Tentamen

DD2385 Software Engineering spring 2015

Friday june 5th 2015 9.00 – 12.00

The exam has two parts with 30 points each

Mark FX:	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 assignments will be added to the part I score.
Alla bonus points will be added to the total score for marks C and A.

del I

You are welcome to write several part I – answers on the same sheet.

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

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

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

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

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

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

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

(8p) **2.** Draw a UML class diagram for the following classes and interface. Open and closed arrowheads must be used correctly. Multiplicities 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 shown in the diagram. *All classes mentioned in the code except ArrayList* must be specified in the diagram. ... means omitted code.

```
class Center {
    ArrayList<Spec> wlist = new ArrayList<Spec>();

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

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

    void tell() {
        for (Spec w:wlist)
            w.amend();
    }
}

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

public interface Spec {
    public void amend();
}

class Looker implements Spec {
    Focus myCenter;
    ...
    Looker (Focus c){ myCenter = c; }
    public void amend() { ... }
    ...
}

class Wiewer implements Spec {
    Focus myCenter;
    ...
    Wiewer (Focus c) { myCenter = c; }
    public void amend() { ... }
    ...
}
```

(1p) **3.** The class names in question 2 are chosen to hint but not reveal in an obvious way which design pattern is used. Which pattern is it?

(2p) **4a.** What characterizes *test driven* software development ?

(1p) **4b.** What is *refactoring* in software development?

(2p) **4c.** Test driven development and refactoring are used in *XP (eXtreme Programming)*. State four additional rules in *XP*.

(1p) **4d.** Describe *Rapid Prototyping* with a few sentences.

5. The following classes are defined

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

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

(1p) **5a.** Create an object:

```
Person miranda = new Athlete();
```

What is printed by `miranda.activity();` ?
What is the rule that determines which method is called?

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

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

Which ONE of the statements below is FALSE?

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()`

"Object of type A" means that the object is created with `new A()`

(2p) **6.** What is an *abstract class* in Java. Why are such classes useful?

(2p) **7.** In the Java libraries are methods that will sort a list of objects, e.g.

```
Collections.sort(myList)
```

A sorting method must be able to compare the objects. Assume that `myList` has objects of class `A` and that you have defined class `A` yourself. How do you specify the comparison in `A` so that the sorting works?

To get the 2 points you must show that you understand the principles used in Java. It is not required to give correct names for methds etc. Neither is Java code required but it may be easier to provide a clear answer if you write some code.

(4p) **8.** Which *four* statements below are correct? Statements about programming relate to Java of course. Give exactly four answers. If you give more that four, only the first four will be considered.

A) To *instantiate* a class `A` means that you write another class that inherits from `A`.

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

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

D) *Polymorfism* is a problem in object oriented programming that is solved by making methods private, i.e. you mark them as `private` in the code.

E) 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`.

F) 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`.

G) A *framework* in Java will contain classes and/or interfaces.

H) All classes in Java inherit from the class `Object` even if you don't write it explicitly in the class definition.

I) The activity of a `Thread` in Java is started when the programmer calls a method called `run()`.

del II

9.

```
class Particle {
    int x, y;
    Particle (int x, int y) {
        thix.x = x; this.y = y;
    }
}
```

(4p) Extend the class `Particle` so that the class itself knows how many objects of `Particle` that have been created. Write a method in `Particle` that returns the number of created objects of `Particle`. Show how to call the method and show the complete new class with the new properties. The number of objects refers to objects created during the current program run. Numbers should not be stored on a file. No other classes beside `Particle` may be used or written.

(4p) **10.** What is a "factory method"? What is its purpose?
Give an example of a situation where it is appropriate with a factory method.
Which Java–*modifyer* is essential to use for a factory method?
Describe how constructors are written in connection to factory methods.

(4p) **11.** Question **6** on part I is about sorting in Java. Assume that it is solved, that objects of class `A` are sorted correctly by

```
Collections.sort(myList)
```

Now we would like to use a library method to sort the list in another way than was defined in class `A` and we are not allowed to change `A` at all. How can this be done? "Another way" of sorting is e.g. to sort on names when the first sorting (defined in `A`) is sorting on social security numbers.
The mechanism in Java for sorting in a second way uses a known design pattern. Which pattern is that?

It is not necessary to name items from the library. Principles are important. If you don't know how it is done in the library, use your Java knowledge to suggest a solution.

12. A file system consists of *files* and *directories*. Directories may contain files and other directories which in turn can contain files and directories etc. accordning to pattern *Composite*. Files are represented by objects of class `File` which is shown below. Directories are represented by class `Directory`. Files and directories have names, stored in instance variable `name` in the common superclass `FileElement`. Notice that `File` is not the same class as in the JAVA API. The question does not deal with "real" file handling or references to real files.

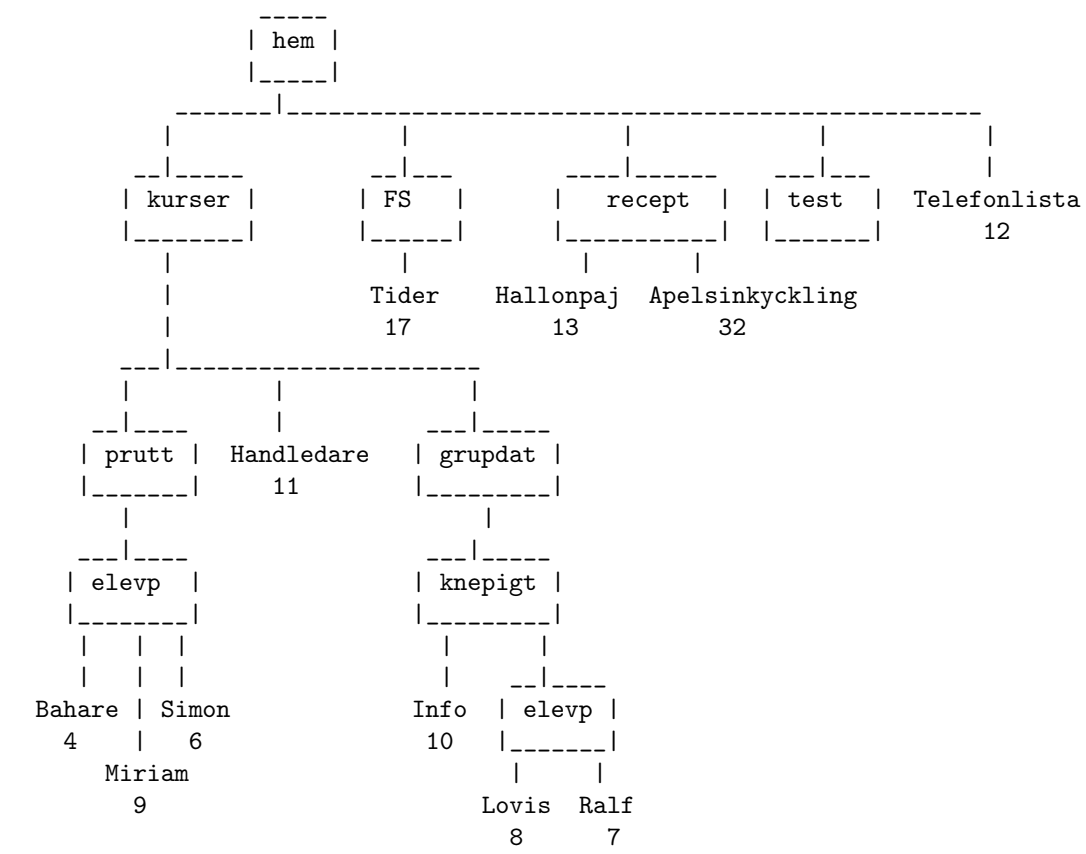
```
abstract class FileElement {

    String name;
    Directory parent = null; // each FileElement except the root must always
                             // have a reference to its parent

    FileElement (String n) {
        name = n;
    }

    public String toString() {
        return name;
    }

    abstract void add(FileElement fe);
    abstract void remove(FileElement fe);
    abstract int size();
  }

class File extends FileElement {
    int size;

    File (String n, int s) {
        super(n);
        size = s;
    }

    void add(FileElement fe) {}
    void remove(FileElement fe) {}

    int size() {
        return size;
    }
}
```

(18p) **8.** Write (part of) class `Directory` according to pattern *Composite*. A `Directory` may contain objects of `Directory` and objects of `File`. `Directory` must include the following methods:

- `add()` to add a `FileElement`–object.
- `remove(FileElement f)` to remove a `FileElement`–object
- `size()` to compute the sum of all file sizes in the directory and all its subdirectories according to *Composite*.
- `findbigdir()` This method must print a list of the contents of the directory sorted by size. The size of a directory is the sum of the sizes of all files in the directory and alls its subdirectories. Look at the example on page 8. You may need to write a new class and/or extend the given the classes.
- `moveup()` to move the contents of a directory to the directory immediately above it and delete the directory. If there is no directory above, the method should have no effect. Look at the example on page 9.
- `moveupR()` to move all files in a directory and all files in all subdirectories to the level above. The directory and its subdirectories must be deleted from the structure.

Perfekt Java-syntax krävs **INTE** för full poäng på någon uppgift.

Example of object structure in question 12.

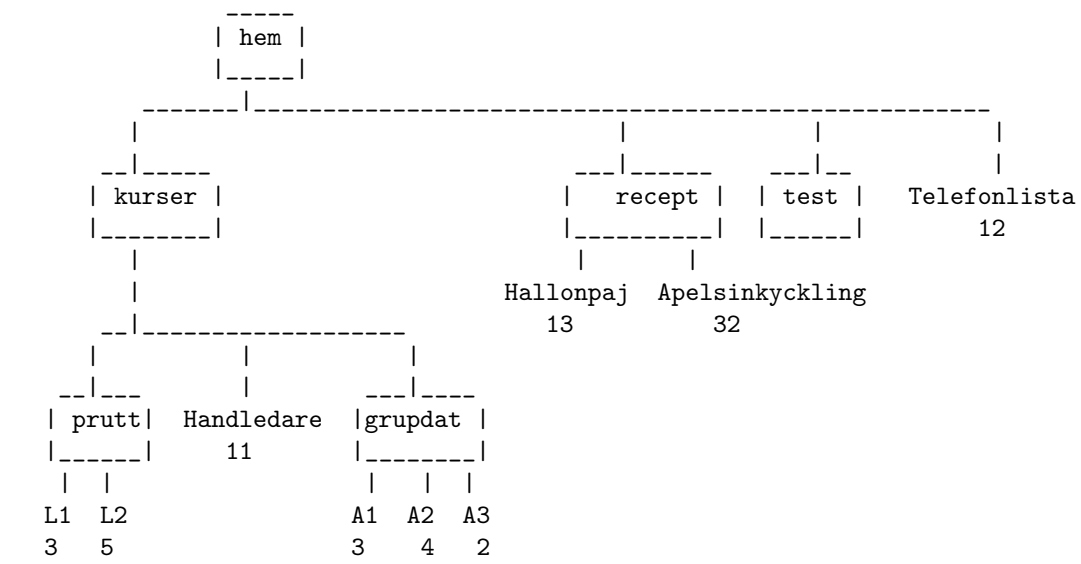


Calling `findBigDir()` on directory `kurser` gives

```
grupdat     25
prutt       19
Handledare  11
```

For directory `hem`

```
kurser       55
recept       45
FS           17
Telefonlista 12
test          0
```



Calling `moveup()` on directory `kurser` above should produce the structure below.