

Further programming Final Report



May 24, 2022

Dao kha tuan

S3877347

Answers for Theory questions:

**Question 1:**

Advantages of Dependency Injection:

* Dependency Injection is necessary to create loose – coupling among classes, so that changes of a class will not have significant changes to other classes, which leads to easier code maintenance and improvement in the future.
* It will separate the creation of an object from its usage, so dependency can be replaced without major changes to the code and boilerplate codes will be reduced significantly.
* Client’s knowledge about the implementation of dependencies is removed, so programs can become more reusable, testable, and maintainable.
* It allows many developers to develop the program concurrently.
* It increases the flexibility of the system.

Disadvantages of Dependency Injection:

* Codes are difficult to trace or read as it separates the construction from the behaviors.
* Requires more upfront development effort
* Can have a negative effect to IDE automation
* Increases number of classes

Strategies to resolve the disadvantages:

* Create a more explicit structure and provide more simple modules so codes can be traced and read easily
* Have a good understanding and preparation of Dependency Injection pattern before developing a relevant program or system.
* Utilize IDE that is compatible and also support Dependency Injection such as IntelliJ, Eclipse, etc.
* Provide fewer number of classes and make them more robust and tight coupling.

**Question 2:**

Architect of Hibernate includes these components:

* SessionFactory: an immutable thread – safe cache of compiled mappings for a single database. It can create an instance of Session
* Session: single-threaded, short-lived object representing a communication between the application and the persistent store. It wraps JDBC java.sql.Connection and works as a factory for org.hibernate.Transaction.
* Persistent objects: Persistent objects are short-lived, single threaded objects that contains persistent state and business function. These can be ordinary JavaBeans/POJOs. They are associated with exactly one Session.
* Transient objects: Transient objects are persistent classes instances that are not currently associated with a Session. They may have been instantiated by the application and not yet persisted, or they may have been instantiated by a closed Session.
* Transaction: Transaction is a single-threaded, short-lived object used by the application to specify atomic units of work. It abstracts the application from the underlying JDBC or JTA transaction. A Session might span multiple Transactions in some cases.
* ConnectionProvider: ConnectionProvider is a factory for JDBC connections. It provides abstraction between the application and underlying javax.sql.DataSource or java.sql.DriverManager. It is not exposed to application, but it can be extended by the developer.
* TransactionFactory: A factory for Transaction instances.

List of databases supported by Hibernate: Oracle, MySQL, PostgreSQL, HSQL Database engine, etc.

**Question 3:**

**Singleton Design Pattern:** is a design pattern that ensures that each class can have only one instance. It is used when only a single instance of a class is required to control the action through execution and prevent concurrent access from many parts of the system. To create a singleton class, we create a class that already have an immutable object of itself and can only get it through a public class.

Graphical user interface, text

Description automatically generated

Figure 1: Singleton class Text

Description automatically generated

Figure 2: Demo of Singleton Design Pattern

**Factory Design Pattern:** is a design pattern that provides an interface for a superclass to create objects and the subclasses can modify the objects that will be created. It is used to extend the internal components of a framework or library. It is also used when the types and dependencies of the objects are unknown to the developers or when developers want to reuse existing objects instead of creating them every time. To implement this pattern, first we create an interface of all the objects required to be built. Then create classes that implement this interface. After that, we create a Factory class that create those objects based on the user’s choice.

Text

Description automatically generated

Figure 3: Product classes and a factory class that creates them

Text

Description automatically generated

Figure 4: Demo of Factory Design Pattern

**Builder Design Pattern:** is a design pattern that creates complex objects with different types and representation using the same construction code. It is used to create objects that have different representations without overloading many constructors. To implement this design pattern, we create a class that use a Builder object to populate its attributes. Then we create a Builder class inside it and provide methods to populate each of its attributes as well as validate and build methods to build the object of that class.

Text

Description automatically generated

Figure 5: Class required to be built with different representations

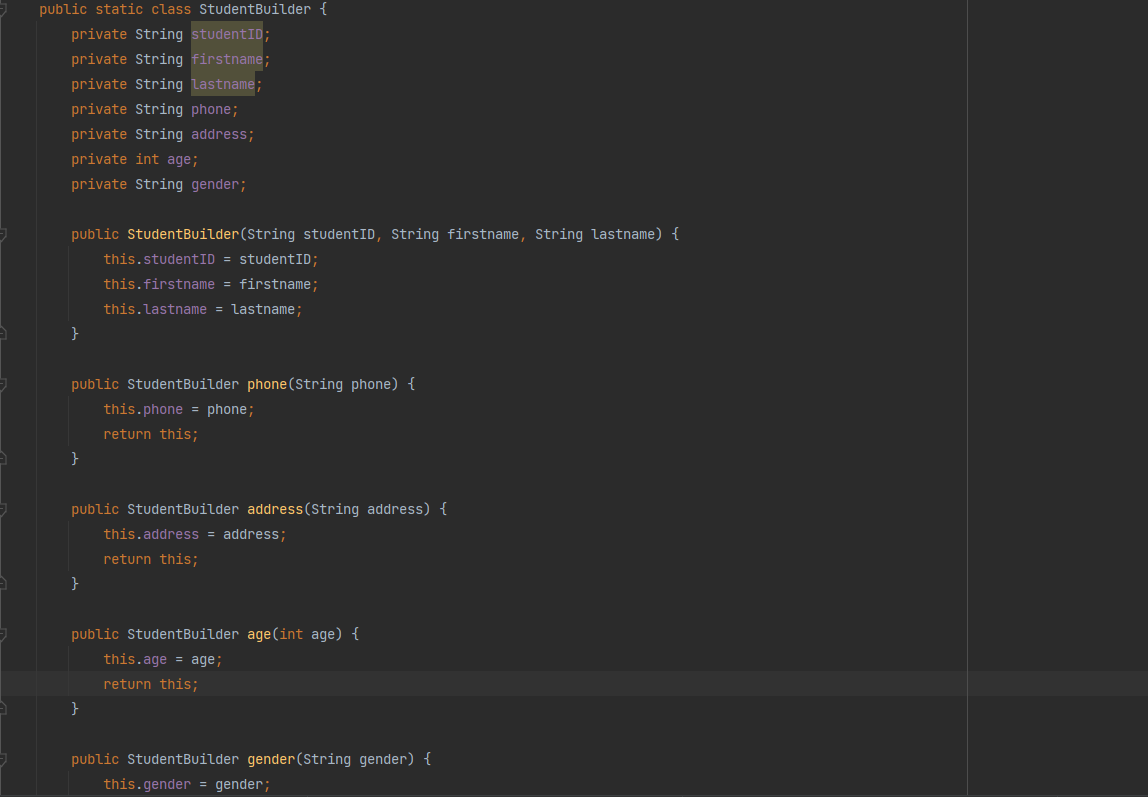


Figure 6: Builder class that populate attributes of the class that contains it

Text

Description automatically generated

Figure 7: build and validate methods of the Builder class

Text

Description automatically generated

Figure 8: Demo of Builder Design Pattern

**Question 4:**

Java IO streams are flows of data between computer programs and I/O devices that we read from or write to them

The application of Decorator Pattern on Java IO stream: some implementations such as BufferedInputStream, ObjectOutputStream, etc. have the constructors that take an instance of the same abstract class or interface to extend more responsibilities and add more extra behaviors to that instance. Thanks to this Decorator Pattern, different concrete classes of Java IO stream can have many flexible combinations with each other without inheritance. For instance, if we wrap the FileOutputStream instance inside ObjectOutputStream instance as below:

new ObjectOutputStream(new FileOutputStream(“myFile.txt”))

The ObjectOutputStream will extend the ability of FileOutputStream, so the data read from the file can be serialized.

**Question 5:**

Advantages of exception handling:

* Program continues to execute after an exception is caught and handled
* Can create meaningful error reports propagated up the call stack.
* Easy to identify the errors and their types

Disadvantages of exception handling:

* It can only detect runtime errors
* Can affect the performance of the program negatively
* The codes that throw exception can break invariants and leave objects in an inconsistent state

Some examples not to use exceptions:

* Exceptions should not be used to handle expected and simple scenarios such as undefined objects or negative values, etc.
* Use exceptions as control flow should be avoided and use other alternatives such as control statements as “if”, “switch” instead
* Exceptions should not be used if there is not third – party developer. Otherwise, the errors should be handled appropriately inside the system.

**Question 6:**

We should favor composition over inheritance: because it is more flexible to change the behaviors of the codes, separates a system into various components and allows them to only use some common behaviors using interfaces. While inheritance can make the codes more complex and provide more boilerplate codes in some situations when there are not many common behaviors among classes.

Scenarios to use inheritance:

* Inheritance is used when there are many common behaviors among classes such as Animal classes of Tiger, Whale, Cat, etc.
* Reduce the redundancy of the codes and reuse the common codes. For instance, Tiger and Cat class can inherit the “walk” method of Animal class.
* Subclasses can add or update the behaviors of the superclass to fulfill its purpose. For instance, Shark class inherit the “swim” method of Fish class but also include the behavior “jump out of water” in “swim” method.