**Week 1 - Software Quality and OOP design**

Development comprises 3 broad activities:

* Analysis
* Design
* Programming

Classes are “concrete entity” (student, course, etc.) or abstract concept or category, to discover classes, look for nouns or significant concepts or artifacts.

1. Association: one class uses another class through attribute level reference (does not imply ownership) (A uses B)

Example: a book can be borrowed by many readers and a reader can borrow many books

1. Aggregation: similar to association, but only refers to attribute level access (not parameter or local variables). It implies part/whole semantics but objects on both sides have independent lifetimes (weak part – whole association). (B is part of A)

Example: a staff can work in a department, a department can have many staffs, but when a department is removed, all staffs will not be removed but can move to other departments

1. Composition: same as aggregation, but the part/whole semantics is enforced, if a whole is deleted, then so are the parts. The “part” object can belong to only one “whole” object as value attribute (strong part – whole association)

Example: a customer can have many electronic wallets and when a customer is removed, the electronic wallets of that customer are removed as well.

1. Dependency: a class uses another class as local variable or an input parameter but not in the attribute. (A uses B)

Example: a shopping cart will depend on the product to perform add operation. When there are changes in the product, the shopping cart may change as well but not vice versa

1. Inheritance: a class inherits all attributes of another class, and also can have its own methods or attributes (A is-a B)

OOP is based on a number of objects working together to perform a function, which can give benefits of easier “code maintenance”, and “reusability”. But a poorly designed OOP program can also be difficult to use and maintain.

Agile development emphasizes an iterative, incremental and evolutionary approach.

Encapsulation indicates that both attributes and methods are contained within a single code module.

Information hiding indicates that private information of a class can only be accessed by its own methods (can have private methods as well). Therefore, outside access to class’s internals are on a “need to know basis” (only some private information can be accessed from outside and no more than that).

The dependence of a class on others is called “coupling”. Many coupling between classes can cause the problems of modifying a class will cause significant changes to many other classes.

Cohesion means that each class has elements with well – defined and logical roles to fulfill its purpose. The object of the class will define one concept or perform one task well.

Cohesion and coupling are trade – off, improve one may have negative effects to others.

**Week 2 - Inheritance and Poly-morphism**

Method overloading is the process of creating many methods with the same name but vary in the type of number of arguments. The actual method invoked is based on the “declared type” at “compile time”

* In inheritance, child class inherits all non – private attributes and methods of parent class even static methods
* Inherited attributes can be modified, and methods are called as if they were part of parent class without having to rewrite or copy them
* This enhances code reusability and extensibility.

The only way to change private attributes and methods in superclass is through the mutators or methods of that superclass.

“Final” keyword indicates that a class or a method cannot be extended or overridden.

Polymorphism enhances code reusability and extensibility by calling the method in a generic way. It can simplify code usage and understandability related classes can be treated in the same way without any conditional codes.

NOTE: a subclass reference is automatically a superclass reference, but not vice versa

Example: class Dog is subclass of class Animal, can convert class Animal to class Dog if class Animal references Dog object. Otherwise, the Animal object cannot be converted to Dog object.

Classes do not inherit constructors

**Week 3 - Abstract Classes and Interfaces**

“Abstract class”:

* cannot be instantiated
* is always inherited and usually enhanced by subclasses
* can have abstract methods
* cannot be instantiated using “new” keyword but can define constructors that can be invoked from subclass, and methods to invoked, implemented, or overridden.

An abstract method:

* is a method signature without any implementation
* can only exist in an abstract class
* a subclass of an abstract superclass must implement all abstract methods of that superclass. Otherwise, it must become abstract class
* can exist in interfaces.

A subclass can be abstract even though the superclass is concrete.

“Interface” is a class – like constructor with only constant and abstract methods. It is similar to abstract class but cannot contain variables and concrete methods.

**Week 4 - Java Collection Framework**

Collection: An Object that groups many elements into a single unit in order to store, retrieve, manipulate and transmit data from one method to one another.

Java Collection Framework: A ‘Unified Architecture’ for representing and manipulating collections. Collection Framework contains:

* Interfaces: abstract data type representing collections, interfaces can be manipulated independently from the details of their representation.
* Implementation: concrete implementations of the collection interfaces, which are reusable data structures
* Algorithms: methods that perform useful computations such as searching and sorting on the objects that implement collection interfaces. They are polymorphic so the same method can have different implementations on appropriate collection interfaces. They are reusable functionality.

Iterator Interface: Iterator allows the caller to delete elements from the underlying collection with well-defined implications during iteration.

List: is an ordered Collection (sometimes called a sequence) and it can contain duplicate elements. It also has additional operations beside ones inherited from Collection:

* Position Access (indexed): manipulate elements based on their numerical position in the list
* Search: search for a specified object in the list and return its numerical position
* List Iteration: extend Iterator semantics to take advantage of the list’s sequential nature
* Range – view: perform arbitrary range operation on the list

Types of implementations of List:

* ArrayList: for general purpose and stored in resizable array
* LinkedList: best performance under particular situations and implements Deque interface
* Vector: reconstructed to implement List and it is also Thread safe.

Functionalities of List interface:

* Provides an ordered collection
* Allow duplicate elements (not as Set)
* Elements can be stored or retrieved using their index or position.
* Has index operations
* Provides a special ListIterator to traverse in both directions.

ArrayList vs LinkedList:

* Similarities:
* Grows dynamically
* Implement List methods
* Differences:

|  |  |
| --- | --- |
| ArrayList | LinkedList |
| Random access | Sequential access |
| Faster at accessing any element in the list | Faster at inserting or deleting any element in the list |
| Slow at inserting element in the middle due to copying overhead | Slow at accessing element due to traversing sequentially |

Set: A Collection that cannot contain duplicate elements, it extends Collection and contains only methods inherited from Collection. But classes implementing Set interface must ensure no duplicates.

Types of implementations of Set:

* HashSet: best performance and stored in hash table but random ordering, it provides a concrete implementation of Set
* TreeSet: order is guaranteed and stored in red – black tree, provides a concrete implementation of SortedSet. Objects inserted must be comparable with each other
* LinkedHashSet: order by insertion order, it extends HashSet with a linked list and supports ordering. It can retrieve element in order insertion.

Map: a collection that maps key to value, it cannot contain duplicate keys and each key can map to at most one value.

* HashMap: best performance and stored in hash table, efficient for locating a value, inserting, or deleting mapping. Entries are not ordered.
* LinkedHashMap: extends HashMap with a linked list and can be accessed in order of insertion.
* TreeMap: order is guaranteed and stored in red – black tree, implements SortedMap interface and efficient for accessing Map in sorted order.
* Hashtable: reconstructed to implement Map

Queue: a first – in, first – out structure but can be ordered arbitrarily

Deque: can be used as both first – in, first – out and last – in, last – out. It extends the Queue interface and new elements can be inserted, retrieved and removed at both ends.

SortedSet: a Set that maintains its elements in ascending order according to their natural order or Comparator provided at SortedSet creation time. It offers operations for:

* Range-view: Performs arbitrary range operations on the sorted set.
* Endpoints: Returns the first or last element in the sorted set.
* Comparator access: Returns the Comparator used to sort the set

SortedMap: a Map that maintains its elements in ascending order according to their natural order or Comparator provided at SortedMap creation time. It offers operations for:

* Range-view: Performs arbitrary range operations on the sorted map.
* Endpoints: Returns the first or last element in the sorted map.
* Comparator access: Returns the Comparator used to sort the map

**Week 5 - OOP for enterprise**

Tight – coupling means that one class is highly dependent on another class and modification of one object in a tight – coupled application can lead to significant changes to other objects.

Loose – coupling is opposite to tight – coupling, indicates that two classes are independent. A loose – coupled application can decrease maintenance and effort.

Dependency Injection is a software design concept that enables a service to be injected to an object in a completely independent way from any client consumption. It indicates that an object should only depends on the Abstract instance and the instance will be implemented and injected into the object at runtime.

Dependency Injection is also called Inversion of Control because in this concept, the object does not need to find and call the dependencies, but they are injected to the object when it is created or returned from a factory method through constructor arguments, arguments to a factory method, or properties that are set on the object instance.

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.

Some frameworks that support Dependency Injection in Java: Spring, Google Guice, Google Play, Google Web Toolkit (GWT), Eclipse RCP, etc.

POJO: (Plain old Java object): a straightforward ordinary type not bound by any restriction or references to any frameworks. It has no naming convention for properties and methods.

JavaBean: same as POJO but follows a strict set of rules around its implementation.

BeanFactory: a factory that produces bean (Java object), which implements factory design pattern. It offers an advanced configuration mechanism to be able to manage any type of object.

ApplicationContext is a sub – interface of BeanFactory. It offers:

* Easier integration with Spring’s AOP features
* Message resource handling (for use in internationalization)
* Event publication
* Application-layer specific contexts such as the WebApplicationContext for use in web applications.

It is responsible for instantiating, configuring, and assembling the beans.

Apache Maven is a software project management and comprehension tool. It is based on the concept of Project Object Model (POM) to manage a project’s build, reporting and documentation from a central piece of information. Maven can be used to build and manage any Java based projects.

The POM contains all necessary information about a project, as well as configurations of plugins to be used during the build process.

**Week 6 - OOP for data persistence**

Object-relational Mapping (ORM) is a programming technique that maps the domain model objects of application to tables of relational database

Some popular ORM tools: Hibernate (the most popular), Toplink, Eclipse Link, Open JPA, etc.

|  |  |
| --- | --- |
| ORM framework | Traditional JDBC |
| Less boilerplate codes | Many boilerplate codes |
| Take care of managing resources and support transaction management, not to worry about resource leaks and data inconsistency of database operations | There are chances of resource leaks and data inconsistency |
| Provide an abstraction layer between application and database | Used for connecting to database and perform CRUD operations at low level |
| Developers can write fewer manual codes to store objects or data to database | Developers must write many manual codes to store objects or data to database |
| Little slower than JDBC | Faster compared to ORM |

Benefits of Hibernate:

* Helps us in mapping joins, collections, inheritance objects and easily visualizing how database tables are represented by model classes.
* It provides a powerful query language (HQL) similar to SQL. However, it is full object – oriented and understands OOP concepts such as inheritance, polymorphism and association.
* Provide integration with some external modules.
* Small learning curve and many documentations.
* It is easy to integrate with other Java EE frameworks.

**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.

JPA: is a programming interface specification for mapping Java objects to a relational database. It is a possible approach to ORM.

**Week 8 – MVC**

Model – View – Controller (MVC) is a software design pattern that allows different aspects of the applications such as input logic, business logic and UI logic to be separated, and creates a loose coupling between these components:

* Model: encapsulate the data of the application and consists POJO in general.
* View: responsible for rendering the data of Model and provides HTML output to the client’s browser for the users to view.
* Controller: responsible for handling users’ requests and building appropriate models, which will be transferred to the View for rendering to users.

Servlet is a web application technology that is used to create a dynamic web application that is situated at server side.

The workflow of Spring Web MVC DispatcherServlet:

* A HTTP request is sent to DispatcherServlet
* After that, DispatcherServlet asks the HandlerMapping to call the suitable Controller
* The Controller will take the request and base on the GET or POST method to call the appropriate service methods to set the model data. After the model data is set based on define business logic, it will return the view name to DispatcherServlet.
* The DispatcherServlet will seek help from ViewResolver to select the defined view for the request
* After view is finalized, the view will be passed with the model data, which will be rendered on the browser.

Representational State Transfer (REST) is an architectural style that uses HTTP requests to access and use data.

**Week 10 - Streams/Serialisation/Files**

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

Some stream implementations in Java:

* FileInputStream: it can read input bytes from a file in file system, it is used to read streams of raw bytes.
* FileOutputStream: it is an output stream to write data/streams of raw bytes to a file.
* BufferedReader: it can read data from the stream and store in an internal buffer.
* BufferedWriter: it uses internal buffer to write data to the output stream
* FileReader: it can read streams of characters from a file
* FileWriter: it can write streams of characters to a file
* ObjectOutputStream: it can write primitive data types and graphs of Java objects to output stream
* ObjectInputStream: deserializes primitive data types and objects previously written using ObjectOutputStream