1. SOLID feature in java

* [**The Single Responsibility Principle**](https://docs.google.com/open?id=0ByOwmqah_nuGNHEtcU5OekdDMkk) : A class should have one, and only one, reason to change.

e.g

public interface Modem

{

public void Dial(string pno);

public void Hangup();

public void Send(char c);

public char Recv();

}

However, there are two responsibilities being shown here. The first responsibility is

connection management. The second is data communication. The dial and hangup func-

tions manage the connection of the modem, while the send and recv functions communi-

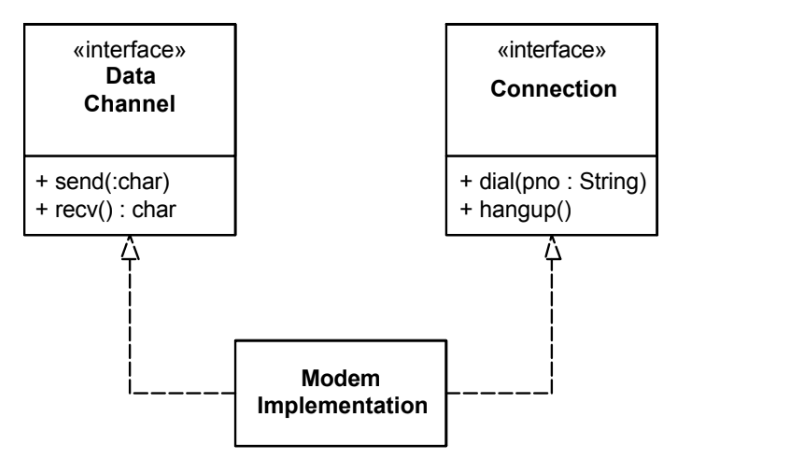
cate data.

Should these two responsibilities be separated? That depends upon how the application is changing. If the application changes in ways that affect the signature of the connection functions, then the design will smell of Rigidity because the classes that call send

and read will have to be recompiled and redeployed more often than we like. In that case

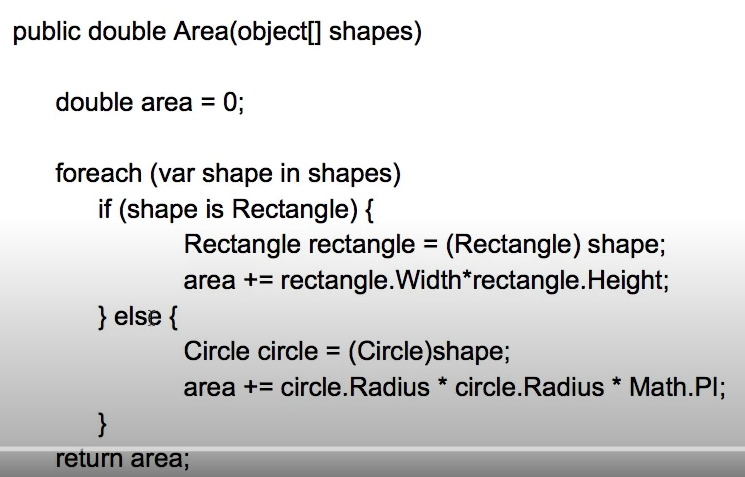
the two responsibilities should be separated as shown in Figure 8-3. This keeps the client

applications from coupling the two responsibilities.



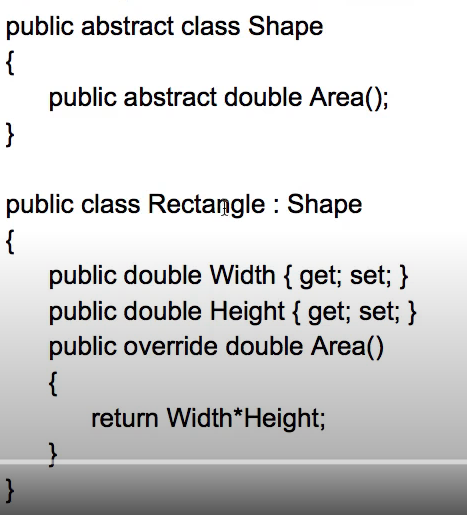
* [**The Open Closed Principle**](http://docs.google.com/a/cleancoder.com/viewer?a=v&pid=explorer&chrome=true&srcid=0BwhCYaYDn8EgN2M5MTkwM2EtNWFkZC00ZTI3LWFjZTUtNTFhZGZiYmUzODc1&hl=en): You should be able to extend a classes behavior, without modifying it.

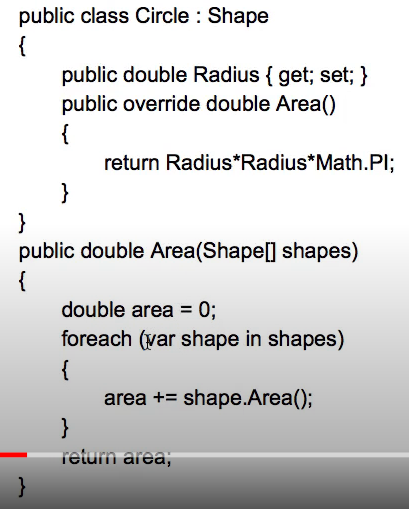
e.g



In this example what if I want to add new shape type like triangle while it work? No.

So to avoid this issue we can use below solution.





* [**The Liskov Substitution Principle**](http://docs.google.com/a/cleancoder.com/viewer?a=v&pid=explorer&chrome=true&srcid=0BwhCYaYDn8EgNzAzZjA5ZmItNjU3NS00MzQ5LTkwYjMtMDJhNDU5ZTM0MTlh&hl=en) : Derived classes must be substitutable for their base classes.

e.g

class rectangle {

void setWidth()

void setheight()

double getWidth()

double getheight()

}

Class square extends rectangle {

void setWidth()

void setheight()

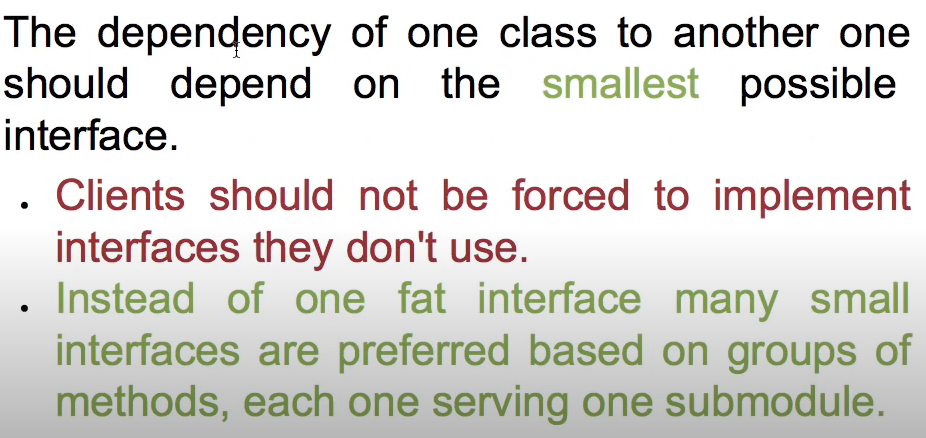
double getWidth()

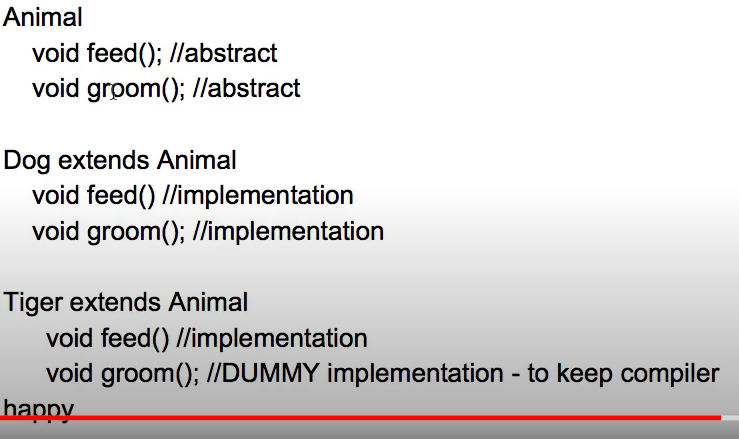
double getheight()

}

In above example class square is reusing the property of class rectangle which is not recommended because just to save few lines of code .

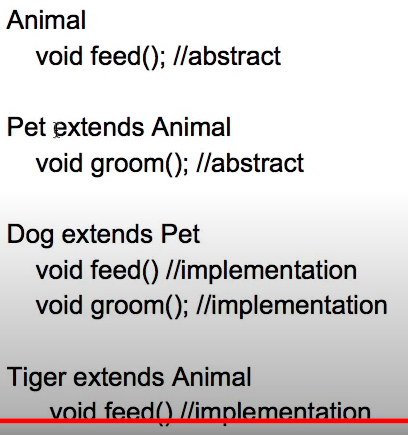
* [**The Interface Segregation Principle**](http://docs.google.com/a/cleancoder.com/viewer?a=v&pid=explorer&chrome=true&srcid=0BwhCYaYDn8EgOTViYjJhYzMtMzYxMC00MzFjLWJjMzYtOGJiMDc5N2JkYmJi&hl=en): Make fine grained interfaces that are client specific





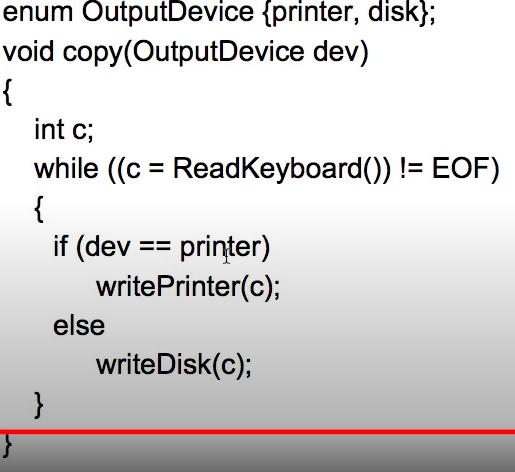
Above example is not recommended because we are implementing groom() unnecessarily

The ideal way to tackle this issue is to have two different interfaces.



* [**The Dependency Inversion Principle**](http://docs.google.com/a/cleancoder.com/viewer?a=v&pid=explorer&chrome=true&srcid=0BwhCYaYDn8EgMjdlMWIzNGUtZTQ0NC00ZjQ5LTkwYzQtZjRhMDRlNTQ3ZGMz&hl=en) : Depend on abstract or interfaces , not on concretions class.

e.g



Issue with above code is you can only write the input which comes from printer or disk but what if we have input from other device apart from printer and disk in that case it will fail so to tackle this issue we should use interface as below

