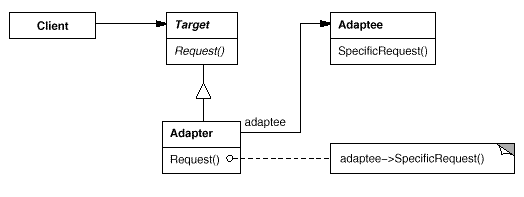
Adapter Pattern

**Intent**  
Convert the interface of a class into another interface clients expect. Adapter (or Wrapper) lets classes work together that couldn't otherwise because of incompatible interfaces. Adapter pattern's motivation is that we can reuse existing software if we can modify the interface.

A class adapter uses multiple inheritance to adapt one interface to another.



1. Adapter pattern relies on object composition.
2. Client calls operation on Adapter object.
3. Adapter calls Adaptee to carry out the operation.
4. In STL, stack adapted from vector:  
   When stack executes push(), underlying vector does vector::push\_back().



Here is an example code:

#include <iostream>

// Desired interface (Target)

class Rectangle

{

public:

virtual void draw() = 0;

};

// Legacy component (Adaptee)

class LegacyRectangle

{

public:

LegacyRectangle(int x1, int y1, int x2, int y2) {

x1\_ = x1;

y1\_ = y1;

x2\_ = x2;

y2\_ = y2;

std::cout << "LegacyRectangle(x1,y1,x2,y2)\n";

}

void oldDraw() {

std::cout << "LegacyRectangle: oldDraw(). \n";

}

private:

int x1\_;

int y1\_;

int x2\_;

int y2\_;

};

// Adapter wrapper

class RectangleAdapter: public Rectangle, private LegacyRectangle

{

public:

RectangleAdapter(int x, int y, int w, int h):

LegacyRectangle(x, y, x + w, y + h) {

std::cout << "RectangleAdapter(x,y,x+w,x+h)\n";

}

void draw() {

std::cout << "RectangleAdapter: draw().\n";

oldDraw();

}

};

int main()

{

int x = 20, y = 50, w = 300, h = 200;

Rectangle \*r = new RectangleAdapter(x,y,w,h);

r->draw();

}

Output from the run:

LegacyRectangle(x1,y1,x2,y2)

RectangleAdapter(x,y,x+w,x+h)

Summary of the code:

1. The client thinks he is talking to a **Rectangle**
2. The target is the **Rectangle** class. This is what the client invokes method on.
3. Rectangle \*r = new RectangleAdapter(x,y,w,h);
4. r->draw();
5. Note that the adapter class uses multiple inheritance.
6. class RectangleAdapter: public Rectangle, private LegacyRectangle {
7. ...
8. }
9. The Adapter **RectangleAdapter** lets the **LegacyRectangle** responds to request (**draw()** on a **Rectangle**) by inheriting BOTH classes.
10. The **LegacyRectangle** class does not have the same methods (**draw()**) as **Rectangle**,   
    but the Adapter(**RectangleAdapter**) can take the **Rectangle** method calls and turn around and invoke method on the **LegacyRectangle**, **oldDraw()**
11. class RectangleAdapter: public Rectangle, private LegacyRectangle {
12. public:
13. RectangleAdapter(int x, int y, int w, int h):
14. LegacyRectangle(x, y, x + w, y + h) {
15. std::cout << "RectangleAdapter(x,y,x+w,x+h)\n";
16. }
18. void draw() {
19. std::cout << "RectangleAdapter: draw().\n";
20. oldDraw();
21. }
22. };

**Adapter** design pattern translates the interface for one class into a compatible but different interface. So, this is similar to the **proxy** pattern in that it's a single-component wrapper. But the interface for the adapter class and the original class may be different.

As we've seen in the example above, this **adapter** pattern is useful to expose a different interface for an existing API to allow it to work with other code. Also, by using adapter pattern, we can take heterogeneous interfaces, and transform them to provide consistent API.

|  |
| --- |
| /\* |
|  |

|  |
| --- |
| \* Example of `adapter' design pattern |
|  |

|  |
| --- |
| \* Copyright (C) 2011 Radek Pazdera |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| \* This program is free software: you can redistribute it and/or modify |
|  |

|  |
| --- |
| \* it under the terms of the GNU General Public License as published by |
|  |

|  |
| --- |
| \* the Free Software Foundation, either version 3 of the License, or |
|  |

|  |
| --- |
| \* (at your option) any later version. |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| \* This program is distributed in the hope that it will be useful, |
|  |

|  |
| --- |
| \* but WITHOUT ANY WARRANTY; without even the implied warranty of |
|  |

|  |
| --- |
| \* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the |
|  |

|  |
| --- |
| \* GNU General Public License for more details. |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| \* You should have received a copy of the GNU General Public License |
|  |

|  |
| --- |
| \* along with this program. If not, see <http://www.gnu.org/licenses/>. |
|  |

|  |
| --- |
| \*/ |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| #include <iostream> |
|  |

|  |
| --- |
| #include <string> |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| typedef int Cable; // wire with electrons :-) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\* Adaptee (source) interface \*/ |
|  |

|  |
| --- |
| class EuropeanSocketInterface |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| public: |
|  |

|  |
| --- |
| virtual int voltage() = 0; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| virtual Cable live() = 0; |
|  |

|  |
| --- |
| virtual Cable neutral() = 0; |
|  |

|  |
| --- |
| virtual Cable earth() = 0; |
|  |

|  |
| --- |
| }; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\* Adaptee \*/ |
|  |

|  |
| --- |
| class Socket : public EuropeanSocketInterface |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| public: |
|  |

|  |
| --- |
| int voltage() { return 230; } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| Cable live() { return 1; } |
|  |

|  |
| --- |
| Cable neutral() { return -1; } |
|  |

|  |
| --- |
| Cable earth() { return 0; } |
|  |

|  |
| --- |
| }; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\* Target interface \*/ |
|  |

|  |
| --- |
| class USASocketInterface |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| public: |
|  |

|  |
| --- |
| virtual int voltage() = 0; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| virtual Cable live() = 0; |
|  |

|  |
| --- |
| virtual Cable neutral() = 0; |
|  |

|  |
| --- |
| }; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\* The Adapter \*/ |
|  |

|  |
| --- |
| class Adapter : public USASocketInterface |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| EuropeanSocketInterface\* socket; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| public: |
|  |

|  |
| --- |
| void plugIn(EuropeanSocketInterface\* outlet) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| socket = outlet; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| int voltage() { return 110; } |
|  |

|  |
| --- |
| Cable live() { return socket->live(); } |
|  |

|  |
| --- |
| Cable neutral() { return socket->neutral(); } |
|  |

|  |
| --- |
| }; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\* Client \*/ |
|  |

|  |
| --- |
| class ElectricKettle |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| USASocketInterface\* power; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| public: |
|  |

|  |
| --- |
| void plugIn(USASocketInterface\* supply) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| power = supply; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| void boil() |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| if (power->voltage() > 110) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| std::cout << "Kettle is on fire!" << std::endl; |
|  |

|  |
| --- |
| return; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| if (power->live() == 1 && power->neutral() == -1) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| std::cout << "Coffee time!" << std::endl; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| }; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| int main() |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| Socket\* socket = new Socket; |
|  |

|  |
| --- |
| Adapter\* adapter = new Adapter; |
|  |

|  |
| --- |
| ElectricKettle\* kettle = new ElectricKettle; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\* Pluging in. \*/ |
|  |

|  |
| --- |
| adapter->plugIn(socket); |
|  |

|  |
| --- |
| kettle->plugIn(adapter); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\* Having coffee \*/ |
|  |

|  |
| --- |
| kettle->boil(); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| return 0; |
|  |

}

<https://gist.github.com/pazdera/1145857>