12 OOP and Functional Programming

12.67 Basic Concepts of Object-oriented Programming

A program written in **procedural languages** is written using a series of **step-by-step** instructions on how to solve the problem.

- Broken down into a number of **smaller modules**.
- The program consists of a series of calls to **procedures or functions**.
- Which in turn call other procedures or functions.

In **object-oriented programming**, the world is viewed as a **collection of objects**, each responsible for its own data and the operations on that data.

- A program creates **objects**, and
- Allows the objects to **communicate with each other** through sending and receiving **messages**.
- All processing is done by objects.

Each object has its own **attributes, state and behaviours** (actions that can be performed by the object).

Classes

A class is a **template for an object**, it defines

- An **attribute** is data associated with the class.
- A **method** is a functionality of the class.
- A **constructor** is used to create objects.

The principle of **information hiding**: other classes cannot directly access the attributes of another class declared private.

Instantiation is the creation of objects - multiple instances of a class each share identical methods and attributes, but the values of attributes will be unique to each instance.

An object **encapsulates** both its state and its behaviours, so that the attributes and behaviours of one object cannot affect the way another object functions.

Inheritance

Subclasses can inherit data and behaviour from a **superclass**.

- The *"is a"* rule asks *"is object A an object B"* before it can inherit from the object.

12.68 Object-oriented Design Principles

Association is a *"has a"* relationship between classes.

- **No ownership** between objects.
- Each have their own lifecycle - can be created and deleted independently.

Aggregation is a type of association.

- A class is a container of other classes.
- The contained class do not have a strong **lifecycle dependency** on the container.

Composition is a stronger form of association.

- If the container is destroyed, every instance of the contained class is also destroyed.
- **Polymorphism** - the programming language's ability to process objects differently **depending on their class**.
- **Overriding** - defining a method with the same name and formal argument types as a method inherited from a superclass.

Composition is generally considered preferable to inheritance as it allows **greater flexibility** - is a less rigid relationship.

Access Modifiers

Information hiding: object's instance variable are hidden so other objects **must use messages** to interact with that object's state.

- **public** - code within any class can see it.
- **private** - only code within the class itself can access it.

Interface

An interface is a **collection of abstract methods** that a group of unrelated classes may implement.

- Methods will only be implemented by a class that implements the interface, not the interface itself.

The strategy of **encapsulate what varies** reduce maintenance and testing effort.

- Using an **interface class** implemented by different classes - code that relies on the interface can **handle any class** implementing the interface.
- If something changes in a program, only that module will need to change.

Advantages of Object-oriented Paradigm

- Forces designer to go through a planning phase, which makes better design and fewer weaknesses.
- **Encapsulation** - source code for an object can be written, tested and maintained independently.
- Details of how methods are implemented is not necessary in order to use it.
- New objects similar to existing ones can easily be created.
- **Re-usability** - tested objects may be used in many different programs.
- **Maintenance** - an OO program is much easier to maintain because of its **rigidly enforced modular structure**.