Cairo University

Faculty of Computers and Information

CS352

Software Engineering II

E _ Website

Review & Test

Team Leader

Mohamed K Amer

Month & Year

5 Mar 2017 at 23:59

CS352: Phase 1– Mohamed K Amer
<E-Website> TA. Omar Khaled

ID	*Student Name*
20130194	**Mohamed Khaled Mohamed Amer**
20140071	**Omnia Shawky Abdullah**
20140118	**Rasha Rabie El-Fauomy**
20140209	**Laila Barakat Mohamed**
20140012	**Ahmed Gamal Mahmoud**

Dr. Amr Kamel	**a.kamel@fci-cu.edu.eg**
Dr. Khadiga Mohamed	**kelbedweihy@fci-cu.edu.eg**

Contents

Instructions [To be removed] .... **Error! Bookmark not defined.**

Review Check List 3

Testing 8

Git repository link 10

CS352: Phase 1– Mohamed K Amer
<E-Website> TA. Omar Khaled

Review Check List

Design Principles

- **The Design Follow The SOLID Principles as :-**

1. **Single Responsibility: Each class handles one role and if we need to make one change we will make it in the class which handles it, Also each class have one responsibility which make the whole code coherent (Cohesion); All concepts are separated in each class.**

2. **Open Close Principle: Open for extension-> we can extend new behaviors to satisfy a changing requirements, Also Closed for modification-> extending the behavior must not result in changing to the source; No global variables, All data is protected.**

3. **Liskov Substitution Principle: All functions have a role and no useless function exist, Also each inheritance is used in a good way and no child class is not related to the super class; Favor composition for inheritance.**

4. **Interface Segregation Principle: No ‘fat’ interfaces, also no non cohesive interfaces. Note: We have a problem here that interfaces are separated and we can’t return to the previous interface; Clients own the interfaces.**

5. **Dependency Inversion Principle: MVC is used here as the receiver depends on the sender.**

- **The Design Follow the OOP Rules as: - We have all classes separated in java classes and have their own methods and attributes.**

- **The Design Simple and Easy to be modified as: - We can modify it any time easily without change the whole code.**

Coding Standards

- **The code understandable and readable as: - The developer follows the programming style principle.**

- **The code follows Java Coding Style** as: - **Indentation is used but not 100%, Spacing, Class Member Ordering, Maximum Line Length, Identifiers (No _ or $).**

- **Indentation used properly.**

- **Variables have good Names.**

Comments

- **No Comments available in this code.**

Code Structure

- **The code follows the design precisely as it does what is the client wants and achieves the business needs (MVC).**

- **No long classes and methods.**
- **No repeated code.**

Error Handling

- **The code handles some errors and exception not all exceptions.**
- **Not used in all functions.**

Logic

- **All loops terminate, also have correct conditions and bounds.**

Overall

- **The design is not bad, also the code.**

Design and Code Checklist

Design Principles

1- Does the design follow SOLID principles?	Yes	90%	Related Issues: ....
2- Does the design follow OOP rules?	Yes	95%	Related Issues: ....
3- Is the design simple and easy to modify?	Yes	95%	Related Issues: ....

Coding Standards

4- Is the code understandable and readable?	Yes	90%	Related Issues: ....
5- Does the code follow Java Coding Style?	Yes	70%	Related Issues: ....
6- Is indentation used properly?	Yes	80%	Related Issues: ....
7- Do variable have good names?	Yes	90%	Related Issues: ....

Comments

8- Is the code commented enough?	No	0%	Related Issues: ....
9- Is every class and method commented?	No	0%	Related Issues: ....
10- Do comments follow Java doc style?	No	0%	Related Issues: ....
11- Is Java doc generated for all the code?	No	0%	Related Issues: ....
12- Are there useless / wrong comments?	----	----	Related Issues: ....

Code Structure

13- Does the code follow the design precisely?	Yes	80%	Related Issues: ....
14- Are there very long classes or methods?	NO	95%	Related Issues: ....
15- Is there repeated code? (put put in a function)	NO	95%	Related Issues: ....

Error Handling

16- Does the code handle errors and exceptions?	Yes	60%	Related Issues: ....
17- Is defensive programming used to avoid errors?	No	40%	Related Issues: ....

Logic

18- Do loops have correct conditions and bounds?	Yes	95%	Related Issues: ....
19- Do loops always terminate?	Yes	95%	Related Issues: ....

Overall

20- Are the design and code of good quality?	Yes	70%

Testing

1. Games Handler Testing Class

Number	Testing function	Description	Result
GamesHundler	AddGame(Game game)	Testing function for add game in Game entity then add it on the file. This test case test the normal adding game scenario	Passed
GamesHundler	LoadGame() WriteGame(String Name)	Testing function for Load, Write Game functions. This test case tests the normal Loading, Creating game scenario.	Failed

2. User Handler Testing Class

Number	Testing function	Description	Result
UserHandler	AddUser(User user)	Testing function for add User in User entity then add it on the file. This test case tests the normal adding User scenario.	Passed Failed only when we delete records and it again.
UserHundler	CheckUserName(User user)	Testing function for check only user name for a user. This test case tests the normal verification user name scenario.	Passed
UserHundler	CheckUser(User user)	Testing function for check user info on the file. This test case tests the normal verification user Scenario.	Passed But have a problem in id attribute We can change it.
UserHundler	GetUser(), Setters, Getters	Test setters, getters for class.	Passed
UserHundler	SelectUser(String UN, String PW)	Test case to check select user behavior from the file.	Failed Always return null.

3. Games Handler Class (Search Game)

Number	Testing function	Description	Result
GamesHundler	GetGames(String name)	Testing function for Get All Games. This test case tests the normal behavior to get, search games scenario.	Failed
GamesHundler	TeacherGames(String name)	Get teacher's games that is created by him.	Failed

Git repository link