EINDHOVEN UNIVERSITY OF TECHNOLOGY
Department of Mathematics & Computer Science

TU/e

Exam Programming Methods, 2IP15, Wednesday 17 April 2013, 09:00–12:00

This *closed-book* exam consists of 16 questions, worth 3 points each, on 2 numbered pages. The final grade is computed as $(2 + s)/5$ rounded to one decimal, where s is the sum of the scores for the questions.

You must provide a to-the-point explanation or motivation for every answer.

---

1. What are the steps in the design technique *Divide & Conquer*?
2. Give five advantages of a modular design over a monolithic design.
3. Give three guidelines for functional decomposition.
4. Why is it important to reason about a method call in terms of the contract rather than the implementation of the method being called?
5. Are design patterns a form of code reuse?
6. A program needs only one instance of class `GameTable`. Currently, the code looks like this (javadoc minimized):

```
1  /** Game table with settable limit. */
2  public class GameTable {
3
4      /** Limit for bets at this table (0 = no limit). */
5      private int limit;
6
7      /** Constructs a table without limit. */
8      public GameTable() {
9          this.limit = 0;
10     }
11
12     /** Sets the table limit. */
13     public void setLimit(final int limit) {
14         this.limit = limit;
15     }
16
17     // Other game table operations
18     . . .
19 }
```

   Apply the *Singleton* design pattern to ensure that a single object of this class is provided. It is to be used in a single-threaded application. Write out the Java code for the singleton version, and clearly indicate what code was added or modified.

7. How is the *Iterator* design pattern a form of abstraction?
8. You are working on a package with multiple classes, but would like to simplify the interface offered to clients of the package. What design pattern would be useful?
9. Are there advantages when implementing the *Adapter* design pattern through inheritance rather than composition?

10. Why is it not a good idea to implement the *Decorator* design pattern through inheritance?

11. Explain the relationship between the *Strategy* design pattern and the *Dependency Inversion Principle*. Draw a UML class diagram for the *Strategy* design pattern, and for its anti-pattern with direct dependency rather than inverted dependency.

12. How is the *Factory Method* design pattern a special case of the *Template Method* design pattern?

13. With the *Observer* design pattern, there are two ways of propagating data to observers: *push* and *pull*. What are the trade-offs?

14. What design pattern has been used in the following piece of code? Give example code for a `ConcreteWaferProcessor` class, using appropriate `System.out.println(...)` for functionality.

```
1  public abstract class AbstractWaferProcessor {
2
3      /** Processes a given wafer. */
4      public void processWafer(final Wafer wafer) {
5          do {
6              conditionWafer(wafer);
7          } while (! measureWafer(wafer));
8          exposeWafer(wafer);
9      }
10
11     /** Conditions a given wafer. */
12     protected abstract void conditionWafer(Wafer wafer);
13
14     /** Measures a given wafer, returning whether this was successful. */
15     protected abstract boolean measureWafer(Wafer wafer);
16
17     /** Exposes a given wafer. */
18     protected abstract void exposeWafer(Wafer wafer);
19
20 }
```

15. What roles are relevant in the *Composite* design pattern and how are they related? Draw a UML class diagram.

16. When using the *Command* design pattern, the command objects respond to an `execute()` method and possibly also an `undo()` method. The preconditions of these methods are specific to the kind of command and its parameters. How are these preconditions guaranteed when using the *Command* design pattern to implement a multi-level undo-redo facility?