CMPT 225: Data Structures & Programming
– Unit 10 –
Design Patterns & Adapters

Dr. Jack Thomas

Simon Fraser University

Spring 2021

Today's Topics

- (Re)visiting Design Patterns
- The Adapter
- Implementing an Example
- Adapting Through Inheritance

Design Patterns

- A set of **best-practices** for **solving a common design problem**.
- Not official, more like the consensus of the programming community from experience. As such, there is no one format or authority.
- The book *Design Patterns: Elements of Reusable Object-Oriented Software* (Gamma et al., 1994) was a milestone in computer science and the 23 patterns it describes are still in use today.

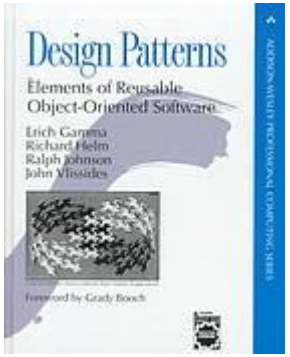

Image credit: https://en.wikipedia.org/wiki/Design_Patterns

The Adapter

- Also called the **Wrapper**, the **Adapter** pattern is used when we want to **alter the interactions of one class to fit with another**.

- The concept is similar to a USB adapter allowing you to charge your phone with your laptop.

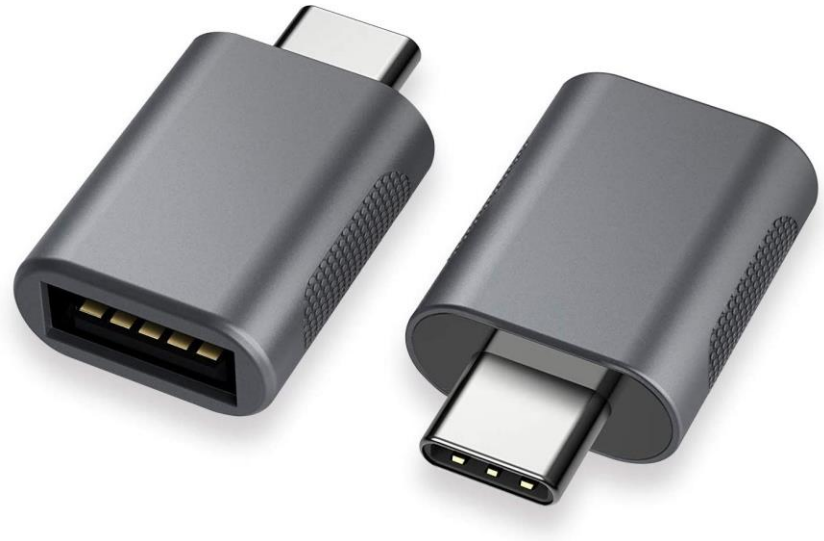

Image credit: https://www.amazon.ca/nonda-Adapter-Thunderbolt-MacBook-Surface/dp/B07XYTHCXV

How an Adapter Works

- A wrapper class is the **interface layer between two classes**, providing functions that convert data from one into the functions used by the other.
- Generally, it will **include the original class** being adapted as a hidden variable, while **presenting the functions of the new class**.
- The work is filling those functions with code that **makes use of the original class's functions**.

Example: From Deque to Stack

- Say we **have** a **Deque**, but what we **need** is a **Stack**.
- “Isn’t a Deque more general than a Stack anyway?” Yes, but **code doesn’t work like that**.
- If a system is expecting to receive a Stack object, it needs to get a Stack object. Remember ADTs and our object-oriented principles, it’s the **expected interface methods that matter**.
- So what we’ll do is make an Adapter class that **implements a Stack’s interface**, so it’ll look like a Stack from the outside, but **contains a Deque** within itself.

Let's Implement That Adapter

```
class DequeStack extends Stack {
    StringDeque secretDeque;

    public String top()
    {
        return secretDeque.getFirst();
    }
    public String pop(){
        String result = secretDeque.getFirst();
        secretDeque.removeFirst();
        return result;
    }
    public void push(int input)
    {
        secretDeque.addFirst(input);
    }
    /*plus size(), isEmpty(), you get the idea*/
}
```

- We're using simplified versions of Stack and Deque in this example that only handle Strings, not any general object.
- Some functions may not need much changed at all, apart from using the right function names, arguments, or return types.

Adaptation Through Inheritance

- One advantage of object-oriented design is that, through inheritance, you can **make your adapted class a recognized subclass** of whatever you're adapting it into.
- Consider our previous example – by formally **extending Stack and overriding its functions**, we can treat our DequeStack as if it were any other Stack.
- The advantage is that **other functions or variables who are expecting to receive a Stack could also receive a DequeStack** and be confident that it will work as intended, without ever know it was originally a Deque.
- IntelliJ/Java tip: **use the @Override tag** to see the functions available from your superclass(es) to be overridden this way.

Recap – Following the Summary Pattern

- **Design patterns** are collections of **best practices** for solving reoccurring design problems.
- The **Adapter** is one such pattern for converting structures of one type to another via translating their interface methods.
- We implemented one example of this **by turning a Deque into a Stack**.
- **Inheritance** in Java lets us make adapters that are members of one class but contain another class.