Object Oriented Programming

Object-oriented programming is a programming paradigm that uses abstraction (in the form of classes and objects) to create models based on the real world environment. An object-oriented application uses a collection of objects, which communicate by passing messages to request services. Objects are capable of passing messages, receiving messages, and processing data. The aim of object-oriented programming is to try to increase the flexibility and maintainability of programs. Because programs created using an OO language are modular, they can be easier to develop, and simpler to understand after development.

Object-Oriented Programming vs. Procedural Programming

Programs are made up of modules, which are parts of a program that can be coded and tested separately, and then assembled to form a complete program. In procedural languages (i.e. C) these modules are procedures, where a procedure is a sequence of statements. In C for example, procedures are a sequence of imperative statements, such as assignments, tests, loops and invocations of sub procedures. These procedures are functions, which map arguments to return statements.

The design method used in procedural programming is called Top Down Design. This is where you start with a problem (procedure) and then systematically break the problem down into sub problems (sub procedures). This is called functional decomposition, which continues until a sub problem is straightforward enough to be solved by the corresponding sub procedure. The difficulties with this type of programming, is that software maintenance can be difficult and time consuming. When changes are made to the main procedure (top), those changes can cascade to the sub procedures of main, and the sub-sub procedures and so on, where the change may impact all procedures in the pyramid.

One alternative to procedural programming is object oriented programming. Object oriented programming is meant to address the difficulties with procedural programming. In object oriented programming, the main modules in a program are classes, rather than procedures. The object-oriented approach lets you create classes and objects that model real world objects.

Classes and Objects

A ***class*** is a collection of objects that have common properties, operations and behaviours. A class is a combination of state (data) and behaviour (methods). In object-oriented languages, a class is a data type, and objects are instances of that data type. In other words, classes are prototypes from which objects are created.

For example, we may design a class Human, which is a collection of all humans in the world. Humans have state, such as height, weight, and hair color. They also have behaviour, such as walking, talking, eating. All of the state and behaviour of a human is encapsulated (contained) within the class human.

An ***object*** is an instance of a class. Objects are units of abstraction. An object can communicate with other objects using messages. An object passes a message to another object, which results in the invocation of a method. Objects then perform the actions that are required to get a response from the system.

Real world objects all share two characteristics; they all have state and behaviour. One-way to begin thinking in an object oriented way is to identify the state and behaviour of real world objects. The complexity of objects can differ, some object have more states and more complex behaviours than other object. Compare the state and behaviour of a television to the states and behaviours of a car.

Software objects are conceptually similar to real world objects. An object stores its state in fields, and it exposes its behaviours through its methods.

A fundamental principle of object oriented programming is encapsulation; the hiding of an objects internal state and requiring all interaction to be performed using the objects methods. (Think of the classes used in VB.Net)

Classes and Objects Example - class Bicycle

There are many types of bicycles, but we can say that each bicycle was built from the same blueprint (or prototype) and each bicycle contains the same components. Your bicycle (a specific bicycle object) is an instance of a class of objects known as bicycles. Describe the state and behaviour of a bicycle.

Fields and Methods

Objects in a class have a number of shared properties/features/attributes. Fields are the variables contained in a class (encapsulated) that are used to represent these properties. For example, class Student may have an integer field to represent graduation year, or a String field to represent the student's Last Name.

Before fields in an object can be assigned values, the object must first be constructed/created. In Java (as an example of an OO language) a special function called a **constructor** is used to create and initialize an instance of a class. The statement:

```
new Student();
```

creates an instance of the class Student, or in other words, a Student object. Each class has operations associated with it. Each Student object has a number of distinct behaviours, such as study, sleep, work, etcetera. The operations associated with a class, together with the attributes of a class are encapsulated within the class. These operations are called the methods of a class. Methods are functions that represent the operations associated with a particular class. Fields and methods are referred to as class **members**.

Encapsulation

Definition: the ability of an object to hide its data and methods from the rest of the world - one of the fundamental principles of OOP. Because objects encapsulate data and implementation, the user of an object can view the object as a black box that provides services. Instance variables and methods can be added, deleted, or changed, but as long as the services provided by the object remain the same, code that uses the object can continue to use it without being rewritten.

Coupling – the degree to which a module (class) depends on other classes. In a good design, you should try to minimize coupling. Classes should be self-contained units that have a low dependence on other classes, meaning understanding and using one class should not require an understanding of another. With low coupling a change in a module will not require changes in other modules.

Related to the concept of coupling is information hiding. Information hiding is the hiding of the implementation in a class or module that are most likely to change; this protects other parts of the program from change if the implementation is changed. Protecting an implementation involves providing a stable interface which other classes can use to access services provided by the class. Often encapsulation and information hiding are used interchangeably.

Example: Car and Driver

Cohesion – is the measure within a module (class) of how well the members work together to provide a specific piece of functionality. Cohesion is measured by how strongly related and focused the responsibilities of a single class are. In good design, the cohesion should be maximized. Cohesion is decreased if the methods of a class have little in common, or methods carry out many activities.

If cohesion is low, modules may be difficult to understand, maintenance is difficult, because change may affect many modules, and modules cannot be reused, because it is unlikely that another application will have a use for the random grouping of functionality.

Example: Oven that has built-in radio and alarm clock.

A well-designed system should maximize cohesion, and minimize coupling.

Inheritance

Multiple classes may share some of the same characteristics, and have things in common with one another, but also may have certain properties that make them dif ferent. Object oriented programming languages allow classes to inherit commonly used state and behaviour from other classes.

Classes in Java occur in **inheritance hierarchies**. These hierarchies consist of parent child relationships among the classes. Inheritance is used to specialize a parent class, but creating a child class (example). Inheritance also allows designers to combine features of classes in a common parent.

Example: Student

Benefits of Object Oriented Programming

1. **Modularity**: The source code for a class can be written and maintained independently of the source code for other classes. Once created, an object can be easily passed around inside the system.
2. **Information-hiding:** By interacting only with an object's methods, the details of its internal implementation remain hidden from the outside world.
3. **Code re-use:** If a class already exists, you can use objects from that class in your program. This allows programmers to implement/test/debug complex, task-specific objects, which you can then use in your own code.
4. **Easy Debugging:** If a particular object turns out to be a problem, you can simply remove it from your application and plug in a dif ferent object as its replacement. This is analogous to fixing mechanical problems in the real world. If a bolt breaks, you replace *it*, not the entire machine.

Summary of Object-Oriented Concepts

1. Everything is an object.
2. Computation is performed by objects communicating with each other, requesting that other objects perform actions. Objects communicate by sending and receiving messages. A message is a request for action, bundled with whatever arguments may be necessary to complete the tasks.
3. Each object has its own memory, which consists of other objects.
4. Every object is an instance of a class. A class simply represents a grouping of similar objects, such as Integers or lists.
5. The class is the repository for behaviour associated with an object. That is, that all objects that are instances of the same class can perform the same actions.
6. Classes are organized into a singly rooted tree structure, called the inheritance hierarchy. Memory and behaviour associated with instances of a class are automatically available to any class associated with a descendant in this tree structure.