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, <u>not the interface itself</u>.

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**.

12.69 Functional Programming

A **programming paradigm** is a style of computer programming, different programming languages support tackling problems in different ways.

- **Procedural programming** have a series of instructions that tell that computer what to do with the input in order to solve the problem.

  Structured programming is a type of procedural programming which uses the programming construct of **sequence, selection, iteration and recursion**. It uses modular techniques to split large programs into manageable tasks.
- **Object-oriented programming** makes it possible to **abstract details of implementation** away from the user, make code **reusable** and programs **easy to maintain**.
- **Declarative programming** is where you write statements to describe the program to be solved, and the language implementation decides the best way of solving it.
- In **functional programming**, functions are used as the fundamental building blocks of a program. Statements are written as a **series of functions** which accept input data as arguments and return an output.

A function is a mapping from a set of inputs, called the domain, to a set of possible outputs, known as the co-domain.

The process of giving particular inputs to a function is known as **functional application**.

In functional programming, a **first-class object** is an object which may

- Appear in expressions.
- Be assigned to a variable.

- Be assigned as an argument.
- Be returned in a functional call.

Features of Functional Programming Languages

- **Statelessness**: In a functional programming language, the **values of variables** cannot change. Variables are staid to be **immutable**, and the program is said to be **stateless**.
- **No side effects**: The only thing that a function can do is calculate something and return a result.

  As a consequence of not being able to change the value of an object, a function that is called twice with the same parameters will always return the same result. This is called **referential transparency**.

  - Makes it relatively easy for programmers to write correct, bug-free programs.
  - A simple function can be proved to be correct, then more complex functions can be built using these functions.
- **Functional composition**: combine two functions to get a new function.
- **Types** are sets of values.
- **Typeclasses** are sets of types.
- A **type variable** represents any type.

12.70 Functional Application

A **higher-order function** is one which either takes a function as an argument or returns a function as a result.

Partial application means binding the values of some inputs to a function to produce another **more specific function**.

- **Map** is a higher-order function that takes a list and the function to be applied to the elements in the list as inputs, and returns a list by applying the function to each element in the old list.
- **Filter** is a higher-order function which takes a **predicate** and a list, this returns the elements within the list that satisfy the boolean condition.
- **Fold** reduces a list into a **single value** using recursion.

12.71 Lists in Functional Programming

A list is a **collection of similar elements** of a similar type, enclosed in square brackets. It is composed of a **head** and a **tail** - the head is the first element of the list, the tail is the remainder of the list.

- The function **null** tests for an empty list.

```haskell
null []  -- True
null [1] -- False
```

- **Prepending** means adding an element to the front of a list.

```haskell
5:[4,3,2,1] -- [5,4,3,2,1]
```

- **Appending** means adding an element to the end of a list.

```haskell
[1,2,3,4] ++ [5] -- [1,2,3,4,5]
```

- **Length** finds the length of the list.

```haskell
length [1,2,3,4,5] -- 5
```