**Adapt Interface**

**Motivation**

Classes can implement an interface, however it may only provide code for some of the interface’s methods. This can lead to empty methods that need to be added in order for the code to compile, but don’t actually do anything on their own. This can cause less efficient communication as the empty classes are unnecessary from a reading standpoint, and can complicate the code by adding methods that don’t really do anything. The Adapter pattern can be used to simplify this code by adding a subclass that only supplies the required code.

**Methods**

1. Create a class that implements the interface and provides do-nothing behaviour. Write a creation method returns a reference to an instance of the Adapter.
2. Delete the empty methods that are there to just implement the interface.
3. Move the methods with the functional code from the interface to the Adapter
4. Remove code declaring that the class implements the interface.
5. Supply instances of the Adapter to those that need it.

**Sample code to refactor**

A website expands a card when the user drags their mouse on it to get more information. This functionality is implemented below. In this case, the mouseMoved() method is empty.

public class CardComponent extends Container implements MouseMotionListener ...

public CardComponent(Card card,Explanations explanations) {

...

addMouseMotionListener(this);

}

public void mouseDragged(MouseEvent e) {

e.consume();

dragPos.x = e.getX();

dragPos.y = e.getY();

setLocation(getLocation().x+e.getX()-currPos.x,

getLocation().y+e.getY()-currPos.y);

repaint();

}

public void mouseMoved(MouseEvent e) {

}

**Risks**

* May complicate code due to numerous nested methods
* Can be hard to implement if the team is not familiar with the Adapter pattern