uSearch

Design Specification

Course Number: SFWRENG 2XB3

Course Name: Software Engineering and Practice: Binding Theory to Practice

Department: Software Engineering

University: McMaster

Date: April 5, 2019

Group: 12

Lab Section: L02

Members: Shesan Balachandran, Bill Nguyen, Dananjay Prabaharan, Nishanth Raveendran

REPORT REVISION HISTORY

It was a single document that was worked on by the whole group online so there is no revision history.

TEAM MEMBERS

<u>**Shesan Balachandran**</u>
Student ID: 400134706
Roles & Responsibilities: Project Manager/Developer
Consent:
By virtue of submitting this document we electronically sign and date that the work being submitted by all the individuals in the group is their exclusive work as a group and we consent to make available the application developed through [CS] or [SE]-2XB3 project, the reports, presentations, and assignments (not including my name and student number) for future teaching purposes.

<u>**Bill Nguyen**</u>
Student ID: 400120876
Roles & Responsibilities: Front End Designer/Developer
Consent:
By virtue of submitting this document we electronically sign and date that the work being submitted by all the individuals in the group is their exclusive work as a group and we consent to make available the application developed through [CS] or [SE]-2XB3 project, the reports, presentations, and assignments (not including my name and student number) for future teaching purposes.

<u>**Dananjay Prabaharan**</u>
Student ID: 400128524
Roles & Responsibilities: Algorithms Developer/Meeting Organizer
Consent:
By virtue of submitting this document we electronically sign and date that the work being submitted by all the individuals in the group is their exclusive work as a group and we consent to make available the application developed through [CS] or [SE]-2XB3 project, the reports, presentations, and assignments (not including my name and student number) for future teaching purposes.

<u>**Nishanth Raveendran**</u>
Student ID: 400121953
Roles & Responsibilities: Algorithms Developer/Scrum Master
Consent:
By virtue of submitting this document we electronically sign and date that the work being submitted by all the individuals in the group is their exclusive work as a group and we consent to make available the application developed through [CS] or [SE]-2XB3 project, the reports, presentations, and assignments (not including my name and student number) for future teaching purposes.

CONTRIBUTIONS

NAME	ROLES	CONTRIBUTIONS	COMMENTS
Shesan Balachandran	Project Manager/Developer	● Recorded project logs and product backlogs ● Developed modules to process and extract data ● Developed the graphing module with testing and documentation ● Contributed to the design of the final presentation ● Contributed the abstract, description of classes/modules and explanation of the decomposition portion to the design specification document	N/A
Bill Nguyen	Front End Designer/Developer	● Designed and developed the graphical user interface ● Developed an android application for our program ● Contributed the view of uses relation and evaluation of design portion to the design specification document	N/A
Dananjay Prabaharan	Algorithms Developer/Meeting Organizer	● Developed search algorithm, search test code and the documentation for it ● Recorded meeting minutes ● Contributed the API and implementation portion to design specification document	N/A
Nishanth Raveendran	Algorithms Developer/Scrum Master	● Developed sorting module, sort test code and the documentation for it ● Recorded sprint backlog and burnout chart ● Contributed to the content of the final project presentation ● Contributed the UML class and state machine diagram portion to design specification document	N/A

EXECUTIVE SUMMARY

Abstract

uSearch is a product searching application that is meant to allow customers to make better decisions when shopping and make the process fast and ecient. Often, we nd ourselves shopping for things at a store and wonder if this is a quality product or if there is a similar but better alternative. This is what uSearch is looking to solve. The app will use the amazon product network dataset that includes all the core information this app will function from such as the products themselves, the ratings, the reviews and the relation between the products. The stakeholders/users of this product could be anyone that has access to a mobile device. Since there are no payment options or any mature content in this application, even children can use this app. The expected outcome of this product is efficient and satisfactory shopping as well as a clean selling process. Users will be quickly presented with critical information such as reviews, ratings, and similar better alternatives if available. They can expect to save a lot of time and most importantly save a lot of money.

TABLE OF CONTENTS

Description of the Classes & the Modules

Product Class

This class is an abstract data type that represents a product. An instance of this class will have different properties like title, asin, group, list of similar products and it's average rating. Tracing back to the requirements, this class allows us model a real product that will then be used throughout the whole program.

Review Class

This class is an abstract data type that represents a review. An instance of this class will have different properties like reviewer name, review text, review summary, review time, review's unix time, the reviewer's rating of the product and the asin. Tracing back to the requirements, this class allows us to model a review that will help pick out the best products.

Graph Class

This class allows us to make a graphical representation of the products and make connections between different products based on similarities. Using a product as a source this class utilizes the breadth first search approach to search all the connected products to the source. Tracing back to the requirements, this class also enables us to be able to suggest similar products to a specific product based on connections and similarities.

Search Module

This module allows us to search products and reviews in a group of products or reviews. The searching can be done through different properties of the product like the asin, the title and the group and through the asin for the review. Tracing back to the requirements, this module provides the core functionality of our application is searching products and reviews for those products.

Sort Module

This module allows us to sort products in a group of products. The sorting can be done through different properties of the product like the asin, the title and the rating. Tracing back to the requirements, this module enables us to perform the binary search by allowing us to sort the products before hand. It also helps pick out the best products based on different properties.

Read Module

This module allows us to read from a file and extract unique strings for each product or review that will be parsed later. This module is strictly designed to extract information from the specific file format. Tracing back to the requirements, this module helps gather information about products and reviews from the data file and will help setup the rest of the application

Create Items Module

This module allows us to make the corresponding abstract data types like product and review from the raw data of strings. This module is strictly designed to extract information from the specific dataset for the corresponding data type. Tracing back to the requirements, this module helps create the instances of the different data types we need for the rest of the application.

Command Line App module and the GUI module

These are the main modules of the application where it is ran. The command line app module provides a command line interface of the application which provides all the functionalities of the application but in the command line. The gui module provides a graphical interface for the user which also provides all the functionalities of the application but in a more user friendly and interactive manner. Tracing back to the requirements, these modules provide the interactive portion of the application.

Explanation of the decomposition of our application

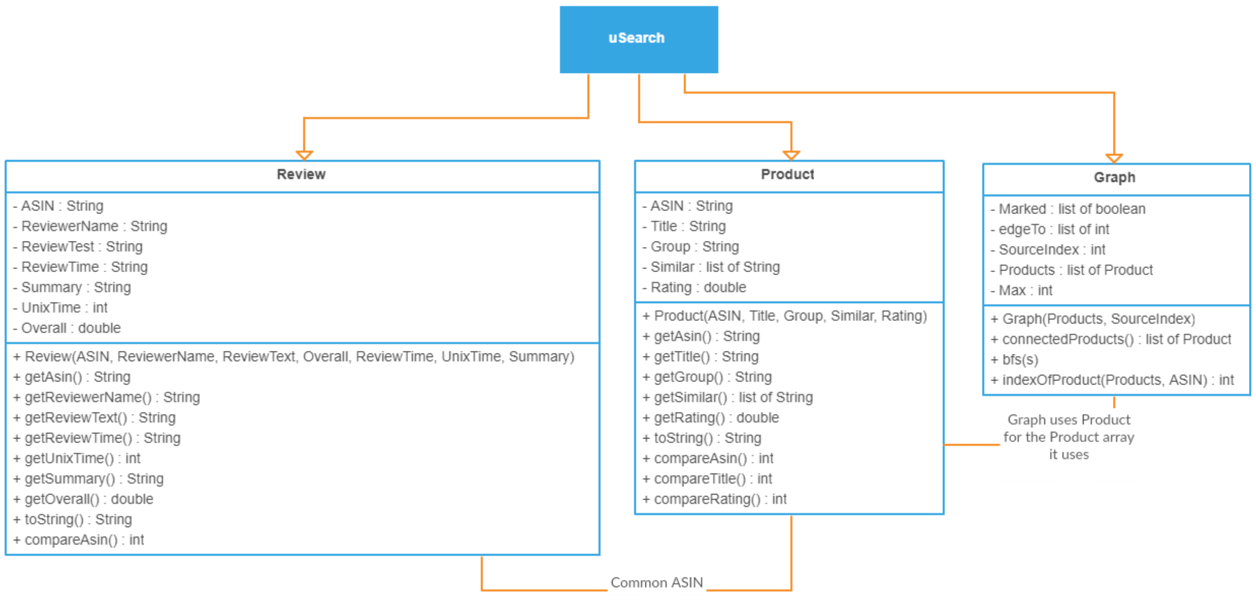
In the brainstorming process we broke down the program into smaller functionalities and different data types that we will need. Based on the two different data sets, we decided to create two types for each, product and review.

Then when we decided on those types we needed a module that can read the data from those files which was the read module. Then after reading the file we wanted to make those types from the strings that was read. This was done using the create items module. The create items module was separated from the read module because we wanted to make a read module that was generic enough to read other data sets with similar patterns, for example we used the same read module to read both the datasets due to its generality. However the creation of the types are completely different for products and the new type creations can easily be added. So this separation between read and create items modules allows to keep a generic read module while creating new functions for the data type creation itself.

Now that all the modules for the setup has been planned out, we needed to figure out the main functionalities of our application. We decided that those will be searching the input product, searching it's reviews and in addition suggesting possible better alternatives similar to the input product. It's clear that we will need a search module and keeping it separate will be easier to modify and test. For the searching we decided to use binary search so the searching array has to be initially sorted hence we will need some sorting module. This module will also be used to sort the similar alternatives to pick out the best ones from those. Like before, keeping the sort module separate will be easier to modify and test. But in order to suggest similar alternatives we will need to create a graph and connect the products and again it is best to create a separate graph module. And lastly we have a main module that will put everything together and another main module but that will have a graphical user interface.

Throughout the implementations, we always decided to create separate modules because it is more organized, made it easier for individual members to work separately, made testing and debugging a lot easier, and lastly it easily lets us modify the inner workings of the module like say change the algorithms while not breaking the rest of the program.

UML Class Diagrams



Description of the interface (API)

public class CreateItems

// Create products from list of strings take from data set.
Product[] createProducts(String[] productstring)

// Create reviews from list of strings take from data set
Review[] createReviews(String[] reviewstring)

public class Graph

// Create connected graph for a source product
Graph(Product[] products, int sourceIndex)

// Get connected products to the source product
Product[] connectedProducts()

public class Product

// Construct the Product
Product (String product_id, int sales_amount)

// Get the Product ID
String get_product_id ( )

// Get the Sales Amount
int get_sales_amount ( )

// Set the Sales Amount
set_sales_amount (int sales_amount)
// String representation of the product
String string_rep ( )

// Compare product to another product
int compareTo (Product j)

public class Read
// Read a string of data line by line
String[] readData(String path, int size)

public class Review
// Construct an instance of Review.
Review(String asin, String reviewerName, String reviewText, double overall, String reviewTime, int unixTime, String summary)

// Get the ASIN of the review.
String getAsin()

// Get the reviewer's name
String getReviewerName()

// Get the text of the review
String getReviewText()

// Get the time of the review
String getReviewTime()

// Get the unix time of the review
int getUnixTime()

// Get the summary of the review
String getSummary()

// Get the reviewer's rating of the product

double getOverall()

// String representation of the Review
String toString ( )

// Compare review to another review
int compareAsin(Review that)

public class Search

// Returns the index of the asin value that needs to be searched in the array
int searchProductAsin(String asin, Product[] a)

// Returns the index of the title value that needs to be searched in the array
int searchProductTitle(String title, Product[] a)

// Returns the index of the group value that needs to be searched in the array
int searchProductGroup(String group, Product[] a)

// Returns the index of the asin value that needs to be searched in the array
int searchReviewAsin(String asin, Review[] a)

public class Sort

// Sorts an array of Products by their Asin using bottom-up merge sort
void sortAsin(Product[] x, int n)

// Sorts an array of Products by their Title using bottom-up merge sort
void sortTitle(Product[] x, int n)

// Sorts an array of Products by their Ratings using bottom-up merge sort
void sortRating(Product[] x, int n)

Description of the Implementation

CreateItems.java

There are no private entities within the CreateItems.java file. The file contains two public functions, **createProducts** and **createReviews**, which return an array of Products and Reviews respectively. The **createProducts** function breaks down a string containing the data of multiple Products and allocates the extracted data into multiple Product object. Similarly, the **createReviews** function breaks down a string containing multiple Review data and allocates the data into multiple Review objects.

Graph.java

There are multiple private variables used in Graph..java:

```
private boolean[] marked;
private int[] edgeTo;
private int sourceIndex;
private Product[] products;
private int max;
```

These variables are used to store the products that they are connected to the source product, the previous connection to each product, the index of the source product, list of all the products, and the maximum depth possible for the recursive depth in the connections. This file contains a private method bfs(int s) that will perform breadth first search on the products to mark all the connected products to the product at the source index. It also contains a private method indexOfProduct(Products[] products, string asin) that finds the index of the a product matching the given asin. The public constructor Graph(Products[] products, int sourceIndex) does most of the work by creating the graph for the products and their connections. Lastly the public method connectedProducts() returns a list of products that are connected to the source product by searching through the marked array that was marked by the bfs.

Product.java

There are multiple private variables used in Product.java:

```
private String asin;
private String title;
private String group;
private String[] similar;
private double rating;
```

These variables are used to store their respective Product object data, utilized in the Product constructor, and interact with the public methods in the class. Apart from the numerous getter methods for each Product property, the Product file contains multiple compare methods designated to compare certain properties such as the title, asin, and ratings. The compare methods would return a negative [positive] number if the first product is less [greater] than the second product. Finally, the Product object contains a toString() method utilized to provide a string representation of the product.

Read.java

There are no private entities within the Read.java file. Within the file, there is only one public method, readData, which is used to read a string of data line by line and output each line into an array of Strings.

Review.java

There are multiple private variables used in Review.java:

```
private String asin;
private String reviewerName;
private String reviewText;
private String reviewTime;
private String summary;
private int unixTime;
private double overall;
```

These variables are used to store their respective Review object data, utilized in the Review constructor, and interact with the public methods in the class. Apart from the numerous getter methods for each Review property, the Review file contains a compare method designated to compare two ASIN. The compare methods would return a negative [positive] number if the first review ASIN is less [greater] than the second review.. Finally, the Review object contains a toString() method utilized to provide a string representation of the review.

Search.java

There are no private entities within the Search.java file. Within the Search.java file, there are multiple public search methods, each designed to search for a specific property from the Product/Review object.

searchProductAsin(String asin, Product[] a) → Searches for a specific Product ASIN

searchProductTitle(String title, Product[] a)→ Searches for a specific Product Title

searchProductGroup(String group, Product[] a) → Searches for a specific Product Group

searchReviewAsin(String asin, Review[] a) → Searches for a specific Review ASIN

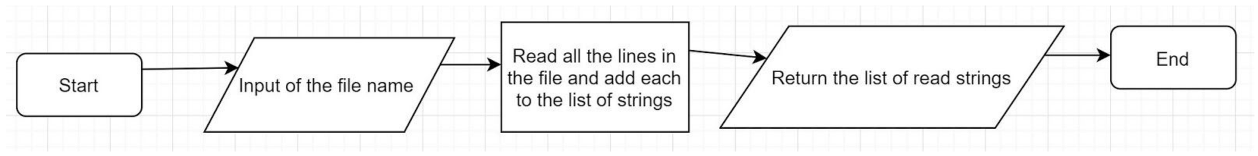
These search methods utilize the **binary search algorithm** to find its specific property.

Sort.java

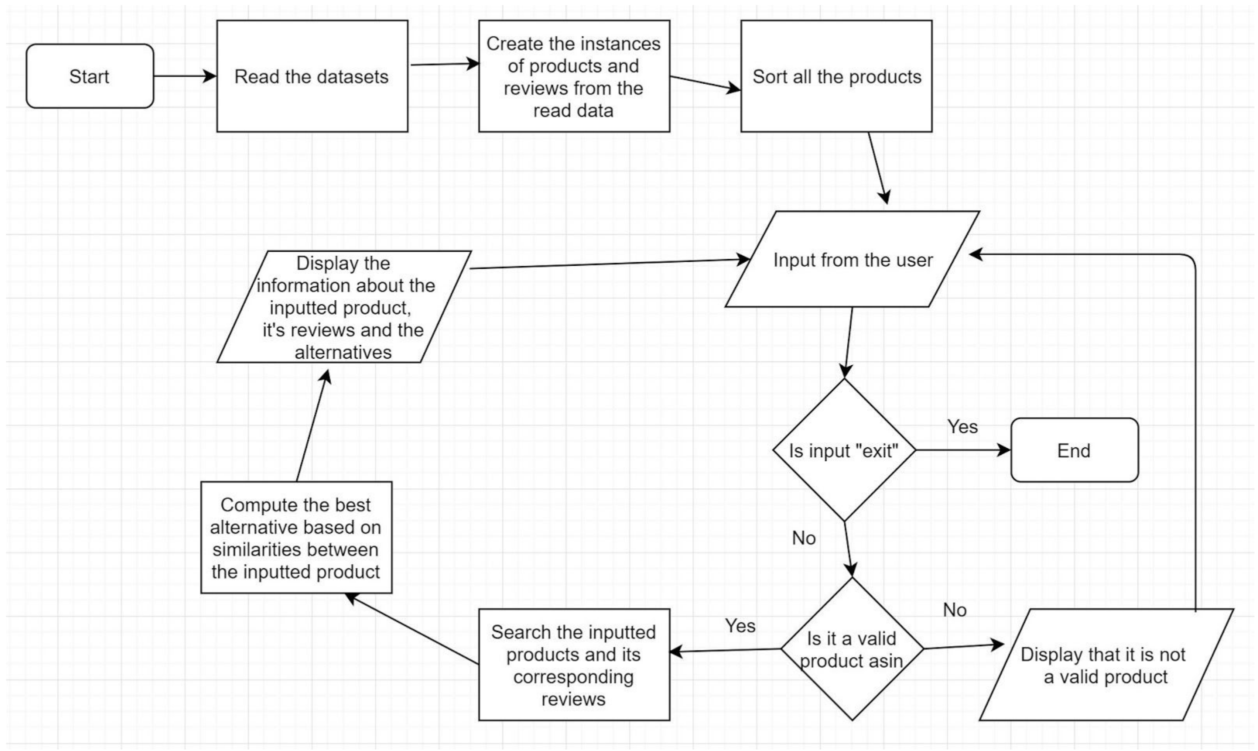
The Sort.java file contains only one private variable - Product[] aux. The auxiliary array is utilized in the merge sort algorithm to copy and store all the elements from the inputted Product array. The array is then utilized afterwards to be merged back into the original array in the sorted order. The Sort.java file also contains the private method **merge(Product[] x, int low, int middle, int high, char type),** which is utilized in the public sort methods to merge an array in a sorted order. There are three public sort methods that sorts a Product object array based on a specific property. These specific properties are the Product ASIN, Title, and Ratings.

UML State Machine Diagram for Two Most Interesting Classes

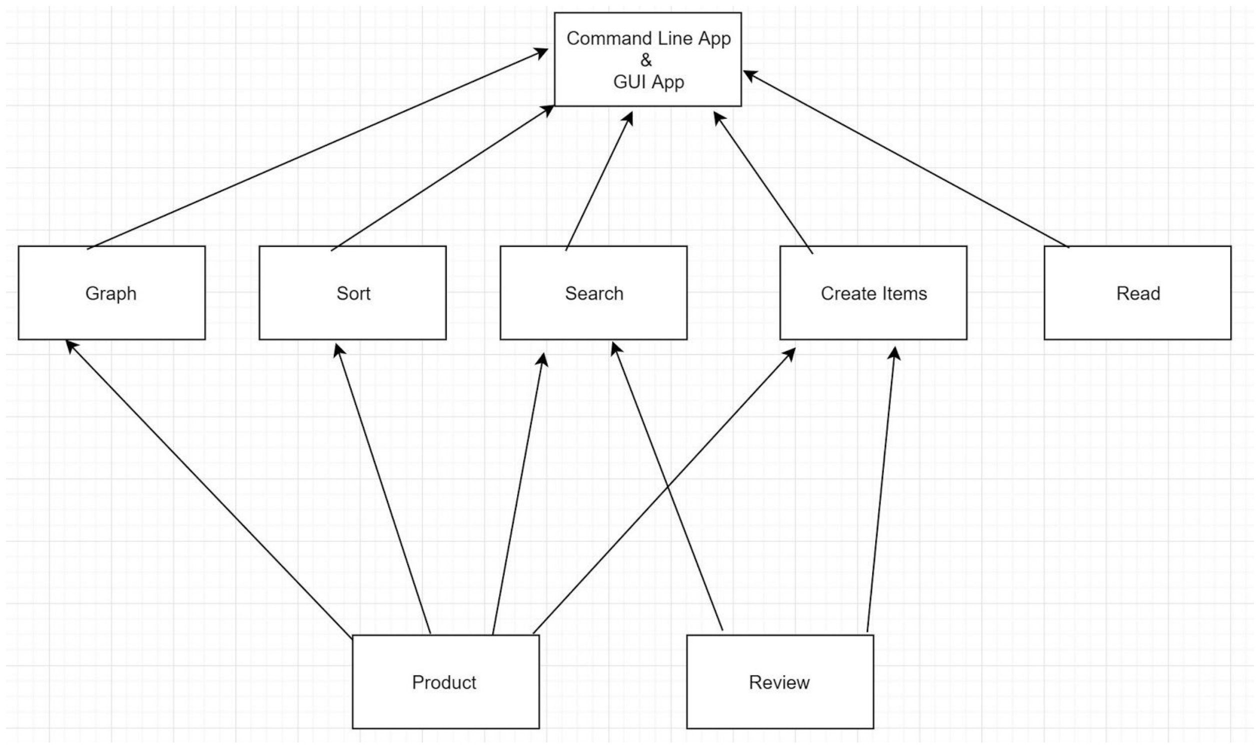
Read.java


Start
Input of the file name
Read all the lines in the file and add each to the list of strings
Return the list of read strings
End

CommandLineApp.java


Start
Read the datasets
Create the instances of products and reviews from the read data
Sort all the products
Input from the user
Is input "exit"
Yes
End
No
Is it a valid product asin
Yes
No
Display that it is not a valid product
Search the inputted products and its corresponding reviews
Compute the best alternative based on similarities between the inputted product
Display the information about the inputted product, it's reviews and the alternatives

View of the uses relationship



Internal review/evaluation of design

Overall we would say the design was pretty good. There were things that were good about the design, things that were bad and things that we can improve on.

Something that was good about our design is that it is modular meaning all the core functionalities are split into different modules. This is really good because it makes it easier to maintain and modify functionalities in the future without breaking the rest of the program. This is also beneficial for testing and we can easily perform unit testing as well.

Something that was bad about our design is that we didn't use a database. We simply read from the dataset and created the objects every time the app launched which made it very slow. We also limited the searching options to searching by the asin only.

And finally something that we can improve upon would things that we did bad. Like we can use a database like mongodb to store our dataset so we have access to it all the time and we don't have to extract the info from the data set every time because it will always be ready. Also we can more functionalities to the search like being able to take in inputs like the product title instead of the just the asin. Lastly, we could also make the user interface better and possibly make an android app instead of just a normal gui.