Software Engineering: Design & Construction

Department of Computer Science
Software Technology Group

TECHNISCHE UNIVERSITÄT DARMSTADT

Practice Exam **May 22, 2015**

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 (9 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 30 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.
- 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	Total
Points				
Max.	12	11	7	30

Name: _____ Matriculation Number: ⌴⌴⌴⌴⌴⌴⌴⌴

Topic 1: Introduction 12P

a) Software Design 1P

What is the difference between hardware engineering and software engineering with respect to measuring the quality?

b) Structured Programming 2P

Name two differences and two similarities between functional programming and traditional object oriented (< Java8) programming languages.

c) Variance 5P

Take a look at the following Scala code example:

```
class MyClass extends AnyRef
class MySubclass extends MyClass
//As a reminder:
//Boolean <: AnyVal <: Any
//everything <: Any
//Nothing <: everything
//Null <: every AnyRef

val f1: (Any) => Any = ???
val f2: (Any) => Boolean = ???
val f3: (MyClass) => Any = ???
val f4: (MySubclass) => Boolean = ???
val f5: (Any) => Nothing = ???
val f6: (MyClass) => Null = ???

val f: (MyClass) => Boolean = // f1 ?, f2 ?, ..., f6 ?
```

Continued on the next page.

Name: _____ Matriculation Number: ⌴⌴⌴⌴⌴⌴⌴⌴

The last line is just a short hand for assigning each variable from f_1 ... f_6 to f.

Now state for each f_i if this assignment is valid. (3P)

Discuss for **one** case why it succeeds or why it is rejected by the compiler.
Use the terminology covariance and contravariance. (2P)

Name: _____ Matriculation Number: ⌴⌴⌴⌴⌴⌴⌴⌴

d) Fragile Base Classes — 4P

Discuss two issues of the following design related to the **fragile base class problem** and give concrete examples for each of them.

Listing 1: Provided Library

```
/** A mutable bank account that ensures that the balance is always positive (>= 0) */
class BankAccount {
  private[this] var balance: Long = 0L

  /** Deposits a given amount of money into the account.
      @param amount The deposited amount of money. If a negative amount is given,
      an AccountException is thrown.
  */
  def deposit(amount: Long): Unit = {
    if (amount < 0) throw new AccountException("amount has to be positive")
    balance += amount
  }

  /** Withdraws the specified amount of money from the account.
      @param amount The amount to be withdrawn.
      If the amount exceeds the balance, an AccountException is thrown.
  */
  def withdraw(amount: Long): Unit = {
    if (balance < amount) throw new AccountException("Not enough money")
    balance -= amount
  }

  /** Transfers the given amount to another account. */
  def transfer(that: BankAccount, amount: Long): Unit = {
    this.withdraw(amount)
    that.deposit(amount)
  }
}
```

Listing 2: Example Usage

```
class LogBankAccount extends BankAccount {
  private var events: List[Long] = Nil
  def log(amount: Long): Unit = {
    events = amount :: events
  }

  override def withdraw(amount: Long): Unit = {
    log(amount)
    super.withdraw(amount)
  }
}
```

Name: _____ Matriculation Number: ⌴⌴⌴⌴⌴⌴⌴⌴

Name: _____ Matriculation Number: ⌴⌴⌴⌴⌴⌴⌴⌴

Topic 2: S.O.L.I.D. 11P

a) 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: ⌴⌴⌴⌴⌴⌴⌴

b) Design Review 7P

In Figure 1, you find an excerpt from a basic dungeon crawler game. 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.

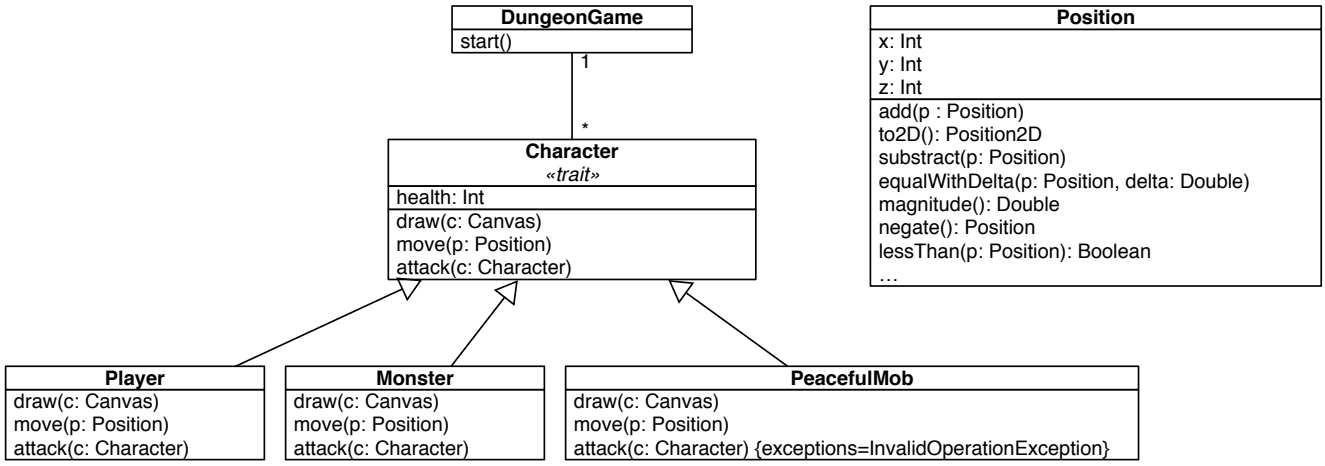

Figure 1: Class diagram for the document hierarchy

Name: _____ Matriculation Number: ⌴⌴⌴⌴⌴⌴⌴

Topic 3: Scala — 7P

a) Mixin Composition and Linearization — 4P

Given the following Scala code example:

```
abstract class Password{
 private[this] var password = "password"
 def set(pwd: String) = { password = pwd }
}

trait Hash extends Password{
 abstract override def set(pwd: String) = super.set(pwd.hashCode.toString)
}

trait Salt extends Password{
  val salt = "xyz"
 abstract override def set(pwd: String) = super.set(salt + pwd)
}

class HashedPassword extends Password with Hash
```

The previous code example contains various Scala traits which were composed in the following code snippet.

```
class DoubleHashWithSalt extends HashedPassword with Salt with Hash
```

Now state

1. the **linearization**
2. the **order of the super calls** for the method `set` of the class `DoubleHashWithSalt`.
3. Does `DoubleHashWithSalt` actually hash the password twice?

Name: _____ Matriculation Number: ⌴⌴⌴⌴⌴⌴⌴⌴

b) Scala vs. Java 1P

What is the difference between *super* calls in Scala compared to the *super* calls in Java?

c) OCP and Pattern Matching 2P

Given the following pattern matching example. Does this implementation support the Open-Closed Principle? Justify your answer.

```
trait Student
class MasterOfArts extends Student
class MasterOfScience extends Student

def getAbbreviation(s: Student) = s match {
  case (_: MasterOfArts)    => "M.A."
  case (_: MasterOfScience) => "M.Sc."
}
```