Programming Paradigms

What is a Programming Language?

- Formal notation for specifying computations, independent of a specific machine
  - Example: a factorial function takes a single non-negative integer argument and computes a positive integer result

- Set of imperative commands used to direct computer to do something useful
  - Print to an output device: printf(“hello world\n”)

Principal Paradigms

- Programming paradigms are the result of people’s ideas about how programs should be constructed
  - … and formal linguistic mechanisms for expressing them
  - … and software engineering principles and practices for using the resulting programming language to solve problems
- A programming paradigm is a fundamental style of computer programming.
- Compare with a software development methodology, which is a style of solving specific software engineering problems.

Principal Paradigms

- A programming paradigm can be understood as an **abstraction of a computer system**, for example the von Neumann model used in traditional sequential computers.
  - Prevalent computer processing model used is the **von Neumann model**, invented by John von Neumann in 1945,
    - Data and program are residing in the **memory**.
    - **Control unit** coordinates the components sequentially following the program's instructions.
    - **Arithmetic Logical Unit** performs the calculations.
    - Input/output provide interfaces to the exterior.
  - The **program** and its **data** are what is **abstracted** in a programming language and translated into machine code by the compiler/interpreter.

Principal Paradigms

- Imperative / Procedural
- Functional / Applicative
- Object-Oriented
- Concurrent
- Logic
- Scripting
- Very few languages are "pure". Most combine features of different paradigms
  - The design goal of multi-paradigm languages is to allow programmers to use the best tool for a job, admitting that no one paradigm solves all problems in the easiest or most efficient way.

Procedural programming

- Often thought as a synonym for **imperative programming**.
  - Specifying the **steps** the program must take to reach the desired **state**.
  - Based upon the concept of the **procedure call**.
  - Procedures, also known as routines, subroutines, methods, or functions that contain a series of computational steps to be carried out.
  - Any given procedure might be called at any point during a program's execution, including by other procedures or itself.
- Using a procedural language, the programmer specifies **language statements** to perform a **sequence of algorithmic steps**.

Procedural programming

- Possible benefits:
  - Often a better choice than simple sequential or unstructured programming in many situations which involve moderate complexity or require significant ease of maintainability.
  - The ability to re-use the same code at different places in the program without copying it.
  - An easier way to keep track of program flow than a collection of "GOTO" or "JUMP" statements (which can turn a large, complicated program into spaghetti code).
  - The ability to be strongly modular or structured.

Object-oriented programming paradigm

- Object-oriented programming (OOP) is a programming paradigm that uses "objects" – data structures encapsulating data fields and procedures together with their interactions – to design applications and computer programs.

- The most important distinction is whereas procedural programming uses procedures to operate on data structures, object-oriented programming bundles the two together, so an "object" operates on its "own" data structure.

- Associated programming techniques may include features such as data **abstraction**, **encapsulation**, **modularity**, **polymorphism**, and **inheritance**.

Object-oriented programming paradigm

- A class defines the abstract characteristics of a thing (object), including that thing's **characteristics** (its attributes, fields or properties) and the thing's **behaviors** (the operations it can do, or methods, operations or functionalities).
  - One might say that a class is a blueprint or factory that describes the nature of something.

- Classes provide **modularity** and **structure** in an object-oriented computer program.

- A class should typically be recognizable to a non-programmer familiar with the **problem domain**, meaning that the characteristics of the class should make sense in context. Also, the code for a class should be relatively self-contained (generally using encapsulation).

Object-oriented programming paradigm

- Collectively, the properties and methods defined by a class are called its members.

- An object is an individual of a class created at run-time trough object **instantiation** from a class.

- The set of values of the attributes of a particular object forms its **state**. The object consists of the **state** and the **behavior** that's defined in the object's class.

- The object is instantiated by implicitly calling its constructor, which is one of its member functions responsible for the creation of instances of that class.

Object-oriented programming paradigm

- Attribute
- An **attribute**, also called data member or member variable, is the data encapsulated within a class or object.
- In the case of a regular field (also called **instance variable**), for each instance of the object there is an instance variable.
- A static field (also called **class variable**) is one variable, which is shared by all instances.
- Attributes are an object's variables that, upon being given values at instantiation (using a **constructor**) and further execution, will represent the state of the object.

Object-oriented programming paradigm

- Attribute

- A class is in fact a data structure that may contain different fields, which is defined to contain the procedures that act upon it. As such, it represents an **abstract data type**.

- In pure object-oriented programming, the attributes of an object are local and cannot be seen from the outside. In many object-oriented programming languages, however, the attributes may be accessible, though it is generally considered bad design to make data members of a class as externally visible.

Object-oriented programming paradigm

- Method

- A **method** is a subroutine that is exclusively associated either with a class or with an object.
  - ***instance*** methods are associated with an object
  - ***class*** or ***static*** methods are associated with a class.

- Like a subroutine in procedural programming languages, a method usually consists of a sequence of programming statements to perform an action, a set of input parameters to customize those actions, and possibly an output value (called the return value).

Object-oriented programming paradigm

- Method
- Methods provide a mechanism for accessing and manipulating the encapsulated state of an object.
- Encapsulating methods inside of objects is what distinguishes object-oriented programming from procedural programming.

Object-oriented programming paradigm

- Method

- The POO favors the use of methods for each and every means of access and change to the underlying data:
  - **Constructors:** Creation and initialization of the state of an object. Constructors are called automatically by the run-time system whenever an object declaration is encountered in the code.
  - **Retrieval and modification of state:** accessor methods are used to access the value of a particular attribute of an object. Mutator methods are used to explicitly change the value of a particular attribute of an object. Since an object's state should be as hidden as possible, accessors and mutators are made available or not depending on the information hiding involved and defined at the class level

Object-oriented programming paradigm

- Method

- The POO favors the use of methods for each and every means of access and change to the underlying data:
  - **Service-providing:** A class exposes some “service-providing” methods to the exterior, who are allowing other objects to use the object’s functionalities. A class may also define private methods who are only visible from the internal perspective of the object.
  - **Destructor:** When an object goes out of scope, or is explicitly destroyed, its destructor is called by the run-time system. This method explicitly frees the memory and resources used during its execution.

Object-oriented programming paradigm

- Inheritance

- Inheritance is a way to compartmentalize and **reuse** code by creating collections of attributes and behaviors (classes) which can be based on previously created classes.
- The new classes, known as **subclasses** (or derived classes), inherit attributes and behavior of the pre-existing classes, which are referred to as superclasses (or ancestor classes). The inheritance relationships of classes gives rise to a hierarchy.

Object-oriented programming paradigm

- Inheritance

- **Multiple inheritance** can be defined whereas a class can inherit from more than one superclass. This leads to a much more complicated definition and implementation, as a single class can then inherit from two classes that have members bearing the same names, but yet have different meanings.
- **Abstract inheritance** can be defined whereas abstract classes can declare member functions that have no definitions and are expected to be defined in all of its subclasses.

Object-oriented programming paradigm

- Abstraction

- **Abstraction** is simplifying complex reality by modeling classes appropriate to the problem, and working at the most appropriate level of inheritance for a given aspect of the problem.
- For example, a class Car would be made up of an Engine, Gearbox, Steering objects, and many more components. To build the Car class, one does not need to know how the different components work internally, but only how to interface with them, i.e., send messages to them, receive messages from them, and perhaps make the different objects composing the class interact with each other.

Object-oriented programming paradigm

- Encapsulation and information hiding

- **Encapsulation** refers to the bundling of data members and member functions inside of a common “box”, thus creating the notion that an object contains its state as well as its functionalities
- **Information hiding** refers to the notion of choosing to either expose or hide some of the members of a class.
- These two concepts are often misidentified. Encapsulation is often understood as including the notion of information hiding.

Object-oriented programming paradigm

- Encapsulation and information hiding

- Encapsulation is achieved by specifying which classes may use the members of an object. The result is that each object exposes to any class a certain interface — those members accessible to that class.
- The reason for encapsulation is to prevent clients of an interface from depending on those parts of the implementation that are likely to change in the future, thereby allowing those changes to be made more easily, that is, without changes to clients.
- It also aims at preventing unauthorized objects to change the state of an object.

Object-oriented programming paradigm

- Polymorphism

- Polymorphism is the ability of a class instance to behave as if it were an instance of another class in its inheritance tree, most often one of its ancestor classes.
  - the ability of objects belonging to **different typ**es to respond to method, field, or property calls of the **same name**, each one according to an appropriate **type-specific behavior**.
- The programmer (and the program) does not have to know the exact type of the object at compile time. The exact behavior is determined at run-time using a run-time system behavior known as **dynamic binding**. Such polymorphism allows the programmer to treat derived class members just like their parent class' members.

Object-oriented programming paradigm

- Polymorphism

- The different objects involved only need to present a **compatible interface** to the clients. That is, there must be public or internal methods, fields, events, and properties with the same name and the same parameter sets in all the superclasses, subclasses and interfaces.
- In principle, the object types may be unrelated, but since they share a common interface, they are often implemented as subclasses of the same superclass.