Victoria University of Wellington
School of Engineering and Computer Science

SWEN221: Software Development

Assignment 6

Due: Monday 25th May @ Mid-night

Bernies Byte Bach

Bernies Byte Bach is small family run business which supplies computer parts to the local community. A web system is needed so Bernie can more easily manage his orders, customers and stock. As is common for web applications, the website interacts which a backend database which is used to store and retrieve the necessary data.

Rather than using an off-the-shelf component, such as MySQL, Bernie decided to develop a database in-house. Bernie completed the initial design, but has now drafted you in to complete the project.

My Database

The source code for the initial design of the system can be downloaded from the SWEN221 course website. This is made up of several files spread across three main packages:

```
com.bytebach.Main
com.bytebach.impl.MyDatabase
com.bytebach.impl.MyDatabaseTests
com.bytebach.model.Database
com.bytebach.model.Table
com.bytebach.model.Field
com.bytebach.model.Value
com.bytebach.model.IntegerValue
com.bytebach.model.BooleanValue
com.bytebach.model.StringValue
com.bytebach.model.ReferenceValue
com.bytebach.model.InvalidOperation
com.bytebach.server.WebServer
```

Several of the classes in `com.bytebach.model` are interfaces. Your task will be to provide implementations for these interfaces in the `com.bytebach.impl` package. The `MyDatabase` class, which implements `Database`, is already provided (although it has no implementation).

A `Database` is simply a collection of `Tables`, each of which has a unique name. Each table has a *schema*, which is a collection of field declarations. Each declaration determines the name and type

of a column in the table, and indicates which are keys. Each table also contains a list of *table rows*, which constitute the data stored in the table. Each row is a list of *values*, each of which must have the correct type according the schema. Users can create new tables by providing their schema, and add/remove/update existing tables by modifying their rows and entries.

A *key field* is a field which is guaranteed to have a unique value in each row of a table; that is, you cannot have two rows with the same value of that field. A *reference* is a special kind of value which refers to a row in another table. References are useful to ensure consistency in the database; for example, if we have an order in the byte bach system, then that order must be associated with a valid customer. Furthermore, if the customer associated with an order is deleted, then *that order is also deleted*. This is known as a *cascading delete* and is necessary to ensure that no reference in the database is *dangling* — i.e. that referring to something which no longer exists.

Finally, a table is permitted to have *multiple key fields*. In such case, it is the combination of key fields which must be unique for each row in a table, rather than an individual field.

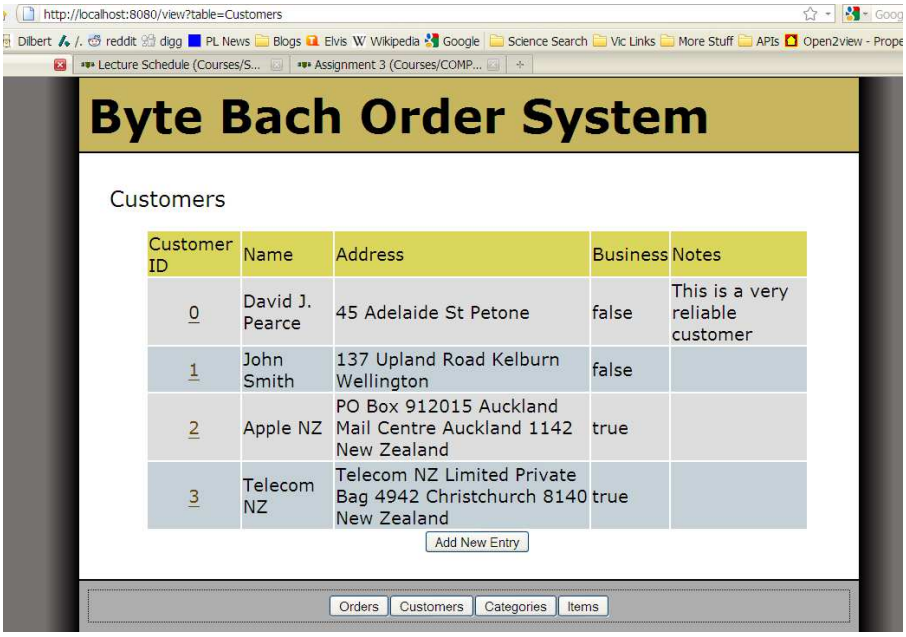
Running the Website

The website is run by a custom server, implemented in `com.bytebach.server.WebServer`. This is complete, and you will not need to modify it. You will need at least a skeleton implementation of `com.bytebach.impl.MyDatabase` before the webserver will run. To run the webserver, use the following command from within the `bin/` directory of your Eclipse project:

```
% java -cp . com.bytebach.Main ../sample.db
```

Here, the example input file `sample.db` is provided to populate the database. You may wish to implement your own input files, or to extend `sample.db`.

At this point, the webserver is running on your local machine[1]. In order, to access the website, you simply point your favourite web-browser at `http://localhost:8080/`. If everything is working correctly, you should see a (very short) introduction message and a navigation bar:



You should spend some time navigating the system. When `MyDatabase` is implemented, you should be able to create new orders, add/remove customers, etc.

[1] If you are using Windows XP, it may ask you to unblock the webserver; you do not need to do that for this assignment

What to Do

You should complete the implementation of `MyDatabase`, such that it adheres with the specification of `Database`. A set of JUnit tests is provided in the `com.bytebach.impl.MyDatabaseTests` to help you check this.

Ultimately, you should aim to get the Byte Bach website running correctly; *however, this is not a requirement for the assignment.* The purpose of the website is simply to illustrate how a database system might be used in practice, and to give you a feel for how web applications work.

NOTE 1: you are not permitted to modify any of the supplied interfaces (e.g. `Table`, `Database`, `Field`, etc)

NOTE 2: you are expected to write additional JUnit tests. The supplied tests DO NOT test all functionality, and should be considered a starting point only.

Submission

Your source files should be submitted electronically via the *online submission system*, linked from the course homepage. The minimal set of required files is:

```
com/bytebach/model/StringValue.java
com/bytebach/model/Database.java
com/bytebach/model/ReferenceValue.java
com/bytebach/model/IntegerValue.java
com/bytebach/model/Table.java
com/bytebach/model/BooleanValue.java
com/bytebach/model/Field.java
com/bytebach/model/InvalidOperation.java
com/bytebach/model/Value.java
com/bytebach/Main.java
com/bytebach/impl/MyDatabase.java
```

You must ensure your submission meets the following requirements (which are needed for the automatic marking script):

1. **Your submission is packaged into a jar file, including the source code**. See the export-to-jar tutorial linked from the course homepage for more on how to do this. *Note, the jar file does not need to be executable.*

2. **The names of all classes, methods and packages remain unchanged**. That is, you may add new classes and/or new methods and you may modify the body of existing methods. However, you may not change the name of any existing class, method or package. *This is to ensure the automatic marking script can test your code.*

3. **All JUnit test files supplied for the assignment remain unchanged.** Specifically, you cannot alter the way in which your code is tested as the marking script relies on this. This does not prohibit you from adding new tests, as you can still create additional JUnit test files. *This is to ensure the automatic marking script can test your code.*

4. **You have removed any debugging code that produces output, or otherwise affects the computation.** *This ensures the output seen by the automatic marking script does not include spurious information.*

Note: Failure to meet these requirements could result in you getting zero marks for the assignment.

Assessment

This assignment will be marked as a letter grade (A+ ... E), based primarily on the following criteria:

- **Correctness (90%)** — does submission adhere to given specification.
- **Style (10%)** — does the submitted code follow the style guide and have appropriate comments (inc. Javadoc)

As indicated above, part of the assessment for the coding assignments in SWEN221 involves a qualitative mark for style, given by a tutor. Whilst this is worth only a small percentage of your final grade, it is worth considering that good programmers have good style.

The qualitative marks for style are given for the following points:

- **Division of Concepts into Classes**. This refers to how *coherent* your classes are. That is, whether a given class is responsible for single specific task (coherent), or for many unrelated tasks (incoherent). In particular, big classes with lots of functionality should be avoided.
- **Division of Work into Methods**. This refers to how well a given task is split across methods. That is, whether a given task is broken down into many small methods (good) or implemented as one large method (bad). The approach of dividing a task into multiple small methods is commonly referred to as *divide-and-conquer*.
- **Use of Naming**. This refers to the choice of names for the classes, fields, methods and variables in your program. Firstly, naming should be consistent and follow the recommended Java Coding Standards (see `http://g.oswego.edu/dl/html/javaCodingStd.html`). Secondly, names of items should be descriptive and reflect their purpose in the program.
- **JavaDoc Comments**. This refers to the use of JavaDoc comments on classes, fields and methods. We certainly expect all `public` and `protected` items to be properly documented. For example, when documenting a method, an appropriate description should be given, as well as for its parameters and return value. Good style also dictates that `private` items are documented as well.
- **Other Comments**. This refers to the use of commenting within a given method. Generally speaking, comments should be used to explain what is happening, rather than simply repeating what is evident from the source code.
- **Additional Tests**. This refers to the presence of additional tests being written. Generally speaking, additional tests should be written to account for limitations in those tests provided.
- **Overall Consistency**. This refers to the consistent use of indentation and other conventions. Generally speaking, code must be properly indented and make consistent use of conventions for e.g. curly braces.

Finally, in addition to a mark, you should expect some written feedback highlighting the good and bad points of your solution.