|  |  |
| --- | --- |
|  | **Cognizant Academy**  **Movie Cruiser**  **C# Specification Document**  **Version 1.0** |
| |  |  |  |  | | --- | --- | --- | --- | |  | **Prepared By / Last Updated By** | **Reviewed By** | **Approved By** | | **Name** | Ramamoorthy Selvam | Vimalathithan Krishnan | Ramadevanahalli Lingachar, Shashidhara Murthy | | **Role** | Learning Solution Designer | Learning Solution Architect | Learning Solution Lead | | **Signature** |  |  |  | | **Date** |  |  |  | |
|  |

Table of Contents

[1.0 Introduction 4](#_Toc11942645)

[1.1 Purpose of this document 4](#_Toc11942646)

[1.2 Definitions & Acronyms 4](#_Toc11942647)

[1.3 Project Overview 4](#_Toc11942648)

[1.4 Scope 4](#_Toc11942649)

[1.5 Intended Audience 4](#_Toc11942650)

[1.6 Hardware and Software Requirement 4](#_Toc11942651)

[1.7 Visual Studio 2015 Project Configuration 5](#_Toc11942652)

[2.0 Class Diagram 5](#_Toc11942653)

[2.1 Model Namespace 5](#_Toc11942654)

[2.2 Util Namespace 6](#_Toc11942655)

[2.2.1 DateUtil.cs 7](#_Toc11942656)

[2.3 Dao namespace 7](#_Toc11942657)

[3.0 Design for View Movie List Admin (EKUC001) 9](#_Toc11942658)

[3.1 Class Diagram 9](#_Toc11942659)

[3.2 MovieDao.cs 9](#_Toc11942660)

[3.3 MovieDaoCollectionImpl.cs 10](#_Toc11942661)

[3.4 MovieDaoCollectionImplTest.cs 10](#_Toc11942662)

[4.0 Design for View Movie List Customer (EKUC002) 12](#_Toc11942663)

[4.1 Class Diagram 12](#_Toc11942664)

[4.2 MovieDao.cs 12](#_Toc11942665)

[4.3 MovieDaoCollectionImpl.cs 12](#_Toc11942666)

[4.4 MovieDaoTest.cs 13](#_Toc11942667)

[5.0 Design for Edit Movie (EKUC003) 14](#_Toc11942668)

[5.1 Class Diagram 14](#_Toc11942669)

[5.2 MovieDao.cs 14](#_Toc11942670)

[5.3 MovieDaoCollectionImpl.cs 14](#_Toc11942671)

[5.4 MovieDaoTest.cs 14](#_Toc11942672)

[6.0 Design for Add to Favorites (EKUC004) 16](#_Toc11942673)

[6.1 Class Diagram 16](#_Toc11942674)

[6.2 FavoritesDao.cs 17](#_Toc11942675)

[6.3 FavoritesDaoCollectionImpl.cs 17](#_Toc11942676)

[7.0 Design for View Favorites (EKUC005) 19](#_Toc11942677)

[7.1 Class Diagram 19](#_Toc11942678)

[7.2 FavoritesDao.cs 20](#_Toc11942679)

[7.3 FavoritesEmptyException.cs 20](#_Toc11942680)

[7.4 FavoritesDaoCollectionImpl.cs 20](#_Toc11942681)

[7.5 FavoritesDaoCollectionImplTest.cs 20](#_Toc11942682)

[8.0 Design for Remove Favorites Item (EKUC006) 22](#_Toc11942683)

[8.1 Class Diagram 22](#_Toc11942684)

[8.2 FavoritesDao.cs 22](#_Toc11942685)

[8.3 FavoritesDaoCollectionImpl.cs 23](#_Toc11942686)

[8.4 FavoritesDaoCollectionImplTest.cs 23](#_Toc11942687)

[9.0 Standards and Guidelines 24](#_Toc11942688)

[9.1 C# 24](#_Toc11942689)

[10.0 Submission 25](#_Toc11942690)

[10.1 Code submission instructions 25](#_Toc11942691)

[11.0 Change Log 26](#_Toc11942692)

# Introduction

## Purpose of this document

The purpose of this document is to define the C# classes related implementation for Movie Cruiser project.

## Definitions & Acronyms

|  |  |
| --- | --- |
| Definition / Acronym | Description |
|  |  |

## Project Overview

Refer Use Case specification document for understanding the functionality and features.



## Scope

Creation of model and data access object classes for movie cruiser application

## Intended Audience

* Product Owner
* Scrum Master
* Application Architect
* Project Manager
* Test Manager
* Development Team
* Testing Team

## Hardware and Software Requirement

1. Hardware Requirement:
   1. Developer PC with 4GB Ram
2. Software Requirement
   1. Git
   2. IE or Chrome
   3. .Net Framework 4.5
   4. Visual Studio Professional Edition 2015
   5. SQL Server enterprise edition 2014

## Visual Studio 2015 Project Configuration

The project cloned from Git needs to be set up as Visual Studio project to make it easier with C# development. Find below the steps to configure Eclipse for movie cruiser project.

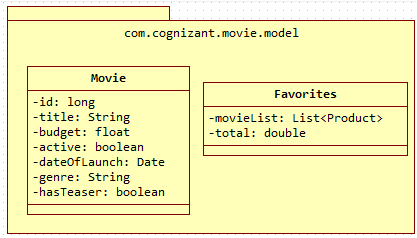
1. Open Visual Studio 2015
2. File > Open > Project/Solution
3. Select the movie cruiser folder in your PC where the movie cruiser code was cloned from <https://code.cognizant.com>
4. Select the solution file ( .sln) to load the project in Visual Studio 2015 IDE.
5. After successful loading of the project, the project folder/files can been seen in the solution explorer available on the left hand side.

# Class Diagram

The classes specified in this document are the primary C# classes that are required for implementation of movie cruiser application. Since ADO.NET module is covered later, the actual database implementation details are postponed to the respective module. The classes in this specification are implemented with hardcoded values and will be consumed by the ASP.NET pages when implementing the next module.

## Model Namespace

Following are the real world objects identified for movie cruiser application. Movie refers to a movie available for viewing online in movie cruiser portal. Favorites will represent customer’s Favorites list to hold the selected movies. Refer the diagram below and create classes accordingly.

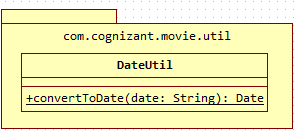


Guidelines for understanding the above class diagram:

1. “com.cognizant.movie.model” represents the namespace
2. Movie and Favorites are classes
3. The content within Movie are instance variables
4. The hypen in each line represents private access specifier
5. For the sake of simplicity the constructors, getter and setter method are not included in the diagram. But it needs to be implemented in code. Code generation option in Eclipse can be used to generate code:
   1. Constructor with option to set all instance variables
   2. Getter and Setter method for each instance variable
   3. Generate toString() method
   4. Generate equals() method which checks for equality based on the ‘id’ attribute

## Util Namespace

Common reusable classes and methods across movie cruiser application will be included in this namespace.



Guidelines for understanding the above class diagram:

1. “com.cognizant.movie cruiser.util” represents the namespace
2. DateUtil is a class
3. Underline denotes static method.

### DateUtil.cs

**convertToDate(date: String): Date**

This method is used to convert date entered in a form to be converted into a Date object.

1. Using **DateTime** and **ParseExact** method to convert the input String into System.DateTime

## Dao namespace

This namespace contains the list of classes that will code to manage the data for movie cruiser application. The methods in Dao classes will be tested using MovieDaoCollectionImplTest and FavoritesDaoCollectionImplTest classes. The Dao interface classes will act as a contract for working with any database. In this specification the implementation of MovieDaoCollectionImpl and FavoritesDaoCollectionImpl will be Collection framework based implementation of Dao interfaces MovieDao and FavoritesDao.



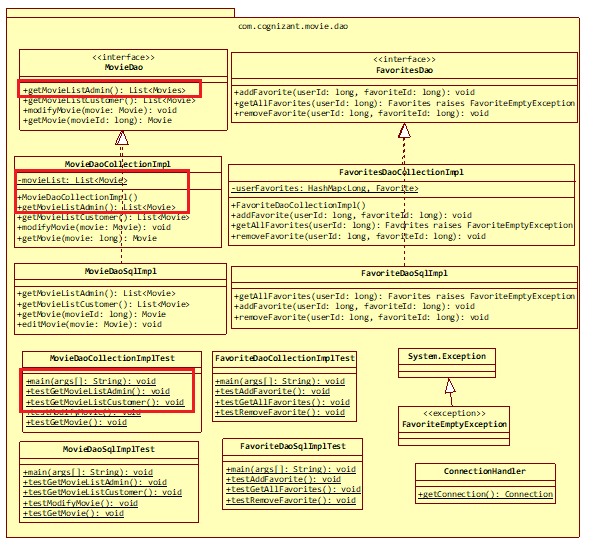
Guidelines for understanding the above class diagram:

1. Identify the namespace, classes, access modifiers, methods and static methods from the above diagram.
2. MovieDao and FavoritesDao are interfaces
3. MovieDaoCollectionImpl and FavoritesDaoCollectionImpl are implementation classes for the interfaces as denoted by the dotted arrow line.
4. MovieDaoCollectionImplTest and FavoritesDaoCollectionImplTest are implementation classes for testing MovieDaoCollectionImpl and FavoritesDaoCollectionImpl.
5. MovieDaoSqlImpl, FavoritesDaoSqlImpl, MovieDaoSqlImplTest, FavoritesDaoSqlImplTest classes will not be implemented in this module. Please ignore these classes for this module.
6. FavoritesEmptyException is an exception class that extends System.Exception.
7. Highlighted classes will be implemented in this module.

# Design for View Movie List Admin (EKUC001)

## Class Diagram

The below diagram denotes the methods that needs to be implemented for this use case. Method wise specification is defined after the diagram.



## MovieDao.cs

Add the method getMovieListAdmin(): List<Movie> in the interface.

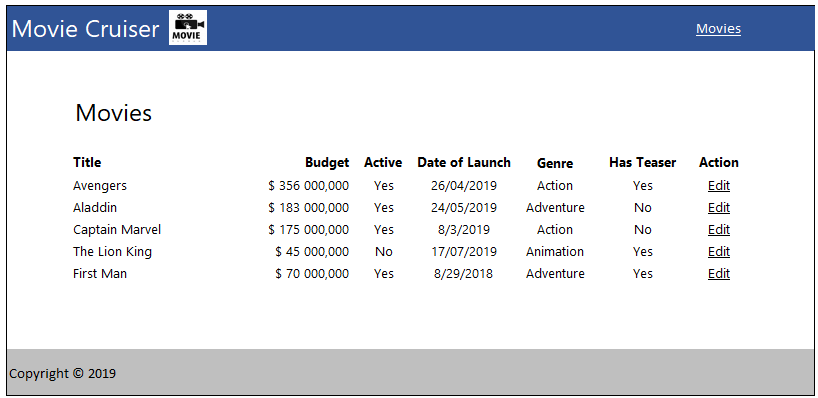
## MovieDaoCollectionImpl.cs

Class for managing data of movies using C# Collections Framework.

**Constructor**

The objective of this constructor is to initialize the movie data that will be displayed in Movie listing screen of Admin.

1. Check if movieList static variable is null or not
2. If it is null perform the steps below:
   1. Create an instance of ArrayList with Movie type
   2. Create multiple Movie instances and add them to movieList. Refer Movie List Admin screen shot from web interface specification and include sample data for movieList based on this sample data.



**getMovieListAdmin(): List<Movie>**

This method returns the list of movies that will be displayed in the Movie listing screen for Admin.

1. Return the movieList

## MovieDaoCollectionImplTest.cs

**main(args[]: String): void**

1. Invoke testGetMovieListAdmin()

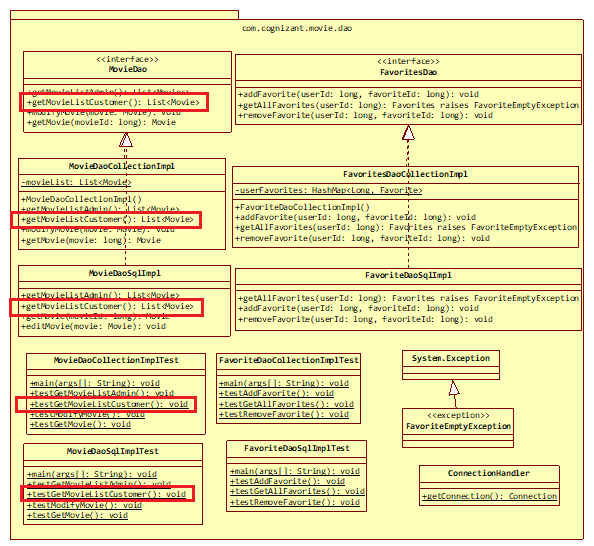
**testGetMovieListAdmin(): void**

1. Instantiate MovieDaoCollectionImpl and assign it MovieDao reference variable movieDao.
2. Invoke movieDao.getMovieListAdmin() and obtain the movieList
3. Iterate through the movieList and display all attributes of each movie.

# Design for View Movie List Customer (EKUC002)

## Class Diagram

The below diagram denotes the methods that needs to be implemented for this use case. Method wise specification is defined after the diagram.



## MovieDao.cs

Add the method getMovieListCustomer(): List<Movie> in the interface.

## MovieDaoCollectionImpl.cs

This class manages the data related to Movies of movie cruiser application. A new method needs to be added for this use case.

**getMovieListCustomer(): List<Movie>**

This method returns the list of movies that will be displayed in the Movie listing screen for Customer.

1. Initialize an ArrayList for type Movie
2. Iterate through movieList and perform the following steps:
   1. Is the expiry date of the movie is after today?
   2. Is the movie active?
   3. If the above conditions satisfy, add the movie into the ArrayList created in the first step.
3. Return the filtered ArrayList

## MovieDaoTest.cs

**main(args[]: String): void**

1. Invoke testGetMovieListCustomer()

**testGetMovieListCustomer(): void**

1. Instantiate MovieDaoCollectionImpl and assign it MovieDao reference variable movieDao.
2. Invoke movieDao.getMovieListCustomer() and obtain the movieList
3. Iterate through the movieList and display all attributes of each movie.

# Design for Edit Movie (EKUC003)

## Class Diagram

The below diagram denotes the methods that needs to be implemented for this use case. Method wise specification is defined after the diagram.

## MovieDao.cs

1. Add method modifyMovie(movie: Movie): void in the interface.
2. Add method getMovie(movieId: long): Movie in the interface.

## MovieDaoCollectionImpl.cs

This class manages the data related to Movies of movie cruiser application. A new method needs to be added for this use case.

**modifyMovie(movie: Movie): void**

This method will be used to change the movie data in the list of movies. This method will be invoked when Customer submits the user form.

1. Iterate through the movieList and find the matching movie
2. Update the matching movie in the ArrayList

**getMovie(movieId: long): Movie**

This method is used to retrieve a particular movie’s detail from the movie list. This method will be invoked when user click on Edit link in movie listing screen of Admin.

1. Iterate through movieList and find the matching movie
2. Return the matching movie from the movieList

## MovieDaoTest.cs

**main(args[]: String): void**

1. Invoke testModifyMovie()

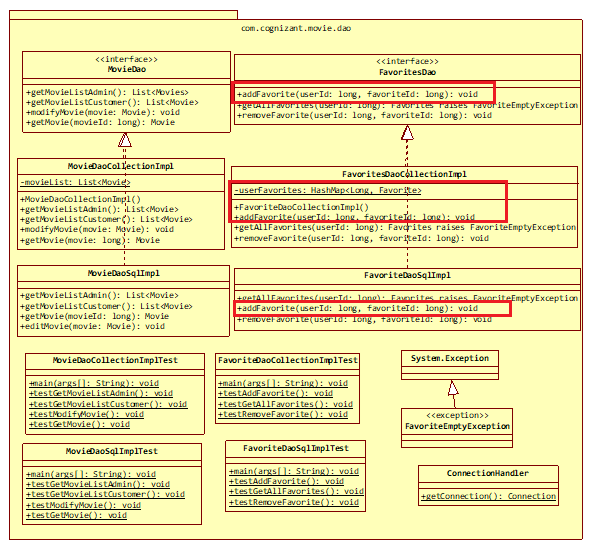
**testModifyMovie(): void**

1. Create an instance for Movie with id matching with one of the movie already added to the movieList.
2. Instantiate MovieDaoCollectionImpl and assign it MovieDao reference variable movieDao.
3. Invoke MovieDao.modifyMovie(movie) by passing the movie created in the first step.
4. Invoke movieDao.getMovie(producId) to read and check if the movie details are modified.

# Design for Add to Favorites (EKUC004)

## Class Diagram

The below diagram denotes the methods that needs to be implemented for this use case. Method wise specification is defined after the diagram.



## FavoritesDao.cs

1. Add method addFavoritesItem(userId: long, movieId: long): void in the interface.

## FavoritesDaoCollectionImpl.cs

This class manages the data related to Favorites of all users of movie cruiser application. A new method needs to be added for this use case.

**Constructor FavoritesDaoCollectionImpl()**

Data for all users will be stored in the HashMap available in Favorites instance. This constructor initialized the Favorites as well as the HashMap within the Favorites, so that the class instance is ready to store values in the HashMap when Customer adds items into the Favorites.

1. Check if the userFavoritess instance variable is null or not
2. If userFavoritess is null then create a new instance of HasMap with type Long and Favorites and assign it to userFavoritess instance variable.
3. The userFavoritess instance variable will hold the Favorites details for each user in a HashMap. The key of this HashMap will have the userId. Each value in the HashMap will be an ArrayList of Movie.

**addFavoritesItem(userId: long, movieId: long): void**

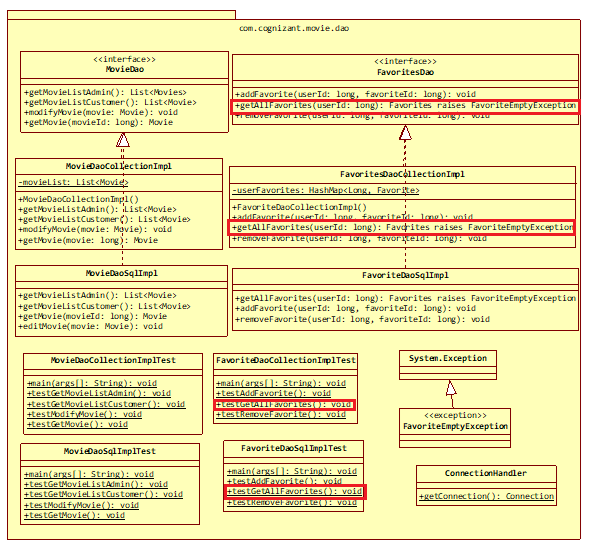
This method is invoked when Customer clicks Add to Favorites link in movie listing screen. This method gets the movie list from the HashMap for the specific user and adds the movie into the movie list. If there is no such user in the HashMap, then a new entry needs to be added in the HashMap with userId as key and new ArrayList of Movies as value.

1. Instantiate MovieDaoCollectionImpl and assign it MovieDao reference variable movieDao.
2. Get the movie using movieDao.getMovie(movieId) method
3. Check existence of user in userFavoritess based on userId
4. If user exists in userFavoritess, perform the steps below:
   1. Get the movieList from the userFavoritess
   2. Add the movie obtained in previous step into movieList
5. If user does not exist in userFavoritess, perform the steps below:
   1. Create a new Favorites instance with new ArrayList
   2. Add the movie obtained in step one and add it to movieList created in previous step
   3. Put the userId and ArrayList of Movie into the userFavoritess

# Design for View Favorites (EKUC005)

## Class Diagram

The below diagram denotes the methods that needs to be implemented for this use case. Method wise specification is defined after the diagram.



## FavoritesDao.cs

1. Add method getAllFavoritesItems(userId: long): void in the interface.

## FavoritesEmptyException.cs

1. Extend this class from C#.lang.Exception and include an empty constructor.

## FavoritesDaoCollectionImpl.cs

This class manages the data related to Favorites of all users of movie cruiser application. A new method needs to be added for this use case.

**getAllFavoritesItems(userId: long): Favorites throws FavoritesEmptyException**

Method to get list of movies added by a customer to Favorites.

1. Get the movieList based on userId from the HashMap of userFavoritess
2. If the returned list is empty
   1. Create new FavoritesEmptyException and throw it
3. If the returned list is not empty
   1. Iterate through the movieList and add up the prices.
   2. Set the total instance variable of Favorites with the added up movie prices.
   3. return Favorites

## FavoritesDaoCollectionImplTest.cs

**main(args[]: String): void**

1. Invoke testAddFavoritesItem()

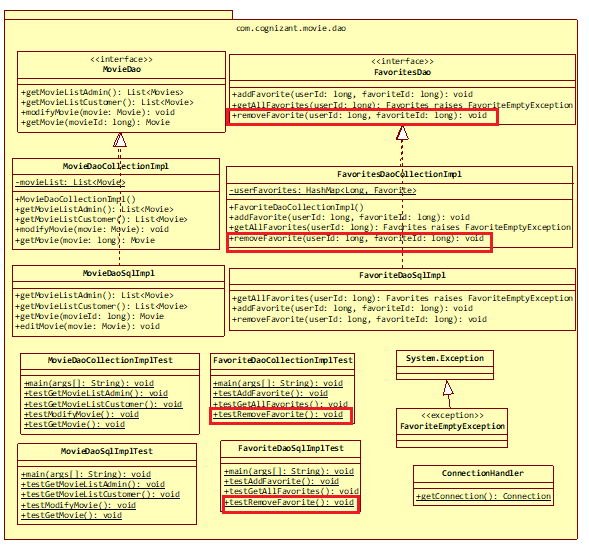
**testAddFavoritesItem(): void**

1. Instantiate FavoritesDaoCollectionImpl and assign it to FavoritesDao reference variable FavoritesDao.
2. Invoke FavoritesDao.addFavoritesItem() method with following parameters
   1. userId: 1
   2. movieId: one of existing movieId in MovieDaoCollectionImpl
3. Invoke FavoritesDao.getAllFavoritesItems() with userId as 1
4. Display the contents of MovieList returned in previous step and check if the added Favorites item is present or not.

# Design for Remove Favorites Item (EKUC006)

## Class Diagram

The below diagram denotes the methods that needs to be implemented for this use case. Method wise specification is defined after the diagram.



## FavoritesDao.cs

1. Add method removeFavoritesItem(userId: long, movieId: long): void in the interface.

## FavoritesDaoCollectionImpl.cs

This class manages the data related to Favorites of all users of movie cruiser application. A new method needs to be added for this use case.

**removeFavoritesItem(userId: long, movieId: long): void**

Method to remove a movie from the Favorites. This will be invoked when Customer clicks Delete link in the Favorites screen.

1. Get the List<Movie> from userFavoritess based on userId
2. Iterate through the List of Movie and perform the below steps
   1. Check if the movieId of each movie from the list matches with this methods input parameter
   2. If movieId matches then remove the movie from the list

## FavoritesDaoCollectionImplTest.cs

**main(args[]: String): void**

1. Invoke testRemoveFavoritesItem()

**testRemoveFavoritesItem(): void**

1. Instantiate FavoritesDaoCollectionImpl and assign it FavoritesDao reference variable FavoritesDao.
2. Invoke FavoritesDao.removeFavoritesItem() method with following parameters
   1. userId: 1
   2. movieId: Same movieId as what was provided when testing add Favorites item.
3. Invoke FavoritesDao.getAllFavoritesItems() with userId as 1
4. Enclose the above method within try catch block with catch block handling FavoritesEmptyException. Check if the catch block is executed, which means that the Favorites item added during testAddFavoritesItem() is removed now and the Favorites is empty, due to which the FavoritesEmptyException is thrown.

# Standards and Guidelines

## C#

1. Ensure that the class names, method names and variable names are followed exactly as specified in the class diagram
2. Ensure that access modifier are in line with the class diagram specification
3. Naming standards to be followed:
   1. Variable
      1. Should be in mixed case with the first letter lowercase and with the first letter of each internal word capitalized (Example: firstName, dateOfBirth)
      2. Variable names should be short, but meaningful
      3. Variable name defined should indicate the purpose to a casual observer
      4. Single character variable names should be avoided except for temporary variables
      5. Temporary variables include i, j, k and m
   2. Class
      1. Class name should be a noun
      2. Class name should be in mixed case with the first letter uppercase and with the first letter of each internal word capitalized
      3. Must use whole words and should not have acronyms or abbreviations

Examples: Employee, TaxCalculator

* 1. Method
     1. Method names should be verbs
     2. Method names should be in mixed case with the first letter lowercase and with the first letter of each internal word capitalized

Example: changeGear(), calculateBalance()

1. Code Formatting
   1. Class Structure
      1. Place the elements of a class in the following order:
         1. Static variables
         2. Instance variables
         3. Constructors
         4. Methods and Getter/Setters
         5. hashCode(), equals(), toString,
   2. Spacing
      1. A space before and after an operator is required
      2. A space before curly braces is required
      3. A space after a comma is required
      4. A space after semicolon in for loop is required
      5. A single line space after a method is required
   3. Curly braces position
      1. Opening curly braces should be in the same line
      2. Closing curly braces should always be in a new line
   4. Tab spacing
      1. Use 4 spaces instead of tab character
      2. Increase a tab character in the lines after opening curly braces
      3. Reduce a tab character on the of closing curly braces
      4. Include one more tab in the wrapped line
   5. Line Width
      1. Width of a line should not exceed 100 characters

# Submission

## Code submission instructions

Once your code is evaluated by the trainer and all the issues reported by the trainer are corrected, the code needs to be submitted to the remote repository. Follow the steps below to submit the code to remote repository.

1. In Windows Explorer go to the movie cruiser folder
2. Right click on the empty space in the right hand side of Windows Explorer and select “Git Bash here”
3. Execute the following commands

To display the added or modified files

git status

To stage the added or modified files

git add .

To display the staged files

git status

To save the code to local repository

git commit -m "C#"

To transfer the changes from local machine to server

git push origin master

1. Successful execution of the above commands will upload the files to the server repository.
2. Login into <https://code.cognizant.com>
3. Click on the project movie cruiser
4. Check if the files that are uploaded correctly with appropriate folder structure.

# Change Log

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Changes Made | | | |
| V1.0.0 | Initial baseline created on 20-May-19 by Ramamoorthy Selvam | | | |
| Vx.y.z | <Please refer the configuration control tool / change item status form if the details of changes are maintained separately. If not, the template given below needs to be followed> | | | |
| **Section No.** | **Changed By** | **Effective Date** | **Changes Effected** |
|  |  |  |  |