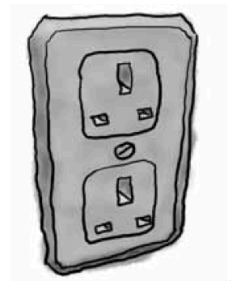
Adapter

Baris Aktemur
CS 534| Ozyegin University

Contents are from “Design Patterns” by Gamma, Helm, Johnson, Vlissides

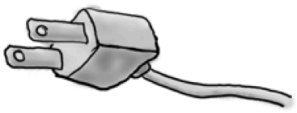
Adapter

- Intent
  - Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.



AC Power Adapter

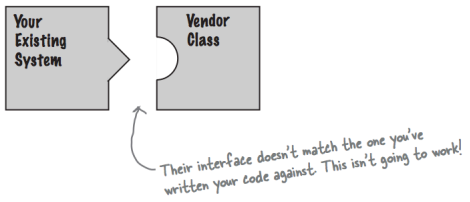
Standard AC Plug

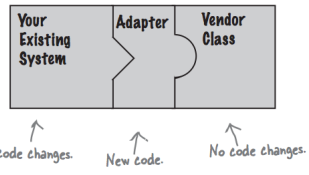


The US laptop expects another interface.

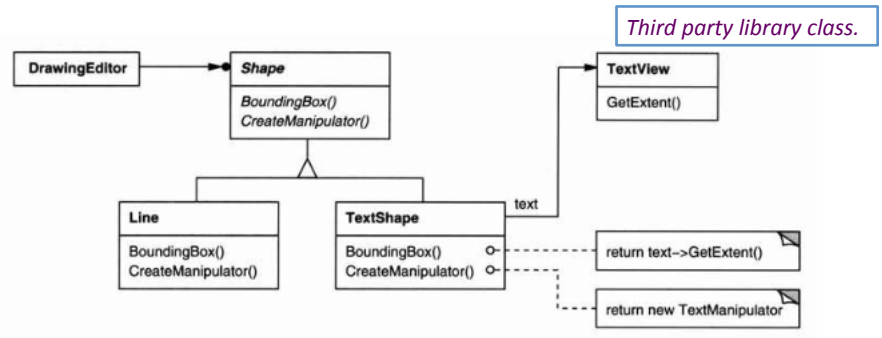
The European wall outlet exposes one interface for getting power.

The adapter converts one interface into another.





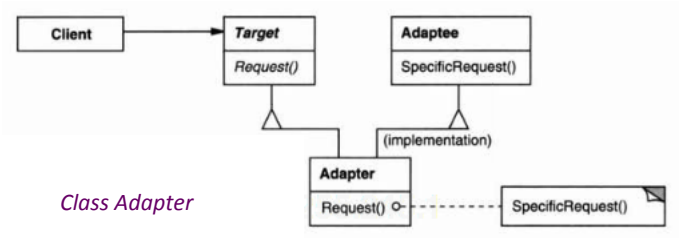
Adapter



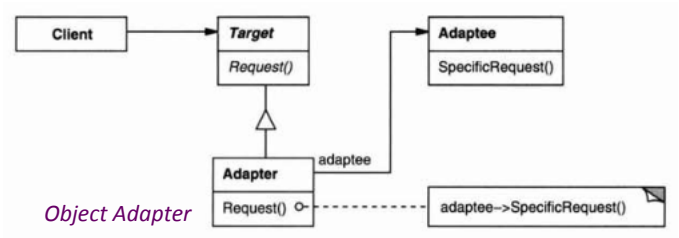
Applicability

- Use the Adapter pattern when
  - you want to use an existing class, and its interface does not match the one you need.
  - you want to create a reusable class that cooperates with unrelated or unforeseen classes, that is, classes that don't necessarily have compatible interfaces.
  - (object adapter only) you need to use several existing subclasses, but it's impractical to adapt their interface by subclassing every one. An object adapter can adapt the interface of its parent class.

Structure



Class Adapter



Object Adapter


```
class Shape {
public:
  Shape();
  virtual void BoundingBox(Point& bottomLeft, Point& topRight)
   const;
  virtual Manipulator* CreateManipulator() const;
};

class TextView {
public:
  TextView();
  void GetOrigin(Coord& x, Coord& y) const;
  void GetExtent(Coord& width, Coord& height) const;
  virtual bool IsEmpty() const;
};
```

Class adapter

```
class TextShape: public Shape, private TextView {
public:
  TextShape();
  virtual void BoundingBox(Point& bottomLeft, Point& topRight) const;
  virtual bool IsEmpty() const;
  virtual Manipulator* CreateManipulator() const;
};

void TextShape::BoundingBox(Point& bottomLeft, Point& topRight) const{
  Coord bottom, left, width, height;
  GetOrigin(bottom, left);
  GetExtent(width, height);
  bottomLeft = Point(bottom, left);
  topRight = Point(bottom + height, left + width);
}

bool TextShape::IsEmpty() const {
  return TextView::IsEmpty();
}

Manipulator* TextShape::CreateManipulator() const {
  return new TextManipulator(this);
}
```


Object adapter

```
class TextShape: public Shape {
public:
  TextShape(TextView*);
  virtual void BoundingBox(Point& bottomLeft, Point& topRight)
   const;
  virtual bool IsEmpty() const;
  virtual Manipulator* CreateManipulator() const;
private:
  TextView* _text;
};

TextShape::TextShape(TextView* t) {
  _text = t;
}
```

can pass an instance of a subclass.

```
void TextShape::BoundingBox(Point& bottomLeft, Point& topRight)
   const {
  Coord bottom, left, width, height;
  _text->GetOrigin(bottom, left);
  _text->GetExtent(width, height);
  bottomLeft = Point(bottom, left);
  topRight = Point(bottom + height, left + width);
}
bool TextShape::IsEmpty() const {
  return _text->IsEmpty();
}

Manipulator* TextShape::CreateManipulator() const {
  return new TextManipulator(this);
}
```