

Inventory Management System Project

By Kiera Walton

Project Outline

The Goal: build an application that an end user can interact with via a command line interface, using a JDBC connection to a GCP My-SQL instance

The application: Inventory Management System

Functionality of application: users can add, update, read details of, and delete customers, items and orders from the system



The Process - Continuous Integration and Technologies Used

Source Code - Java

Build Tool - Maven

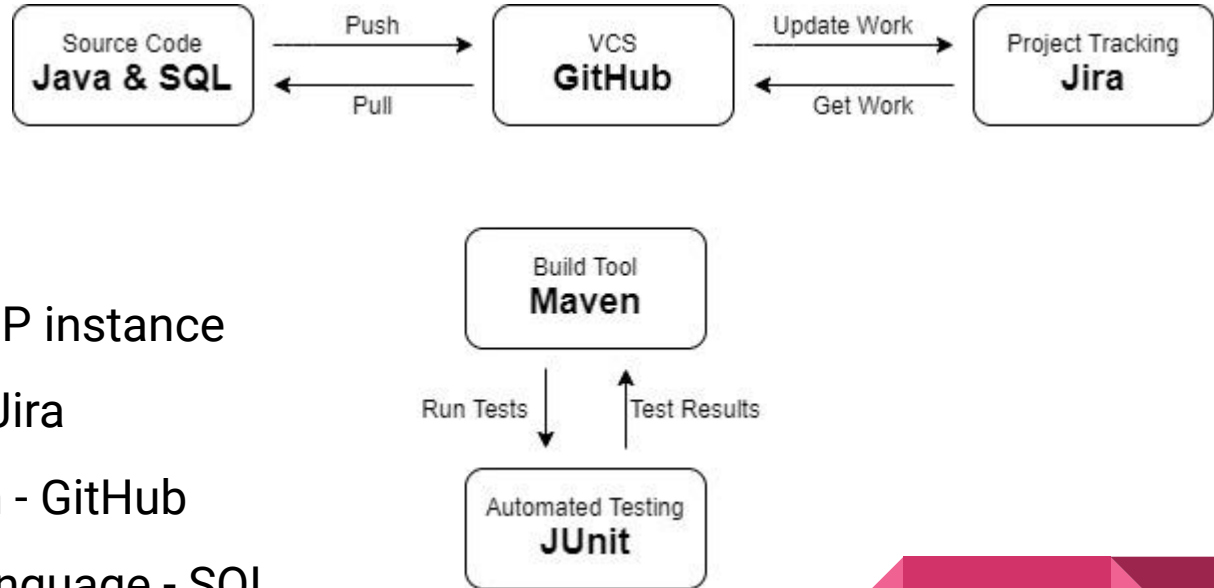
Testing - JUnit

JDBC connection to GCP instance

Project Management - Jira

Version Control System - GitHub

Relational Database Language - SQL



Project Management - Jira

‘As a user, I want to be able to create, read, update and delete items from the database so that I can manage the items that are in the system’

Projects / Inventory Management System / IMS board

Kanban board

Search: KW Only My Issues Recently Updated

BACKLOG 3

- Connect to GCP (instead of using MySQL workbench) IMS-13
- UML diagram IMS-14
- SonarQube IMS-18

SELECTED FOR DEVELOPMENT 3

- User Story 4 (As a user, I should be able to enter an item quantity and item ID to my order, and retrieve a total item price in return, so that I do) IMS-4
- User Story 5 (As a user, I should be able to interact with the application via a command line interface, so that I can easily put entries into the) IMS-5
- Presentation IMS-12

IN PROGRESS 2

- Order Testing IMS-10
- Complete ReadMe file IMS-11

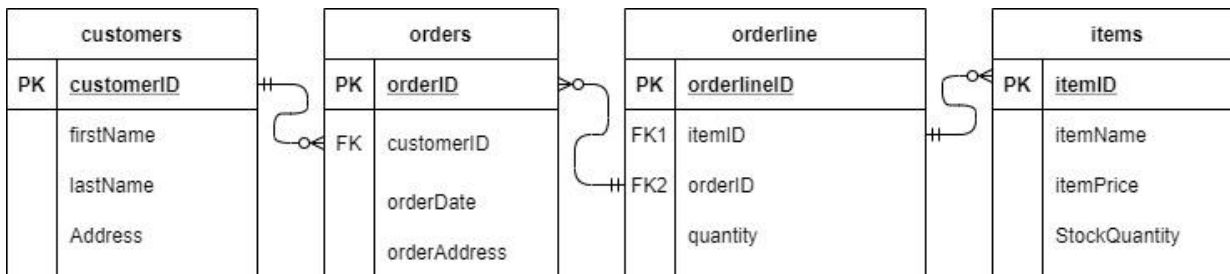
DONE 10

- User Story 1 (As a user, I want to be able to create, read, update and delete items from the database so that I can manage the items that are in the system) IMS-1
- User Story 2 (As a user, I want to be able to create, read, update and delete items from the item table in the database, so that I can have) IMS-2
- User Story 3 (As a user, I want to be able to create, read update and delete orders from the database, so that I can manage the orders that) IMS-3
- Complete ERD for SQL tables IMS-6
- Complete Risk Assessment IMS-7
- Customer Testing IMS-8

User Stories 4 and 5 can be selected for development because they satisfy ‘definition of ready’ criteria

Relational Database - SQL

ERD:



DDL:

```
1 create database if not exists ims;
2 create table if not exists ims.customers(id int unique primary key auto_increment, first_name varchar(40), surname varchar(40));
3 create table if not exists ims.items(itemID int unique primary key auto_increment, itemName varchar(40), itemPrice varchar(40));
4 create table if not exists ims.orders(orderID int unique primary key auto_increment, customerID int, totalPrice varchar(10), foreign key(customerID) references ims.customers(id));
5 create table if not exists ims.orderline(orderlineID int unique primary key auto_increment, itemID int, orderID int, quantity int, foreign key(itemID) references ims.items(itemID), foreign key(orderID) references ims.orders(orderID));
```

DML:

Creating statements
to execute SQL
commands using a
JDBC connection

```
@Override
public Item update(Item item) {
    try (Connection connection = DriverManager.getConnection(jdbcConnectionUrl, username, password);
        Statement statement = connection.createStatement();) {
        statement.executeUpdate("UPDATE items SET itemName = '" + item.getItemName()
            + "', itemPrice = '" + item.getItemPrice() + "' WHERE itemID = '" + item.getItemID() + "'");
        return readItem(item.getItemID());
    } catch (Exception e) {
        LOGGER.debug(e.getStackTrace());
        LOGGER.error(e.getStackTrace());
    }
    return null;
}
```

Connecting To A GCP Instance

```
init("jdbc:mysql://35.234.155.64:3306/ims",
```

These tables were generated on my GCP instance - they correspond to the DDL sql schema file in my maven project and demonstrate the success of this file

```
mysql> show tables;
+-----+
| Tables_in_ims |
+-----+
| customers      |
| items          |
| orderline      |
| orders         |
+-----+
4 rows in set (0.01
```

```
mysql> describe items;
```

Field	Type	Null	Key	Default	Extra
itemID	int(11)	NO	PRI	NULL	auto_increment
itemName	varchar(40)	YES		NULL	
itemPrice	varchar(40)	YES		NULL	

```
mysql> describe orderline;
```

Field	Type	Null	Key	Default	Extra
orderlineID	int(11)	NO	PRI	NULL	auto_increment
orderID	int(11)	YES	MUL	NULL	
itemID	int(11)	YES	MUL	NULL	
quantity	int(11)	YES		NULL	

VCS - Github

Committing



Utilising the Feature-Branch Model: I integrated code continuously throughout the development process through github, branching off a dev branch into feature branches corresponding to user stories and PBIs

Resolving Merge Conflicts

```
admin@DESKTOP-0DE92GB MINGW64 ~/Desktop/ims-project/ims-demo (orderfeature)
$ git merge orderfeature dev
Auto-merging src/main/java/com/qa/ims/persistence/domain/Custom
CONFLICT (content): Merge conflict in src/main/java/com/qa/ims/persistence/domain/Custom.java
Auto-merging src/main/java/com/qa/ims/persistence/dao/ItemDaoMysql.java
Automatic merge failed; fix conflicts and then commit the result.

admin@DESKTOP-0DE92GB MINGW64 ~/Desktop/ims-project/ims-demo (orderfeature|MERGE
NG)
$ git add .

admin@DESKTOP-0DE92GB MINGW64 ~/Desktop/ims-project/ims-demo (orderfeature|MERGE
NG)
$ git add .

admin@DESKTOP-0DE92GB MINGW64 ~/Desktop/ims-project/ims-demo (orderfeature|MERGE
NG)
$ git commit -m "fixing merge conflicts"
[orderfeature c2eb1c7] fixing merge conflicts

admin@DESKTOP-0DE92GB MINGW64 ~/Desktop/ims-project/ims-demo (orderfeature)
$ git checkout dev
Switched to branch 'dev'
```

Branching

```
admin@DESKTOP-0DE92GB MINGW64 ~
$ git branch
* dev
  itemfeature
  itemtesting
  master
  orderfeature
  ordertesting
  updateorderfeature
```

Resetting a Commit

```
admin@DESKTOP-0DE92GB MINGW64 ~/Desktop/ims
$ git reset --hard HEAD~1
HEAD is now at 9b70cc1 order and orderline
d manually
```

Build Tool - Maven

A Build Automation tool that manages the project's build; it addresses how the software is built and the dependencies it has on other projects

Pom.xml is a file that describes the software that is being built, it's dependencies on other modules, plugins it requires etc

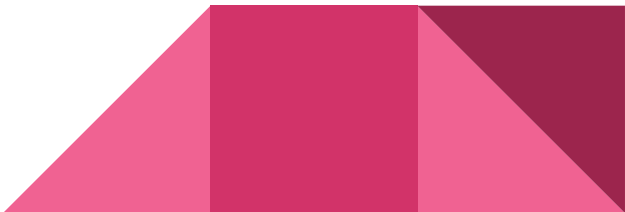
Example: dependencies required for testing shown in the pom.xml

```
<!-- https://mvnrepository.com/artifact/org.mockito/mockito-core -->
<dependency>
  <groupId>org.mockito</groupId>
  <artifactId>mockito-core</artifactId>
  <version>3.2.4</version>
  <scope>test</scope>
</dependency>

<!-- https://mvnrepository.com/artifact/junit/junit -->
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.12</version>
  <scope>test</scope>
</dependency>
```


Source Code - Java

Some of the main concepts and principles within java which I used throughout are as follows:

- OOP - inheritance, encapsulation, abstraction, polymorphism
 - Access Modifiers and Constructors
 - Logging
 - Generics
 - Scanners
 - Try Catch Blocks and Exceptions
 - Casting
 - Enums
 - SOLID principles
- 

Focus: User Story

As a user, I want to create, read, update and delete orders from the system, so that I can manage and track the orders that are in the database



Tracking Workflow - Orders and Orderline

```
@Override
public Order create() {
    LOGGER.info("enter the customerID for the order you wish to create");
    Long customerID = Long.valueOf(getInput());
    LOGGER.info("enter the total price of your order");
    Double totalPrice = Double.valueOf(getInput());
    Order order = orderService.create(customerID, totalPrice);
    LOGGER.info("order created, ID: " + order.getId());
}
```

Two different controller classes, one for orders and one for orderline, both implementing crud controllers for their corresponding domain (order/orderline)

Problem: goes against MVP - too many features on the system...users of the system should be able to enter all order details in one place

```
@Override
public Orderline create() {
    LOGGER.info("please enter the orderID");
    Long orderID = Long.valueOf(getInput());
    LOGGER.info("please enter the itemID");
    Long itemID = Long.valueOf(getInput());
    LOGGER.info("please enter the quantity of the item in the order");
    int quantity = Integer.parseInt(getInput());
    Orderline orderline = orderlineService.create(new Orderline(orderID, itemID, quantity));
    LOGGER.info("orderline record created");
    return orderline;
}
```

Tracking Workflow - Integrating Order and Orderline, Part 1

Using 2 Attributes in my Order Controller

```
public class OrderController implements CrudController<Order> {  
  
    public static final Logger LOGGER = Logger.getLogger(OrderCon  
  
    private CrudServices<Order> orderService;  
    private CrudServices<Orderline> orderlineService;
```

```
@Override  
public Order create() {  
    LOGGER.info("enter the customerID for the order you wish to create");  
    Long customerID = Long.valueOf(getInput());  
    LOGGER.info("enter the total price of your order");  
    Double totalPrice = Double.valueOf(getInput());  
    Order order = orderService.create(new Order(customerID, totalPrice));  
    LOGGER.info("order created, with orderID: " + order.getOrderID());  
  
    String answer = "yes";  
    while (answer.equalsIgnoreCase("yes")) {  
        LOGGER.info("enter the itemID of the item you wish to add");  
        Long itemID = Long.valueOf(getInput());  
        LOGGER.info("enter the quantity of this item you would like to add");  
        Integer quantity = Integer.parseInt(getInput());  
        Orderline orderline = orderlineService.create(new Orderline(order.getOrderID(), itemID, quantity));  
        LOGGER.info("enter yes to add more items, or enter 'no' to finish order");  
        answer = getInput();  
    }  
    LOGGER.info("order created");  
    return order;
```

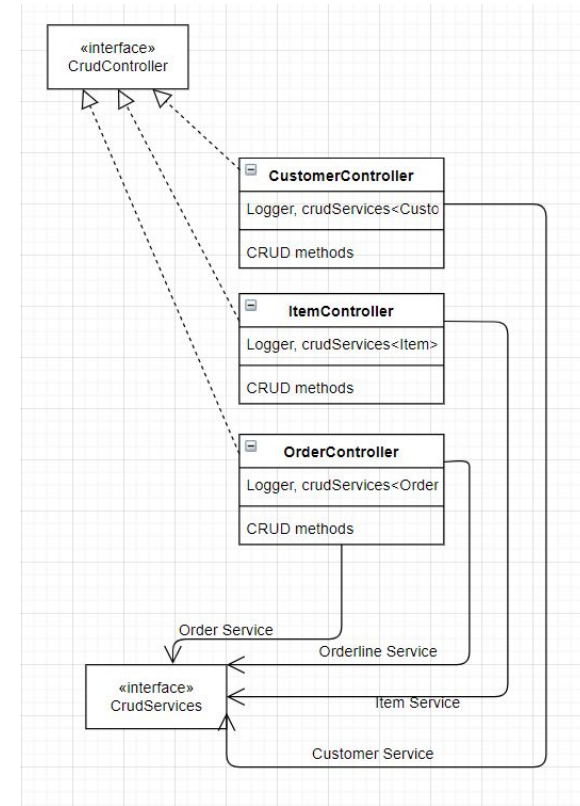
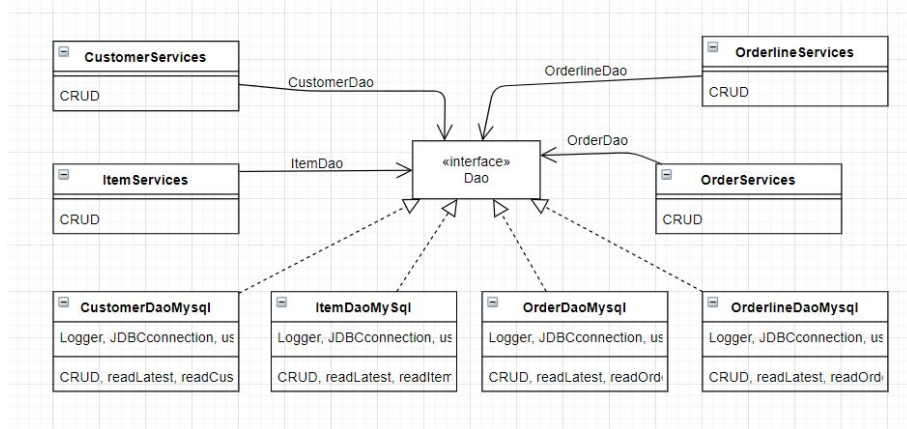
Order create method from order services

```
@Override  
public Order create(Order order) {  
    return orderDao.create(order);  
}
```

The orderline create method from
orderline services

```
@Override  
public Orderline create(Orderline orderline) {  
    return orderlineDao.create(orderline);  
}
```

Class Relationships



Tracking Workflow - Integrating orders and orderline, part 2

I attempted to integrate orders and orderlines fully, so that I would no longer need the orderline classes - this is the process I followed

1: New Interfaces

```
public interface CrudOrderController<T> {  
    List<T> readAllOrders();  
    List<T> readAllOrderlines();  
    T create();  
    T addToOrder();  
    T removeFromOrder();  
    T changeQuantityofItemInOrder();  
    void delete();  
}  
  
public interface DaoOrderline<T> {  
    List<T> readAllOrders();  
    List<T> readAllOrderlines();  
    T create(T t);  
    T addToOrder(T t);  
    void removeFromOrder(long id);  
    T changeQuantityofItemInOrder(T t);  
    void delete(long id);  
}
```

2: Implementing the New Interfaces

```
public class OrderController implements CrudOrderController<Order> {  
  
    public class OrderDaoMysql implements DaoOrderline<Order> {
```

3: New Enum Class for Order Actions

```
public enum ActionsForOrders {  
    CREATE("To create an order"), READORDERS("To read orders"),  
    ADD("To add to an order"), REMOVE("To remove from an order")  
  
    public static final Logger LOGGER = Logger.getLogger(ActionsForOrders.class);  
    private String descriptionOfAction;  
  
    private ActionsForOrders(String descriptionOfAction) {  
        this.descriptionOfAction = descriptionOfAction;  
    }  
  
    // describing t  
    public void doActionForOrders(CrudOrderController<Order> crudOrderController, String orderAction) {  
        switch (orderAction) {  
            case CREATE:  
                crudOrderController.create();  
                break;  
            case READORDERS:  
                crudOrderController.readAllOrders();  
                break;  
            case READORDERLINES:  
                crudOrderController.readAllOrderlines();  
                break;  
            case ADD:  
                crudOrderController.addToOrder();  
                break;  
            case REMOVE:  
                crudOrderController.removeFromOrder();  
                break;  
        }  
    }  
}
```

```
case ORDER:  
    OrderController orderController = new OrderController(  
        new OrderServices(new OrderDaoMysql(username, password)));  
    doActionForOrders(orderController, orderAction);  
    break;
```


Testing - JUnit

- Unit Tests were used to test this application
- Individual units of source code are tested to validate whether code performs as expected

Element	Coverage
▼ 📁 src/main/java	64.8 %
> 📁 com.qa.ims.persistence.d	65.3 %
> 📁 com.qa.ims.controller	43.8 %
> 📁 com.qa.ims	15.6 %
> 📁 com.qa.ims.persistence.d	94.5 %
> 📁 com.qa.ims.utils	0.0 %
> 📁 com.qa.ims.services	100.0 %

```
@Test
public void equalsWithNull() {
    assertFalse(item.equals(null));
}

@Test
public void createItemWithItemID() {
    assertEquals(1L, item.getItemID(), 0);
    assertEquals("candle", item.getItemName());
    assertEquals(5.0, item.getItemPrice(), 0);
}

@Test
public void equalsWithDifferentObjects() {
    assertFalse(item.equals(new Object()));
}

@Test
public void checkEquality() {
    assertTrue(item.equals(item));
}
```

Note: low % on controller was a time management issue

Testing - Using Mockito

Allows for the creation of test double objects - mock objects. Used to mock interfaces so that a dummy functionality can be added to a mock interface that can be used in unit testing

Using a second parameter in verify method

```
@Test
public void deleteTest() {
    // doesn't return anything - just verify that it calls the c
    String itemID = "1";
    Mockito.doReturn(itemID).when(itemController).getInput();
    itemController.delete();
    Mockito.verify(itemServices, Mockito.times(1)).delete(1L);
}
```

Creating mock objects and testing them

```
@RunWith(MockitoJUnitRunner.class)
public class ItemControllerTest {

    @Mock
    private ItemServices itemServices;

    @Spy
    @InjectMocks
    private ItemController itemController;

    @Test
    public void readAllTest() {
        ItemController itemController = new ItemController(itemServices);
        List<Item> items = new ArrayList<>();
        items.add(new Item("candle", 5.0));
        items.add(new Item("plant", 12.0));
        items.add(new Item("chair", 30.0));
        items.add(new Item("table", 50.0));
        Mockito.when(itemServices.readAll()).thenReturn(items);
        assertEquals(items, itemController.readAll());
    }
}
```

Using the created mock object & giving it a reference variable

Then calls the method as our actual value, giving it items as our expected

Defining the functionality - when the method is called, then return items

The Final Product: A Live Demo



Sprint Review and Retrospective

What could be Improved?

Test coverage at 80%

Fully integrate orders and orderlines

- I was very close, but lack of time meant I could not get there

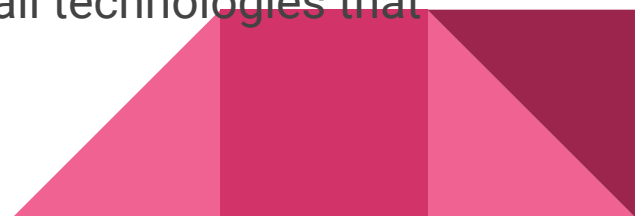
Total price calculator - have a method that calculates total price so that users don't have to input it themselves

What went well?

My java code developed significantly from the start of the development process to the end

Concept of MVP from the users perspective is adhered to

Successfully applied newly learned knowledge of all technologies that were specified



Any Questions?

