四种常见设计模式的代码示例以及其特点概述：

1.

单例模式（Singleton）：

Python代码示例：

class Singleton:

    \_instance = None

    @staticmethod

    def getInstance():

        if Singleton.\_instance == None:

            Singleton()

        return Singleton.\_instance

    def \_\_init\_\_(self):

        if Singleton.\_instance != None:

            raise Exception("This class is a singleton!")

        else:

            Singleton.\_instance = self

特点：单例模式确保一个类仅有一个实例，并提供一个全局访问点。它通常用于需要严格控制资源访问或只能存在一个实例的场景，如数据库连接、日志记录等。

2.

观察者模式（Observer）：

Java代码示例：

public interface Observer {

    public void update();

}

public class ConcreteObserver implements Observer {

    public void update() {

        System.out.println("Observer notified.");

    }

}

public class Subject {

    private List<Observer> observers = new ArrayList<Observer>();

    public void attach(Observer observer) {

        observers.add(observer);

    }

    public void notifyAllObservers() {

        for (Observer observer : observers) {

            observer.update();

        }

    }

}

特点：观察者模式定义了对象间的一种一对多依赖关系，当一个对象（主题）的状态发生改变时，所有依赖于它的对象（观察者）都会自动得到通知并更新。它常用于实现事件驱动系统。

3.

工厂模式（Factory Method）：

C#代码示例：

public interface IAnimal {

    void Speak();

}

public class Dog : IAnimal {

    public void Speak() {

        Console.WriteLine("Dog says: Bow-Wow.");

    }

}

public class AnimalFactory {

    public IAnimal CreateAnimal() {

        return new Dog();

    }

}

特点：工厂方法模式提供一个接口用于创建对象，但允许子类决定实例化哪一个类。工厂方法使得类的实例化延迟到子类。它有利于代码解耦，提高了程序的灵活性和可扩展性。

4.

策略模式（Strategy）：

JavaScript代码示例：

class Strategy {

    doOperation(num1, num2) {

        throw new Error("This method must be overwritten!");

    }

}

class OperationAdd extends Strategy {

    doOperation(num1, num2) {

        return num1 + num2;

    }

}

class Context {

    constructor(strategy) {

        this.strategy = strategy;

    }

    executeStrategy(num1, num2) {

        return this.strategy.doOperation(num1, num2);

    }

}

特点：策略模式定义了一系列算法，并将每一个算法封装起来，使它们可以互换。策略模式让算法独立于使用它的客户端，使得客户端可以选择和更改算法，而不需要修改任何代码。