**SSE 659 Project 2**

Refactoring the Deli App

Mike Dumas, Kei’Shawn Tention, Jewl Johnson

**Abstract**

Demonstrate knowledge of topics covered in Chapters 3-12 of the text Refactoring – Improving the Design of Existing Code. Each team member will submit his or her own report.

Kei’Shawn Tention’s Submission

March 23, 2015

Table of Contents

1. Introduction 3

2. Development Tools 3

2.1 Eclipse 3

2.2 GitHub 3

3. What is Refactoring? 3

3.1 What are we Refactoring? 3

4. TDD in Conjunction with Refactoring 4

4.1 Deli App Tests 4

5. Chapter 7 – Moving Features Between Objects 6

5.1 Overview 6

5.2 Implementation 6

6. Chapter 8 – Organizing Data 6

6.1 Overview 6

6.2 Code Clean Up 6

6.3 Implementation 8

Beef.java Class Diagram 9

Ham.java Class Diagram 9

Chicken.java Class Diagram 9

7. Conclusion 9

8. Appendix 9

8.1 Test Results 9

8.2 Source Code 9

8.2.1 Beef.java Class file 9

8.2.2 Ham.java Class file 9

8.2.3 Chicken.java Class file 9

8.2.4 Complete and Original testing.java Class file 10

8.2.5 Original loadProducts() method 18

8.3 Non-Direct Log 21

# 1. Introduction

The purpose of Project 2 is to demonstrate one’s knowledge of the topics covered in Chapters 3 – 12 of Refactoring: Improving the Design of Existing Code, written by Martin Fowler.

# 2. Development Tools

## 2.1 Eclipse

## 2.2 GitHub

# 3. What is Refactoring?

## 3.1 What are we Refactoring?

The Deli Training application is a rather complex application. The classes that make up the application are as followed:

* *main.java*
* *Products.java*
* *Results.java*
* *testing.java*
* *GameEngine.java*

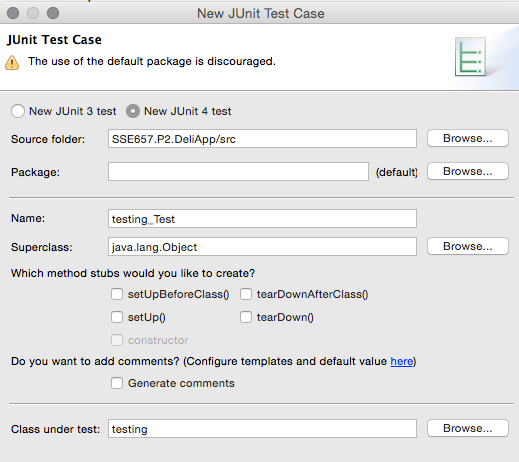
For the most part, the classes drive the GUI aspect of the Deli Application. The source code for each class be viewed in [Appendix 8.2](#_8.2_Source_Code). The two main classes that control the functionality of the Deli Application are *testing.java* and *GameEngine.java*.

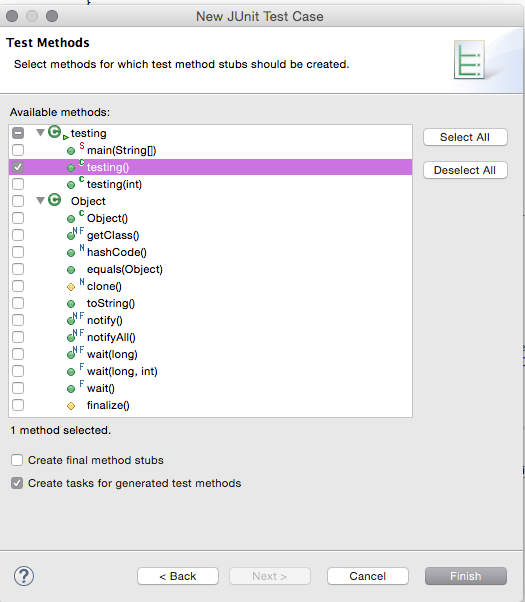
After thoroughly examining each class file, we noticed that the *testing.java* and *GameEngine.java* are the classes that refactoring could be applied to. In *testing.java*, we noticed that the method *loadProducts()*  was rather lengthy, which was an indication that refactoring was necessary. Also within that method, we recognized duplicate code. In *GameEngine.java*, we recognized the excessive use of conditional statements and concluded that refactoring can be performed to simplify the code.

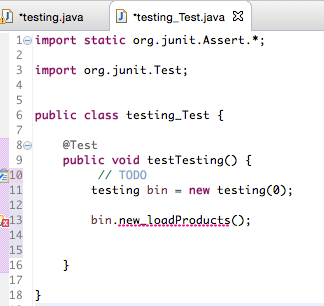
Each member of the team had their own particular area of concern to refactor, hence the separate reports. One are I want to address and refactor pertains to the Organizing Data, which is covered in Chapter 8 of Martin Fowler’s text, Refactoring: Improving the Design of Existing Code. The second area of concern is Moving Features Between Objects, which is covered in Chapter 7 of the same text. [Section 5](#_5._Chapter_7) and [Section 6](#_6._Chapter_8) will discuss my knowledge and implementation of Chapters 7 and 8, respectively.

# 4. TDD in Conjunction with Refactoring

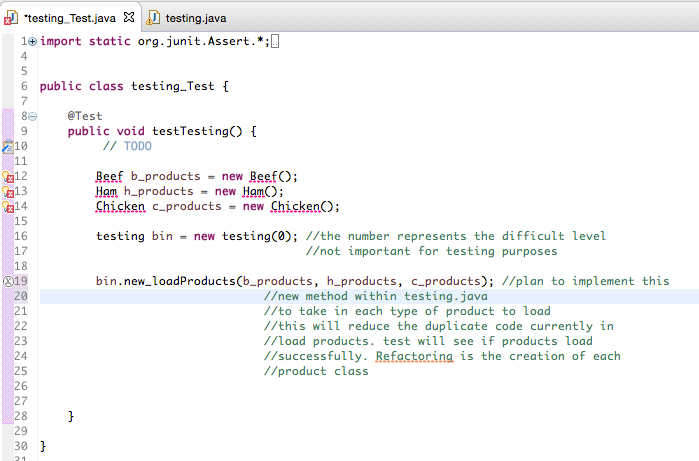
## 4.1 Deli App Tests







***CURRENT CODE AS OF 3/23/15 3:40PM.***

******

# 5. Chapter 7 – Moving Features Between Objects

## 5.1 Overview

## 5.2 Implementation

# 6. Chapter 8 – Organizing Data

## 6.1 Overview

## 6.2 Code Clean Up

Before implementing the concepts behind data organization, I notice that the *testing.java* class contained unnecessary lines of code. It seems that the lines of code were initially used to do validation within the class file. The side-by-side view of the code refactoring is show below.

|  |  |
| --- | --- |
| Original *testing.java* | Modified *testing.java* |
| **…..**  **…..**  **…..**  **public** **class** testing {  **protected** JFrame frmDeliTrainingApplication;  Random generator = **new** Random(System.*currentTimeMillis*());  **int** max = 0, rand\_product = 0, rand\_product2 = 0, rand\_option = 0, score = 0, questions = 0, difficulty = 0;  String name = "", name2 = "", type = "", type2 = "";  GameEngine Engine = **new** GameEngine();  Results resultsWindow;  /\*\*  \* Launch the application.  \*/  **public** **static** **void** main(String[] args) {  EventQueue.*invokeLater*(**new** Runnable() {  **public** **void** run() {  **try** {  testing window = **new** testing();  window.frmDeliTrainingApplication.setExtendedState(JFrame.***MAXIMIZED\_BOTH***);  window.frmDeliTrainingApplication.setVisible(**true**);  } **catch** (Exception e) {  e.printStackTrace();  }  }  });  }  /\*\*  \* Create the application.  \*/  **public** testing() {  loadProducts();  pickProducts();  initializeTesting();  }  **public** testing(**int** difficulty){ //Constructor passing in level of difficulty  **this**.difficulty = difficulty;  loadProducts();  pickProducts();  initializeTesting();  }  **private** **void** loadProducts() {  …..  …..  ….. | **…..**  **…..**  **…..**  **public** **class** testing {  **protected** JFrame frmDeliTrainingApplication;  Random generator = **new** Random(System.*currentTimeMillis*());  **int** max = 0, rand\_product = 0, rand\_product2 = 0, rand\_option = 0, score = 0, questions = 0, difficulty = 0;  String name = "", name2 = "", type = "", type2 = "";  GameEngine Engine = **new** GameEngine();  Results resultsWindow;  **public** testing(**int** difficulty){ //Constructor passing in level of difficulty  **this**.difficulty = difficulty;  loadProducts();  pickProducts();  initializeTesting();  }  **private** **void** loadProducts() { …..  ….. |

## 6.3 Implementation

The goal of this project is refactor the *testing.java* and *GameEngine.java* classes. The Deli App is full functional as is, however we found many areas that could be improved. In order to limit the amount of duplicate code and organize data better, I decided to created individual product classes. This will allow the *loadProducts()* method in *testing.java* to be a more flexible and intuitive block of code. For the purpose of this project and skill demonstration, I choose 3 products to refactor. They are the *beef product*, *ham product*, and the *chicken product.*  The complete and original *loadProducts()* can be viewed in [Appendix 8.2.5](#_8.2.5_Original_loadProducts())

Below are the class diagrams for each product, respectively.

### Beef.java Class Diagram

See [Appendix 8.2.1](#_8.2.1_Beef.java_Class) for complete source code.

### Ham.java Class Diagram

See [Appendix 8.2.2](#_8.2.2_Ham.java_Class) for complete source code.

### Chicken.java Class Diagram

See [Appendix 8.2.3](#_8.2.3_Chicken.java_Class) for complete source code.

# 7. Conclusion

# 8. Appendix

## 8.1 Test Results

## 8.2 Source Code

### 8.2.1 Beef.java Class file

### 8.2.2 Ham.java Class file

### 8.2.3 Chicken.java Class file

### 8.2.4 Complete and Original testing.java Class file

**import** java.awt.EventQueue;

**import** javax.print.DocFlavor.URL;

**import** javax.swing.JFrame;

**import** javax.swing.JTabbedPane;

**import** javax.swing.JScrollPane;

**import** javax.swing.JDesktopPane;

**import** javax.swing.JLabel;

**import** javax.swing.ImageIcon;

**import** javax.swing.JFormattedTextField;

**import** javax.swing.JButton;

**import** java.awt.event.ActionListener;

**import** java.awt.event.ActionEvent;

**import** java.io.File;

**import** java.security.CodeSource;

**import** java.util.ArrayList;

**import** java.util.HashMap;

**import** java.util.List;

**import** java.util.Map;

**import** java.util.Random;

**import** java.util.zip.ZipInputStream;

**import** java.awt.event.ComponentAdapter;

**import** java.awt.event.ComponentEvent;

**import** java.awt.event.MouseAdapter;

**import** java.awt.event.MouseEvent;

**import** java.awt.Color;

**import** javax.swing.UIManager;

**import** java.awt.SystemColor;

**import** java.awt.Font;

**public** **class** testing {

**protected** JFrame frmDeliTrainingApplication;

Random generator = **new** Random(System.*currentTimeMillis*());

**int** max = 0, rand\_product = 0, rand\_product2 = 0, rand\_option = 0, score = 0, questions = 0, difficulty = 0;

String name = "", name2 = "", type = "", type2 = "";

GameEngine Engine = **new** GameEngine();

Results resultsWindow;

/\*\*

\* Launch the application.

\*/

**public** **static** **void** main(String[] args) {

EventQueue.*invokeLater*(**new** Runnable() {

**public** **void** run() {

**try** {

testing window = **new** testing();

window.frmDeliTrainingApplication.setExtendedState(JFrame.***MAXIMIZED\_BOTH***);

window.frmDeliTrainingApplication.setVisible(**true**);

} **catch** (Exception e) {

e.printStackTrace();

}

}

});

}

/\*\*

\* Create the application.

\*/

**public** testing() {

loadProducts();

pickProducts();

initializeTesting();

}

**public** testing(**int** difficulty){ //Constructor passing in level of difficulty

**this**.difficulty = difficulty;

loadProducts();

pickProducts();

initializeTesting();

}

**private** **void** loadProducts() { //Convert All Censored Beef Pictures into Beef Product Objects Objects

File Folder = **new** File( testing.**class**.getResource("/ProductImages/CensoredImages/Meats/Beef/").getPath() );

File [] beefs = Folder.listFiles();

**for**(**int** index = 0; index < beefs.length; index++) {

Map<String, String> properties = **new** HashMap<String, String>();

name = beefs[index].getName();

properties.put("name", name);

properties.put("type", "Beef");

Engine.getProducts().add(**new** Product(properties));

}

//Convert All Censored Bologna Pictures

Folder = **new** File( testing.**class**.getResource("/ProductImages/CensoredImages/Meats/Bologna/").getPath() );

File [] bolognas = Folder.listFiles();

**for**(**int** index = 0; index < bolognas.length; index++) {

Map<String, String> properties = **new** HashMap<String, String>();

name = bolognas[index].getName();

properties.put("name", name);

properties.put("type", "Bologna");

Engine.getProducts().add(**new** Product(properties));

}

//Convert All Censored Chicken Pictures

Folder = **new** File( testing.**class**.getResource("/ProductImages/CensoredImages/Meats/Chicken/").getPath() );

File [] chickens = Folder.listFiles();

**for**(**int** index = 0; index < chickens.length; index++) {

Map<String, String> properties = **new** HashMap<String, String>();

name = chickens[index].getName();

properties.put("name", name);

properties.put("type", "Chicken");

Engine.getProducts().add(**new** Product(properties));

}

//Convert All Censored Ham Pictures

Folder = **new** File( testing.**class**.getResource("/ProductImages/CensoredImages/Meats/Ham/").getPath() );

File [] hams = Folder.listFiles();

**for**(**int** index = 0; index < hams.length; index++) {

Map<String, String> properties = **new** HashMap<String, String>();

name = hams[index].getName();

properties.put("name", name);

properties.put("type", "Ham");

Engine.getProducts().add(**new** Product(properties));

}

//Convert All Censored Italian Pictures

Folder = **new** File( testing.**class**.getResource("/ProductImages/CensoredImages/Meats/Italian/").getPath() );

File [] italians = Folder.listFiles();

**for**(**int** index = 0; index < italians.length; index++) {

Map<String, String> properties = **new** HashMap<String, String>();

name = italians[index].getName();

properties.put("name", name);

properties.put("type", "Italian");

Engine.getProducts().add(**new** Product(properties));

}

//Convert All Censored Turkey Pictures

Folder = **new** File( testing.**class**.getResource("/ProductImages/CensoredImages/Meats/Turkey/").getPath() );

File [] turkeys = Folder.listFiles();

**for**(**int** index = 0; index < turkeys.length; index++) {

Map<String, String> properties = **new** HashMap<String, String>();

name = turkeys[index].getName();

properties.put("name", name);

properties.put("type", "Turkey");

Engine.getProducts().add(**new** Product(properties));

}

//Convert All Censored Wurst Pictures

Folder = **new** File( testing.**class**.getResource("/ProductImages/CensoredImages/Meats/Wurst/").getPath() );

File [] wursts = Folder.listFiles();

**for**(**int** index = 0; index < wursts.length; index++) {

Map<String, String> properties = **new** HashMap<String, String>();

name = wursts[index].getName();

properties.put("name", name);

properties.put("type", "Wurst");

Engine.getProducts().add(**new** Product(properties));

}

//Convert All Censored Cheese Pictures into Cheese Product Objects

File cheeseFolder = **new** File( testing.**class**.getResource("/ProductImages/CensoredImages/Cheeses/").getPath() );

File [] cheeses = cheeseFolder.listFiles();

**for**(**int** index = 0; index < cheeses.length; index++) {

Map<String, String> properties = **new** HashMap<String, String>();

name = cheeses[index].getName();

properties.put("name", name);

properties.put("type", "Cheese");

Engine.getProducts().add(**new** Product(properties));

}

}

**private** **void** initializeTesting() {

/\*\*

\* Initialize the contents of the frame.

\*/

frmDeliTrainingApplication = **new** JFrame();

frmDeliTrainingApplication.getContentPane().setForeground(Color.***WHITE***);

frmDeliTrainingApplication.getContentPane().setBackground(Color.***WHITE***);

frmDeliTrainingApplication.setTitle("Deli Training Application - Testing Mode");

frmDeliTrainingApplication.setBounds(100, 100, 1274, 688);

frmDeliTrainingApplication.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);

frmDeliTrainingApplication.getContentPane().setLayout(**null**);

**final** JLabel Option1 = **new** JLabel("");

**final** JLabel Option2 = **new** JLabel("");

**final** JFormattedTextField QuestionText = **new** JFormattedTextField();

**final** JFormattedTextField ScoreCounter = **new** JFormattedTextField();

Option1.setBackground(Color.***WHITE***);

Option1.setForeground(Color.***WHITE***);

Option1.setFont(UIManager.*getFont*("Menu.font"));

Option1.addMouseListener(**new** MouseAdapter() {

@Override

**public** **void** mouseClicked(MouseEvent arg0) {

updateScore(0);

pickProducts();

updatePage(Option1,Option2,QuestionText,ScoreCounter);

}

});

Option1.setIcon(**new** ImageIcon(testing.**class**.getResource("/ProductImages/CensoredImages/Meats/Beef/BLondonBroilTopRound.jpg")));

Option1.setBounds(0, 50, 600, 600);

frmDeliTrainingApplication.getContentPane().add(Option1);

QuestionText.setFont(**new** Font("Tahoma", Font.***PLAIN***, 19));

QuestionText.setBorder(**null**);

QuestionText.setSelectedTextColor(**new** Color(255, 255, 255));

QuestionText.setSelectionColor(Color.***WHITE***);

QuestionText.setForeground(Color.***BLACK***);

QuestionText.setDisabledTextColor(Color.***WHITE***);

QuestionText.setCaretColor(Color.***WHITE***);

QuestionText.setBackground(**new** Color(255, 255, 255));

QuestionText.setEditable(**false**);

QuestionText.setBounds(10, 0, 600, 46);

frmDeliTrainingApplication.getContentPane().add(QuestionText);

Option2.addMouseListener(**new** MouseAdapter() {

@Override

**public** **void** mouseClicked(MouseEvent e) {

updateScore(1);

pickProducts();

updatePage(Option1,Option2,QuestionText,ScoreCounter);

}

});

Option2.setIcon(**new** ImageIcon(testing.**class**.getResource("/ProductImages/CensoredImages/Cheeses/BWhiteAmericanCheese.jpg")));

Option2.setBounds(648, 50, 600, 600);

frmDeliTrainingApplication.getContentPane().add(Option2);

QuestionText.setText("Select the: "+ name.subSequence(0, name.indexOf('.')));

updatePage(Option1,Option2,QuestionText,ScoreCounter);

ScoreCounter.setText("Score: 0/0");

ScoreCounter.setSelectionColor(Color.***WHITE***);

ScoreCounter.setSelectedTextColor(Color.***WHITE***);

ScoreCounter.setForeground(Color.***BLACK***);

ScoreCounter.setFont(**new** Font("Tahoma", Font.***PLAIN***, 19));

ScoreCounter.setEditable(**false**);

ScoreCounter.setDisabledTextColor(Color.***WHITE***);

ScoreCounter.setCaretColor(Color.***WHITE***);

ScoreCounter.setBorder(**null**);

ScoreCounter.setBackground(Color.***WHITE***);

ScoreCounter.setBounds(1140, 0, 108, 46);

frmDeliTrainingApplication.getContentPane().add(ScoreCounter);

}

**private** **void** updatePage(JLabel Option1, JLabel Option2, JFormattedTextField QuestionText, JFormattedTextField ScoreCounter)

{

rand\_option = generator.nextInt((1+1));

QuestionText.setText("Select the: "+ name.subSequence(0, name.indexOf('.')));

ScoreCounter.setText("Score: "+score+"/"+questions);

questions++;

//Determine Where to display correct picture

**if**(rand\_option < 1) //then display correct picture in Option1 box

{

**if**(type.compareTo("Cheese")!=0)//Determine which folder correct picture is in

Option1.setIcon(**new** ImageIcon(testing.**class**.getResource("/ProductImages/CensoredImages/Meats/"+type+"/"+name)));

**else**

Option1.setIcon(**new** ImageIcon(testing.**class**.getResource("/ProductImages/CensoredImages/Cheeses/"+name)));

**if**(type2.compareTo("Cheese")!=0)//Determine which folder incorrect picture is in

Option2.setIcon(**new** ImageIcon(testing.**class**.getResource("/ProductImages/CensoredImages/Meats/"+type2+"/"+name2)));

**else**

Option2.setIcon(**new** ImageIcon(testing.**class**.getResource("/ProductImages/CensoredImages/Cheeses/"+name2)));

}

**else**

{

**if**(type.compareTo("Cheese")!=0)//Determine which folder correct picture is in

Option2.setIcon(**new** ImageIcon(testing.**class**.getResource("/ProductImages/CensoredImages/Meats/"+type+"/"+name)));

**else**

Option2.setIcon(**new** ImageIcon(testing.**class**.getResource("/ProductImages/CensoredImages/Cheeses/"+name)));

**if**(type2.compareTo("Cheese")!=0)//Determine which folder incorrect picture is in

Option1.setIcon(**new** ImageIcon(testing.**class**.getResource("/ProductImages/CensoredImages/Meats/"+type2+"/"+name2)));

**else**

Option1.setIcon(**new** ImageIcon(testing.**class**.getResource("/ProductImages/CensoredImages/Cheeses/"+name2)));

}

}

**private** **void** updateScore(**int** selection) {

**if**(selection == rand\_option)

score++;

}

**private** **void** showResults(){

resultsWindow = **new** Results(score,questions);

resultsWindow.setVisible(**true**);

**this**.frmDeliTrainingApplication.dispose();

}

**private** **void** pickProducts(){

Product product1 = Engine.getUnpicked();

Product product2;

**if**(product1 == **null**)

//No more unpicked products

showResults();

**else**

{

name = product1.properties.get("name");

type = product1.properties.get("type");

**if**(difficulty == 0) //if easy

{

product2 = Engine.getProduct(Engine.getProducts().indexOf(product1));

type2 = product2.properties.get("type");

}

**else**

{

product2 = Engine.getProduct ( type, Engine.getProducts().indexOf(product1) );

type2 = type;

}

name2 = product2.properties.get("name");

}

}

}

### 8.2.5 Original loadProducts() method

**private** **void** loadProducts() { //Convert All Censored Beef Pictures into Beef Product Objects Objects

File Folder = **new** File( testing.**class**.getResource("/ProductImages/CensoredImages/Meats/Beef/").getPath() );

File [] beefs = Folder.listFiles();

**for**(**int** index = 0; index < beefs.length; index++) {

Map<String, String> properties = **new** HashMap<String, String>();

name = beefs[index].getName();

properties.put("name", name);

properties.put("type", "Beef");

Engine.getProducts().add(**new** Product(properties));

}

//Convert All Censored Bologna Pictures

Folder = **new** File( testing.**class**.getResource("/ProductImages/CensoredImages/Meats/Bologna/").getPath() );

File [] bolognas = Folder.listFiles();

**for**(**int** index = 0; index < bolognas.length; index++) {

Map<String, String> properties = **new** HashMap<String, String>();

name = bolognas[index].getName();

properties.put("name", name);

properties.put("type", "Bologna");

Engine.getProducts().add(**new** Product(properties));

}

//Convert All Censored Chicken Pictures

Folder = **new** File( testing.**class**.getResource("/ProductImages/CensoredImages/Meats/Chicken/").getPath() );

File [] chickens = Folder.listFiles();

**for**(**int** index = 0; index < chickens.length; index++) {

Map<String, String> properties = **new** HashMap<String, String>();

name = chickens[index].getName();

properties.put("name", name);

properties.put("type", "Chicken");

Engine.getProducts().add(**new** Product(properties));

}

//Convert All Censored Ham Pictures

Folder = **new** File( testing.**class**.getResource("/ProductImages/CensoredImages/Meats/Ham/").getPath() );

File [] hams = Folder.listFiles();

**for**(**int** index = 0; index < hams.length; index++) {

Map<String, String> properties = **new** HashMap<String, String>();

name = hams[index].getName();

properties.put("name", name);

properties.put("type", "Ham");

Engine.getProducts().add(**new** Product(properties));

}

//Convert All Censored Italian Pictures

Folder = **new** File( testing.**class**.getResource("/ProductImages/CensoredImages/Meats/Italian/").getPath() );

File [] italians = Folder.listFiles();

**for**(**int** index = 0; index < italians.length; index++) {

Map<String, String> properties = **new** HashMap<String, String>();

name = italians[index].getName();

properties.put("name", name);

properties.put("type", "Italian");

Engine.getProducts().add(**new** Product(properties));

}

//Convert All Censored Turkey Pictures

Folder = **new** File( testing.**class**.getResource("/ProductImages/CensoredImages/Meats/Turkey/").getPath() );

File [] turkeys = Folder.listFiles();

**for**(**int** index = 0; index < turkeys.length; index++) {

Map<String, String> properties = **new** HashMap<String, String>();

name = turkeys[index].getName();

properties.put("name", name);

properties.put("type", "Turkey");

Engine.getProducts().add(**new** Product(properties));

}

//Convert All Censored Wurst Pictures

Folder = **new** File( testing.**class**.getResource("/ProductImages/CensoredImages/Meats/Wurst/").getPath() );

File [] wursts = Folder.listFiles();

**for**(**int** index = 0; index < wursts.length; index++) {

Map<String, String> properties = **new** HashMap<String, String>();

name = wursts[index].getName();

properties.put("name", name);

properties.put("type", "Wurst");

Engine.getProducts().add(**new** Product(properties));

}

//Convert All Censored Cheese Pictures into Cheese Product Objects

File cheeseFolder = **new** File( testing.**class**.getResource("/ProductImages/CensoredImages/Cheeses/").getPath() );

File [] cheeses = cheeseFolder.listFiles();

**for**(**int** index = 0; index < cheeses.length; index++) {

Map<String, String> properties = **new** HashMap<String, String>();

name = cheeses[index].getName();

properties.put("name", name);

properties.put("type", "Cheese");

Engine.getProducts().add(**new** Product(properties));

}

}

## 8.3 Non-Direct Log

|  |  |  |  |
| --- | --- | --- | --- |
| Time/Task Log | Activity Type: Non Direct | |  |
| Date | Duration (in minutes) | Specific Activity/Task |  |
| Feb-23 | 60 | Read some of Chapter 3 Refactoring - Improving the Design. Most of it I seen before |  |
| Feb-25 | 60 | Read some of Chapter 4 Refactoring - Improving the Design |  |
| Feb-27 | 60 | Read Chapter 5 Refactoring - Improving the Design |  |
| Feb-28 | 60 | Read Chapter 5 Refactoring - Improving the Design |  |
| Mar-1 | 60 | Read Chapter 6 Refactoring - Improving the Design |  |
| Mar-2 | 60 | Read Chapter 6 Refactoring - Improving the Design |  |
| Mar-3 | 60 | Read Chapter 7 Refactoring - Improving the Design |  |
| Mar-4 | 60 | Read Chapter 7 Refactoring - Improving the Design |  |
| Mar-5 | 60 | Updating my log and reading the repo Mike Dumas created for project 2 |  |
| Mar-10 | 60 | Read part of Chapter 8 to see how it can be applied to the Deli Training App |  |
| Mar-11 | 60 | Read the first half of Mike Dumas's report for Project 2 of SSE 657 | *These reports are Deli App specific* |
| Mar-12 | 90 | Read the 2nd half of Mike Dumas's report for Project 2 of SSE 657 |
| Mar-16 | 120 | Read the first half of Mike Dumas's report for Project 3 of SSE 657 |
| Mar-17 | 120 | Read the 2nd half of Mike Dumas's report for Project 3 of SSE 657 |
| Mar-19 | 60 | Explored the code of the Deli App |  |
| Mar-20 | 120 | Read Chapter 8 of Refactoring - Improving the Design and figured out what areas of code could be refactored |  |
| Mar-21 | 120 | Further analyzed the driving system behind the Deli App (GameEngine.java) and reasearched how the Map interface works Refactored code in the Testing.java class |  |
| Mar-22 | 240 | Visting Refactoring.com to obtain an overview of Chapter 8. Structured classes for the different types of products the Deli App consisted of. Created a JUnit test for testing.java |  |
| Mar-23 | 120 | Modified JUnit test for testing.java and structured report |  |
| **TOTAL** | **1650** |  |  |
| **REQUIRED** | **4500** |  |  |
| **REMAINING** | **1390** |  |  |