Software Engineering: Design & Construction

Department of Computer Science
Software Technology Group

TECHNISCHE UNIVERSITÄT DARMSTADT

Practice Exam **January 16, 2014**

First Name	
Last Name	
Matriculation Number	
Course of Study	
Department	
Signature	

Permitted Aids

- Everything except electronic devices
- Use a pen with document-proof ink (No green or red color)

Announcements

- Make sure that your copy of the exam is complete (11 pages).
- Fill out all fields on the front page.
- Do not use abbreviations for your course of study or your department.
- Put your name and your matriculation number on all pages of this exam.
- The time for solving this exam is 45 minutes.
- All questions of the exam must be answered in English.
- You are not allowed to remove the stapling.
- All tasks can be processed independently and in any order.
- You can use the backsides of the pages if you need more space for your solution.
- Wrong answers can lead to a subtraction of points.
- Make sure that you have read and understood a task before you start answering it.
- It is not allowed to use your own paper.
- Sign your exam to confirm your details and acknowledge the above announcements.

Topic	**1**	**2**	**3**	**4**	**5**	**Total**
Points						
Max.	**5**	**13**	**10**	**10**	**7**	**45**

Name: ____________________ Matriculation Number: ⌴⌴⌴⌴⌴⌴⌴

Topic 1: Introduction 5P

a) Software Design 1P

What is the greatest challenge in designing and building software when compared to designing and building bridges?

b) Structured Programming 1P

Which language features do structured programming languages offer beyond assembly languages?

c) Variance 1P

Variance of type parameters is related to which S.O.L.I.D. principle?

Name: ______________________________ Matriculation Number: ⌴⌴⌴⌴⌴⌴⌴

d) Fragile Base Classes 2P

Describe the Fragile Base Class Problem in one sentence and give an example.

Name: ____________________ Matriculation Number: ⌴⌴⌴⌴⌴⌴⌴

Topic 2: S.O.L.I.D. 13P

a) Interface Segregation vs Dependency Inversion 2P

The Interface Segregation and Dependency Inversion principles have in common that they are both about defining interfaces. They address different problems, however. Explain that difference in one or two sentences.

b) Origins of the Interface Segregation Principle (ISP) 4P

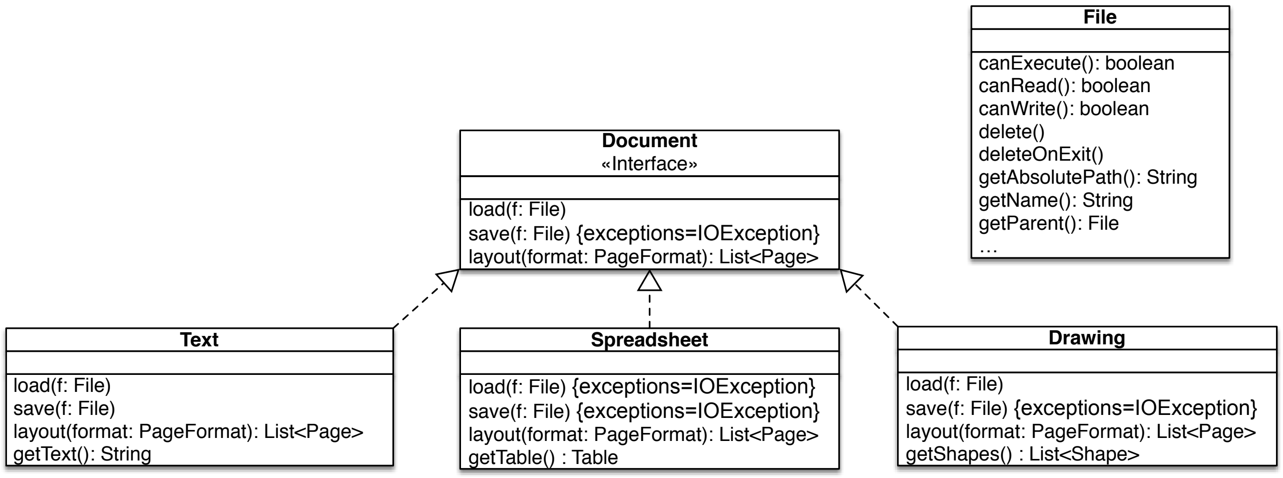
Folklore says that the ISP was invented by Robert C. Martin while working for Xerox on a flexible printer system that would not only print but also perform a variety of printer-related tasks like stapling and faxing. There was one main *Job* class that was used by most other tasks. During the development of the system, making a modification to the Job class therefore affected many other classes and a compilation cycle would take a long time. This made it very time-consuming to modify the system. Since the big Job class had many methods specific to a variety of different clients, the solution was to introduce multiple smaller interfaces for each client. A client would then only operate on a specific, small interface to the Job class. Modifications to the Job class would then affect a smaller number of interfaces and therefore less classes had to be recompiled.

From this description, it sounds like there are other S.O.L.I.D. principles that could have been violated by Xerox's design. Which ones, why, and how would you redesign the system to adhere to them?

Name: ______________________ Matriculation Number: ⌴⌴⌴⌴⌴⌴⌴

c) Design Review 7P

In Figure 1, you find a basic document hierarchy for an office suite. Note that not every class is shown. Assume that there are no checked exceptions, but that the given exceptions are complete, i.e., a method will not throw any exception that is not given in its signature. For each S.O.L.I.D. principle, say how the design violates or honors it (as much as you can tell from the diagram). Suggest improvements and draw a UML diagram that incorporates your improvements.



File
canExecute(): boolean
canRead(): boolean
canWrite(): boolean
delete()
deleteOnExit()
getAbsolutePath(): String
getName(): String
getParent(): File
…
Document
«Interface»
load(f: File)
save(f: File) {exceptions=IOException}
layout(format: PageFormat): List<Page>
Text
load(f: File)
save(f: File)
layout(format: PageFormat): List<Page>
getText(): String
Spreadsheet
load(f: File) {exceptions=IOException}
save(f: File) {exceptions=IOException}
layout(format: PageFormat): List<Page>
getTable() : Table
Drawing
load(f: File)
save(f: File) {exceptions=IOException}
layout(format: PageFormat): List<Page>
getShapes() : List<Shape>

Figure 1: Class diagram for the document hierarchy

Name: ______________________ Matriculation Number: ␣␣␣␣␣␣␣

Name: ____________________ Matriculation Number: ⌴⌴⌴⌴⌴⌴⌴

Topic 3: Scala **10P**

a) Scala Features 3P

Name three features beyond closures, operator overloading, infix/prefix/postfix methods that Scala offers that Java 7 does not offer?

b) Closures 3P

Name three ways in which Scala closures are different from anonymous classes in Java. Hint: think about what would be different or difficult if you had to implement the following `loopUntil` control flow operator in Java:

```
def loopUntil(cond: =>Boolean)(body: =>Unit) =
  while(!cond) {
    body
  }

def example() {
  var i = 0
  loopUntil(i == 100) {
    println(i)
    if (i == 50) return;
    i += 1
  }
  println("Done.")
}
```

Name: ______________________ Matriculation Number: ⌴⌴⌴⌴⌴⌴⌴

c) Dynamic Dispatch and Pattern Matching 4P

Scala supports both dynamic dispatch and pattern matching. Imagine you implement the same application twice, once only allowing dynamic dispatch and methods (no external functions!), as in

```
trait Animal {
  def hello()
}

class Cat extends Animal {
  def hello() { println("Meow.") }
}

class Dog extends Animal {
  def hello() { println("Woof.") }
}
```

and once only allowing pattern matching and functions (no methods!) as in

```
trait Animal
class Cat extends Animal
class Dog extends Animal

def hello(a: Animal) = a match {
  case c: Cat => println("Meow.")
  case d: Dog => println("Woof.")
}
```

Describe in at most 4 sentences how the two implementations are different in terms of supporting the Open-Closed Principle. Hint: think of which forms of extensions each implementation supports and doesn't support.

Name: ______________________________ Matriculation Number: ␣␣␣␣␣␣␣

Topic 4: Reactive Programming 9P

a) Reactive? 3P

Which of the following programs are *reactive* and which are not? Why?

1. The "javac" command line command
2. Eclipse
3. a pulse rate monitor.

b) Glitches 3P

Explain what a glitch is and how a reactive framework such as REScala can prevent them from occurring?

Name: ______________________ Matriculation Number: ⌴⌴⌴⌴⌴⌴⌴

c) Temperature Alarm in REScala 3P

Given a temperature event stream `temp` of type `Event[Int]`, write a program that creates a string signal `status` which has value `"green"` when the last value from `temp` was less than `25`, `"red"` when the last value is greater than `40` and `"yellow"` otherwise.

Topic 5: Design Patterns 7P

a) Strategies in Functional Programming Languages 1P

Which feature makes it easy to implement the Strategy design pattern in Scala (and functional programming languages)?

4cm

Name: ______________________ Matriculation Number: ⌴⌴⌴⌴⌴⌴⌴

b) Sorting Collections 6P

Sketch two designs for a generic `SortableCollection` trait in Scala that supports sorting. Assume that you have a base class `Collection[A]` which is covariant in `A`. The outcome of the sorting method should be a fresh collection and the order of elements depends on how elements are compared. One of your designs should use the template pattern and the other the strategy pattern to parameterize the sorting process with a comparison operation. Note: You do not have to implement the sorting method, but indicate in a comment where you use the comparison operation.

Discuss the advantages and disadvantages of each design in not more than 4 sentences.