Design Patterns

MSc in Communications Software

Produced by

Eamonn de Leastar (edeleastar@wit.ie)

Department of Computing, Maths & Physics
Waterford Institute of Technology

http://www.wit.ie

http://elearning.wit.ie

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

eLearning support unit

Adapter

Design Pattern

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.
- Also known as Wrapper

Motivation (1)

- An Algorithm is designed to search an in-memory data structure via an Iterator object.

- This can search any collection that exposes an iterator interface:

```
static boolean linearSearch (Iterator it, Object searchItem)
{
  while (it.hasNext())
  {
    if (it.next().equals(searchItem))
    {
      return true;
    }
  }
  return false;
}
```

```
ArrayList arrayList = new ArrayList();

Object searchObject = // Some object to search for
boolean found = linearSearch(arrayList.iterator(), searchObject);
```

Motivation (2)

```
ObjectInputStream objectStream = new ObjectInputStream (new FileInputStream("t.ser"));
```

- We would like to use the same linear search method to search a stream of objects stored in a file on disk.

```
found = linearSearch (objectStream, searchObject);
```

- objectStream is not an Iterator, nor does it support Iterators…
- … so we introduce a new class – ObjectIterator – which “adapts” and ObjectStream to support Iterator interface:

```
ObjectIterator objIterator = new ObjectIterator (new FileInputStream("t.ser"));
```

- The original linearSearch() can now directly access the object stream on disk – via the adapter.

```
found = linearSearch (objIterator, searchObject);
```

```
class ObjectIterator extends ObjectInputStream implements Iterator
{
  private boolean at_end_of_file = false;

  public ObjectIterator(InputStream src) throws IOException
  {
    super(src);
  }

  public boolean hasNext()
  {
    return at_end_of_file == false;
  }

  public Object next()
  {
    try
    {
      return readObject();
    }
    catch (Exception e)
    {
      at_end_of_file = true;
      return null;
    }
  }

  public void remove()
  {
    throw new UnsupportedOperationException();
  }
}
```

ObjectIterator Adapter Class

Motivation (3)

- ObjectIterator is a “Class” Adapter that adapts an ObjectInputStream to implement the Iterator interface.

- Existing methods that examine a set of objects by using an Iterator can now examine objects directly from a file.
  - i.e. the method doesn’t know or care whether it’s reading from a file or traversing a Collection of some sort,

- Useful when implementing an Object cache that can overflow to disk, for example.
  - i.e. don’t need to write two versions of the object-reader method, one for files and one for collections.

```
class WrappedObjectIterator implements Iterator
{
  private boolean at_end_of_file = false;
  private final ObjectInputStream in;

  public WrappedObjectIterator(ObjectInputStream in)
  {
    this.in = in;
  }

  public boolean hasNext()
  {
    return at_end_of_file == false;
  }

  public Object next()
  {
    try
    {
      return in.readObject();
    }
    catch (Exception e)
    {
      at_end_of_file = true;
      return null;
    }
  }

  public void remove()
  {
    throw new UnsupportedOperationException();
  }
}
```

Wrapped adapter class

Motivation (4)

- WrappedObjectIterator is an “Object” Adapter version of ObjectIterator that uses containment rather than inheritance.

- The “Class” Adapter, since it is an ObjectInputStream that implements Iterator, can be used by any method that knows how to an use either ObjectInputStream or Iterator.

- The “Object” Adapter, since it encapsulates the input stream, cannot be used as an ObjectInputStream, but you can use the input stream for a while, temporarily wrap it in a WrappedObjectIterator to extract a few objects, then pull the input stream out again.

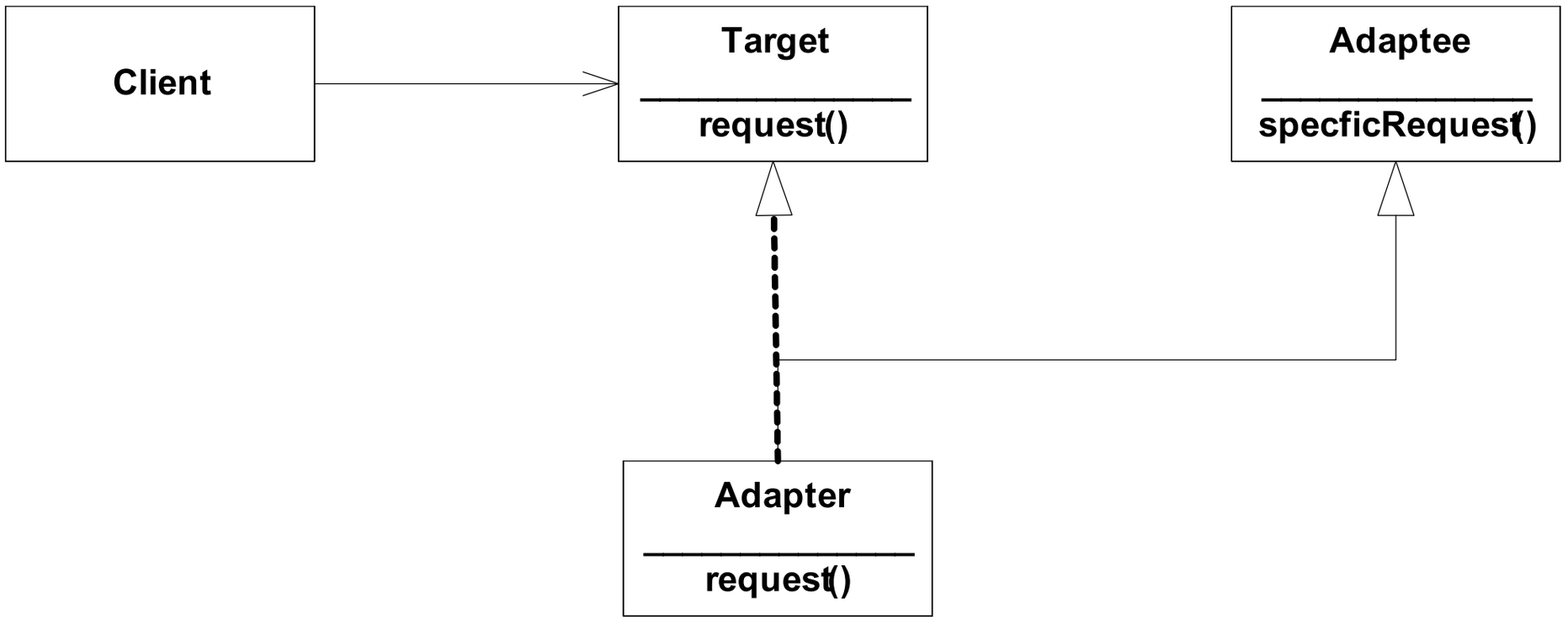
Applicability

- When you want to use an existing class, and its interface does not match the one you need.

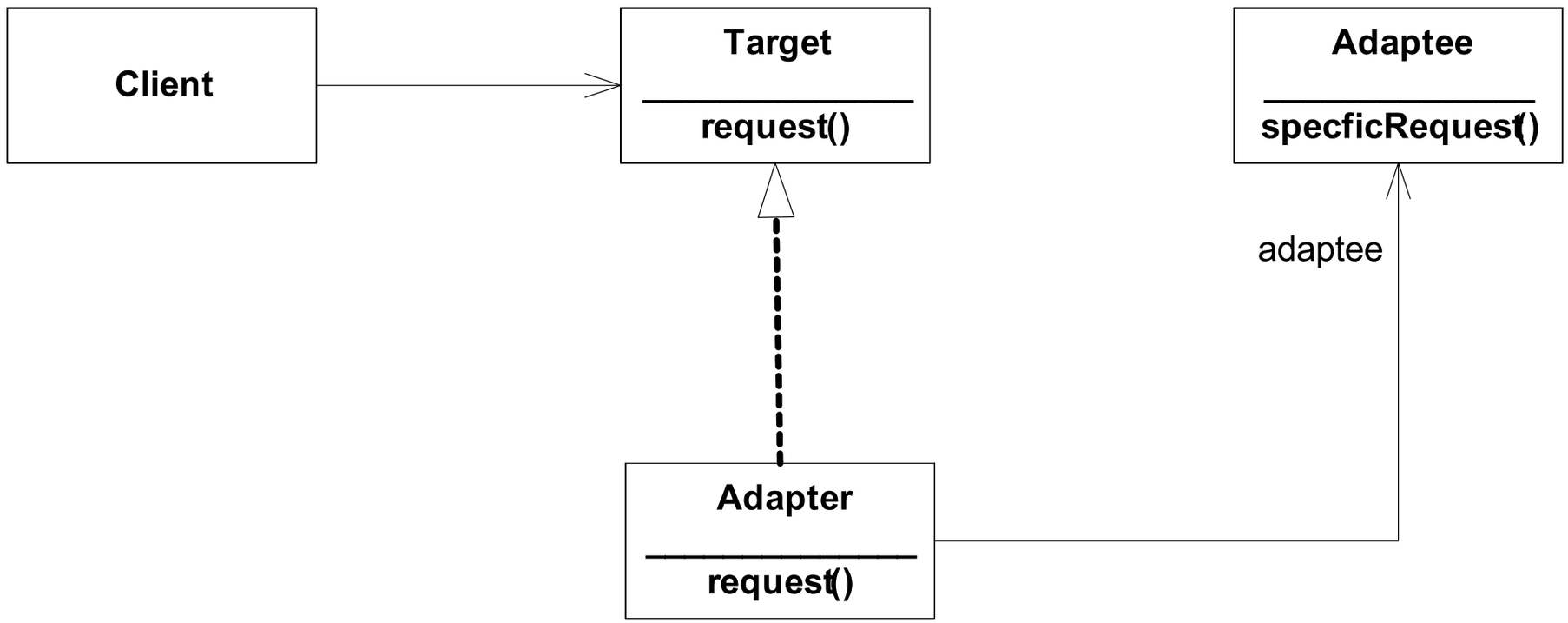
- When you want to create a reusable class that cooperates with unrelated or unforeseen classes, that is, classes that don't necessarily have compatible interfaces.

- (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



Structure: Object Adapter



Participants

- Target (Iterator)
  - defines the domain-specific interface that Client uses.
- Client (linearSearch)
  - collaborates with objects conforming to the Target interface.
- Adaptee (ObjectInputStream)
  - defines an existing interface that needs adapting.
- Adapter (ObjectIterator)
  - adapts the interface of Adaptee to the Target interface.

Collaborations

- Clients call operations on an Adapter instance.
- In turn, the adapter calls Adaptee operations that carry out the request.

Consequences: Class Adapter

- Adapts Adaptee to Target by committing to a concrete Adaptee class.
- As a consequence, a class adapter won't work when we want to adapt a class and all its subclasses.
- Lets Adapter override some of Adaptee's behavior, since Adapter is a subclass of Adaptee.
- Introduces only one object, and no additional pointer indirection is needed to get to the adaptee

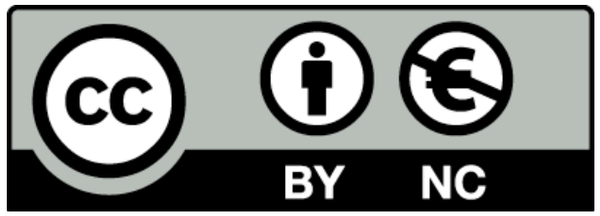
Consequences: Object Adapter

- Lets a single Adapter work with many Adaptees—that is, the Adaptee itself and all of its subclasses (if any).

- The Adapter can also add functionality to all Adaptees at once.

- Makes it harder to override Adaptee behavior.

- It will require subclassing Adaptee and making Adapter refer to the subclass rather than the Adaptee itself.



Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

eLearning support unit