Object Oriented Programming

Overview of Object-Oriented Programming:

Classes - OOP makes uses of classes to represent objects these classes have attributes (known as instance variable) which store pieces of data about that object. they also have methods which perform operations on these attributes e.g. a User class may have a userName attribute and a method to update that username. These classes can be though of as instructions on how to create an object, they become an object when they are instantiated e.g. User user = new User(). A class can be thought of as the blue prints for a car and an object as the car itself.

Coupling - How much is one class dependent on another? You want to keep this to a minimum in order to increase reuseability of your code and maintainability. If you make a change in a class that many other classes are dependent on then this will cascade through to all those clases forcing you to update them all and retest them. One focus of design patterns is to provide ideas for how to reduce coupling in your code.

Interfaces - Allow you to build loosely couple software as they proveide "A contract that specifies the capabilities that a class should provide". E.g. if you open a restaurant you will be dependent on having a chef but not a particular chef - in this case the Interface idescribes the role of a chef and a class will implement these capabilites e.g John the Chef. But john could be replaced by Mary the Chef if necessary. You should attempt to "program to an interface" rather than a "concrete implementation" to achieve loose coupling.

4 Principles of OOP

Encapsulation - private access modifiers should be used for instance variables and getter and setter methods are used to access and set these variables from outside of the class. This leaves the classes instanvce variables and methods Encapsulated by the class.

Abstraction - "Reduce Complexity by hiding unncesary details" E.g.When using a Remore Control on your tv implemetation details such as how the transistors work are abstracted away / hidden from you all you need to care about is the interface i.e. what happens when you hit the volume up button. In your code this mean you should only reveal the methods that are necesary for the user to carry out the functionallity necesay all implementation details / methods should be made private. Additionally this will enable you to update the implemnation details of your functionallity without have to update classes that use that functionality reducing coupling.

Inheritance - Mechanism to reduce code duplication / increase reuse across classes by implementing common behaviours in a base / parent class and then extend the parent in children classes which inherit the common functiality and then extend it by adding additional functioality.

Polymorphism - Poly = "Many" and morphism = "Forms" . I.e an abstract class and method can defined in a parent class this method will then be implemented in children classes. You can then use this parent class as a varable in a method an depending on what child class is passed to the method the particular child method will be selected at run time allowing the one method to take many forms and redcing coupling by programming to an interface - the abstarct parent class and method

Design patterns take the above principles and provide examples of how they are best used in given situations. they have been built up over time based on developers experiences using OOP.