Design Constructs: Part I

Structure of Programs

- Expression → often evaluates and gives you a result. Eg. adding two values and getting the result back.
- Statement → often performs an action. Eg. printing something.
- Compound Blocks → these are a collection of statements or expressions.
- Conditional → usually a branch. If so & so, do this, else, do something else.
- Loops → For loop, While loop, Do while loop.
- Functions/Methods → can have an expression, can have multiple Compound Statements.
- Classes → collection of functions, can have variables, methods.
- Packages/Namespaces
  - Packages → usually a logical distinction
  - Namespaces → usually a physical distinction
  - The difference matters on the context/language.

Paradigms

Object Oriented Paradigm

- We create objects — which are a collection of a certain amount of data and behaviour around that data.
- Pillars of the paradigm
  - **Abstraction**
    - A way to represent a certain idea/concept.
  - **Encapsulation**

        - A way to make that a particular implementation is kept separate from interface. We want to keep what we do with an abstraction separate from how the abstraction implements.
        - We can change the implementation without making any changes to the interface.
        - Eg. cars of different models and makes have the same interface — that is, if you can drive a Chevrolet, you can also drive a Toyota.
    - **Inheritance**
        - Gives the ability to take an abstraction and build further on that abstraction by extending it.
        - Weakest of the four pillars.
    - **Polymorphism**
        - Given an object or a reference to an object, the method that is called is not based on the type of the reference, but based on the real type of the object at runtime.
        - **The most important feature in OOP.**
            - It is polymorphism that gives extensibility in OO systems.
        - Strongest of the four pillars.
- Polymorphism combined with Encapsulation is the real powerful pillar and the other two are there to support these.
- When a function depends on an interface, it is more reusable than when it depends on a class.