INFORMATICS INSTITUTE OF TECHNOLOGY

UNIVERSITY OF WESTMINSTER

Informatics Institute of Technology

Department of Computing

BEng: Software Engineering

MODULE: 4COSC010C.3 Programming Principles II

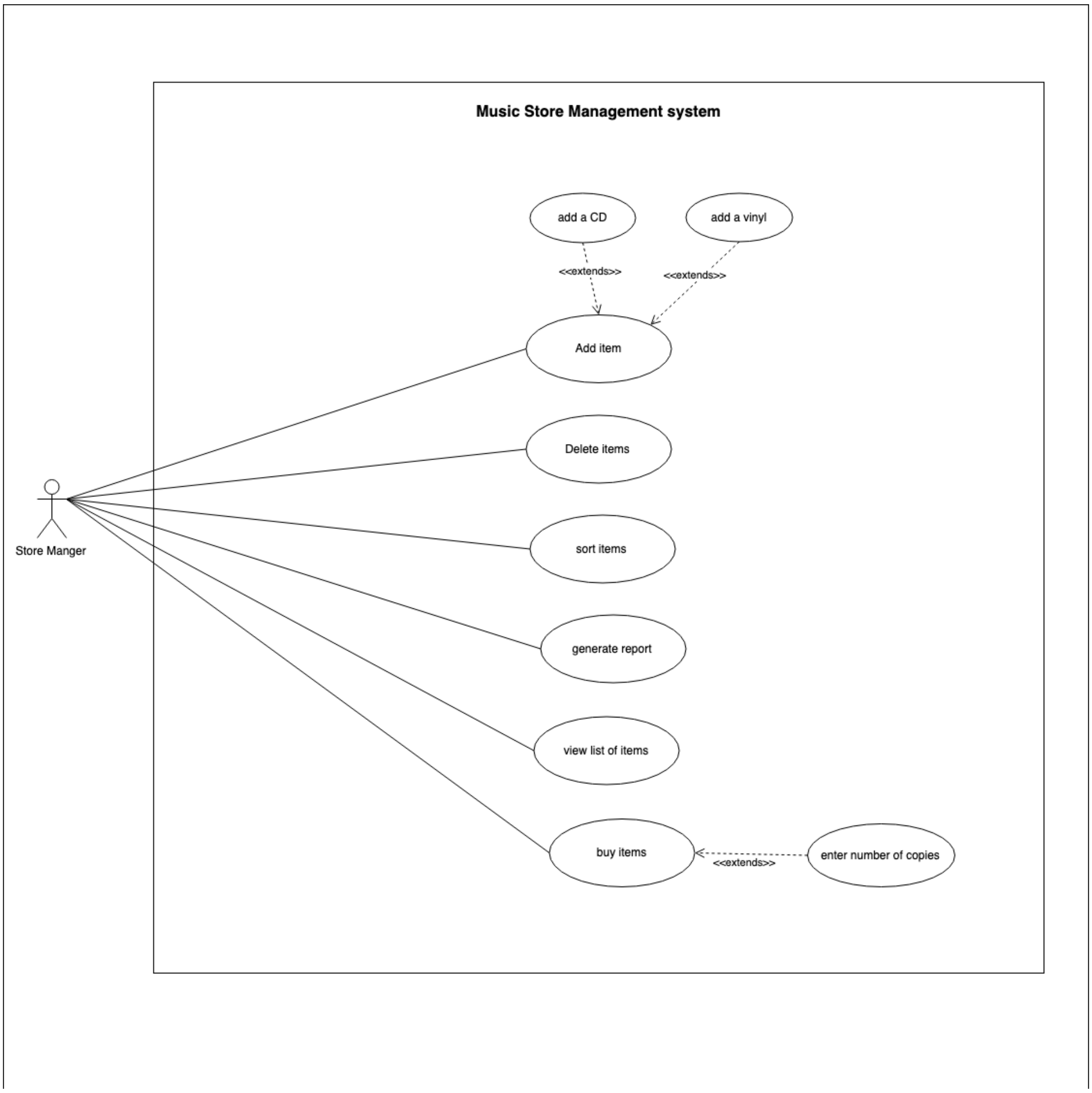
Coursework Report

Tutorial Group	: Group D
Student IIT ID	: 2018516
Student UoW ID	: w1742286
Student Name	: Nujitha Wickramasurendra

Table of Contents

Use Case Diagram……………………………………………………………………………………….3
Use Case Description……………………………………………………………………………………..4
Class Diagram……………………………………………………………………………………………..10
Activity Diagram…………………………………………………………………………………………...11
Test Case…………………………………………………………………………………………………
Black Box Test……………………………………………………………………………………12
White Box Test……………………………………………………………………………………14
Code……………………………………………………………………………………………………21

1. Use Case Diagram



2. Use Case Descriptions

Use Case ID	**1**
Use Case Name	Buy items
Use Case Description	Store manager can buy items by selecting the item ID. The user can also buy more than one copy
Actors	Store Manager
Pre Conditions	1. Items should have been added to the store 2. User should have selected the needed item and the number of copies
Post conditions	1. The quantity in the store should be deducted by the number of copies ordered by the user from the selected item type 2. Customer should be informed about the total via the interface itself
Priority	High
Includes	View total price, check item availability
Extends	Enter number of copies
Path	
Primary Path	1. Customer selects an item and the number of copies 2. Check if that item is available and the number of items requested by the user. 3. Deduct the quantity from the store 4. View the total amount for the user in the interface
Alternate Path	N/A
Exception Path	1. Store manager selects an item and the number of copies 2. Check if that item is available and the number of items requested by the user. 3. Display a message to the user about the unavailability of items in the store 4. Re-prompt the customer to select an item and number of copies required
Assumptions	No system runtime failures

Use Case ID	**2**
Use Case Name	Add item
Use Case Description	Store manager can add items to the store
Actors	Store manager
Pre Conditions	1. The store should have not been full. (should not exceed the given maximum stock level of 1000)
Post conditions	1. Added items should be appended to the database store 2. Database store should update at the instance
Priority	High
Includes	Check free space in the store
Extends	Add a cd, Add a vinyl
Path	
Primary Path	1. Prompt the menu interface for the user 2. Check the status of the database (Space Available / Full) 3. Get the details of the item 4. Add items to the database
Alternate Path	N/A
Exception Path	1. Prompt the interface for the user 2. Check if the store status is currently full 3. Notify the store manager with a message 4. Ask if he wants to delete some existing items or cancel the process of adding items 5. Redirect the user to the menu
Assumptions	No system runtime failures

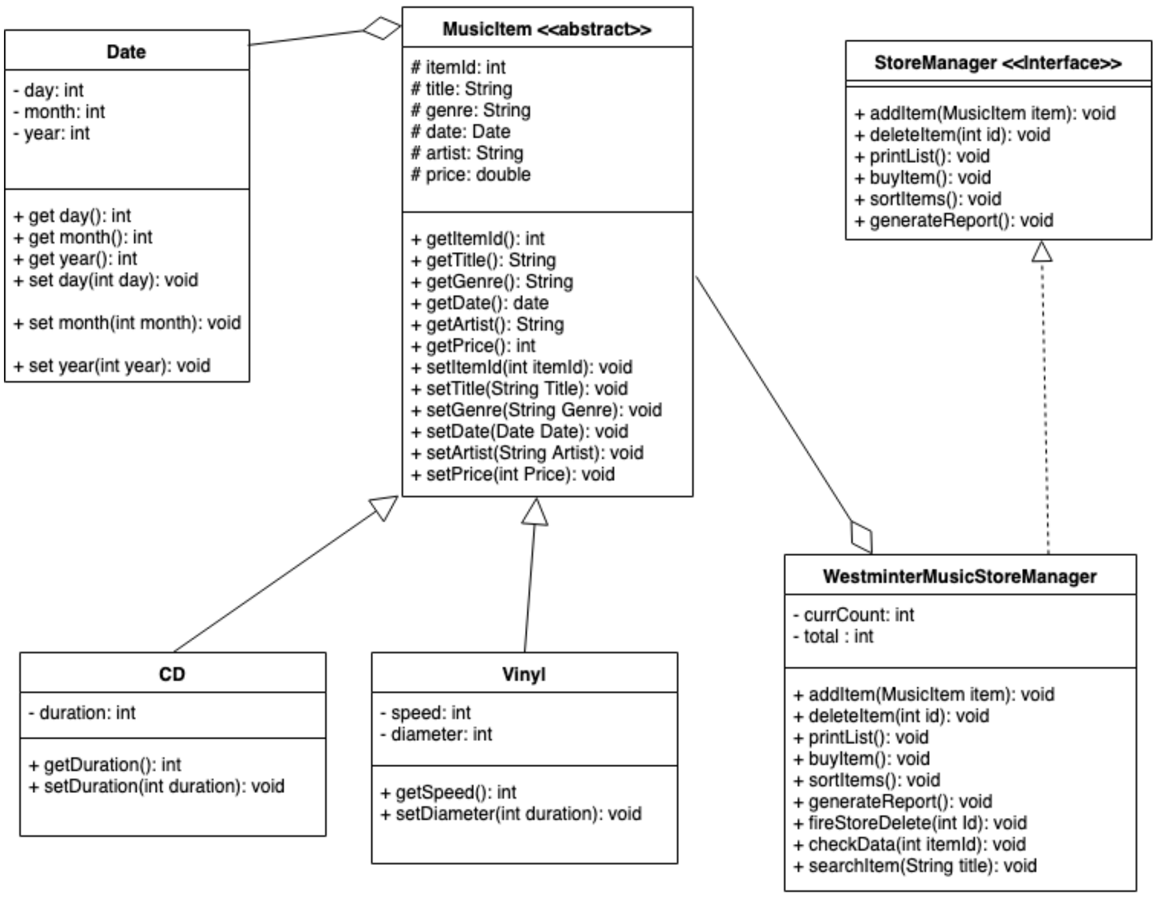
Use Case ID	**3**
Use Case Name	Delete items
Use Case Description	Store manager can delete items from the store
Actors	Store manager
Pre Conditions	1. Store should be at least consisted of a single item
Post conditions	1. Removed item should be deleted with its all relevant information stored in the database
Priority	High
Includes	Check free space in the store
Extends	Add a cd, Add a vinyl
Path	
Primary Path	1. Prompt the menu interface for the user 2. Check the status of the database (Items Available / Null) 3. Prompt the user to find the item he wants to delete 4. Remove item from the database
Alternate Path	N/A
Exception Path	1. Prompt the menu interface for the user 2. Check the status of the database (Items Available / Null) 3. If Null notify the user with a message 4. Ask if the user wants to add an item to the store or cancel the process of deleting items
Assumptions	No system runtime failures

Use Case ID	4
Use Case Name	Sort items
Use Case Description	Store manager can sort the items in the store
Actors	Store manager
Pre Conditions	1. Store should be at least consisted of a single item
Post conditions	1. Items in the store should be sorted together with its relevant information, in the ascending order
Priority	High
Includes	Check free space in the store
Extends	N/A
Path	
Primary Path	1. Prompt the menu interface for the user 2. Check the status of the database (Items Available / Null) 3. Perform the sorting process
Alternate Path	N/A
Exception Path	1. Prompt the menu interface for the user 2. Check the status of the database (Items Available / Null) 3. If Null notify the user with a message 4. Redirect the user to the menu
Assumptions	No system runtime failures

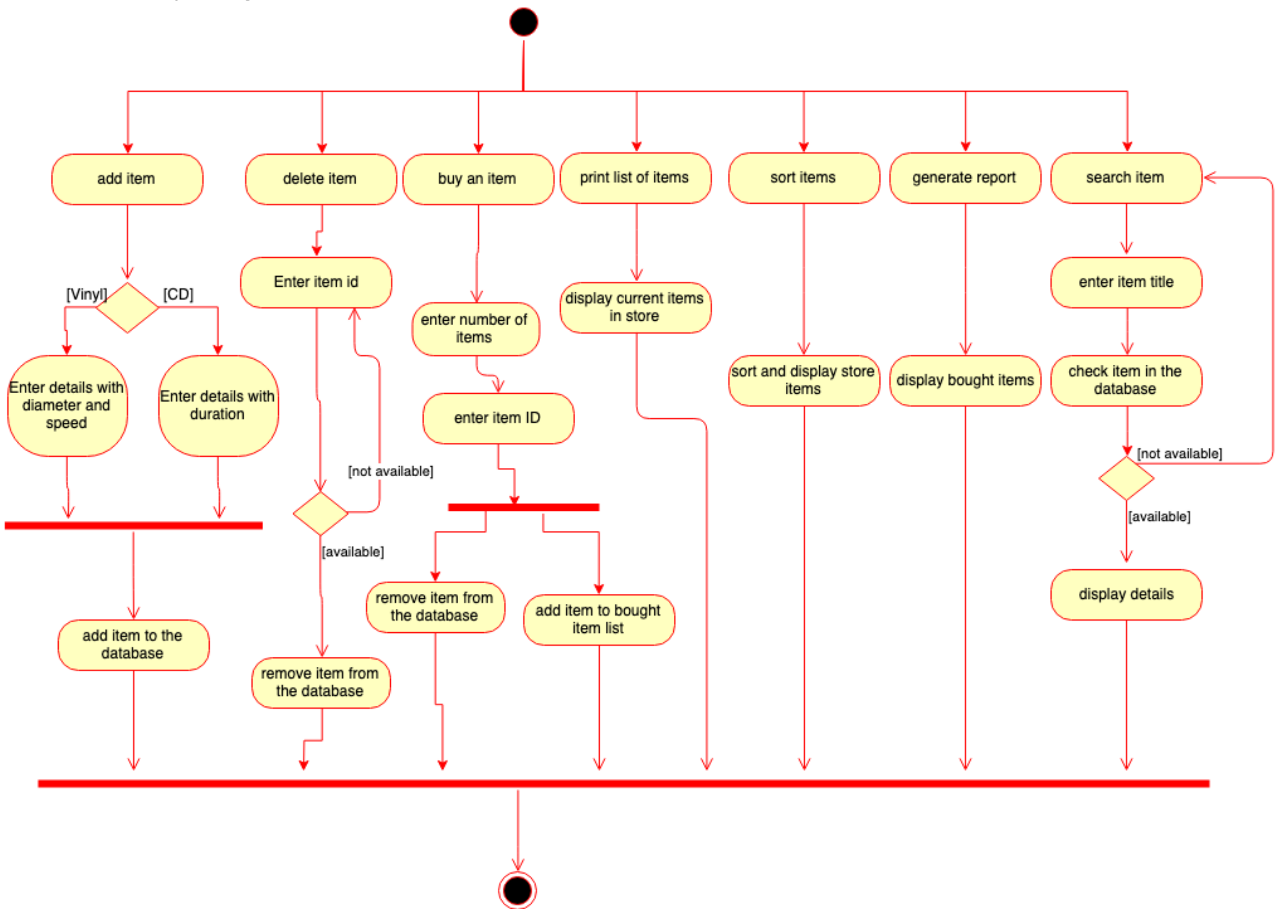
Use Case ID	**5**
Use Case Name	Generate a report
Use Case Description	Store manager can generate a report regarding the items already sold
Actors	Store manager
Pre Conditions	1. Items should have been sold
Post conditions	1. Display the title, ID, price and the selling time / date
Priority	High
Includes	Check free space in the store
Extends	N/A
Path	
Primary Path	1. Prompt the menu interface for the user 2. Check the status of the database (Items Available / Null) 3. Perform the sorting process
Alternate Path	N/A
Exception Path	1. Prompt the menu interface for the user 2. Check the status of the database (Items Available / Null) 3. If Null notify the user with a message 4. Redirect the user to the menu
Assumptions	No system runtime failures

Use Case ID	**6**
Use Case Name	View list of items
Use Case Description	Store manager can view the current items in the store
Actors	Store manager
Pre Conditions	1. Items should have been added to the store
Post conditions	1. Display the item ID and the item type in the console itself
Priority	High
Includes	Check free space in the store
Extends	N/A
Path	
Primary Path	1. Prompt the menu interface for the user 2. Check the status of the database (Items Available / Null) 3. Display the relevant information
Alternate Path	N/A
Exception Path	1. Prompt the menu interface for the user 2. Check the status of the database (Items Available / Null) 3. If Null notify the user with a message 4. Redirect the user to the menu
Assumptions	No system runtime failures

3. Class Diagram



4. Activity Diagram



5. Test Case
    a. Black Box

        i. 1) Main menu input validation
        ii.

Input	Expected Value	Actual Output	Bug?
0 [boundary value]	Invalid, re-enter	Invalid, re-enter	No
9 [boundary value]	Invalid, re-enter	Invalid, re-enter	No
1 [valid]	Pass	Pass	No
7 [valid]	Pass	Pass	No
"hello"	Invalid	Error	Yes

2) Date Validation (day and month)

Input	Expected Value	Actual Output	Bug?
0 [boundary value]	Invalid, re-enter	Invalid, re-enter	No
32 [boundary value]	Invalid, re-enter	Invalid, re-enter	No
1	Pass	Pass	No
7	Pass	Pass	No
12	Pass	Pass	No
13	Invalid, re-enter	Invalid, re-enter	No
"hello"	Invalid	Error	Yes

3) Search an item with its title

Input	Expected Value	Actual Output	Bug?
Maroon5	Item not found	Item not found	No
photograph	Pass	Pass	No
8	Item not found	Item not found	No

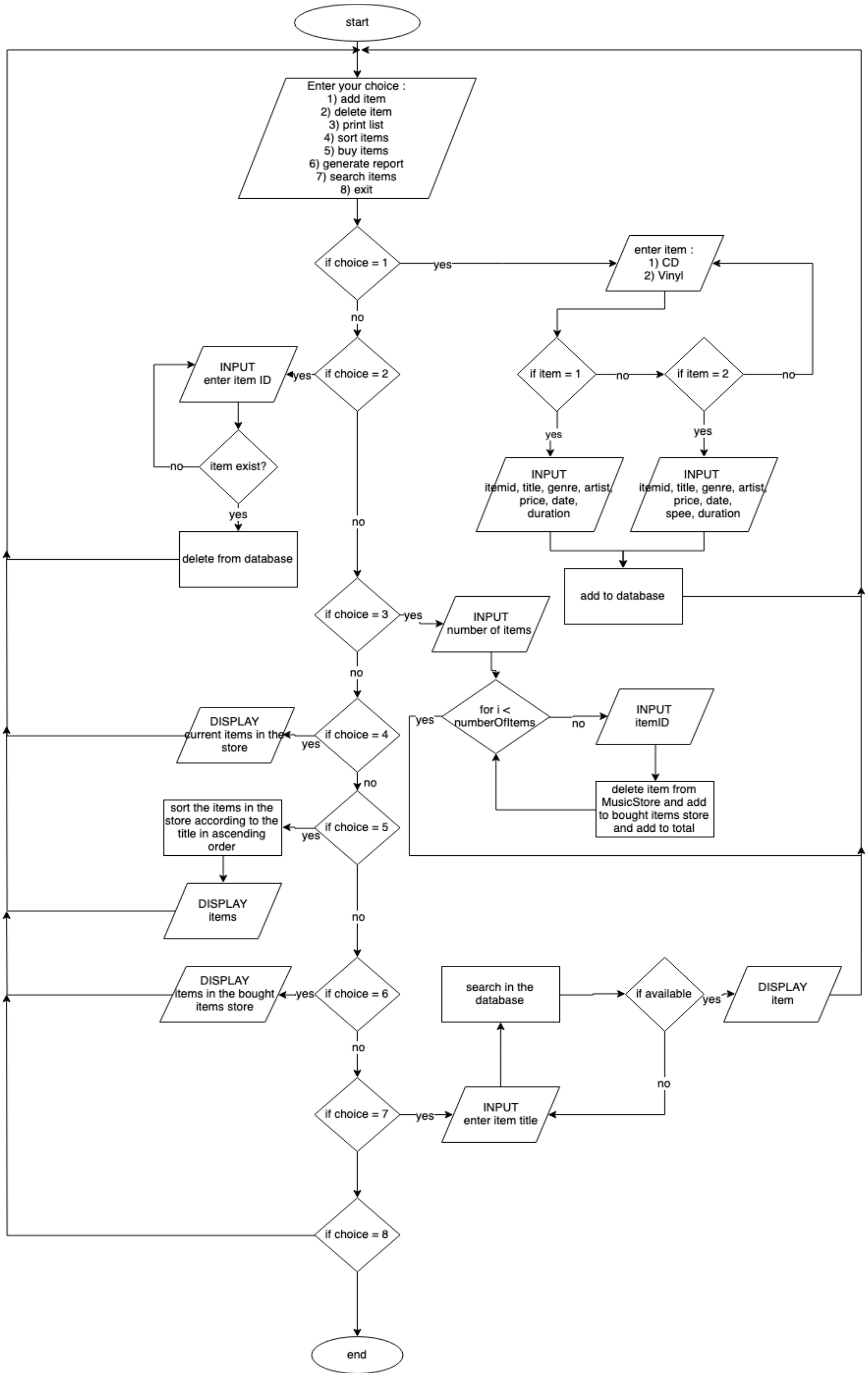
4) Buy items (user should enter a valid item id)

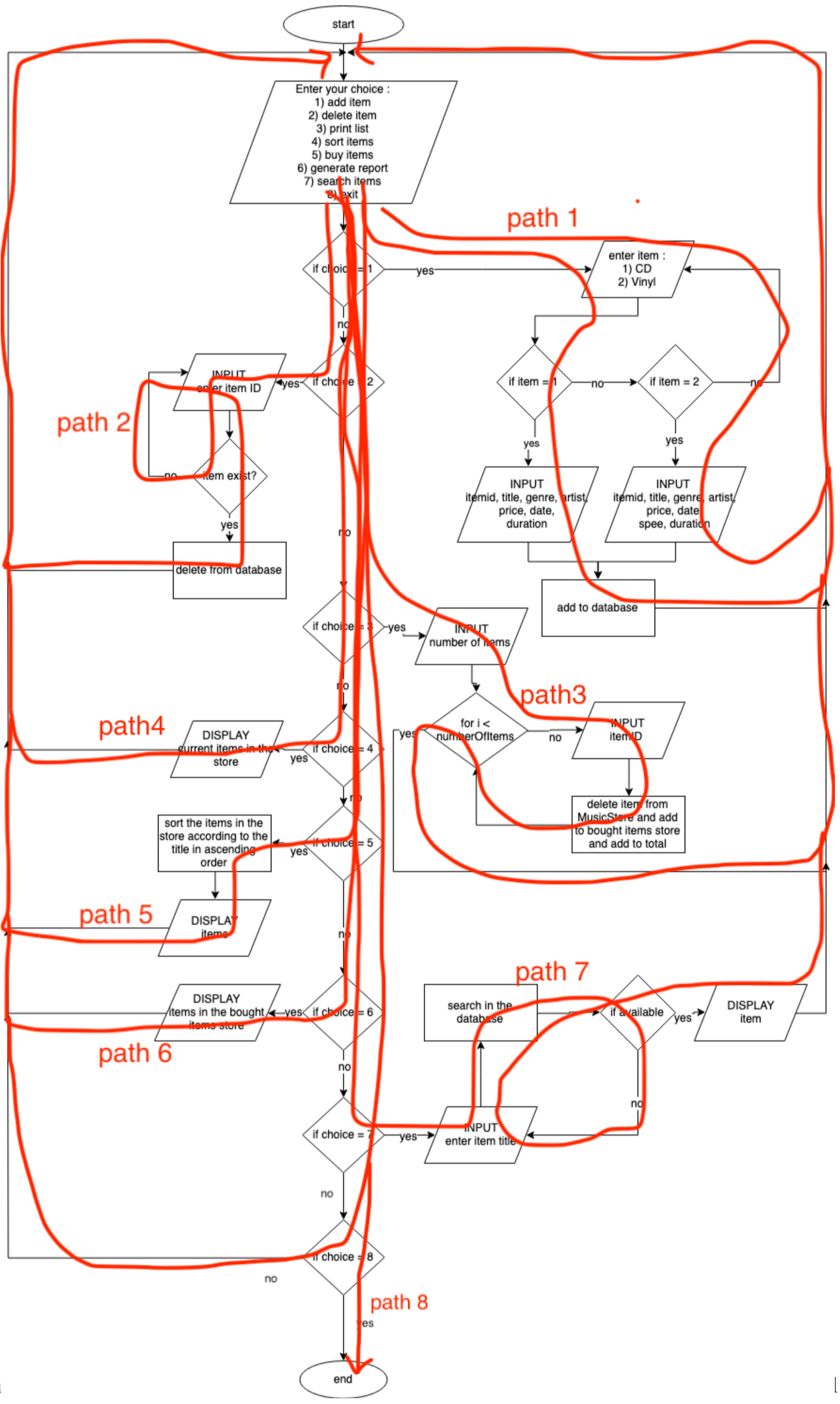
Input	Expected Value	Actual Output	Bug?
70	Item not found	Item not found	No
1	Pass	Pass	No
photograph	Item not found	Item not found	No

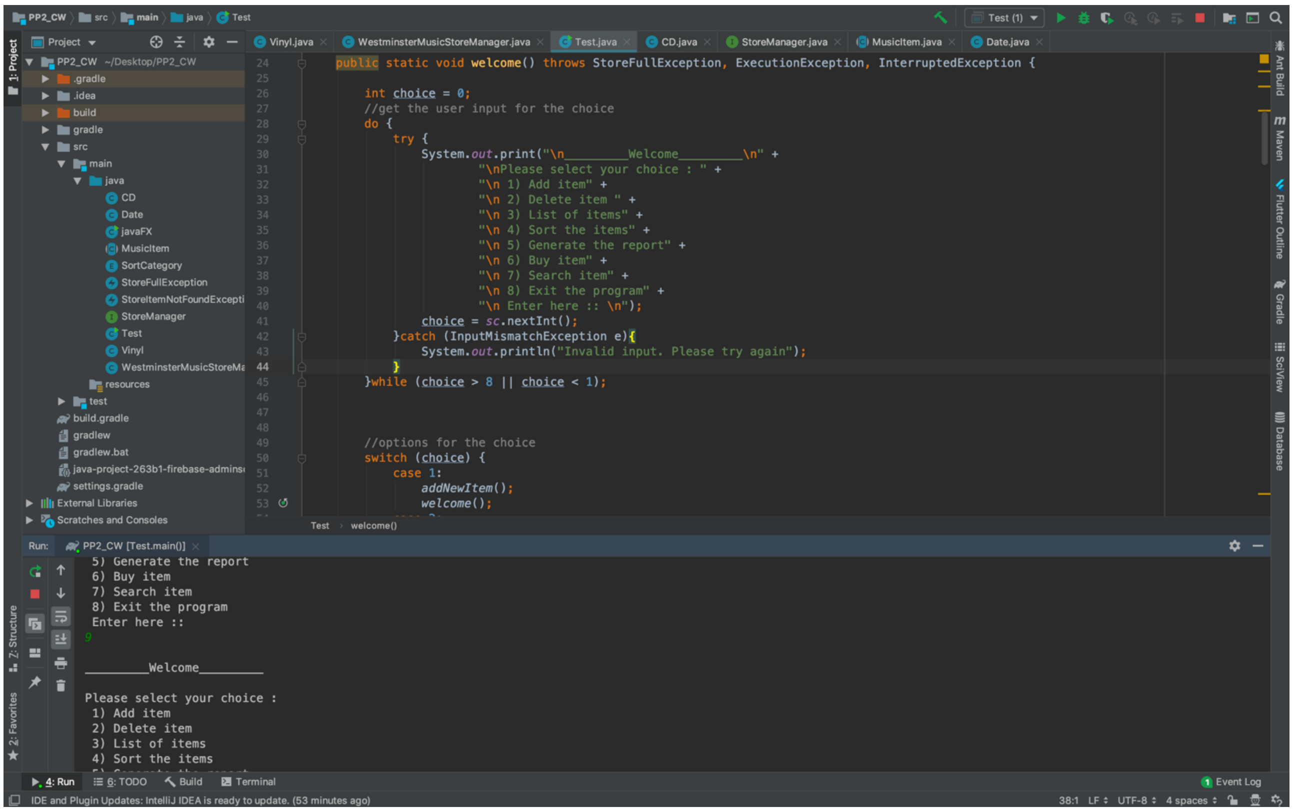
5) Delete items (user should enter a valid item id)

Input	Expected Value	Actual Output	Bug?
70	Item not found	Item not found	No
1	Pass	Pass	No
photograph	Item not found	Item not found	No

b. White Box

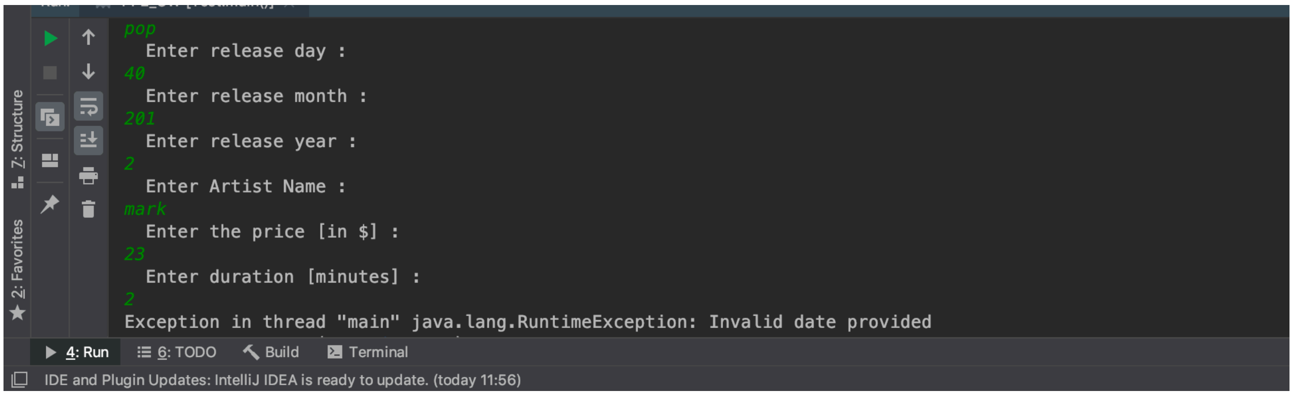

start
Enter your choice :
1) add item
2) delete item
3) print list
4) sort items
5) buy items
6) generate report
7) search items
8) exit
if choice = 1
yes
enter item :
1) CD
2) Vinyl
no
if choice = 2
yes
INPUT
enter item ID
item exist?
no
yes
delete from database
if item = 1
no
if item = 2
no
yes
yes
INPUT
itemid, title, genre, artist, price, date, duration
INPUT
itemid, title, genre, artist, price, date, spee, duration
add to database
no
if choice = 3
yes
INPUT
number of items
no
for i < numberOfItems
yes
no
INPUT
itemID
delete item from MusicStore and add to bought items store and add to total
DISPLAY
current items in the store
yes
if choice = 4
no
sort the items in the store according to the title in ascending order
yes
if choice = 5
DISPLAY
items
no
DISPLAY
items in the bought items store
yes
if choice = 6
search in the database
if available
yes
DISPLAY
item
no
no
if choice = 7
yes
INPUT
enter item title
if choice = 8
end


start
Enter your choice :
1) add item
2) delete item
3) print list
4) sort items
5) buy items
6) generate report
7) search items
8) exit
if choice = 1
yes
no
enter item :
1) CD
2) Vinyl
if item = 1
no
if item = 2
no
yes
yes
INPUT
itemid, title, genre, artist, price, date, duration
INPUT
itemid, title, genre, artist, price, date, spee, duration
add to database
if choice = 2
yes
INPUT
enter item ID
item exist?
no
yes
delete from database
no
if choice = 3
yes
INPUT
number of items
for i < numberOfItems
yes
no
INPUT
itemID
delete item from MusicStore and add to bought items store and add to total
no
if choice = 4
yes
DISPLAY
current items in the store
no
if choice = 5
yes
sort the items in the store according to the title in ascending order
DISPLAY
items
no
if choice = 6
yes
DISPLAY
items in the bought items store
no
if choice = 7
yes
INPUT
enter item title
search in the database
if available
yes
DISPLAY
item
no
no
if choice = 8
no
yes
end
path 1
path 2
path3
path4
path 5
path 6
path 7
path 8

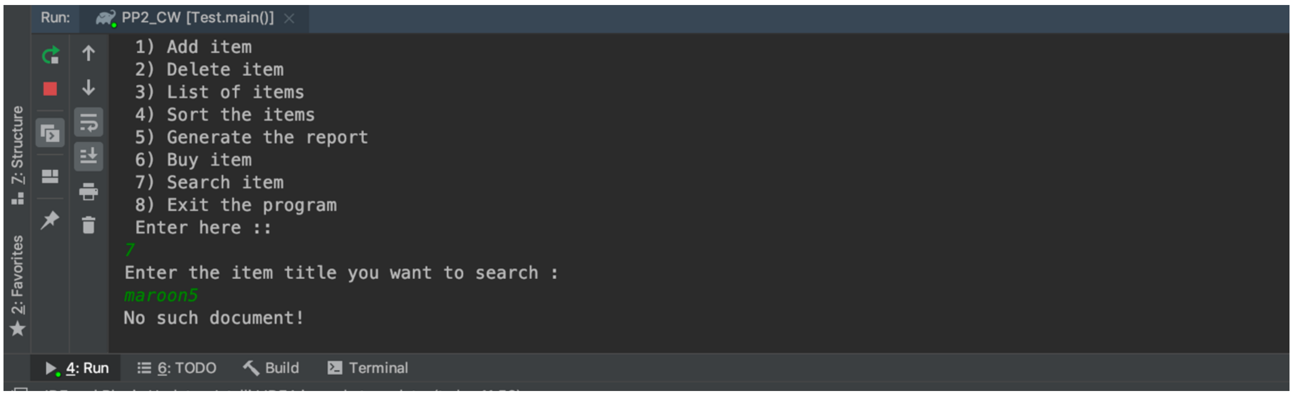


```java
public static void welcome() throws StoreFullException, ExecutionException, InterruptedException {

    int choice = 0;
    //get the user input for the choice
    do {
        try {
            System.out.print("\n_________Welcome_________\n" +
                    "\nPlease select your choice : " +
                    "\n 1) Add item" +
                    "\n 2) Delete item " +
                    "\n 3) List of items" +
                    "\n 4) Sort the items" +
                    "\n 5) Generate the report" +
                    "\n 6) Buy item" +
                    "\n 7) Search item" +
                    "\n 8) Exit the program" +
                    "\n Enter here :: \n");
            choice = sc.nextInt();
        }catch (InputMismatchException e){
            System.out.println("Invalid input. Please try again");
        }
    }while (choice > 8 || choice < 1);



    //options for the choice
    switch (choice) {
        case 1:
            addNewItem();
            welcome();
```

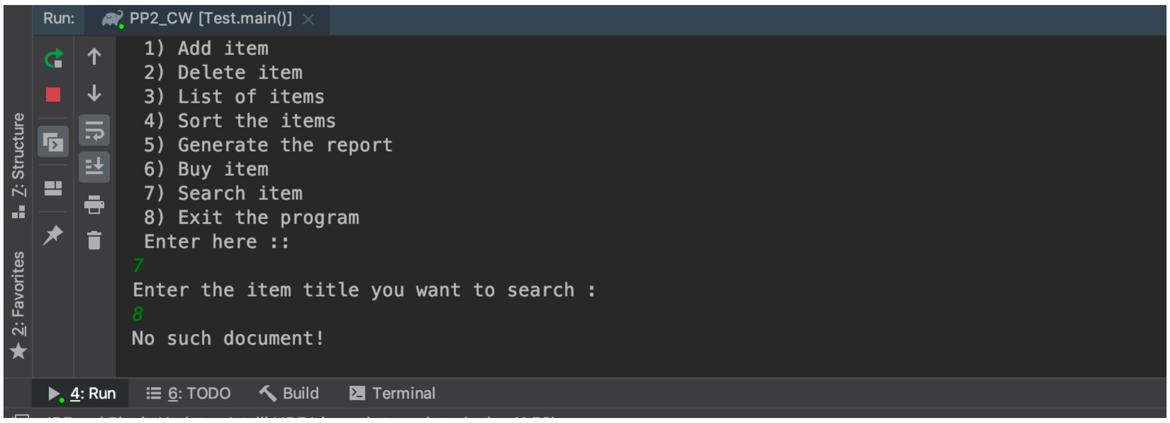
```
 5) Generate the report
 6) Buy item
 7) Search item
 8) Exit the program
 Enter here ::
9

_________Welcome_________

Please select your choice :
 1) Add item
 2) Delete item
 3) List of items
 4) Sort the items
```



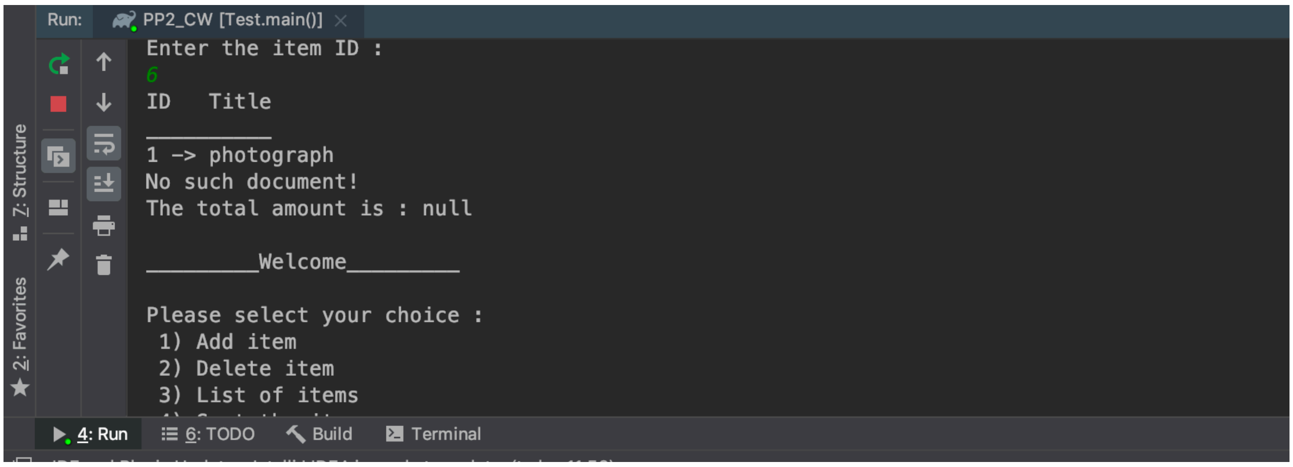
```
pop
  Enter release day :
40
  Enter release month :
201
  Enter release year :
2
  Enter Artist Name :
mark
  Enter the price [in $] :
23
  Enter duration [minutes] :
2
Exception in thread "main" java.lang.RuntimeException: Invalid date provided
```



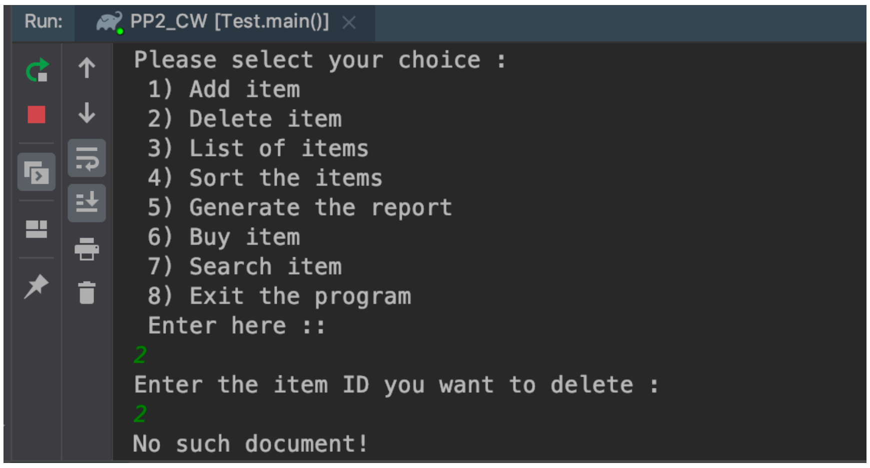
```
 1) Add item
 2) Delete item
 3) List of items
 4) Sort the items
 5) Generate the report
 6) Buy item
 7) Search item
 8) Exit the program
 Enter here ::
7
Enter the item title you want to search :
maroon5
No such document!
```



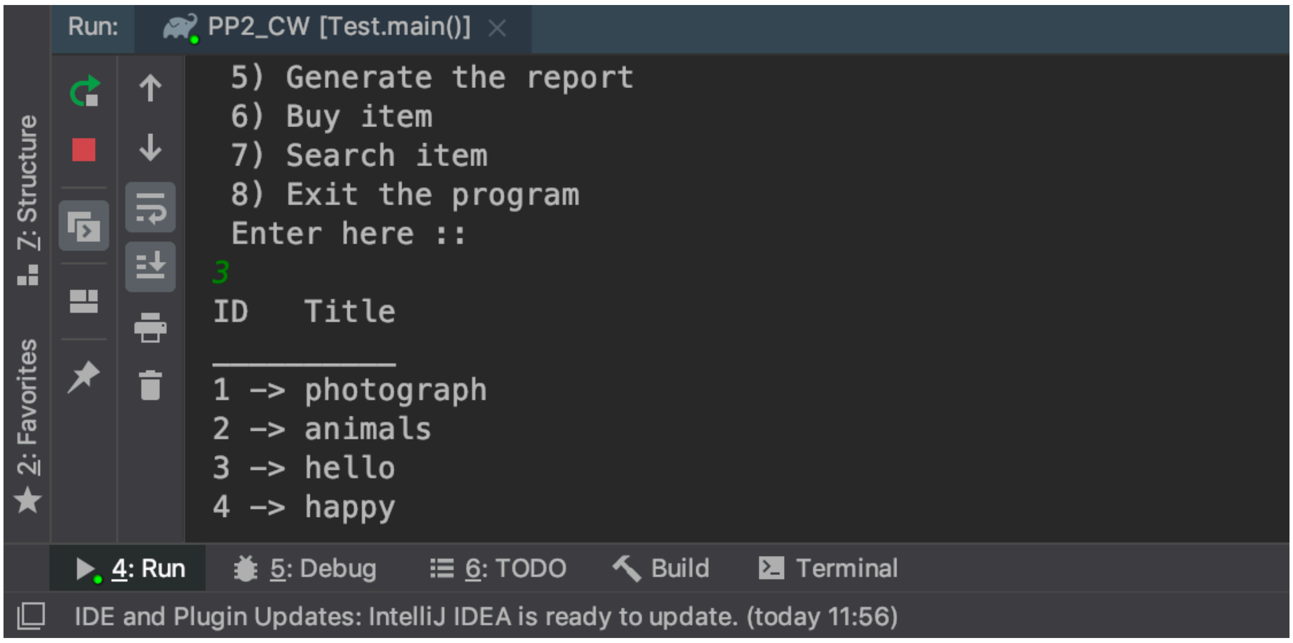
```
 1) Add item
 2) Delete item
 3) List of items
 4) Sort the items
 5) Generate the report
 6) Buy item
 7) Search item
 8) Exit the program
 Enter here ::
7
Enter the item title you want to search :
8
No such document!
```



```
Enter the item ID :
6
ID   Title
__________
1 -> photograph
No such document!
The total amount is : null

_________Welcome_________

Please select your choice :
 1) Add item
 2) Delete item
 3) List of items
```

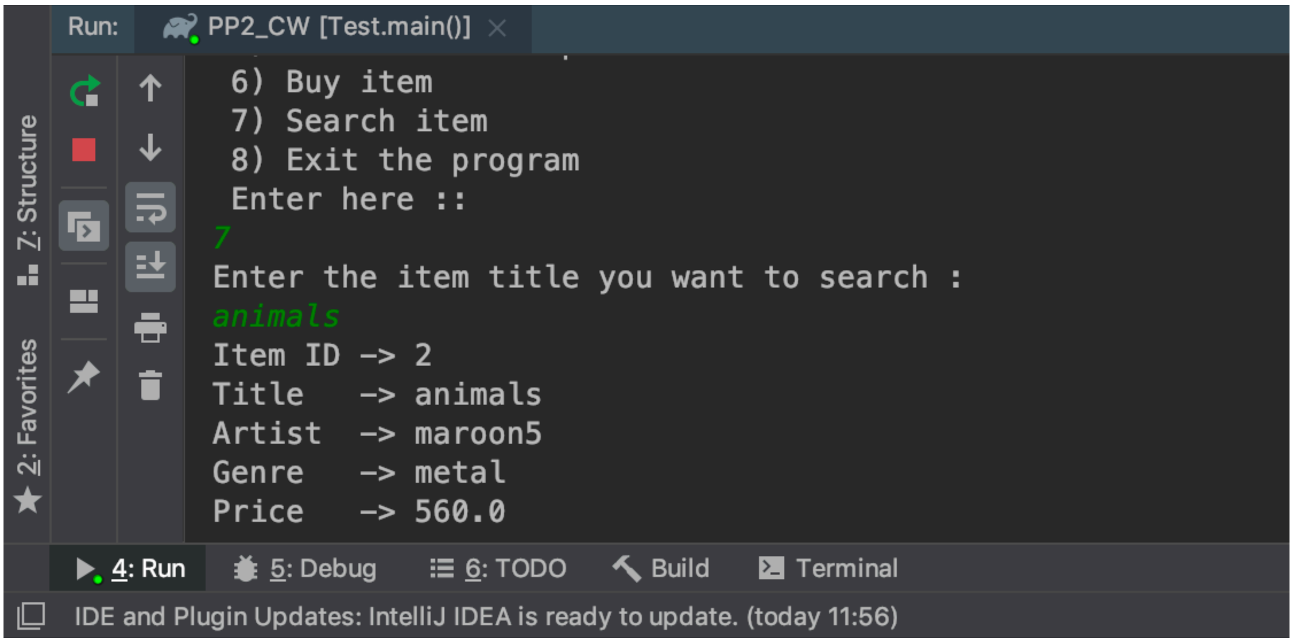


```
Please select your choice :
 1) Add item
 2) Delete item
 3) List of items
 4) Sort the items
 5) Generate the report
 6) Buy item
 7) Search item
 8) Exit the program
 Enter here ::
2
Enter the item ID you want to delete :
2
No such document!
```



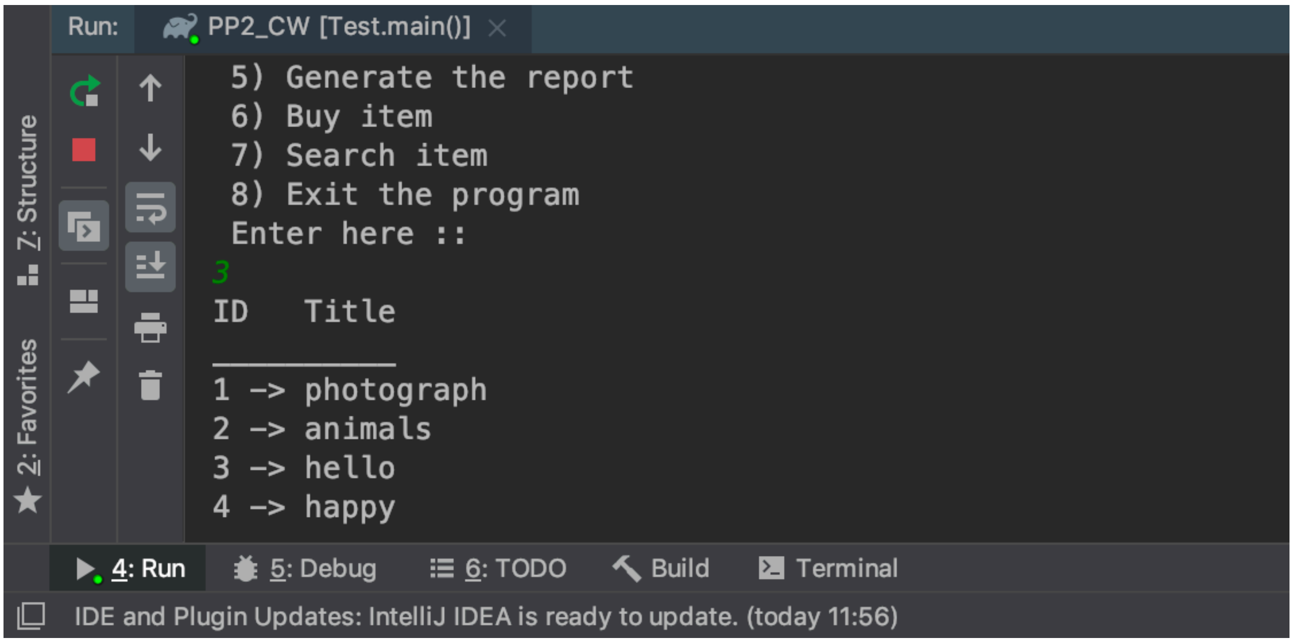
```
  5) Generate the report
  6) Buy item
  7) Search item
  8) Exit the program
  Enter here ::
3
ID   Title
__________
1 -> photograph
2 -> animals
3 -> hello
4 -> happy
```

IDE and Plugin Updates: IntelliJ IDEA is ready to update. (today 11:56)



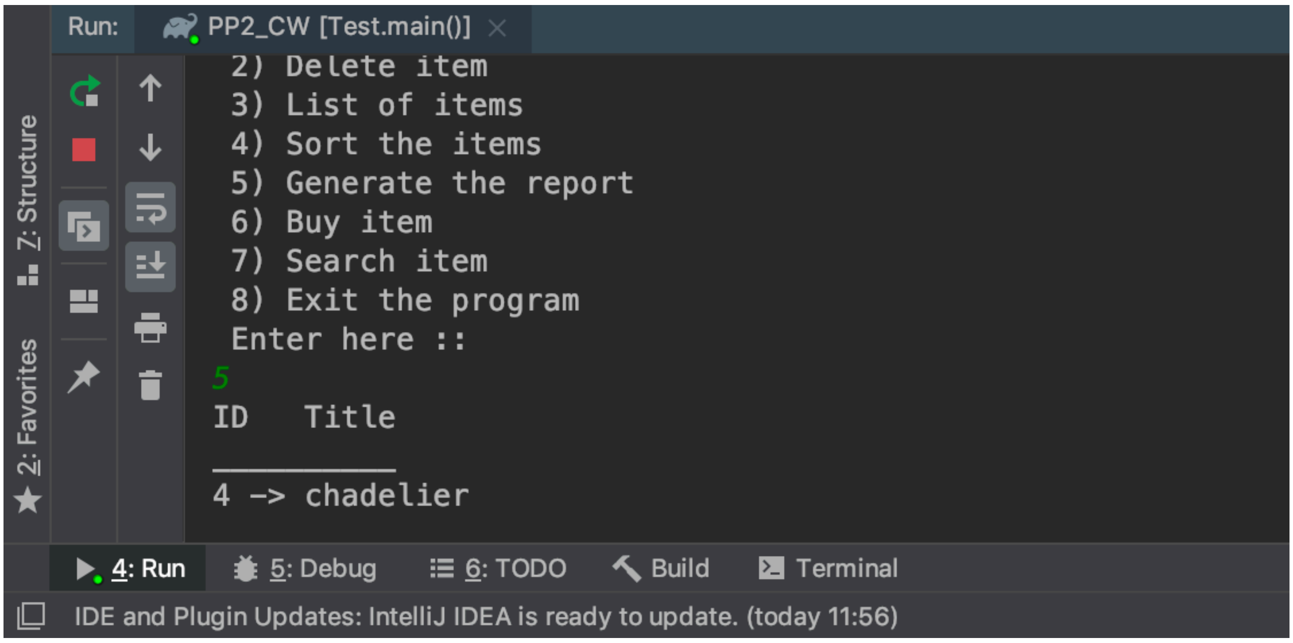
```
  6) Buy item
  7) Search item
  8) Exit the program
  Enter here ::
7
Enter the item title you want to search :
animals
Item ID -> 2
Title   -> animals
Artist  -> maroon5
Genre   -> metal
Price   -> 560.0
```

IDE and Plugin Updates: IntelliJ IDEA is ready to update. (today 11:56)



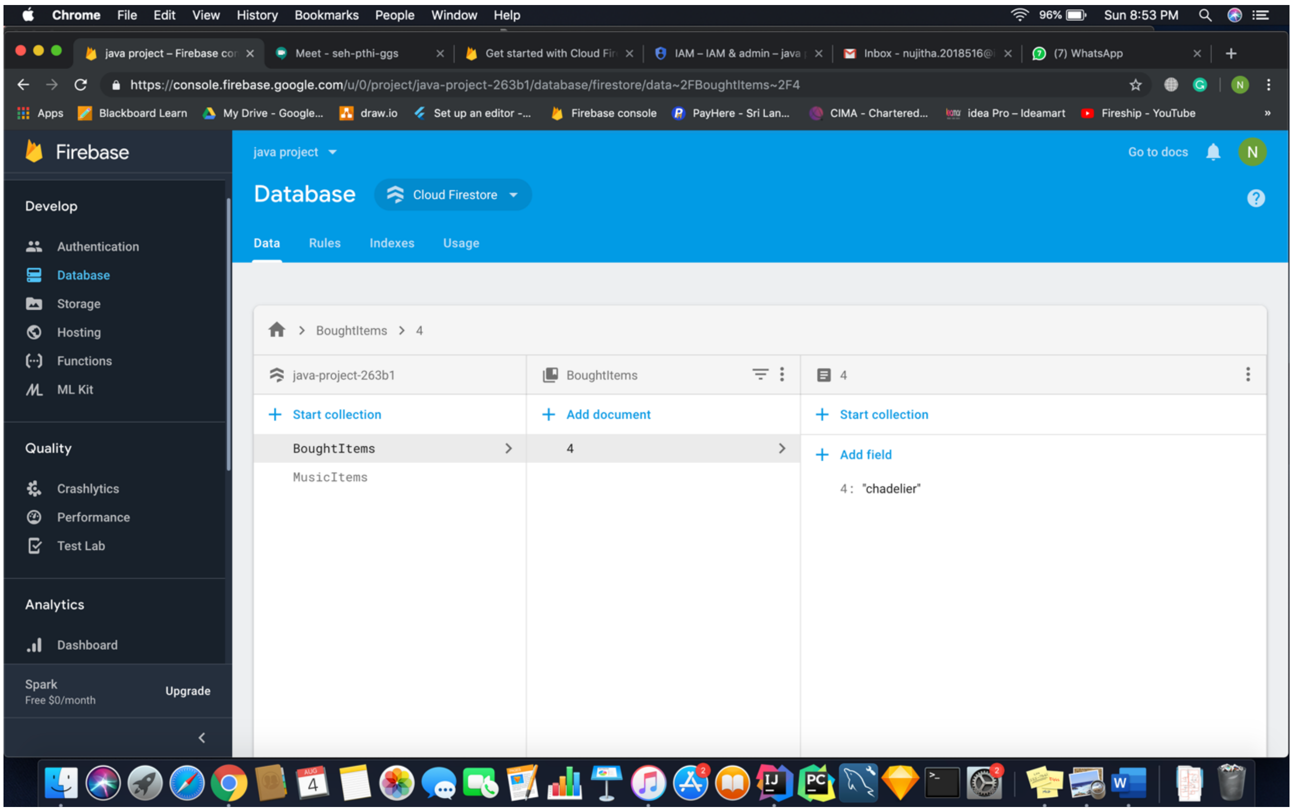
```
  5) Generate the report
  6) Buy item
  7) Search item
  8) Exit the program
  Enter here ::
3
ID   Title
__________
1 -> photograph
2 -> animals
3 -> hello
4 -> happy
```

IDE and Plugin Updates: IntelliJ IDEA is ready to update. (today 11:56)



```
  2) Delete item
  3) List of items
  4) Sort the items
  5) Generate the report
  6) Buy item
  7) Search item
  8) Exit the program
  Enter here ::
5
ID   Title
__________
4 -> chadelier
```

IDE and Plugin Updates: IntelliJ IDEA is ready to update. (today 11:56)



https://console.firebase.google.com/u/0/project/java-project-263b1/database/firestore/data~2FBoughtItems~2F4

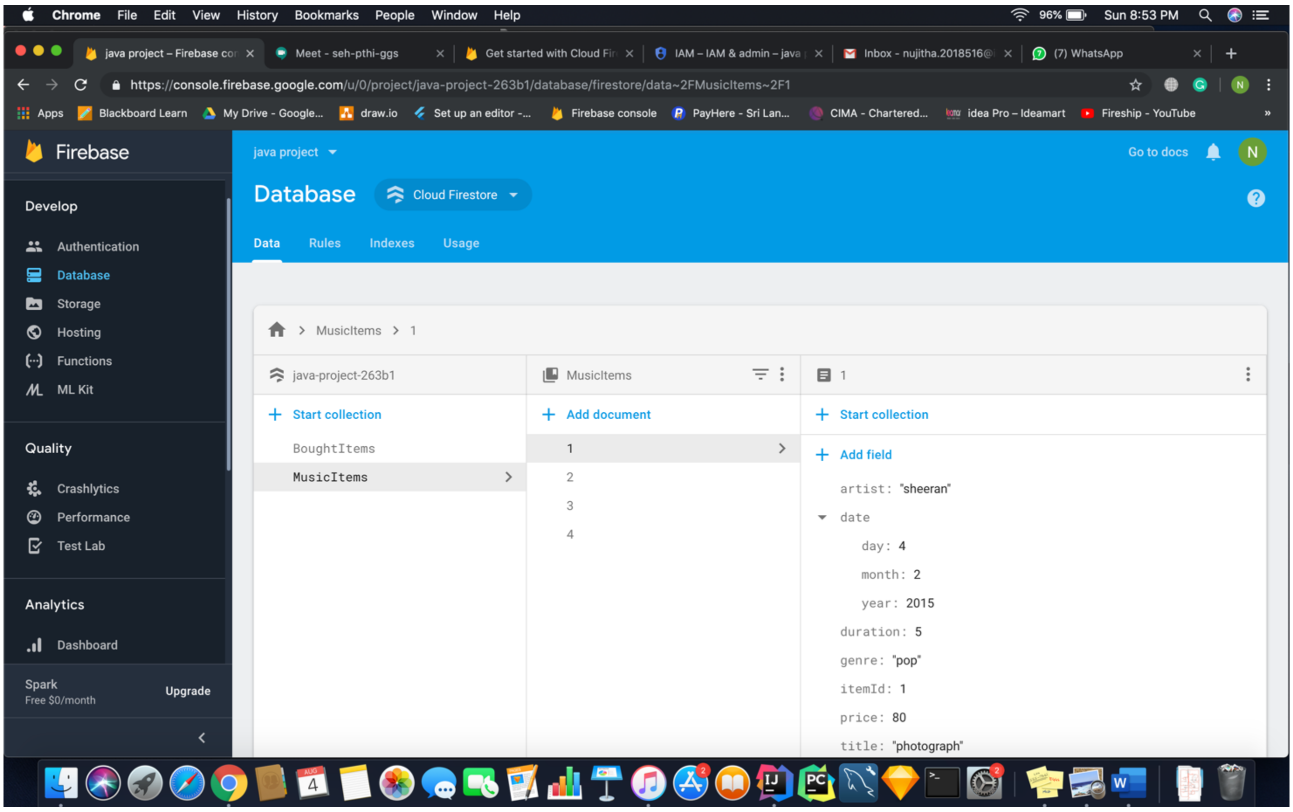
BoughtItems > 4

java-project-263b1 | BoughtItems | 4

BoughtItems
MusicItems

4

4: "chadelier"



https://console.firebase.google.com/u/0/project/java-project-263b1/database/firestore/data~2FMusicItems~2F1

MusicItems > 1

java-project-263b1 | MusicItems | 1

BoughtItems
MusicItems

1
2
3
4

```
artist: "sheeran"
date
    day: 4
    month: 2
    year: 2015
duration: 5
genre: "pop"
itemId: 1
price: 80
title: "photograph"
```

Code

Test.java

```
import com.google.cloud.firestore.Firestore;

import java.util.InputMismatchException;
import java.util.Scanner;
import java.util.concurrent.ExecutionException;

public class Test {

    private static Scanner sc = new Scanner(System.in);
    private static Firestore db;

    private static WestminsterMusicStoreManager westminsterMusicStoreManager;

    public static void main(String[] args) {
        westminsterMusicStoreManager = new WestminsterMusicStoreManager();
        try {
            welcome();
        } catch (StoreFullException | ExecutionException | InterruptedException
e) {
            e.printStackTrace();
        }
    }

    public static void welcome() throws StoreFullException, ExecutionException,
InterruptedException {

        int choice = 0;
        //get the user input for the choice
        do {
            try {
                System.out.print("\n_________Welcome_________\n" +
                        "\nPlease select your choice : " +
                        "\n 1) Add item" +
                        "\n 2) Delete item " +
                        "\n 3) List of items" +
                        "\n 4) Sort the items" +
                        "\n 5) Generate the report" +
                        "\n 6) Buy item" +
                        "\n 7) Search item" +
                        "\n 8) Exit the program" +
                        "\n Enter here :: \n");
                choice = sc.nextInt();
            } catch (InputMismatchException e) {
                System.out.println("Invalid input. Please try again");
            }
        } while (choice > 8 || choice < 1);

        //options for the choice
        switch (choice) {
            case 1:
                addNewItem();
                welcome();
            case 2:
                deleteItem();
                welcome();
            case 3:
                try {
                    listOfItems();
                } catch (ExecutionException | InterruptedException e) {
                    e.printStackTrace();
                }
                welcome();
            case 4:
                sortItems();
                welcome();
            case 5:
                generateReport();
                welcome();
            case 6:
                buyItem();
                welcome();
            case 7:
                searchItem();
                welcome();
            case 8:
                System.out.println("\n ________Thank You________");
                System.exit(0);

            default:
                System.out.println("you might have mistaken, check again");
        }
    }

    //method to add an item
    private static void addNewItem() {

        System.out.println("Do you want to ad  a CD or a Vinyl? " +
                "\n 1) Add a CD" +
                "\n 2) Add a Vinyl" +
                "\nEnter here : ");
        int choice = sc.nextInt();
        if (choice == 1) {

            System.out.println("please enter the following information :");

            //get the item id
            System.out.println("  Enter the item ID : ");
            int itemId = sc.nextInt();

            //get the item title
            System.out.println("  Enter the item Title : ");
            String title = sc.next();

            //get the item genre
            System.out.println("  Enter the item Genre : ");
            String genre = sc.next();

            //enter date
            System.out.println("  Enter release day : ");
            int day = sc.nextInt();
            System.out.println("  Enter release month : ");
            int month = sc.nextInt();

            System.out.println("  Enter release year : ");
            int year = sc.nextInt();

            //enter artist
            System.out.println("  Enter Artist Name : ");
            String artist = sc.next();

            //enter price
            System.out.println("  Enter the price [in $] : ");
            int price = sc.nextInt();

            //enter duration
            System.out.println("  Enter duration [minutes] : ");
            int duration = sc.nextInt();

            MusicItem cd = new CD(itemId, title, genre, new Date(day, month,
year), artist, price, duration);

            westminsterMusicStoreManager.addItem(cd);


        } else if (choice == 2) { //option for the vinyl

            System.out.println("please enter the following information :");

            //get the item id
            System.out.println("  Enter the item ID : ");
            int itemId = sc.nextInt();

            //get the item title
            System.out.println("  Enter the item Title : ");
            String title = sc.next();

            //get the item genre
            System.out.println("  Enter the item Genre : ");
            String genre = sc.next();

            //enter date
            System.out.println("  Enter release day : ");
            int day = sc.nextInt();

            System.out.println("  Enter release month : ");
            int month = sc.nextInt();

            System.out.println("  Enter release year : ");
            int year = sc.nextInt();

            //enter artist
            System.out.println("  Enter Artist Name : ");
            String artist = sc.next();

            //enter price
            System.out.println("  Enter the price [in $]: ");
            int price = sc.nextInt();

            //enter speed
            System.out.println("  Enter the speed [KB/sec] : ");
            int speed = sc.nextInt();

            //enter diameter
            System.out.println("  Enter the Diameter [cm]: ");
            int dia = sc.nextInt();

            //create the object
            MusicItem vinyl = new Vinyl(itemId, title, genre, new Date(day,
month, year), artist, price, speed, dia);

            westminsterMusicStoreManager.addItem(vinyl);

        } else {
            System.out.println("You entered an invalid input. Please re-enter");
            //recursive calling
            addNewItem();
        }

    }

    //method to delete an item
    private static void deleteItem() {
        System.out.println("Enter the item ID you want to delete : ");
        int deleteThis = sc.nextInt();
        westminsterMusicStoreManager.deleteItem(deleteThis);
    }

    //displays the items which are currently in the store
    private static void listOfItems() throws ExecutionException,
InterruptedException {
        westminsterMusicStoreManager.printList();
    }

    //sort the items in the order of title
    private static void sortItems() throws ExecutionException,
InterruptedException {
        westminsterMusicStoreManager.sortItems();
    }

    //buy items method
    private static void buyItem() throws ExecutionException,
InterruptedException {
        //view list of items before buying
        westminsterMusicStoreManager.printList();
        System.out.println("How many items do you want to buy? ");
        int noOfItems = sc.nextInt();
        //loop for number of items entered
        for (int i = 0; i < noOfItems; i++) {
            System.out.println("Enter the item ID :");
            int id = sc.nextInt();
            westminsterMusicStoreManager.checkData(id);
        }
        westminsterMusicStoreManager.buyItems();
    }

    //displays the bought items
    private static void generateReport() throws ExecutionException,
InterruptedException {
        westminsterMusicStoreManager.generateReport();
    }
    //prompts for the user to search an item
    public static void searchItem() throws ExecutionException,
InterruptedException {
        System.out.println("Enter the item title you want to search : ");
        String search = sc.next();
        westminsterMusicStoreManager.searchItem(search);
    }
}
```

WestminsterMusicStoreManager.java

```
import com.google.api.core.ApiFuture;
import com.google.auth.oauth2.GoogleCredentials;
import com.google.cloud.firestore.*;
import com.google.firebase.FirebaseApp;
import com.google.firebase.FirebaseOptions;
import com.google.firebase.cloud.FirestoreClient;

import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.util.*;
import java.util.concurrent.ExecutionException;

public class WestminsterMusicStoreManager implements StoreManager {

    private int currCount = 0;
    //firestore initializing
    private static Firestore db;
    private static final int count = 1000;
    public Double total;

    public WestminsterMusicStoreManager() {
        try {
            setupFirebase();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void addItem(MusicItem item) {
        currCount++;

db.collection("MusicItems").document(String.valueOf(item.getItemId())).set(item)
;
    }

    @Override
    public void deleteItem(int id) {
        try {
            firestoreDelete(id);
            currCount--;
            System.out.println("Number of free spaces left : " + (count -
currCount));
        } catch (ExecutionException | InterruptedException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void printList() throws ExecutionException, InterruptedException {
        // Create a query against the collection.
        Query query = db.collection("MusicItems");
        // retrieve  query results asynchronously using query.get()
        ApiFuture<QuerySnapshot> querySnapshot = query.get();
        System.out.println("ID   Title\n__________");

        for (DocumentSnapshot document : querySnapshot.get().getDocuments()) {
            System.out.println(document.getId() + " -> " +
document.getString("title"));
        }
    }

    @Override
    public void sortItems() throws ExecutionException, InterruptedException {
        // Create a reference to the MusicItems collection and query against the
collection.
        Query query = db.collection("MusicItems").orderBy("title");
        // retrieve  query results asynchronously using query.get()
        ApiFuture<QuerySnapshot> querySnapshot = query.get();
        System.out.println("ID   Title\n__________");
        //displays all the items in the store after sorting them out
        for (DocumentSnapshot document : querySnapshot.get().getDocuments()) {
            System.out.println(document.getId() + " -> " +
document.getString("title"));
        }
    }

    @Override
    public void buyItems() {
        System.out.println("The total amount is : " + total);
    }

    @Override
    public void generateReport() throws ExecutionException, InterruptedException
{
        // Create a query against the collection.
        Query query = db.collection("BoughtItems");
        // retrieve  query results asynchronously using query.get()
        ApiFuture<QuerySnapshot> querySnapshot = query.get();
        System.out.println("ID   Title\n__________");

        for (DocumentSnapshot document : querySnapshot.get().getDocuments()) {
            System.out.println(document.getId() + " -> " +
document.getString(document.getId()));
        }
    }

    //setting up the firebase connection
    private static void setupFirebase() throws IOException {
        InputStream serviceAccount = new FileInputStream("java-project-263b1-
firebase-adminsdk-f3f30-a5df91161f.json");
        GoogleCredentials credentials =
GoogleCredentials.fromStream(serviceAccount);
        FirebaseOptions options = new FirebaseOptions.Builder()
                .setCredentials(credentials)
                .build();
        FirebaseApp.initializeApp(options);
        db = FirestoreClient.getFirestore();
    }

    //method to delete items from the firebase
    public void firestoreDelete(int id) throws ExecutionException,
InterruptedException {
        //refers the given document in the firestore for the given id
        DocumentReference docRef =
db.collection("MusicItems").document(String.valueOf(id));
        ApiFuture<DocumentSnapshot> future = docRef.get();
        DocumentSnapshot document = future.get();
        if (document.exists()) {
            db.collection("MusicItems").document(String.valueOf(id)).delete();
            System.out.println("Item deleted successfully..!!");
        } else {
            System.out.println("No such document!");
        }
    }

    //check if such item exists in the firestore before buying
    public void checkData(int itemId) throws ExecutionException,
InterruptedException {
        printList();
        //refers the given document in the firestore for the given id
        DocumentReference docRef =
db.collection("MusicItems").document(String.valueOf(itemId));
        ApiFuture<DocumentSnapshot> ref = docRef.get();
        DocumentSnapshot document = ref.get();
        if (document.exists()) {
            Map<String, Object> BoughtItems = new HashMap<>();
            BoughtItems.put(document.getId(), document.getString("title"));

db.collection("BoughtItems").document(String.valueOf(itemId)).set(BoughtItems);
            firestoreDelete(itemId);
            System.out.println("Item added to the cart...");
        } else {
            System.out.println("No such document!");
        }
    }

    public void searchItem(String title) throws ExecutionException,
InterruptedException {
        //asynchronously retrieve multiple documents
        ApiFuture<QuerySnapshot> future =
db.collection("MusicItems").whereEqualTo("title", title).get();
        // future.get() blocks on response
        List<QueryDocumentSnapshot> documents = future.get().getDocuments();
        for (DocumentSnapshot document : documents) {
            if (document.exists()) {
                System.out.println("Item ID -> " + document.getId());
                System.out.println("Title   -> " + document.getString("title"));
                System.out.println("Artist  -> " +
document.getString("artist"));
                System.out.println("Genre   -> " + document.getString("genre"));
                System.out.println("Price   -> " + document.get("price"));
                break;
            } else {
                System.out.println("No such document!");
            }
        }
    }

}
```

CD.java

```
import java.util.Objects;

public class CD extends MusicItem {

    private int duration;

    public CD(int itemId, String title, String genre, Date date, String artist,
int price, int duration) {
        super(itemId, title, genre, date, artist, price);
        this.duration = duration;
    }

    public void setDuration(int duration) {
        this.duration = duration;
    }

    public int getDuration() {
        return duration;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        if (!super.equals(o)) return false;
        CD cd = (CD) o;
        return duration == cd.duration;
    }

    @Override
    public int hashCode() {
        return Objects.hash(super.hashCode(), duration);
    }

    @Override
    public String toString() {
        return "CD{" +
                "duration=" + duration +
                ", itemId=" + itemId +
                ", title='" + title + '\'' +
                ", genre='" + genre + '\'' +
                ", date=" + date +
                ", artist='" + artist + '\'' +
                ", price=" + price +
                '}';
    }
}
```

MusicItem.java

```
import java.util.Objects;

public abstract class MusicItem implements Comparable<MusicItem> {
    protected int itemId;
    protected String title;
    protected String genre;
    protected Date date;
    protected String artist;
    protected double price;

    public MusicItem(int itemId, String title, String genre, Date date, String
artist, int price) {
        this.itemId = itemId;
        this.title = title;
        this.genre = genre;
        this.date = date;
        this.artist = artist;
        this.price = price;

    }

    public int getItemId() {
        return itemId;
    }

    public void setItemId(int itemId) {
        this.itemId = itemId;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getGenre() {
        return genre;
```

```java
    }

    public void setGenre(String genre) {
        this.genre = genre;
    }

    public Date getDate() {
        return date;
    }

    public void setDate(Date date) {
        this.date = date;
    }

    public String getArtist() {
        return artist;
    }

    public void setArtist(String artist) {
        this.artist = artist;
    }

    public double getPrice() {
        return price;
    }

    public void setPrice(double price) {
        this.price = price;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        MusicItem musicItem = (MusicItem) o;
        return itemId == musicItem.itemId &&
                Double.compare(musicItem.price, price) == 0 &&
                Objects.equals(title, musicItem.title) &&
                Objects.equals(genre, musicItem.genre) &&
                Objects.equals(date, musicItem.date) &&
                Objects.equals(artist, musicItem.artist);
    }

    @Override
    public int hashCode() {
        return Objects.hash(itemId, title, genre, date, artist, price);
    }

    @Override
    public int compareTo(MusicItem o) {
        //sort using title
        return title.compareTo(o.getTitle());

    }

    @Override
    public String toString() {
        return "MusicItem{" +
                "itemId=" + itemId +
                ", title='" + title + '\'' +
                ", genre='" + genre + '\'' +
                ", date=" + date +
                ", artist='" + artist + '\'' +
                ", price=" + price +
                '}';
    }
}
```

Date.java

```
public class Date {
    private int day;
    private int month;
    private int year;

    //constructor
    public Date(int day, int month, int year) {
        if(day >= 1 && day <= 31){
            this.day = day;
        } else {
            throw new RuntimeException("Invalid date provided");

        }

        if (month < 13 && month > 0){
            this.month = month;
        }else {
            throw new IllegalArgumentException("Check again..!! You entered an
invalid month");
        }

        this.year = year;

    }

    public int getDay() {
        return day;
    }

    public void setDay(int day) {
        this.day = day;
    }

    public int getMonth() {
        return month;
    }

    public void setMonth(int month) {
        this.month = month;
    }
    public int getYear() {
        return year;
    }

    public void setYear(int year) {
        this.year = year;
    }

}
```

Vinyl.java

```
import java.util.Objects;

public class Vinyl extends MusicItem {

    private int speed; //data type is double
    private int diameter; //date type is double

    public Vinyl(int itemId, String title, String genre, Date date, String
artist, int price, int speed, int diameter) {
        super(itemId, title, genre, date, artist, price);
        this.speed = speed;
        this.diameter = diameter;
    }

    public int getSpeed() {
        return speed;
    }

    public int getDiameter() {
        return diameter;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        if (!super.equals(o)) return false;
        Vinyl vinyl = (Vinyl) o;
        return speed == vinyl.speed &&
                diameter == vinyl.diameter;
    }

    @Override
    public int hashCode() {
        return Objects.hash(super.hashCode(), speed, diameter);
    }
    @Override
    public String toString() {
        return "Vinyl{" +
                "speed=" + speed +
                ", diameter=" + diameter +
                ", itemId=" + itemId +
                ", title='" + title + '\'' +
                ", genre='" + genre + '\'' +
                ", date=" + date +
                ", artist='" + artist + '\'' +
                ", price=" + price +
                '}';
    }
}
```

StoreMnager.java

```
import java.util.concurrent.ExecutionException;

public interface StoreManager {
    void addItem(MusicItem item);

    void deleteItem(int id);

    void printList() throws ExecutionException, InterruptedException;

    void sortItems() throws ExecutionException, InterruptedException;

    void buyItems() throws ExecutionException, InterruptedException;

    void generateReport() throws ExecutionException, InterruptedException;

}
```