Object-Oriented Programming – Summary of Key Terms

Definitions of some of the key concepts in Object Oriented Programming (OOP)

Examples are given in italics. Cross-references are underlined.

Term	Definition
Abstract Data Type	A user-defined data type, including both attributes (its state) and methods (its behaviour). An object oriented language will include means to define new types (see class) and create instances of those classes (see object). It will also provide a number of primitive types.
Aggregation	Objects that are made up of other objects are known as aggregations. The relationship is generally of one of two types: • Composition – the object is composed of other objects. This form of aggregation is a form of code reuse. *E.g. A Car is composed of Wheels, a Chassis and an Engine* • Collection – the object contains other objects. *E.g. a List contains several Items; A Set several Members*.
Attribute	A characteristic of an object. Collectively the attributes of an object describe its state. *E.g. a Car may have attributes of Speed, Direction, Registration Number and Driver.*
Class	The definition of objects of the same abstract data type. In Java `class` is the keyword used to define new types.
Dynamic (Late) Binding	The identification at run time of which version of a method is being called (see polymorphism). When the class of an object cannot be identified at compile time, it is impossible to use static binding to identify the correct object method, so dynamic binding must be used.
Encapsulation	The combining together of attributes (data) and methods (behaviour/processes) into a single abstract data type with a public interface and a private implementation. This allows the implementation to be altered without affecting the interface.
Inheritance	The derivation of one class from another so that the attributes and methods of one class are part of the definition of another class. The first class is often referred to the base or parent class. The child is often referred to as a derived or sub-class. Derived classes are always 'a kind of' their base classes. Derived classes generally add to the attributes and/or behaviour of the base class. Inheritance is one form of object-oriented code reuse. *E.g. Both Motorbikes and Cars are kinds of MotorVehicles and therefore share some common attributes and behaviour but may add their own that are unique to that particular type.*

Interface	The behaviour that a class exposes to the outside world; its public face. Also called its 'contract'. In Java `interface` is also a keyword similar to class. However a Java interface contains no implementation: it simply describes the behaviour expected of a particular type of object, it doesn't so how that behaviour should be implemented.
Member Variable	See attribute
Method	The implementation of some behaviour of an object.
Message	The invoking of a method of an object. In an object-oriented application objects send each other messages (i.e. execute each others methods) to achieve the desired behaviour.
Object	An instance of a class. Objects have state, identity and behaviour.
Overloading	Allowing the same method name to be used for more than one implementation. The different versions of the method vary according to their parameter lists. If this can be determined at compile time then static binding is used, otherwise dynamic binding is used to select the correct method as runtime.
Polymorphism	Generally, the ability of different classes of object to respond to the same message in different, class-specific ways. Polymorphic methods are used which have one name but different implementations for different classes. *E.g. Both the Plane and Car types might be able to respond to a turnLeft message. While the behaviour is the same, the means of achieving it are specific to each type.*
Primitive Type	The basic types which are provided with a given object-oriented programming language. *E.g. int, float, double, char, boolean*
Static(Early) Binding	The identification at compile time of which version of a polymorphic method is being called. In order to do this the compiler must identify the class of an object.