EINDHOVEN UNIVERSITY OF TECHNOLOGY
Department of Mathematics & Computer Science

TU/e

Exam Programming Methods, 2IP15, Wednesday 26 June 2013, 14:00–17:00

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 abstraction mechanisms are available in the programming language Java to apply *Divide & Conquer*?
2. Why is it *not* a good idea to write monolithic programs? Give five reasons.
3. Why is it important to reason about a method implementation in terms of the contract rather than in terms of the actual calls of that method?
4. Give three benefits of developing test cases *before* implementing a class and its methods.
5. What is an Abstract Data Type (ADT), and what does it take to implement an ADT?
6. Given are the interfaces of a legacy and a Next-Generation wafer processor:

```
1  /** Interface of a legacy wafer processor. */
2  public interface IWaferProcessor {
3
4      /** Conditions a given wafer. */
5      void conditionWafer(Wafer wafer);
6
7      /** Measures a given wafer. */
8      void measureWafer(Wafer wafer);
9
10     /** Exposes a given wafer. */
11     void exposeWafer(Wafer wafer);
12
13 }
```

```
1  /** Interface of a Next-Generation wafer processor. */
2  public interface IWaferProcessorNG {
3
4      /** Loads and conditions a given wafer. */
5      void loadWafer(Wafer wafer);
6
7      /** Measures the loaded wafer. */
8      void measureWafer();
9
10     /** Exposes the loaded wafer, and returns it. */
11     Wafer exposeWafer();
12
13 }
```

Write a class `WaferProcessorAdapter` that adapts a legacy wafer processor (given as an object) to be usable through the Next-Generation interface.

7. Why is it *not* a good idea to implement the *Decorator* design pattern through inheritance?
8. In what ways is a singleton class better than using just a global variable?
9. Explain the *Iterator* design pattern.
10. When is it appropriate to apply the *State* design pattern?
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. What is the intent of the *Factory Method* design pattern?
13. Why is it useful that the update method in the Observer interface includes a reference to the subject (the object being observed)? Does the observer not know what object it is observing?

```
1     interface Observer {
2         void update(Subject subject, Data data);
3     }
```

14. What roles are relevant in the *Composite* design pattern and how are they related? Draw a UML class diagram.
15. 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?
16. Give three benefits of applying design patterns.