# 依赖倒置

定义：

1、上层模块不应该依赖底层模块，它们都应该依赖于抽象。  
2、抽象不应该依赖于细节，细节应该依赖于抽象。

抽象不应该依赖细节，细节应该依赖抽象。通过依赖倒置，可以减少类与类之间的耦合性，提高系统的稳定性，提高代码的可读性和可维护性，并且能够降低修改程序所造成的风险。

优势：（结合实例）

修改前

public class hym {

    public void studyJavaCourse() {

        System.out.println("hym 同学正在学习「Java」课程");

    }

    public void studyDesignPatternCourse() {

        System.out.println("hym 同学正在学习「软件工程」课程");

    }

}

public static void main(String[] args) {

    hym Hym = new hym();

    hym.studyJavaCourse();

    hym.studyDesignPatternCourse();

}

1. 有效控制影响范围

由于hym同学热爱学习，随着学习兴趣的 “暴增”，可能会继续学习 AI（人工智能）的课程。这个时候，因为「业务的扩展」，要从底层实现到高层调用依次地修改代码。

我们需要在hym类中新增 studyAICourse() 方法，也需要在高层调用中增加调用，这样一来，系统发布后，其实是非常不稳定的。显然在这个简单的例子中，我们还可以自信地认为，我们能 Hold 住这一次的修改带来的影响，因为都是新增的代码，我们回归的时候也可以很好地 cover 住，但实际的情况和实际的软件环境要复杂得多。

最理想的情况就是，我们已经编写好的代码可以 “万年不变”，这就意味着已经覆盖的单元测试可以不用修改，已经存在的行为可以保证保持不变，这就意味着「稳定」。任何代码上的修改带来的影响都是有未知风险的，不论看上去多么简单。

2、增强代码可读性和可维护性

另外一点，你有没有发现其实加上新增的 AI 课程的学习，他们三节课本质上行为都是一样的，如果我们任由这样行为近乎一样的代码在我们的类里面肆意扩展的话，很快我们的类就会变得臃肿不堪，等到我们意识到不得不重构这个类以缓解这样的情况的时候，或许成本已经变得高得可怕了。

3、降低耦合

进行交流的双方都依赖于抽象——也就是接口——来进行交换，这样一来耦合度就大为降低。

## 根据依赖倒置修改后

public interface ICourse {

    void study();

}

public class JavaCourse implements ICourse {

    public void study() {

        System.out.println("hym 同学正在学习「Java」课程");

    }

}

public class DesignPatternCourse implements ICourse {

    public void study() {

        System.out.println("hym 同学正在学习「设计模式」课程");

    }

}

public class hym {

    public void study(ICourse course) {

        course.study();

    }

}

public static void main(String[] args) {

    hym Hym = new hym();

    hym.study(new JavaCourse());

    hym.study(new DesignPatternCourse());

}