POO

- Class Design Guidelines
  - Modeling Real World Systems
    - In contrast of what a structured programming is able to do, Object Oriented Design (OOD) is an evolve. OO programming allows us to encapsulate the data and behavior of the objects. This modifies everything, even the mindset required to program. Using Grady Booch's classes that literally model real world objects.
  - Identifying the Public Interfaces
    - The public class interface should be minimum, the minimum necessary to accomplish what it needs to be done with the interface. If the interface is not quite complete, or has accessibility problems where the user is able to affect the methods or data of the object, it could lead us to having many problems with maintaining of the system and errors.
    - The minimum Public Interface
    - Hiding the Implementation
      - When the public interface is well designed there's no need to know how it does what it does. Hiding and creating an opacity wall for this implementation must provide to the user, concerns that he needs, but don't need to know how are they being performed.
  - Designing Reduced Constructors
    - How the class is going to initiate is an important issue, thereby the constructor must provided the object in creation an initial safe state. Considering, the constructor must initialize the object, set its fields, and manage the memory. As designer of the class, must provide a proper constructor.
  - Designing Error Handling into a Class
    - Since every system is going to face unforeseen problems, the designer must anticipate potential errors including code to deal with them, the system must never crash whether it expects it, self or gracefully exits without any data loss.
  - Documenting a Class and Using Comments
    - The documentation is never taken seriously or ignored. The lack of proper documentation and comments will make the system larger, the programmer of the class is able to work with a class without any documentation, but when this class is reused by another programmer, it is not only hard for the class to grow in code.
  - Building Objects with the Intent to Cooperate
    - A class is destined to interact with other classes, it must be aware of the fact that, for the designing has to be aware of the fact.
  - Designing with Reuse in Mind
    - The code of a class should be created with the reuse in mind, so this is delegated to the design when one will create classes that will serve to others in the future.
  - Designing with Extensibility in Mind
    - One of the important aims of OO programming is the reuse of code. a class will get new features, new methods, and new data, without affecting the existing objects and inherit wonders.
    - Making names Description
    - Abstracting out Nonportable Code
      - If a System designed to run specifically in certain hardware the code must be abstract. The system must be able to be able to execute certain things from the primary data.
    - Providing a way to Copy and Compare Objects
      - The copy and comparation is not a simple thing, you can copy and compare a object in many ways, it is the designing process you must take this in count.
    - Keeping the scope as Small as Possible
      - Using abstraction and implementation you will be able to keep the scope as small as possible. Allows maintaining the class, then it and maintain it is easier.
    - A class should be responsible for a Set
      - An object responsible for itself, so no matter if the data of objects are identical, depends of the object how the system will behave.
  - Designing with Maintainability in Mind
    - When a class is useful and consise the maintainability is easier. Separate pieces of code are easie to maintain than larger pieces, with this in mind it best to reduce the interdependent code. This work with the concept where designers should write code that is easy to the implementation of the object, the public interface is untouchable.
    - Using interaction in the Development Process
      - The code is created in small increments, then build estimated. This process iterates until the class has proper interfaces. This doesn't apply only to code. Also to the design it self.
    - Testing the interface
      - The minimal implementation of the interface is called stubs. Stubs are used to test the interface without writing a real code. It's all a simulation. This allows to find design and coding problems, which will modified and the result is an evolution.
  - Using object Persistence
    - The concept of maintaining the state of an object, which allows to be saved to a data and access to it later. This is a common problem in OO programming problem, as the object must be saved for later. The can happen with a flat file system, that allow us to save an object by serializing it, or using a database, which is more complex. The object oriented database, the most reliable, but not a great thing to do, because the relational databases must be converted.
- Object
  - Attributes
    - Defined by dot that will store the object, which tied up to the behavior.
    - Acces Modifier
      - The access to de data and methods is different for every class, or in this case the object, public, private are the main modifiers, but that the default or presented depends of the language programming at.
  - Behavior
    - Objects contain functions or methods that are used to perform operations with the data and other objects. This uses are resolved by calling them or sending them a message.
    - Implementation
      - The public implementation is the one that the client is able to see, also the one that conforms the interface and the private is inaccessible considered part of the internal implementation.
    - Overloading Methods
      - This functionality allows the programmer to use a methods in different ways, by specifying the signature in a different way.
    - Private Implementation Methods
      - Not all the methods in a class should be public, some of them should be private. The private methods are meant to be part of the implementation. These methods, are called internally from the class itself.
    - Public Interface Methods
      - The accessor methods and the constructor are declared public and part of the interface. They are singled out because of their specific importance to interconnect the class.
    - Operator Overloading
      - This means that you can change the meaning of an operator. This can be pretty good or pretty badly used, for example, the operator + go to do with the meanings of the operators.
  - Encapsulation
    - Interfaces
      - Interfaces define the communication between objects.
    - Getters and Setters
      - Let the creator control the access tools the data and methods of the object.
      - Based on the concept of data hiding, an object should not manipulate directly another object, that's getter and setter let indirectly controlled access to an object data.
  - State
    - The data stored within an object represents the state of the object.
- Class
  - Inheritance
    - A class is like a template of an object. The basic of how the object is built.
    - Multiple Inheritance
      - A class can inherit from multiple classes. Since objects are supposed to represent real world things, multiple inheritance is a powerful tool to model complex relationships. But it increases the complexity of the system, for everyone.
  - Composition
    - Allows the data to inherit attributes and methods of another class, we can create objects new class by aggregating the common attributes and behaviors.
  - Superclasses and Subclasses
    - Acts like a father, that meant it contains all of the attributes and behaviors that are common in the classes that inherit from it. The Subclass or child class is the extension of the superclass.
  - Polymorphism
    - The Superclass has to be well designed so the child classes can efficiently inherit all the determinate attributes.
  - Abstraction
    - When a message is sent to the object, the object must have a method to answer the specific message.
  - Constructors
    - Depends of the language your working on, but usually the constructor has the same name of the class. constructor does not have a return value.
    - Calls
      - At the time an object is being created, the constructor is called, that allocates the memory required for the object. Then the arguments are being passed as specified in the constructor interface.
    - Structure
      - The main function of the constructor is to initialize the memory allocated for the new object, the object should be created in a safe, stable and safe state.
    - Default
      - When the constructor is not declared in a class, the class will work anyway, there is constructor provided. this constructor creates the object and initializes the attributes to default.
    - Multiple Constructors
      - Depending of how you want to create an object, depends of the situation, you can provide different ways to create a completely related to overloading a method.
    - The Design
      - It's pretty well known that all attributes that is be private, and that is a good idea. Remember that the constructor is not a good idea. Remember that the constructor is there so the application is in a stable state. Attributes don't act in the same ways as others, so during the design, you need to identify a proper way to initialize all of them.
  - The Name of the Class
    - The name of the class should be representative of what the class is, it should be a good idea, what the class does and how it interacts within larger systems.
  - Comments
    - A class, or better, a class should be the function of a class, or pretty much anything else.
  - Attributes
    - Attributes represent the state of the object, so they are the most important part of the class. Depends of the concept public or private, public means that only the declaring object is able to access to it. The data means that there will only be one copy of the attribute for all the objects instantiated from the class.
  - Accessors
    - A class should be very protective of it's attributes. But there will be times when an object needs to access to another object's attributes but not directly.
- OO vs Procedural
  - Scope
    - Multiple objects can be instanciated from a singles class, each object with it's own space of memory, attributes and methods. There are three types of attributes.
    - Local Attributes
      - Are owned by an specific method. you can have an attribute that has the same name in two different methods, but the scopes still were two separate attributes, even if you have the same name.
    - Object Attributes
      - When the attribute has to be shared to different methods in the same object. The can be several object with the same name of attributes, but the attributes belong to the object in reality.
    - Class Attributes
      - By making an attribute static two or more objects can share attributes. By declaring the attribute static there's a single piece of memory for all objects. Changes made by one object, there's a single copy shared to all objects of the class.
  - Interface and Implementation
    - The services presented to an end user compose the interface.
    - The implementation details are hidden from the user.
  - Design of the interface
    - Determining the users.
      - It depends of what the user wants, but the software developer has to decide if the product is usable.
    - Object Behavior
      - From the point of view of the users, you must identify the purpose of each object, design it so it would perform it properly.
    - Environmental Constraints
      - The limitations of what an object can do is often set by the environment. factor. So it's pretty important to think of it.
    - Public interfaces
      - Once you have gathered all the information of the users, behaviors, environments, its time to determine the public interfaces you have to offer to the users.
    - The implementation
      - After all the design has been done, and you know the public interfaces, it's time to create the implementation. Remember that the interface is not public. So the user might never see any of the methods that compose the implementation.
  - Error Handling
    - Since nothing can be perfect, it's good practices to get ready if anything happens. The are four thing that you can do in case of getting any troubles working with OOP.
    - Ignoring the Problem
      - Ignoring the problems is never a good thing to do. As a software developer your code is it will be maintained available.
    - Checking for Problems and Aborting the Application
      - When a potential problem appears abort the application, it means that application exits. It's not a bad idea. It's better than application crashes and put the application in a more stable state.
    - Checking for Problems and Attempting to Recover
      - The application tries to fix it self. It's not always easy to determine when a problem first appears, but it should be part of the designing of the class.
    - Throwing an Exception
      - Provided from most of the languages, exceptions are a powerful way to deal with errors. When an error is detected, and the system takes care of it...
      - Mainly used by the keywords catch and throw.
- Object Wrappers
  - Let the programmer create an object that has involved code enclosing it inside.
- unified Modeling Language
  - The benefits of Modeling Software
    - Since every project is an abstraction of an idea, the development team must create a conceptual model to transform that idea into a functional system.
    - Model
      - A model is simplification of reality, or a description of static and dynamic subject area. An abstract conceptualization of some entity.
    - Why model software?
      - The main benefit from modeling a software is that the development team can comprehend the system that they're developing.
    - Communicate decisions to the project stakeholders
      - Visualize ideas or existing systems.
      - Conceptualize the kind of system is to develop.
      - since every project is different and has its own level of abstraction, the communication is a fundamental part. The models provided comprehensive, what the system is about.
    - Document the benefits of Modeling software
      - not everyone involved in the working team is involved in the requirements gathering and one way to communicate this information to them is documenting this process.
    - Specify the structure and behavior elements of a system
      - In the design and implementation workflows, the structural dynamic must be specified. The solution model point is to represent the complete architecture of the system.
  - OOSD as Model Transformations
    - From the mental models, we can conclate that during the requirements gathering, and analysis workflows to requirements model, the non-functional requirements into the Architecture model.
    - Functional requirements
      - This type of requirements are transformed into a Design model, that defines the abstract components of the solution.
    - UML
      - Is a graphical language for visualizing, specifying, constructing and documenting artifacts of a software intensive system.
      - UML Diagrams
        - Use Case Diagram
          - represents the set of high level behaviors that the system must perform for a given actor
        - Class Diagram
          - A collection of Software classes and their interrelationships.
        - Object Diagram
          - Represents a runtime snapshot of software objects and their relationships.
        - Communication Diagram
          - The collection of objects that work together to support some system behavior.
        - Sequence Diagram
          - Represents a time oriented perspective of an object communication.
        - Activity Diagram
          - represents a flow of activities that might be performed by either a systems or an actor.
        - State Machine
          - The set of states that an object might experience in its lifetime, triggers that transition the object form one state to another.
        - Component Diagram
          - Collection of physical software components and their relationships.
        - Deployment diagram
          - Collection of components and shows how these are distributed across one or more nodes.
        - Package diagram
          - Collection of another modeling elements and diagrams.
        - Interaction Overviews
          - Activity diagram where nodes can represent interaction diagram fragments, can for have sequence, communication, timing or interaction overview.
        - Timing Diagram
          - represents the changes in state or value
        - Composite Structure Diagram
          - Represents the internal structure of a classifier, usually in form of parts.
        - Profile Diagram
          - Additional diagram types or extend existing diagrams with additional notations.
      - Components
        - Diagrams
        - Relationships
        - Usage
        - Views
        - Elements
      - UML elements and connectors
        - Package
          - Used to group together any UML elements and diagrams, elements therefore they can be nested.
        - Note
          - Allows textual nodes to be added to any aspect of a diagram. Can be classes, methods, components, actors,associations, etc.
        - Dependency
          - The typical dependency can be attachment line with a stereotype.
        - Stereotypes
          - Are used to declare a more specific type of elements or connector type
            - Actor
              - Defines a coherent set of interaction users of an entity can play when interacting with the entity.
            - Create
              - Create is a stereotyped usage dependency, denoting that one client cognitive creates of the supplie classes.
            - Extend
              - Indicates the instance of use case B, by an specified behavior from case A.
            - Import
              - Subtype 1.

XMind: ZEN | Trial Mode