BMI Analysis System Phase 2: Design Document

Version 1.0

XYZZY Software

July 2018

Revision History

Version	Date	Author	Comments
1.0	04/07/18	D. Jarvis	Initial draft

Please email d.jarvis@cqu.edu.au with any corrections / requests for clarification.

Table of Contents

1. Introduction

XYZZY Software has been approached to develop a system to assist in the calculation and analysis of BMI (Body Mass Index) values. A phased implementation strategy was adopted. This document represents the current state of the design for the Phase 2 system. Note that the system must be developed using Java Technologies (NetBeans, Swing, Java DB) and it must conform to the specification provided in this document.

2. Requirements

Because of the simplicity of the user requirements, the corresponding use cases are not duplicated here, as would be the case in a normal XYZZY design document. The purpose of the system is to assist a health professional in the calculation and analysis of BMV values. A Java desktop application is to be developed, driven by a simple Swing GUI. Interaction between the database and the application is to be via JDBC. The NetBeans IDE is to be used for development. Java DB must be used as the database.

The Phase 1 requirements were as follows:

1. Start the application and connect to the database. If a connection cannot be established, the application must exit.
2. Close the database connection and stop the application
3. Display all records
4. Display all records where the BMI value is within a specified range
5. Calculate the BMI value and its corresponding category for all patients in the database and update all records.
6. The application must conform to the MVP (Model View Presenter) pattern.

It has been decided that in Phase 2, browsing of query results is to be supported using the MVC pattern. However, updating of records will still use the MVP pattern. Also, support is to be provided for a second BMI calculator. The Phase 2 requirements thus become:

1. Start the application and connect to the database. If a connection cannot be established, the application must exit.
2. Close the database connection and stop the application
3. Browse (cyclically) all records using MVC
4. Browse (cyclically all records where the BMI value is within a specified range using MVC
5. Calculate the BMI value and its corresponding category for all patients in the database and update all records using MVP.
6. Support the two methods of calculating BMI detailed below

The database design and sample data are provided in Section 4. Data validation is not required at this stage. However, basic preconditions must be satisfied for each requirement and if these are not satisfied, an appropriate message is to be displayed. These preconditions are specified in Section 8.

The standard method for calculating BMI values is as follows:

BMI = weight (in kgs) / (height (in metres))^2

It is now believed that a more realistic formula is the following:

BMI = 1.3*weight (in kgs) / (height (in metres))^2.5

(http://www.healthyweightforum.org/eng/calculators/new-bmi/) The categorisation remains the same for both the standard and new formulae:

Range	Category
0.0-18.5	Underweight
18.5-25.0	Normal Weight
25.0-30.0	Overweight
30.0-35.0	Obesity (Class 1)
35.0-40.0	Obesity (Class 2)
40.0-100.0	Morbid Obesity

Table 1. BMI Categories for Adults

Note that BMI is a floating point number. For ease of comparison in database queries, BMI10, which is an integer quantity, is stored in the database. BMI10 is defined as integer defined as follows:

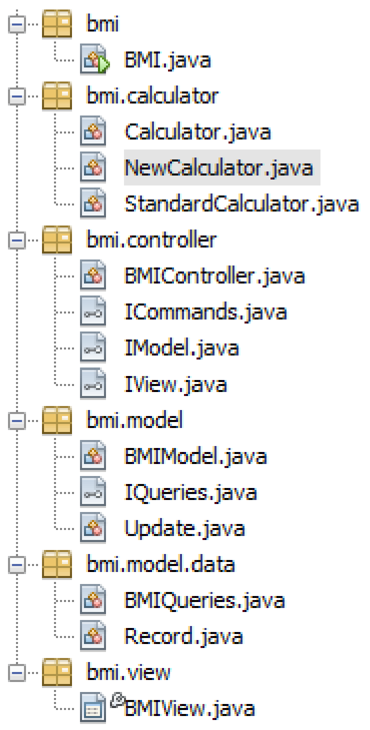
BMI10 = (int) ((BMI+0.05)*10.0) and

BMI = (double) ((BMI10/10 +(BMI10%10 / 10.0))

The user views and enters BMI values and not BMI10 values.

3. Architecture

A three-layered architecture combining both the MVP and MVC patterns, as indicated in Requirements 3-5, is to be employed in Phase 2. Given the simplicity of the architecture, an architecture diagram is not provided. Layers are to be modelled as packages – the package structure for the application is illustrated in Figure 1.



```
bmi
    BMI.java
bmi.calculator
    Calculator.java
    NewCalculator.java
    StandardCalculator.java
bmi.controller
    BMIController.java
    ICommands.java
    IModel.java
    IView.java
bmi.model
    BMIModel.java
    IQueries.java
    Update.java
bmi.model.data
    BMIQueries.java
    Record.java
bmi.view
    BMIView.java
```

Figure 1. Package structure

4. Database / Data Access Design

The SQL script that will be used to test the application is given below.

```
DROP TABLE RECORDS;

CREATE TABLE RECORDS

(

        SUBJECTID VARCHAR (8),

        HEIGHT REAL,

        WEIGHT REAL,

        BMI10 INT,

        BMICATEGORY VARCHAR (32),

        PRIMARY KEY (SUBJECTID)

);

INSERT INTO RECORDS(SUBJECTID,HEIGHT,WEIGHT,BMI10,BMICATEGORY)

        VALUES

        ('S01',1.8,70.0,0,'?'),

        ('S02',1.8,40.0,0,'?'),

        ('S03',1.8,100.0,0,'?'),

        ('S04',1.8,200.0,0,'?'),

        ('S05',1.8,81.0,0,'?'),

        ('S06',1.5,80.3,0,'?'),

        ('S07',1.5,59.9,0,'?'),

        ('S08',1.5,48.7,0,'?'),

        ('S09',1.5,48.3,0,'?'),

        ('S10',1.5,51.1,0,'?');
```

As there is only one table, an ERD is not provided.

Data access will be via JDBC using prepared statements. Normally, in an XYZZY design document, we would specify the queries in this section and relate them to the methods exposed by the BMIQueries class described in Section 6. However, for this project, we have decided to leave query formulation to the implementers.

5. GUI Design

Developers are free to use the NetBeans GUI Builder or alternatively, they can hand code the complete GUI. An indicative GUI is presented in Figure 2 and Table 2. Developers are free to use this design as is or to do things differently.

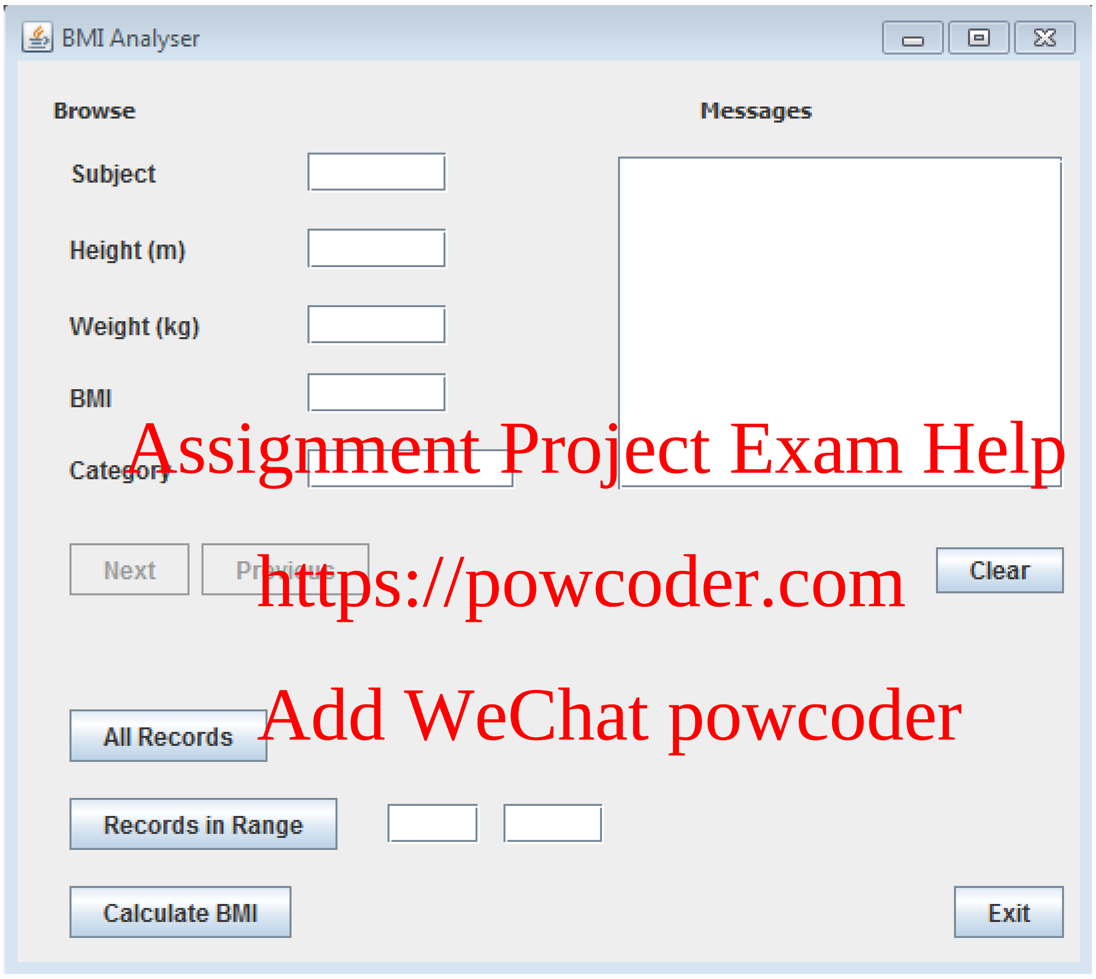
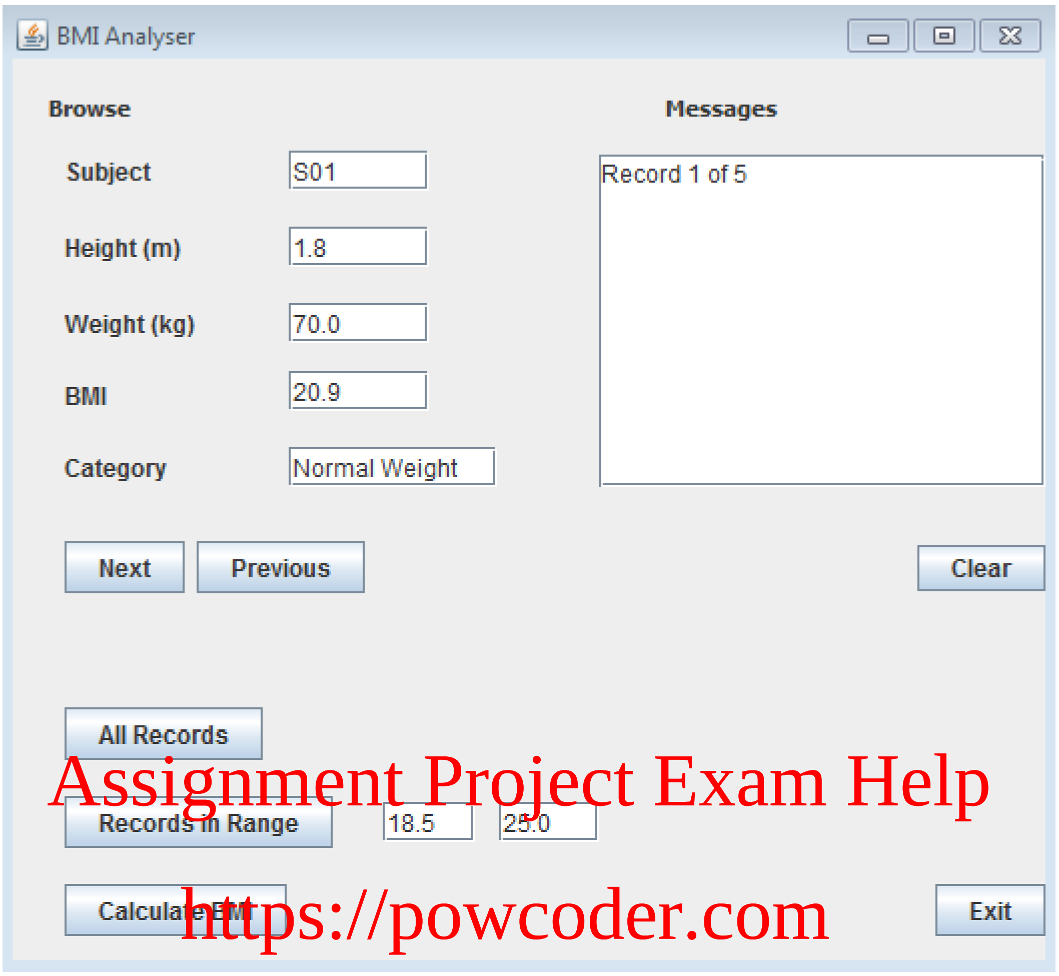


Figure 2. Indicative GUI

Functionality	Swing Components
Messages	JTextArea
Record output, Input	JLabel, JTextField
Operations	JButton

Table 2. Mapping of GUI functionality to Swing component types.

When outputting results from queries (ie All Records and Records in Range), use the browsing fields and messages area, as illustrated in Figure 3. Results from commands (ie Calculate BMI) just use the messages area.



Subject: S01
Height (m): 1.8
Weight (kg): 70.0
BMI: 20.9
Category: Normal Weight
Records in Range: 18.5 25.0

```
Record 1 of 5
```

Figure 3. Output after Calculate BMI and Records in Range actions have been performed.

Note that because of the simplicity of the GUI in Figure 2, screen shots for the realisation of each requirement are not shown as would normally occur in an XYZZY design document. Rather, the required actions are summarised in Table 3.

Requirement	Button	Inputs Required
1	n/a	n/a
2	Exit	None
3	All Records	None
4	Records in Range	Range of BMI values – see below
5	Calculate BMI	None
6	n/a	n/a

Table 3. Mapping of requirements to actions

For Requirement 4, note that clicking of the Records in Range button will result in the invocation of a single method.

If two numbers are specified as input, then they are to be treated as the lower bound (left) and upper bound (right) of an inclusive range. Having the upper bound less than the lower bound is to be treated as an error. Having both numbers the same is not an error – they are the endpoints of a range of length zero.

If one number is specified, then if it is the right input, then the left input is set to 0.0. If it is the left input that is specified, then the right input is set to the value of the left input.

6. Class Diagram

The class diagram for the application is illustrated in Figure 4.

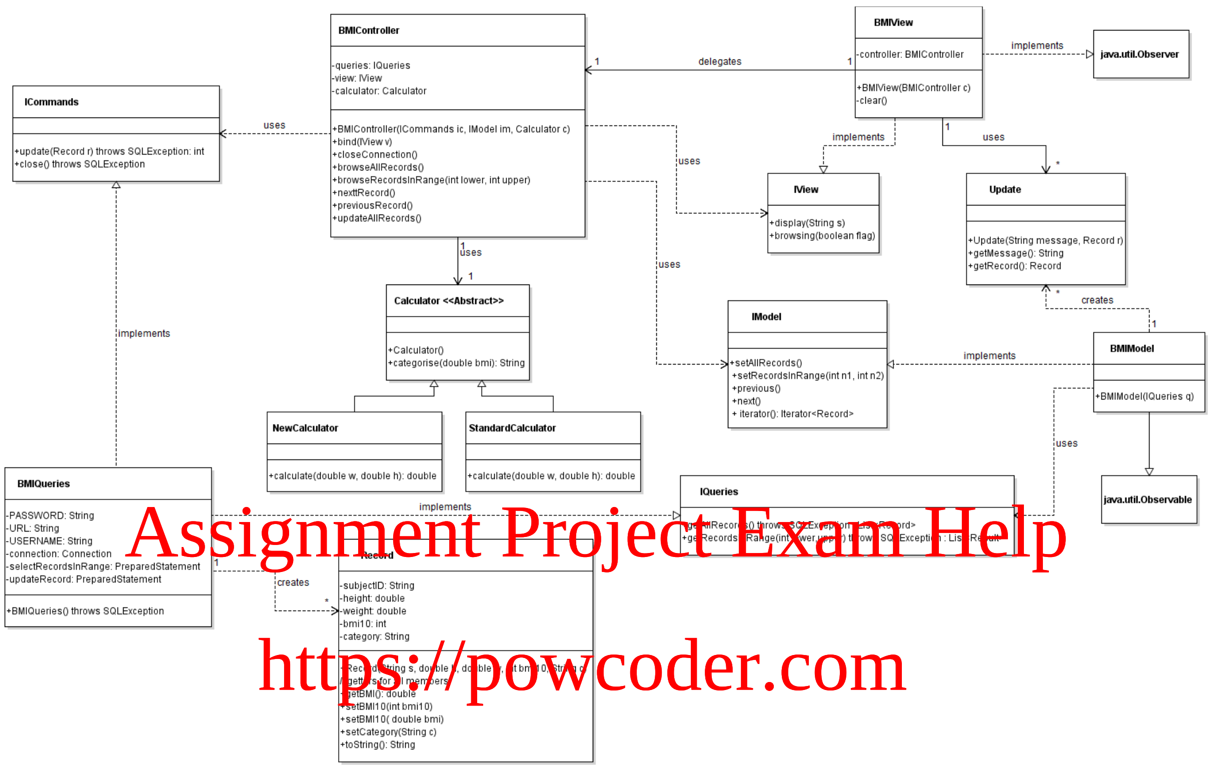


Figure 4: Class diagram

The BMI class and its create dependencies (BMIView, BMIController, BMIModel, BMIQueries and Calculator) are not shown in Figure 4 because of the increased complexity that they would have added to the diagram. Additionally, note that

1. The methods in the IQueries interface from Phase 1 is now partitioned into 2 interfaces – IQueries and ICommands. BMIQueries implements both interfaces. IQueries is used by BMIModel; ICommands by BMIController.
2. The IModel interface (implemented by BMIModel and used by BMIController) defines a method java.util.Iterator <Record> iterator(), which unlike the other methods in the interface, returns a value. The reason for this is that the other methods are there to support MVC enabled browsing, whereas iterator() is needed to support BMI calculation and record updating. In this regard, iterator() will need to execute BMIQueries.getAllRecords() and then invoke iterator() on the returned List. The resulting Iterator object is then returned.
3. Updating of the browsing fields and the messages area is achieved by the creation of Update objects in BMIModel. An Update object is then passed to notifyObservers(), which then invokes BMIView.update(), passing the object as one its two parameters.

The mapping to packages/layers is provided in Section 3. Note that Figure 4 represents a refactoring of the class structure favoured by the NetBeans GUI Builder. Rather than having GUI creation in the

GUI class definition (through the provision of a main() method, we have chosen to place GUI creation in a separate main class. If you are using the NetBeans GUI builder, all that this means is that the main() method that the Builder generates is moved into the BMI class. Also note that interaction between the BMIQueries class and the JBC library classes is not shown, as is normal for class diagrams. Finally, formal UML syntax is not strictly followed. In particular, we feel felt that method signatures as employed in NetBeans are clearer than their UML counterparts.

In terms of implementation, note carefully where BMI10 (integer) values are used. As noted earlier, the user views and enters BMI (double) values with one decimal point of precision. With the Calculator class, no specific implementation approach is specified, but feel free to use an enum.

7. Sequence Diagrams

Omitted.

8. Test Plan

The following tests will be performed prior to acceptance:

Requirement	**Test**	**Comment**
1	Operation performs correctly given correct preconditions	
1	Handles no database/incorrect database	Application is to exit
2	Operation performs correctly	
3	Operation performs correctly	Code inspection for MVC
4	Operation performs correctly given correct preconditions	Code inspection for MVC. No data validation checks required.
4	Handles incomplete field entry	
4	Input variants as specified in Section 5 are handled correctly	
5	Operation performs correctly	Code inspection for MVP
6	Polymorphism used	Code inspection.

Table 3. Acceptance tests

Note:

1. By preconditions, we mean that the required values are available.
2. Acceptance tests coincide with class unit tests, so no separate unit testing is required.
3. Testing will be conducted with the database specified in Section 4.

Acknowledgements

The UML diagrams were created using Violet (http://horstmann.com/violet/)