1 = new Mystery(1729);
        Mystery m2 = new Mystery(666);  // *** question 8b) ***
        Mystery m3 = new Mystery(42);
        System.out.print(m1 + " ");
        System.out.print(m2 + " ");
        System.out.print(m3);
        System.out.println();
        // System.out.println(spell);   // *** question 8c) ***
    }
}

Output is: 42 42 42
```

(2p) **8a.** In the main method, three objects of `Mystery` are created with different parameters but all three are printed as `42`. Explain why! What change(es) should be made to class `Mystery` in order get the printout `1729 666 42` without changing the print statements.

(1p) **8b.** If the printout of object `m1` is made at the first comment in the main method above, what value will be printed?

(1p) **8c.** If the last line of the main method is uncommented, will the program compile without errors? The question refers to the program above, not after possible changes according to **8a**.

(4p) **9.** Explain polymorphism and dynamic binding in object oriented programming. How can they be useful to a programmer?

10. Define a method that filters a list. From a list of objects, produce another list of objects that all satisfy a condition. E.g.

– From a list of integers, create a new list with only the odd numbers
– From a list of objects representing persons, create a list with persons within a certain age range.
– From a list of words, create a new list with only words shorter than a limit.

(3p) **10a.** Describe a general solution, *not* solutions for the specific examples.

(3p) **10b.** Apply the solution from **10a** to the following special case: The list contains `String`– objects and the condition is that the text strings are shorter than `n`.

11. *Decorator pattern, decorate a queue*

This is an `interface` for a queue.

```
interface IQueue<E> {
    public boolean isEmpty();
    public void put(E obj);
    public E get();
}
```

Assume there is a class `Queue` implementing `IQueue<String>`. You need *not write* class `Queue`. The task here is to write an abstract `Decorator` class and two concrete decorators.

- **Decorator 1:** Keep track of the number of elements in the queue. Supply a method to retrieve the number of elements. Remember that the queue is not necessarily empty when it is decorated.
- **Decorator 2:** Make it possible to
  - Clear the queue (remove all elements)
  - Remove all elements from the queue that satisfy a condition. The condition must be a parameter of the the method.

The interface `IQueue<E>` defines the basic operations of the queue and must not be changed. No other operations are allowed on the queue.

(4p) **11a.** Chose your own names for the decorators and draw a UML diagram with `IQueue, Queue`, the abstract `Decorator`–class and the two concrete decorators. The diagram must follow the standard *Decorator* pattern.

(4p) **11b.** Define the abstract *Decorator*–class according to pattern *Decorator*.

(4p) **11c.** Define the class for concrete Decorator 1 according to pattern *Decorator*.

(4p) **11d.** Define the class for concrete Decorator 2 according to pattern *Decorator*.

- To get all the points, your code must clearly demonstrate the pattern properties.
- However, some points may be given even if you don't master the pattern.
- Perfect Java syntax is not required.
- *Tip:* The basic queue operations must work for objects of the decorator classes.