

ADvance Programming

Final Assignment



June 21, 2023

Abdallah Daradkeh

21110446

A picture containing text, font, logo, symbol

Description automatically generated

Contents

[Part 1: 1](#_Toc135561154)

[1.1 Investigate object orientated characteristics and SOLID principles 1](#_Toc135561155)

[1.2 Impact of clean coding techniques on the use of data structures and operations while writing algorithms 3](#_Toc135561156)

[1.3 Analyse design patterns 3](#_Toc135561157)

[1.4 Evaluate SOLID development principles impact on object orientated application development 3](#_Toc135561158)

[Part 2: 3](#_Toc135561159)

[2.1 Class Diagram building 3](#_Toc135561160)

[2.2 Testing Methodology 3](#_Toc135561161)

[2.3 Design Refinement 3](#_Toc135561162)

[Part 3: 3](#_Toc135561163)

[3.1 Build a system (Code) 3](#_Toc135561164)

[3.2 Solid principles, clean coding techniques, and programming patterns effectiveness on application development 3](#_Toc135561165)

[3.3 Pros and Cons of forms Automatic testing of applications and software systems, + examples from the developed system 3](#_Toc135561166)

[Part 4: 3](#_Toc135561167)

[4.1 different methods of implementing automatic testing as designed in the test plan examination 3](#_Toc135561168)

[4.2 Automatic testing implementation of developed System 3](#_Toc135561169)

[4.3 Developer produced VS vendor provided automatic testing tools for applications & software systems 3](#_Toc135561170)

# Part 1:

## 1.1 Investigate object orientated characteristics and SOLID principles

Object-Oriented: the main concept of object oriented is object, it revolves around this concept, which objects are instances from classes.

Object Oriented characteristics:

* **Inheritance**: It is a method that enable a class to inherit another class properties (attributes and methods), it is enable overriding for methods (the same method in the inherited class with different implementation). So this is lead to code reuse, and build relationships between classes hierarchically.
* **Encapsulation**: It is a method that we use to prevent the direct access to some attributes, and we can use it for methods also. So access to the data happened through methods( getters and setters for example). So objects can access directly to some of this class data.
* **Polymorphism**: It is that the objects get many forms. SO objects of a base class(super class) can take the form of a derived class(sub class). And it has many forms: allowing many methods(two or more) to have the same name and different parameters which is overloading. And the derived class( sub class) to have the same method(exactly) as the base class(super class) with different implementation which called Override.
* **Constructors**: They are the special methods which are used to initialize class objects. So, when we want to create an instance from a class we use it. The object is allocated in memory when we initialized it by the constructor. There are the implicit constructors, constructor without parameters is implicitly called without manually calling it. And explicit constructors which called manually.
* **Abstract**: The concept of abstraction it refers to hide the implementation details. Like interfaces and abstract classes, we hide the implementation of methods in it. We focuse on main and important features and hide the implementation details.
* **Interface:** It is a class which is full abstract, So all method inside it are an abstract methods by default and has public as a modifier access. And variables has public as modifier access, and static and final by default.And it support multiple inheritance functionality
* **Collections**: java.util package is the package that have these collections in it. These collections are used to manipulate group of inputs(objects). It store, process and manipulate with multiple objects. Each collection has its pros and cons and its features. For example: vector is Synchronized while ArrayList is not synchronized, that mean that vector is allowed to one thread only can access the code, while ArrayList multiple third at the same time can access the code. So, ArrayList is faster than vector since vector only permit to only one thread to access the code and the next thread must wait until the first thread finished and so on. In terms of Data growth, ArrayList increase its size by 50% of its current size while vector do it by 100%. So ArrayList is better in this. And so on to the other collections.
  + **Some Collections**:
    - **1**. ArrayList
    - **2**. Vector
    - **3**. Hashmap
* **Static**: It is a keyword used to declare attributes(variables and methods) of the class. If a variable or method has static keyword so:
  + It is called static variable/class variable \_ static method/class method
  + Can be directly accessed through using the class name without the need to a class’s instance.
  + Static methods can only accessed to static variables.
  + Static method/variable is shared between all classes instances
* **Class** **relationships**: It is how classes are related/connected to each other, There are many relationships that class can made between each other:

**1) Dependencies: This is when a class depends on other class in some way, if the independent class effect the dependent class may effect, and the class using the other class as an argument in an operation signature**. This relationship is represented as a dashed row. Like that >>> independent class ----------> dependant class

2) Generalization: It represents is-a relationship between classes, where a class (subclass) inherits attributes and behaviors of the other class.It is represented by solid linw with arrowhead pointing to the superclass. Such as:

A picture containing line, screenshot, plot, diagram

Description automatically generated

3) Realization: It represents a relashionship where the relationship is between a class which is an interface and the other class that implements it. It is donated such as the following

A graph paper with a dotted line

Description automatically generated with low confidence

4) Association: Representeing of a relationship between two classes(seperated which the relationship is established through their objects. There are two types of association: binary(when connects two classes exactly. Ternary: when more than two classes are connected. There are role(teacher, student…) and multiplicity(many : 1,1:1,1:many…) which we can apply for association:

It is donated as the following:

A picture containing text, line, screenshot, font

Description automatically generated

5) Aggregation: It is a relationship (which is has a relationship), so if we remove the parent class the child class can still work alone or can still exist independently of it. It is denoted as:

A black line on a white background

Description automatically generated with low confidence

6)Composition: it make a belongs to or part-of relationship, where there two classes, one classe(child) can’t exist independently of the other(parent). It is donated as the following:

A picture containing line, screenshot, white

Description automatically generated

* **SOLID principles**:

It is a principles introduced by Robert C. Marten, it serves as a guidelines to the developers in order to design softwares and programs which are easy to understand, and extend, and maintain.

The SOLID is stands for :

1. **Single Responsibility Principle**: Which mean that a class should have only a single responsibility(it is responsible only for one issue), it is very helpful in terms of that he achieve high cohesion(How relations between software/class components are) and that classes are manageable(manage organized and smaller classes are easier) and loose coupling(Low interdepence between classes). Coupling( Level of how the various software/classes are inter-dependency). And this will help in testing the program since it will generate fewer test cases.

2. **Open-Closed Principle**: This principle that a software component have to be open for extension, but it closed to change/modify. So if we want to make updates to the program, it does not require to change the current code as possible as can. And at the same time if we want to make new updates, it must be extendable to add them. So the code is become flexible and more maintainable.

3**. Liskov Substitution**: It is to write the code in a way that to the ability can take their subtypes place, In a way that the program correctness does not affect. It is a gudelines for inheritance and polymorphisim. So we ensure that we can use objects of derived classes interchangeably with their base classes objects. It promotes reuse of the code. And by giving guidline for the proper use of polymorphism and inheritance it promotes robustness.

4) **Interface Segregation Principle:** It is a principle that we create focused interfaces, which mean that every interface is designed for its users and they should not be forced to depend on interfaces that they do not use. So the interfaces that class are implemented must be suitable for the users requirements and needs, and should not have unrelated functionalities which users don’t need. And this help to not have empty methods, and increase cohesion of interface

5) **Dependency Inversion Principle:** It is principle that mention that High-level modules(classes) should not depend on the low-level modules(classes), and details(both high and low level modules) should depend on abstractions, but abstractions should not depend on details, which aim to decrease the interdependencies between classes. This makes the system flexible and extensible and easier to test and support code reusability since the classes are become more more independently contains

## 1.2 Impact of clean coding techniques on the use of data structures and operations while writing algorithms

The clean coding techniques lead to **understandable and readable data structures**, since it is include to use pronounceable and meaningful variable names, and it is also includes to use names which are searchable( such as: SecondsInTheDay, instead of: 86400000), and what also increase the previous and increase and attain **data strucutre clarity**. And one of clean code techniques that functions should do one thing which avoid existence of unneeded context and the **data strucure become more readable and more higher quality**. And it says that functions should say the job that they do which optimize **data strucutres readability** and make them more clarity. Also since they(clean code techniques) said that functions should do one things(each one), so this ensure isolated operations and modularity(which mean to break complex operations down into smaller). Which make it more easier for testing and debugging and code reusing. Also it is **increase the efficiency and performance of data structures** since clean coding encourages developers to write efficient algorithms and it is emphasize to select the most appropriate data structure for the problem you want to solve and it takes the time and complexity in consideration and this is through also avoiding unkneaded overhead(for example: many functions with the same job) which leading to faster algorithms.

## 1.3 Analyse design patterns

Okay let’s analyse design these design patterns:

1) creational Design Patterns: it focus on mechanisms of object creation, it is provide ways to create objects more flexible.

Example on Creational Design patterns **is factory method**:

In factory method we create an interface which all subclasses implements the interface, which enable polymorphisim characteristics, so enables to create objects and choose which subclass that we want to instantiate the object to it. It optimizes loose coupling between the code and the created objects, and encapsulates the logic in which the process of object creation is done.

A screenshot of a computer

Description automatically generated

2) Structural Design Patterns: it is focus on the composition of classes, it is how the combination of different objects and classes together, so we achieving multiple goals completely. It gives solution to organize and manage the relationships between classes.

For an example on Structural Design patterns is Proxy:

In proxy, we access to the functionality of a class through another class. So by using proxy design patterns we restricted clients access to the main resources and we provide them just with the needed information through another class(Proxy class).

A screenshot of a computer

Description automatically generated with low confidence

3) Behavioural Design Patterns: it focus on communication between objects and objects behavioural and provide patterns to the objects to communicate with each other:

An example of behavioural design pattern is Strategy:

Strategy makes you able to create many classes which have an interchangeable algorithms, or define a family of algorithms which are interchangeable. So you select a specific strategy or algorithm to run at runtime depending on the behavior that takes place.

A picture containing text, screenshot, diagram, line

Description automatically generated

## 1.4 Evaluate SOLID development principles impact on object orientated application development

SOLID development principles:

1) Single Responsibility Principle: It is mention that class should have only one reason in order to change, so class should do one responsibility(have single responsibility), that lead to code which is easier to maintain and test and understand, impacts are:

* Optimized testability: since it is generate smaller class, and focused class, and it is easier to test , leading to better unit testing.
* Optimized the code organization: since it is generate more cohesive classes, break down complex classes into smaller, this is optimize code organziztioon and code readability.
* Optimized maintainability: Since changes done to one responsibility are less possibilities to affect responsibilities which are unrelated to it.

2) Open/Closed Principle: it says that code must be closed to modify but open to extension. It encourages classes which can extend without the need to modify the current code.

Impacts are:

* Scalability, and flexibility: since open/closed principle allows developers to add new features or functions or behaviors by extending classes or implement new classes, and this is without the need to modify the current codebase.
* Reduces bugs that can effect the codebase: by avoiding changes and modifications to the codebase, it minimizes the risk that, a new bugs can appear in the codebase since the codebase, there are no modifications happening on him.
* Encourages modularity: since it encourages to create smaller, reusable components which can be extended independently, which is taken us to more modular codebase and more maintaibale codebase.

3) Liskov Substitution Principle: It states that superclass’s object should be replaceable with objects of its subclasses, in away that the correctness of the program does not affect.

Impacts:

* It enables the use of polymorphism, so different classes are interchangeably by this principle through their base class which is shared between them or interface.
* Violations the liskove substitution principle can lead to an unexpected behaviour when we substitute the objects, which cause bugs which can be difficult to track.

4) Interface Segregation Principle: it states that clients should not be forced to depend on interface that he does not use.

Impact:

* It reduces coupling between classes, since it encourages focused intefaces. This lead to that modifications that take place on interface will have minimal impacts on other classes and system components.
* It reducing unrelated and unnecessary dependencies by encourages focused interface, and the ability to select the interfaces that they need, this support the reuse of the code.
* It optimized and enhaces testability since it is encourages focused interfaces.

5) Dependency Inversion Principle (DIP): it states that high-level modules should not depend on low-level modules. Abstraction should not depend on details, but details( both (low and high level)) should depend on abstraction.

Impact:

# Part 2:

## 2.1 Class Diagram building

A picture containing text, diagram, line, plan

Description automatically generated

I will submit the UML file the diagram if there is anything unclear in the picture above.

## 2.2 Testing Methodology

* Requirement of the System:

1) Doctor should add medicine, and treatment description to the system.

2) Receptionist should add patient to the system, print Bill, and print all appointments at the clinic.

3) Nurse should Generate a daily report of the clinic by print information on file.

4) Patient should have discount based on their category.

* Testing Planning

Testing approach: I will use euivalence Partioning as my approach to test. I will test the functional of each method by give it invalid and valid output and check the results. This will allow us to higher quality code. And find defects on program and check everything is going write as better as we can. I will perform unit testing on the application.

* Our test cases:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Reciptionist class | Function being tested | | Testcases | Valid output | | | Invalid output | Expected result |
|  | printAppointment | | Check choosing from array  []appointment | Put index > i = 5  Output>= 10:00  and  Output<= 6:30 | | | Let index i=30  Output >6:30  i = -1  output < 10:00 | Valid output: 12:30-1  Invalid Output:  Second: error out of range  Third: out of bounds error |
|
| ArrayList<Patient> | | Check of two objects | All elements of object = all elements of object | | | Not all element of object = all elements of other object, or no element equal | Valid: True  Invalid: false,error |
| Setters and getters | | check equality of input and actual output | Input = output | | | Input != output | Valid: true,  Invalid,false |
|  | |  |  | | |  |  |
| addMedicine | |  | User input = 444, mmm  Our input = 444 | | | User input = 444, mmm  Our input = 123 | Valid:true, function will go forward and implement the rest process.  Invalid: Function will stop its implementation |
| Compare between medicine user enter and actual medicine | User: 123456789, Ibuprofen 200mg | | | User: 12344\*,  1234 | Valid: true  Invalid: Error |
| treatmentDiscription | |
|  | | Compare between discription user enter and actual medicine | User= 123456789,Fever | | | User: 12344\*,1234 | Valid: true  Invalid: False, “invalid output” |
| registration | | Check elements being added to patients Arraylist | Elements with String type = a, elements with int type = 1 | | | Elements with String type = 123,  Elements with int type = a | Valid: process complete, elements has been added.  Invalid: error, function will stop immediatly |
| Compare between two object | reciptionist = reciptionist | | | reciptionist!=reciptionist | Valid: true, Invalid: false : |
| Check if and else if conditions | User = ordinary/student/consultant | | | User = aa | Valid: true, continue  Invalid: Error. |
| printBill | | Check the if statement | User input = 123456789  Actually = 123456789 | | | User input = ddd, 123456789 | Valid: function will go forward and implement the rest process.(print all bill details)  Invalid: Function will stop its implementation |
| writePatientInfo | | Compare between information enter and information on the file | User: Enter information based on each data type | | | User: Enter information conflicts with the data type for example String value =111 | Valid: true  Invalid: False, Stop the program immediatly |
|  | |  |  | | |  |  |
| postAnnoouncement | | Compare between message user enter and actual message | User= mmm, message = mmm | | | User = Hello sear,message = fff | Valid: true  Invalid: false |
| notifyObservers | | Compare between entered message and stored message | User = mmm,  System = mmm. | | | User = 123\*, | Valid: true  Invalid: false |
|  |  | | |  |  | |
|  |  |  | | |  |  | |
|  |  |  | | |  |  | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Patient | discount | Check discount validation | User = 111,(ordinary)  User = 111,(consultant)  User = 111,(student)  System = 111 | User = 111,(ordinary)  User = 111,(consultant)  User = 111,(student)  System = 1111 | Valid: true and return discount based on type(50%,20%,0%)  Invalid: error, invalid input |

* I will use java junit to implement the test cases.
* When issues or defects are determined, document them.
* Ensure that testing does not affect the program work.

## 2.3 Design Refinement

A picture containing text, diagram, plan, line

Description automatically generated

I will submit it with the files.

# Part 3:

## 3.1 Build a system (Code)

## 3.2 Solid principles, clean coding techniques, and programming patterns effectiveness on application development

* It makes the code more maintainable, more professional in terms of performance and storage issues, code become more readability, support modular design.
* It makes the code more adaptive to accommodate changes without the need for huge modifications.
* Support coding reusability. It support the creation of components which are reusability.
* This makes test process easier. Easier to write unit tests.
* It support collaboration, since it is include common standardization and understanding of coding practices. So, code review become more efficient and the whole development process.

## 3.3 Pros and Cons of forms Automatic testing of applications and software systems, + examples from the developed system

Pros:

* Test cases execution become more efficient and rapid. It can manipulate large volumes of test data rapidly. Can testing different fields of values, which will be a waste of time manually.(Efficiency)
* Ensure consistenct and repeatable results. Since they are build in scripts and test conditions which are ready predefined.
* Comprehensive coverage of scenarios and test cases which may be challeng and hard in manual testing only. It provides validations and error handling.

Cons:

* You are going to start with a time and effort consuming a little bit. Since you need to write test scripts, setting up and prepare the environment of testing, maintain test case and test suites.
* Maintenance may increase the test cases number, and increase the huge of automated tests, since when you maintain or update, you make changes in code which may leading to bug or error.
* You may document an issue that did not exist, which take time to inestigate non issues. And test may fails to detect the actual proble. Which give you false sese of security.
* It lacks the intuition of human and the understanding of the human. It is hard to assess the visual aspects overall and the usability or these forms can be associated with subjective user experience.

From the developed system: There are a huge number of test cases, which can be very exhausted to manipulate it manually. Also, While I try to maintain problems, there are new problems appear while I maining. In the first, I have to write a lot of codes line and prepare the environment before started. And maintaint test cases. I found a difficult while I am trying to defect some problems with automation testing, so I was having to check and follow error by myself manually.

# Part 4:

## 4.1 different methods of implementing automatic testing as designed in the test plan examination

1)

* Unit Testing:
* Objective: testing individual components of code to verify their functionality.
* Advantages:
  + It allow us to detect defects early in the development cycle.
  + It improves the quality and maintainability of code.
  + It provides rapid feedback on changes that take place on the code.
* Limitations: It focuses on single components(isolated), to make comprehensive testing it will take time, it may not cover all the test cases and scenarios.
* Integration Testing:
* Objective: the interactions between different components, it test them.
* Advantages:
  + Identify integration errors and issues
  + Validate the system behavior
  + It ensure that the integrated components has the correct functioning it intended to have.
* Limitations:
  + The test can be challenging, because of the dependencies between components.
  + It is consume time and may be complex
* System Testing:
  + Objective: Evaluate the whole software system, to ensure that the system complianc with specific requirements.
* Advantages:
  + Identify errors which are related to system integration, system performance, system usability.
  + Verify the comprehensive functionality and scenarios related to user.
* Limitations
  + It takes time, especially for system which is complex
  + Defects and scenarios, it may not cover all of them.
* Regression Testing:
* Objective: In this test, we ensure that updates and modifications do not come up with new defects.
* Advantages:
  + It maintain the stability of the system, and it support system reliability.
  + It save time and effort
  + Ensure that after changes, the existing functionality still work as expected.
* Limitations:
  + It maybe need a huge effort to maintain and update test cases
  + It can takes time
* Performance Testing:
* Objective: It measures the performance of the system, its scalability, how the system interact and behaviour under different load conditions.
* Advantages:
  + Identify performance issues, the resource limitations, and the bottlenecks.
  + It evaluate the stability of system under stress.
  + It allow to make planning of the capacity and optimization
* Limitations:
  + It may not cover all aspects of performance.
  + The test setup is complex and data management is also complex
  + This type of test requires realist workload scenarios in order to get accurate results

Combination of methods:

In the dental clinic project, in order to ensure the best testing process, So, We may combine all methods.

## 4.2 Automatic testing implementation of developed System

## 4.3 Developer produced VS vendor provided automatic testing tools for applications & software systems

Differences:

1) Ownership and control:

* Developer-produced: he has full control over the design functionalites and other thing such as customization of the tools since these tools are produced within the organiation
* Vendor-Provided: It developed and maintained by third-partiy vendors. There are limitings in terms of controlling these tools underlying code and features.

2)Customization and flexibility:

* Developer-produced: he has more flexibility and he is more able to customize the tool’s behavior and make it integrate with existing systems, and increase its capapbilies.
* Vendor-provided: he has less flexibility and less customizable by comparing them to the Developer produced tools.

3) Integration with development process:

* Developer produced: These tools integrated with the development workflow. Because they are developed in the organization. And they often be very related and tightly coupled with Orangnizaton practices.
* Vendor-Provided: the integration with development process is depending on the tools of vendors. Some of them may require adjustments to fit with the workflows of the development process of their own.

4) Support and Maintenance:

* Developer-Produced: Te maintaince and support are the responsibility of development team since they have deep understanding of their internals.(immediate support)
* Vendor-provided: It is the responsibility of vendor to support and maintenance them. Support volume depends on cpapbilites of vendors

5) Cost and Licensing:

Developer-produced: no licening costs or few, since they have only the cost of development effort which required to create and maintain these tools.

Vendor-Produced: It involves licensing fees based on features of the tools the usage the size of organization

# References

Images references: