Victoria University of Wellington
School of Engineering and Computer Science

SWEN221: Software Development

Assignment 4

Due: 23:59pm Sunday 28th May

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 with 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, and may need to be implemented in the `com.bytebach.impl` package. The `MyDatabase` class, which implements `Database`, is already provided (although it has no implementation). The specifications are as follows:

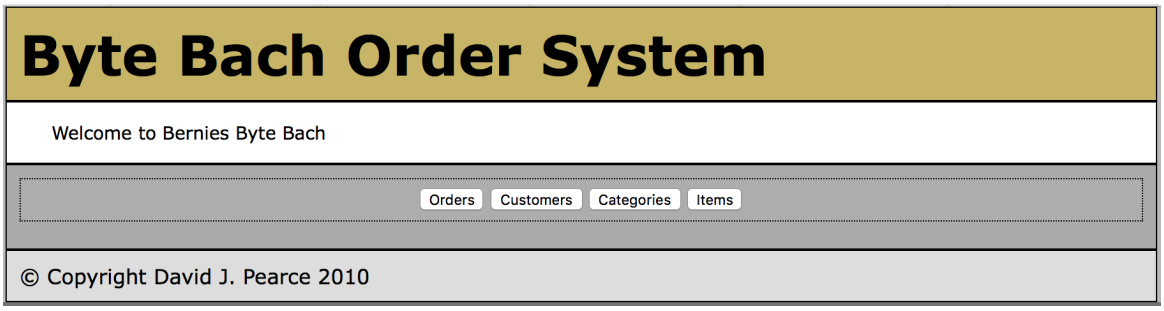
- A `Database` is simply a collection of `Table`s, each of which has a unique name.

- Each table contains a list of `Field`s and a list of *table rows*, which constitute the data stored in the table.
- Each field determines the name and type of a column in the table, and indicates whether it is the *key field*. 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 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.
- Each row is a list of `Value`s, whose types are determined by the field list of the table. That is, the ith value in the value list must have the type of the ith field in the field list.
- A variety of value types are provided in the `com.bytebach.model` package, such as `IntegerValue` and `BooleanValue`. It is worth noting that a `ReferenceValue` is a special type 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.
- You should implement the following functionalities:
  - Create new tables by providing their field lists.
  - Add a new row to a table.
  - Update/Modify the entries of an existing row to a table.
  - *Cascading delete* a row from a table.
  - Delete a table.

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, first run the `Main()` method in `com.bytebach.Main`.

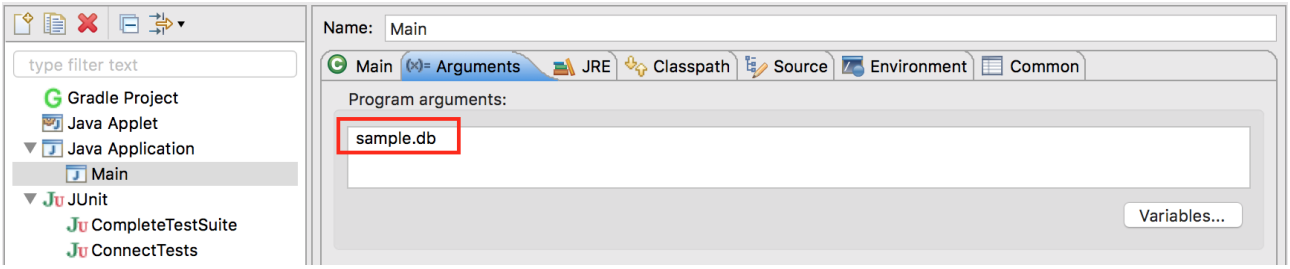
At this point, the webserver is running on your local machine[1]. In order to access the website, you simply point your favourite web-browser at `http://localhost:8080/`. If everything is working correctly, you should see a (very short) introduction message and a navigation bar:



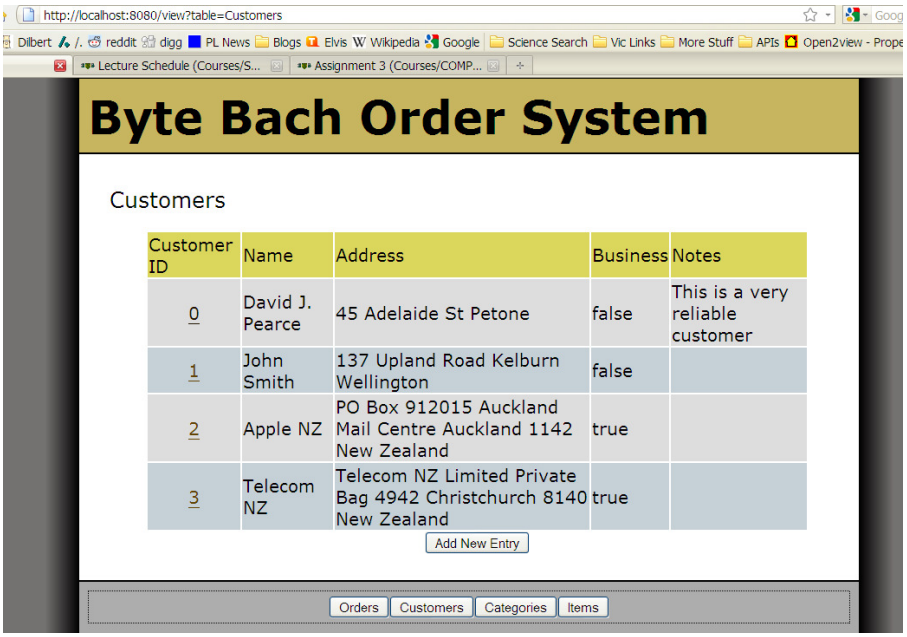
[1] If you are using Windows XP, it may ask you to unblock the webserver; you do not need to do that for this assignment

None of the buttons in the navigation bar is working properly at this stage. It is your work to make all of them working properly.

An initial database `sample.db` is provided for testing your system. You can populate it by setting the following run configuration:



After populating the initial database, when you click the `Customers` button, you should be able to see the following screen, as well as being able to create new orders, add/remove customers, etc.



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.

One of the challenges in this assignment lies in designing a suitable representation of objects to implement the functionality described in the `Database` interface. Figure 1 illustrates on possible approach, and it's worth making a few points about this:

- **Classes**. The suggested structure includes the classes `MyDatabase`, `MyTable` and `MyRow`. Here, `MyDatabase` implements `Database` and `MyTable` implements `Table`.
- **Back References**. The `MyTable` and `MyRow` classes each contain a *back reference*, also known as a *parent pointer*. For a `MyRow` instance, the back reference links to the `MyTable` instance

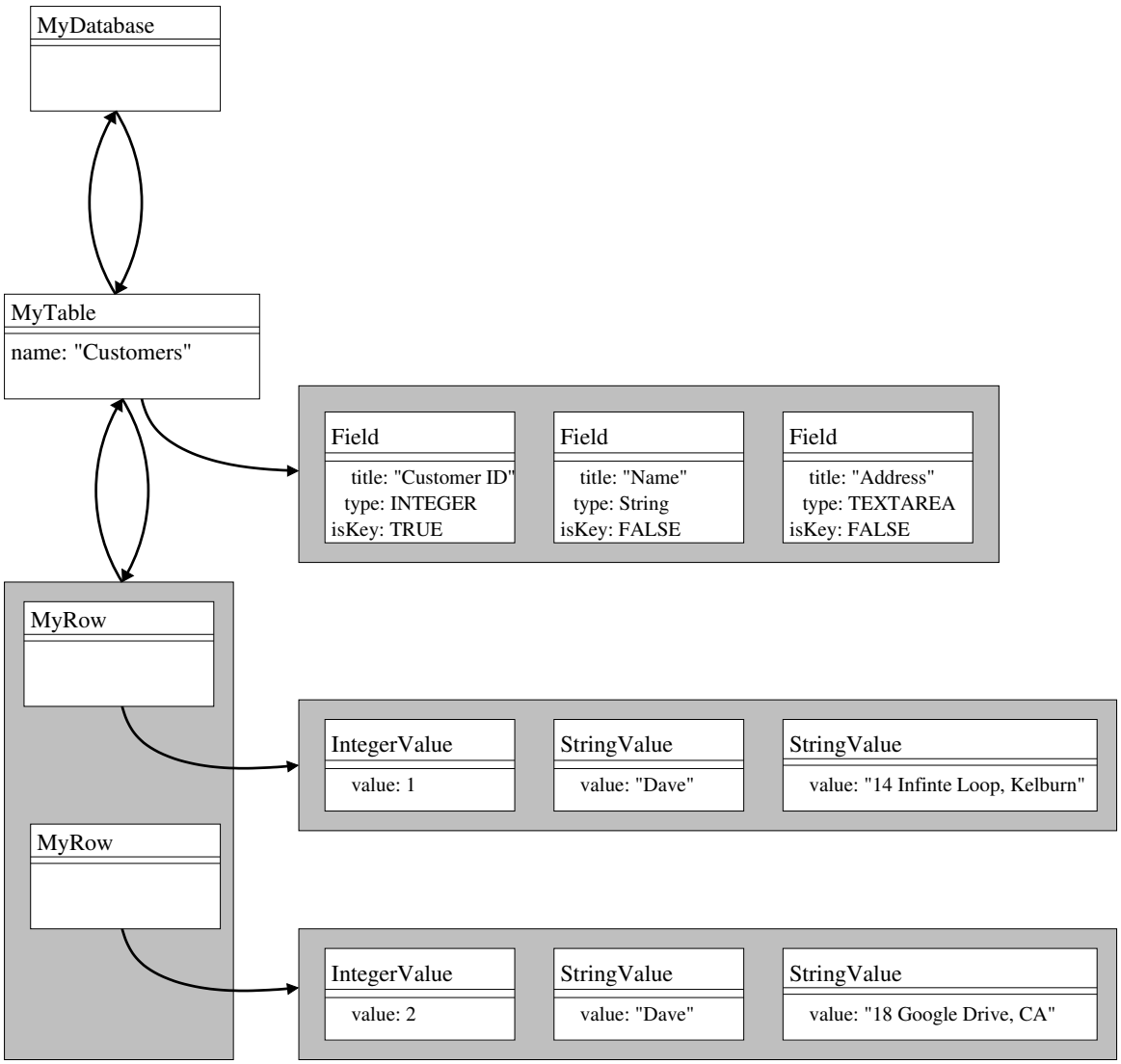


Figure 1: Illustrating the object layout for a database containing one table called "Customers". This table has three fields, and currently has rows of data within it. Observe that rows and tables have *back references* to their parent object.

containing that row. For a `MyTable` instance, the back reference links to the `MyDatabase` instance containing that table. The back references are needed so that these objects can access information in their parents. For example, a `MyRow` instance might wish to check how many fields are required for the given database, or which are key fields, etc.

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**. *Note, the jar file does not need to be executable.* See the following Eclipse tutorials for more on this:

   `http://ecs.victoria.ac.nz/Support/TechNoteEclipseTutorials`

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 testing mechanism supplied with the assignment remain unchanged.** Specifically, you cannot alter the way in which your code is tested as the marking script relies on this. However, this does not prohibit you from adding new tests. *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 your submission being reject by the submission system and/or zero marks being awarded.

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 SWEN 221 involves a qualitative mark for coding style. Coding style is an important aspect of coding since all code should be expected to be revisited again, i.e., it is important that you are someone else can make sense of the code it in the future and change it without accidentally introducing errors.

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

- **Division of Functionality into Classes**. This refers to how *cohesive* your classes are. That is, whether a given class is responsible for single specific task (has high cohesion), or for many unrelated tasks (has low cohesion). In particular, big classes with lots of weakly related 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 reusable methods (good) or implemented as one large monolithic method (bad).
- **Self-Explanatory Naming**. This refers to the choice of names for the classes, fields, methods and variables in your program. First, 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 code.
- **JavaDoc Comments**. This refers to the use of JavaDoc comments on classes, fields and methods. We expect all `public` and `protected` items to be properly documented. For example, when documenting a method, an appropriate description should be given, including despriptions for (if present) its parameters and return value. Good style dictates that `private` features are documented as well.
- **Code Comments**. This refers to the use of commenting within a given method. Comments should be used to elaborate the purpose of the code, rather than simply repeating what is evident from the code already.
- **Layout**. This refers to the consistent use of indentation and other layout conventions. Code must be properly indented and make consistent use of conventions e.g. for placing curly braces.

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