Inversion of Control

Name: _____ ID: _____

These questions do not have a formal, definitive answer. They are meant to be food for thoughts. Feel free to seek answers on browsing the Internet, talking to other software developers or reading books.

Today we will be going over some good practices of software development. In particular, we will answer some questions about the *Single Responsibility Principle*, and about *Inversion of Control*. In order to reach these concepts, we will develop a small server, that returns the grade of students, over a socket connection.

1. Given that we will be using sockets to implement communication, let's start with the basic: "what is a socket?" question...

2. Students will be represented by the class below. public class Student implements Serializable private static final long serialVersionUID = 1L; public final String name; public final double grade; public final long key; public Student(long key, String name, double grade) this.name = name; this.grade = grade; this.key = key; public String toString() return name + "" + "(" + key + "): " + grade; Notice that it implements the `Serializable` interface. Why?

3. Information about the students will be stored remotely, and we will access it through a pattern called *Data Access Object* (DAO). The interface of our DAO is given below. public interface DAO¡K, V¿ V get(K key); void add(K key, V value); void delete(K key); void update(K key, V value); What is a DAO, and what is the advantage of using it?

4. On the server side, we will have a database, that has a very simple implementation: our database is just a table that associates numbers to instances of the `Student` class. Take a look into the implementation: public class GradeServer public static void main(String[] args) Map¡Long, Student¿ students = new HashMap¡Long, Student¿(); loadDB(students); ServerSocket serverSocket = null; try serverSocket = new ServerSocket(4444); while (true) new StudentHandlerThread(serverSocket.accept(), students).start(); serverSocket.close(); catch (IOException e) e.printStackTrace(); System.exit(-1); By now, all that our server will do is to receive messages containing numbers and to send messages containing student data. These functions are implemented by the `StudentHandlerThread` class. Give an implementation for this class.

5. We need an implementation of our DAO. So far, we only ask for information regarding a student. In the next question we will be able to delete, update and insert new entries. So, how the implementation of the `get` method of our DAO would be like? Remember that we must be able to talk to an instance of `StudentHandlerThread`.

6. We now need to implement the other methods of the DAO. We will need some sort of *communication protocol*. How to re-implement the class `StudentHandlerThread` so that it will handle different types of requests?

7. Take a look into the class `dao.ex2.Dao2`, in the supplementary course material. The implementation of this class has a lot of problems. Which are this problems, and how to improve them?

8. The class `dao.ex3.Dao3`, also available in the supplementary material, improves on `dao.ex2.Dao2` by doing some *code factorization*. But it still has problems. The most serious of these problems is the *session per operation* approach that it adopts. What is this problem, and how to improve it?

9. One of the ways to solve the session per operation problem is to create two methods, one to open, and another to close the connection. The DAO would, then, be used as in the method below:

```
public static void main(String args[]) {
  System.out.println("Running the client 4");
  Dao4 d = new Dao4();
  d.openConnection("localhost", 4444);
  Student s = d.get(200934878L);
  System.out.println(s);
  d.add(160355385L, new Student(160355385L, "Caravaggio", 98.0));
  s = d.get(160355385L);
  System.out.println(s);
  d.update(160355385L, new Student(160355385L, "Michelangelo", 99.5));
  s = d.get(160355385L);
  System.out.println(s);
  d.closeConnection();
}
```

This approach, although apparently elegant, involves many considerations.

(a) Where should the `openConnection` method be inserted?

(b) What about closing the connection? How to enforce that the connection will be eventually closed?

(c) After all: what are the problems with this approach?

10. Conceptually, one of the problems with the approach we last saw is that the `DAO` has two responsibilities: it is in charge, not only, of manipulating student data, but it is also in charge of handling connections. This goes against the *Single Responsibility Principle*. Which principle is this one?

11. One way to solve the problems enumerated above is through *Inversion of Control* (IoC). Normally, an object o_1 calls a method m_2 on an object o_2, which will call a method m_3 on an object o_3 and so forth. However, in the IoC approach, o_1 calls m_2 on o_2, passing o_3 to o_2, possibly as a parameter of m_2. In this way, o_2 has the flexibility of calling m_3 whenever it needs it. In terms of an analogy, this is what happens in job interviews: you do the interview, and leave your telephone with the interviewer, who will contact you in case he needs it. How the IoC approach would help in this case?

12. The principle of Inversion of Control is also known as the *Hollywood Principle*. Why?

13. Take a look into the class `dao.ex5.Dao5`. This class uses IoC to remove the responsibility of handling connections from the DAO. However, the connection handling code is hard to reuse. We can factor the connection code out via the *Command* design pattern. Describe this pattern using an UML class diagram.

14. In the supplementary material we have an example command, which has the following code:

```
public class CommandImpl implements Command {
  public void execute(Dao6 d) {
    // Execute the DAO operations
    Student s = d.get(200934878L);
    System.out.println(s);
    d.add(160355385L, new Student(160355385L, "Caravaggio", 98.0));
    s = d.get(160355385L);
    System.out.println(s);
    d.update(160355385L, new Student(160355385L, "Michelangelo", 99.5));
    s = d.get(160355385L);
    System.out.println(s);
  }
}
```

Write a new command implementation, that reads a file `"st.txt"`, containing a list of students – one student per line. Each line has the format `id, name, grade`. The command must register all these students in the remove database.

15. The last implementation, based on the command pattern, abides to the single responsibility principle. How the different responsibilities, the handling of data and the handling of connections, has been divided among the different software components?

16. Frameworks are heavily based on inversion of control. `Java.swing` is a good example of a framework. Give examples of IoC in this framework.

17. The program in `dao.ex6.Dao6` implements a form of *Dependency Injection* known as *settler injection*. Describe this technique.

18. There are other ways to implement dependency injection, for instance, we have the *constructor injection* method. Could you change `dao.ex6.Dao6` to use this technique, instead of the settler injection model?