The concepts of OOP are briefly described here:

• Encapsulation is the combination of the data and actions that are related to an object. For

example, a BankAccount type might have data, such as Balance and AccountName, as well as

actions, such as Deposit and Withdraw. When encapsulating, you often want to control what

can access those actions and the data, for example, restricting how the internal state of an

object can be accessed or modified from the outside.

• Composition is about what an object is made of. For example, a Car is composed of different

parts, such as four Wheel objects, several Seat objects, and an Engine.

• Aggregation is about what can be combined with an object. For example, a Person is not part

of a Car object, but they could sit in the driver’s Seat and then become the car’s Driver—two

separate objects that are aggregated together to form a new component.

• Inheritance is about reusing code by having a subclass derive from a base or superclass. All

functionality in the base class is inherited by, and becomes available in, the derived class. For

example, the base or super Exception class has some members that have the same implemen-

tation across all exceptions, and the sub or derived SqlException class inherits those members

and has extra members that are only relevant when a SQL database exception occurs, like a

property for the database connection.

• Abstraction is about capturing the core idea of an object and ignoring the details or specifics.

C# has the abstract keyword that formalizes this concept but do not confuse the concept of

abstraction with meaning the use of the abstract keyword because it is more than that. The

concept of abstraction can also be achieved using interfaces. If a class is not explicitly abstract,

then it can be described as being concrete. Bases or superclasses are often abstract; for example,

the superclass Stream is abstract, and its subclasses, like FileStream and MemoryStream, are

concrete. Only concrete classes can be used to create objects; abstract classes can only be used

as the base for other classes because they are missing some implementation. Abstraction is

a tricky balance. If you make a class more abstract, more classes will be able to inherit from

it, but at the same time, there will be less functionality to share. A real-world example of ab-

straction is the approach car manufacturers have taken to electric vehicles (EVs). They create

a common “platform” (basically just the battery and wheels) that is an abstraction of what all

EVs need, and then add on top of that to build different vehicles like cars, trucks, vans, and so

on. The platform on its own is not a complete product, like an abstract class.

• Polymorphism is about allowing a derived class to override an inherited action to provide

custom behavior.