Object Adapter

Linda Marshall and Vreda Pieterse

Department of Computer Science
University of Pretoria

23 September 2014

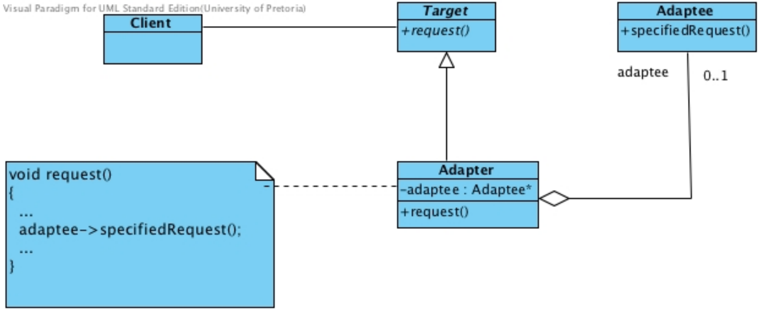
Overview

1. Identification
2. Structure
3. Discussion
4. Participants
5. Related Patterns
6. Examples
    - Billboard
    - Maths

Name and Classification: Adapter (Object and Class Structural)

Intent: “Convert an interface of a class into another interface clients expect. Adapter lets classes work together that couldn’t otherwise because of incompatible interfaces. ” GoF(139)

Object Adapter



```
void request()
{
  ...
  adaptee->specifiedRequest();
  ...
}
```

- Used to modify exsiting interfaces make it work after it has been designed
- Object Adapter makes use of object composition to delegate to Adaptee.

Target

- Domain specific interface used by the client

Adapter

- Adapts the interface of Adaptee to the Target interface

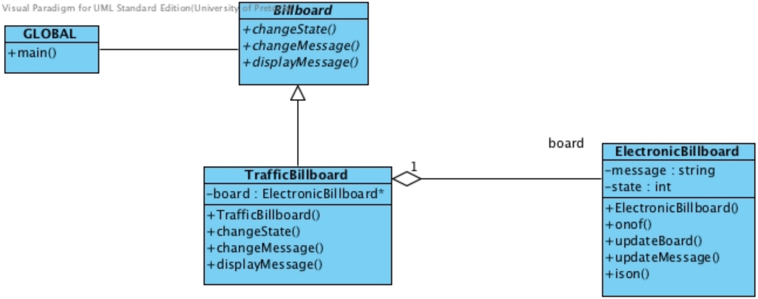
Adaptee

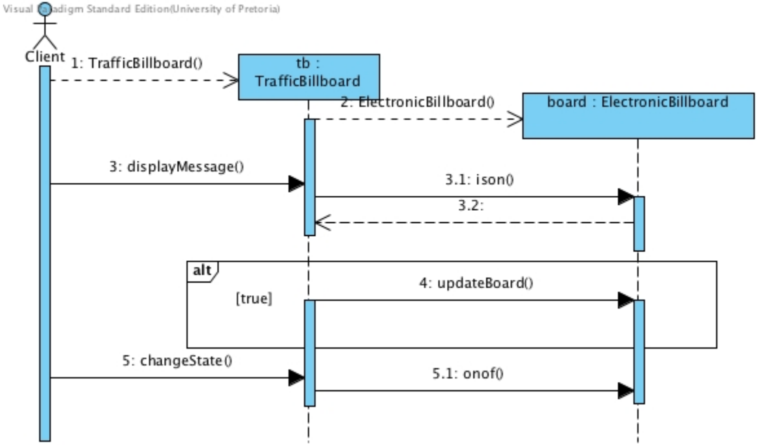
- The existing interface that needs to be adapted

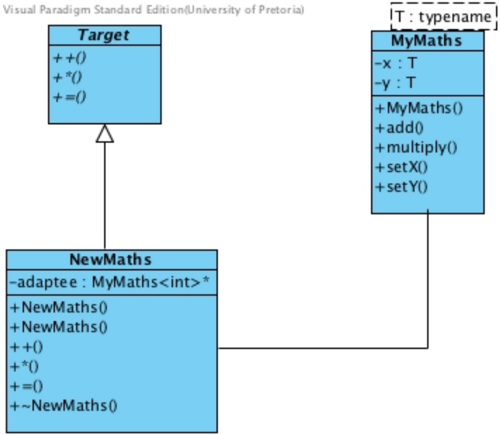
Client

- Manipulates objects conforming to the interface specified by the abstract class Target

- **Bridge** () : Structurally they are similar. However their intent is different, the Adapter changes the interface while the Bridge separates the implementation from the interface.
- **Decorator** () : Enhances an object without changing the interface.
- **Proxy** () : Defines a surrogate of to an object without changing its interface.


Visual Paradigm for UML Standard Edition(University of Preto
GLOBAL
+main()
Billboard
+changeState()
+changeMessage()
+displayMessage()
TrafficBillboard
-board : ElectronicBillboard*
+TrafficBillboard()
+changeState()
+changeMessage()
+displayMessage()
1
board
ElectronicBillboard
-message : string
-state : int
+ElectronicBillboard()
+onof()
+updateBoard()
+updateMessage()
+ison()


Visual Paradigm Standard Edition(University of Pretoria)
Client
1: TrafficBillboard()
tb : TrafficBillboard
2: ElectronicBillboard()
board : ElectronicBillboard
3: displayMessage()
3.1: ison()
3.2:
alt
[true]
4: updateBoard()
5: changeState()
5.1: onof()



Target

++()
+()*
+=()

T : typename

MyMaths

-x : T
-y : T

+MyMaths()
+add()
+multiply()
+setX()
+setY()

NewMaths

-adaptee : MyMaths<int>*

+NewMaths()
+NewMaths()
++()
+*()
+=()
+~NewMaths()

```
#ifndef MYMATHS_H
#define MYMATHS_H

template <typename T>
class MyMaths {
public:
    MyMaths(T, T);
    T add ();
    T multiply();
 //protected:
    void setX(T);
    void setY(T);
  private:
    T x;
    T y;
};

#include "MyMaths.cpp"

#endif
```

```
template <typename T>
MyMaths<T>::MyMaths(T v1, T v2)
{
    x = v1;
    y = v2;
}

template <typename T>
T MyMaths<T>::add()
{
    return x + y;
}


template <typename T>
T MyMaths<T>::multiply()
{
    return x * y;
}
```

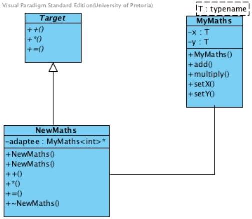
```
template <typename T>
void MyMaths<T>::setX(T object)
{
    x = object;
}

template <typename T>
void MyMaths<T>::setY(T object)
{
    y = object;
}
```

T must be:

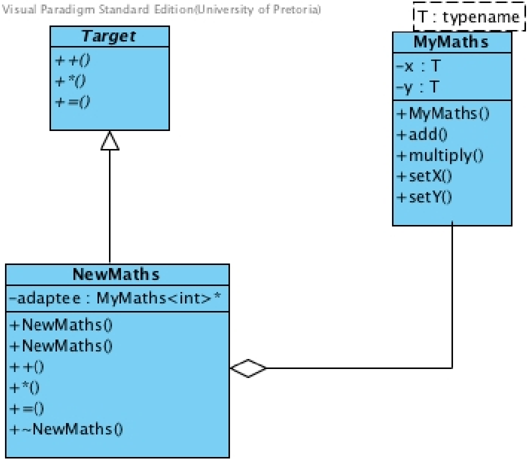
- assignable
- copy constructible; and
- operators + and $*$ must be defined

```
#ifndef TARGET_H
#define TARGET_H

class Target {
public:
    virtual int operator+(int) = 0;
    virtual int operator*(int) = 0;
    virtual int operator=(int) = 0;
};

#endif
```

```
#ifndef NEWMATHS_H
#define NEWMATHS_H

#include "Target.h"
#include "MyMaths.h"

class NewMaths : public Target
{
public:
    NewMaths();
    NewMaths(int);
    virtual int operator+(int);
    virtual int operator*(int);
    virtual int operator=(int);
    ~NewMaths();
private:
    MyMaths<int>* adaptee;

};

#endif
```



Visual paradigm draws the association between `NewMaths` and `MyMaths` without a specific type.



Target

++()
+()*
+=()

T : typename

MyMaths

-x : T
-y : T

+MyMaths()
+add()
+multiply()
+setX()
+setY()

NewMaths

-adaptee : MyMaths<int>*

+NewMaths()
+NewMaths()
++()
+*()
+=()
+~NewMaths()

```
#include <iostream>
#include "Target.h"
#include "NewMaths.h"

using namespace std;

int main()
{
    Target* obj = new NewMaths(4);

    int temp;
    temp  = (*obj +3);
    cout << temp << endl;

    *obj = 10;
    temp  = (*obj + 3);
    cout << temp << endl;

    return 0;
}
```