Strategy Pattern

When you want to define a class that will have one behavior that is similar to other behaviors in a list.
Define a family of algorithms, encapsulate each one, and make them interchangeable. The pattern lets the algorithms vary independently from clients that use it.

For Example : Sub-classes should be able to choose from several flying behaviours; Not Flying, Fly with Wings, Fly with thrusters.

Observer (Subscriber) Pattern

When you want several objects to receive an update when a certain object changes.
Loose coupling is achieved, the subject doesn't need to know anything about the observers.
The observers may receive unnecessary updates in a drawback.
The Subject maintains a list of observers which register with the subject and notifies them automatically of a state change.

Factory Pattern

When we need any of the several possible implementations of a super class at runtime.

For Example : EnemyShipFactory.makeEnemyShip(String name); // RocketShip, LaserShip

Singleton Pattern

When you want to ensure only one instance of an object in runtime.