**Dependency Injection (DI)**

Dependency Injection (DI) is a design pattern that allows a class to receive its dependencies from an external source rather than creating them itself.

This promotes loose coupling, making the code more modular, testable, and maintainable.

---------手手腳腳都由external source比---------

Keyword:

design pattern

Inversion of Control (IoC)

Benefits:

Loose Coupling

Maintainability

Injection Types:

1. constructor injection

2. Setter Injection

3. Interface Injection:

Constructor Injection:

Dependencies are provided to a class via constructor.

Main Body class; (Dependent Class)

Arm and foot class; (Dependency)

Use a main class to inject arm foot into body

public class Engine {

public String start(){

return "Engine started";

}

}

public class Car {

private Engine engine;

public Car(Engine engine){

this.engine = engine;

}

public String drive(){

return engine.start();

}

}

public class Main {

public static void main(String[] args){

Engine engine = new Engine();

Car car = new Car(engine);

System.out.println(car.drive());

}

}

Setter injection

Dependencies are provided to a class through setter methods rather than through the constructor.

The benefits of setter injection:

* Late binding: Create an object, can inject the dependency later. Allow more complex initialization scenarios.

public class Cup(){

public void putCup(){

System.out.println(“Put a cup here”);

}

}

public class Desk(){

private Cup cup;

public void setCup(Cup cup){

this.cup = cup;

}

}

public class MainTest(){

Desk desk = new Desk();

Desk.setCup(new Cup());

}

Interface Injection

Holiday class is dependent on Trip interface.

You can inject Trip interface implementation into Holiday class by constructor / setter. (In this example, only use constructor)

public interface Trip() {

void plan();

void order();

void go();

}

public class TripImplement implements Trip() {

public void plan(){

println(“Will plan to go Japan”);

}

public void order(){

println(“Find hotel and plane with price go down”);

}

public void go(){

println(“Apply AL then go”);

}

}

public class Holiday() {

private final Trip trip; //init a trip interface implementation object by Trip

public Holiday (Trip t){

this.trip = t; // inject tool by constructor

}

public void tripExcute(){

trip.plan();

trip.order()

trip.go();

}

}

public static void main(String[] args) {

Trip t = new TripImplement();

Holiday h = new Holiday(t);

h.tripExcute();

}

This will print:

Will plan to go Japan

Apply AL then go

Find hotel and plane with price go down

Framework that provides DI:

Spring

Java EE

Guice: A lightweight framework provided by Google

Constructor and setter injection pros:

Late binding

The object is init, but can inject arm and legs later

loose coupling

promotes dependencies between components in a system.

Components can be developed, tested, and deployed independently, making it easier to manage large systems.

Interface injection pros:

loose coupling

Immutability: Dependencies can be declared as final,

ensuring they are set only once and cannot be changed later.