# Design Document Template and Example

The **Design Document** is a **Microsoft** **Word** **document** that describes the overall design of the solution to a programming task. It provides a statement of the problem, lists the inputs the program requires and the outputs it produces, identifies the classes into which the task is divided and lists the responsibilities of each. It has a UML diagram that outlines the classes and their relationships to each other. The design includes the algorithms used in the program unless they are self-evident in the code (for example, the algorithm for a getter method need not be described unless there is more to it than simply returning the value of a specific attribute). The design document concludes with a set of test cases that will thoroughly test the solution you develop along with the expected (correct) output for each of the test cases. The document may include pictures, graphs, diagrams, or other items which have been copied from other sources (be sure to document your sources in this case).

## Document Layout

The design document (using MS Word with diagrams copied/pasted into the document if needed) should have 6 sections as described in this table.

|  |  |
| --- | --- |
| Layout of the Design Document | |
| Section 1: | Statement of the problem to be solved |
| Section 2: | List of inputs, outputs, and processing required |
| Section 3: | Identification of classes needed and a list of their individual responsibilities |
| Section 4: | UML class diagram showing the classes and depicting the relationships among the classes |
| Section 5: | Design of the main and major algorithms |
| Section 6: | List of test cases: the data and expected results used to verify the final product is working as expected |

### Test Cases/Trace Table

A list of test cases is a list of all inputs with which you plan to test the program along with the correct answers that you expect the program to produce. The list of test cases should include some invalid and/or boundary cases that you expect the program to handle appropriately. For example, if the problem to be solved is determining the GPA by dividing the number of credits earned by the number of hours completed, the following table shows one appropriate set of test cases.

Table 1: Example Test Case/Trace Table

|  |  |  |
| --- | --- | --- |
| Test Input Values | | Expected Result |
| Prompt | **Test Input** |  |
| Enter the Title of the song: | Red Red Wine |  |
| Enter the artist: | UB40 |  |
| Enter the Release Date of the song: | 2000 |  |
| Enter the Playback time of the song: | 3.03 |  |
| Enter the Genre of the song: | pop |  |
| Enter the cost of the song: | 1.29 |  |
| Enter the File Size of the song: | 2.89 |  |
| Enter the file name of the Album photo: | Photo.jpg | =================================================  MP3 Title: Red Red Wine  Artist: UB40  Release Date: 2000 Genre: POP  Download Cost: 1.29 File Size: 2.89  Song Playtime: 3.03 Album Photo: photo.jpg  ================================================= |
| menu | 2 | There are no songs stored. Please add a new song. |
| menu | 1 | Enter the name of the collection:, Enter your first name:, Enter your last name:, Enter the date: |
| Menu | 3 | Please enter the number of the member that you want to change: |
| Menu | 4 | Please enter the number of the song that you want to remove: |
| Menu | 5 | =================================================  MP3 Title: Red Red Wine  Artist: UB40  Release Date: 2000 Genre: POP  Download Cost: 1.29 File Size: 2.89  Song Playtime: 3.03 Album Photo: photo.jpg  ================================================= |
| Menu | 6 | Enter the title of the song that you are searching for. (Case sensitive) |
| Menu | 7 | Enter the Artist of the song that you are searching for. (Case sensitive) |
| Menu | 8 | Which type would you like to sort by: ROCK, POP, JAZZ, COUNTRY, CLASSICAL, OTHER" |
| Menu | 9 | Here is all songs ordered alphabetically: |
| Menu | 10 | \*Popup window asking the user to select the file they want to save their collection to |
| Menu | 11 | \*Popup window asking the user to select the file they want to import their collection from |
| Menu | 12 | \*Orders collection from shortest to longest playtime |

## Example of a Design Document

Problem: Project 1 – 1260 Summer

### List of Inputs, Outputs, and Processing Required

INPUTS

* Song title – String
* Song artist – String
* Song Release Date – String
* Playback time – double
* Song Genre – Genre (ENUM)
* Song Cost – double
* Song filesize – double
* Song Album - String
* User name – String
* Name of Collection – String
* FirstName – String
* LastName – String
* CreationDate – String
* 1-13 – menu inputs - Integer

OUTPUTS

* Formatted toString of song data

PROCESSING

* Ordering array althabetticaly
* Ordering array by song length
* Searching for artist
* Searching for title
* Searching for genre
* Saving file
* Reading file

### Identification of Classes and Their Responsibilities

**Class name:** **MP3Driver (driver)**

Responsibilities:

* inputting the song details
* inputting and validating the song details
* outputting the song details in a user friendly format
* creating collection

**Class name:** **MP3**

Responsibilities:

* knowing all details of the song data
* formatting toString for output

**Class name:** **Genre**

Responsibilities:

* Created genre values as enum type

**Class name:** **MP3Tracker**

Responsibilities:

* Creates collections
* Adds songs to collections
* Edit songs
* Remove songs
* Show song index
* Search by title
* Search by artist
* Search by genre
* Formatted toString for output
* Save MP3 objects to file
* Populate collection from file
* Recognize when save is needed

### UML Class Diagram

This diagram may omit common methods that essentially all classes have such as constructors and getters/setters, but they must be present in the code.

### 

### Algorithms

Class: MP3Tracker

**public** **int** sortTitle (**int** size) {

**int** max = 0;

**for**(**int** n=1; n < size; n++) //compares titles to get the greatest value

**if** (collection.get(n).getSongTitle().compareTo(collection.get(max).getSongTitle())> 0) {

max = n;

}

**return** max;

}

**public** **void** sortTitleDis() {

**for** (**int** i = collection.size(); i > 1; i--) {

**int** m = sortTitle(i);

**if**(m != i-1) {

MP3 temp = collection.get(m); //swaps values to sort greater to the top

collection.set(m, collection.get(i-1));

collection.set(i-1, temp);

}

}

System.***out***.println(collection + "\n"); //prints sorted array

}

**public** **void** findTitle(String title) {

**for**(MP3 mp3 : collection) {

**if**(mp3.getSongTitle() != **null** && mp3.getSongTitle ( ).contains (title)) {

System.***out***.println(mp3.toString ( ));

}

}

}

**public** **void** findArtist(String artist) {

**for**(MP3 mp3 : collection) {

**if**(mp3.getArtist ( ) != **null** && mp3.getArtist ( ).contains (artist)) {

System.***out***.println(mp3.toString ( ));

}

}

}

**public** String genreType(Genre genre) {

String n = "";

**for**(MP3 mp3 : collection) {

**if**(mp3.getGenre ( ) == genre) {

n += mp3.toString ( ) + "\n\n";

}

}

**return** n;

}

**public** **void** maxLength() {

**for**(**int** i = 0; i < collection.size()-1; i++) {

**int** n = i;

**for**(**int** j = i + 1; j < collection.size(); j++) {

**if**(collection.get (n).getPlaybackTime ( ) > collection.get (j).getPlaybackTime ( )) {

n = j;

}

}

MP3 tmp = collection.get (i);

collection.set (i, collection.get (n));

collection.set(n, tmp);

}

}

**public** **static** **void** fillFromFile() {

File fileIn = **new** File("D:\\1260 Summer WS\\Eclipse Workspace\\MP3 - Project 1\\MP3TrackerData\\MusicList.txt");

Scanner file = **null**;

**try** {

file = **new** Scanner(fileIn);

}

**catch**(Exception e1){

System.***out***.println (e1.getMessage ( ) );

}

**while** (file.hasNextLine ( )) {

String line = file.nextLine ( );

String[] split = line.split ("\\|");

**try** {

MP3 m = **new** MP3(split[0], split[1], split[2], Double.*parseDouble* (split[3]), **null**, Double.*parseDouble* (split[5]), Double.*parseDouble* (split[6]), split[7]);

*collection*.add (m);

}

**catch** (NumberFormatException e)

{

e.printStackTrace();

}

**catch** (Exception e)

{

e.printStackTrace();

}

}

}

**public** **static** **void** saveToFile(String fileName, String filePath) **throws** FileNotFoundException{

JFileChooser chooser = **new** JFileChooser("./MP3TrackerData/"); //prompts user to pick a file

**int** result = chooser.showSaveDialog(**null**);

**if** (result != JFileChooser.***APPROVE\_OPTION***) {

System.***out***.println("No file Selected - TERMINATED");

System.*exit*(0);

}

fileName = chooser.getSelectedFile().getPath();

PrintWriter writer = **null**;

**try** {

writer = **new** PrintWriter(fileName);

//write header line

writer.println(*collectionName* + "|" + *firstName* + "|" + *lastName* + "|" + *creationDate*);

//write member details

**for**(**int** n =0; n < *collection*.size(); n++) { //adds members to file based on how many members are created

MP3 m = *collection*.get(n);

writer.println(m.getSongTitle ( ) + "|" + m.getArtist ( ) + "|" + m.getReleaseDate ( ) + "|" + m.getPlaybackTime ( ) + "|" + m.getGenre ( )

+ "|" + m.getCost ( ) + "|" + m.getFileSize ( ) + "|" + m.getAlbumCover ( ));

*saveNeeded* = **false**;

}

} **catch** (FileNotFoundException e) {

// **TODO** Auto-generated catch block

System.***out***.println("File not saved: " + e.getMessage()); //prints error message if file not found

} **finally**{

**if** (writer != **null**)

writer.close();

}

}

Class: MP3Driver

### Test Cases

The following table should have additional test values including some border values , but this gives enough for illustration purposes in this tutorial.

Table 2 - Test Values

|  |  |  |
| --- | --- | --- |
| Test Input Values | | Outpu |
|  |  |
| 2 |  | There are no songs stored. Please add a new song. |
| 13 |  | Thank you -USER- for using MP3 |
| \*Index Value that is non-exsistant |  | **EXCEPTION** |
|  |  |  |
|  |  |  |
|  |  |  |