Elementary Design Patterns

Dr. Isaac Griffith Idaho State University

Outcomes

Idaho State University | Computer Science

After today's lecture you will be able to:

- To describe the structure, implement, and know when to use the following design patterns:
  - Iterator
  - Singleton
  - Adapter

Elementary Design Patterns

CS 2263

Reusing Experience

- As our experience grows it is logical to want to reuse that experience in future projects
- Unfortunately most applications tend to exhibit relatively little similarity
  - But, upon deeper inspection, we can see a number of similar issues at the design level
- As we gain exposure to problems common to multiple scenarios
  - Our ability to identify these problems increases
  - Our ability to provide solutions to these problems quickly increases
  - We begin to recognize commonalities in these solutions
    - Sets of classes with similar functionalities and relationships

Design Patterns?

Idaho State University | Computer Science

- In O-O we divide a system into objects and the classes that create them
- Thus, the task in designing an OO System is to recognize the
  - classes
  - interfaces
  - and relationships
- Necessary to solve a specific design problem
- The application of this approach has lead to the identification of **design patterns**
- **Design Pattern:** An O-O solution (set of classes, interfaces, and their relationships) to a commonly occurring design problem
  - It can also be thought of as an encapsulation of design experience or knowledge
  - It also defines a lexicon of OO design concepts

Elementary Design Patterns

Idaho State University | Computer Science

In this lecture we will study the following three design patterns:

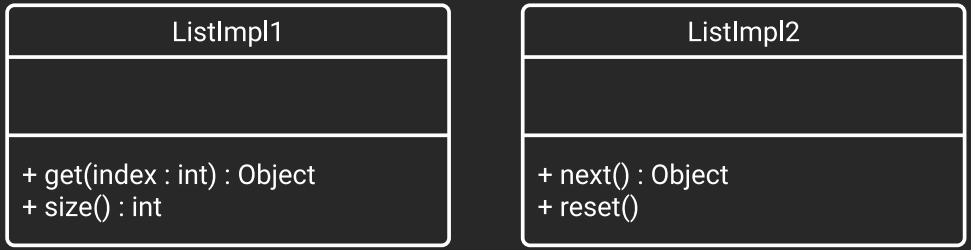
- **Iterator Pattern:** Used to traverse a collection regardless of the means by which the collection is implemented.
- **Singleton Pattern:** Used when it is known that we need one and only one instance of a class.
- **Adapter Pattern:** Used to adapt existing classes to a new interface.

Iterator Pattern

CS 2263

Introducing Iterator

- Let us say we have two List implementations:



Iterator Motivation

Idaho State University | Computer Science

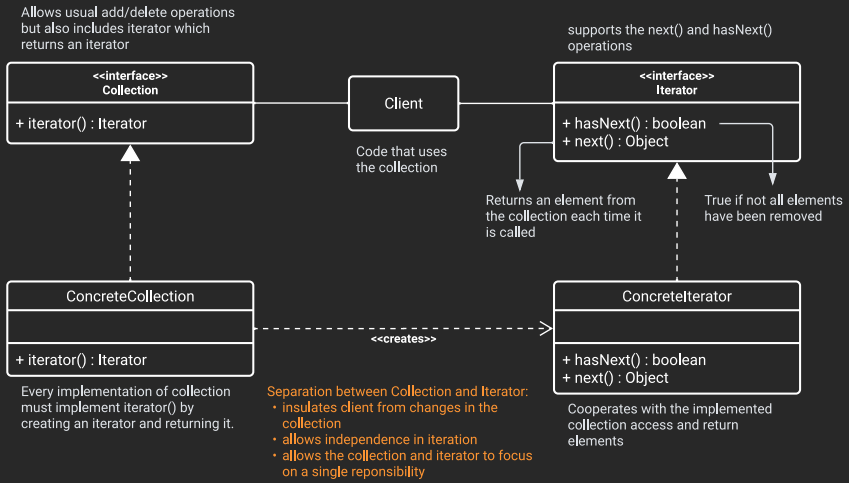
- Numerous applications need to maintain collections of objects
- Yet, depending on the needs of a particular application, the specific data structure may change
- Common Data Structures (as you know from CS 2235) are:
  - Lists
  - Sets
  - Maps
  - Queues
  - Deques
  - Stacks
  - Binary Trees
  - B-Trees
  - Graphs

Iterator Motivation

Idaho State University | Computer Science

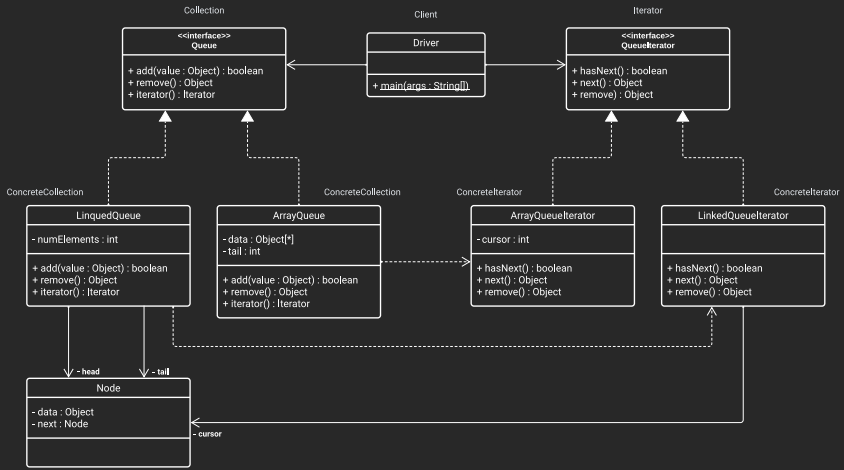
- The traversal of each of these structures and their various implementations, tend to be specific to those implementations
  - Yet, if we require every collection to provide a means of traversal, we end up violating both the SRP and LC principles
    - **SRP** - The collection both stores and traverses
    - **LC** - Client of the collection needs to be intimately aware of how to traverse
- Thus, we need a new object – **Iterator** which provides standard methods to traverse a collection.

Iterator Structure



Iterator Implementation

Idaho State University | Computer Science



Let’s See the Code!

Singleton Pattern

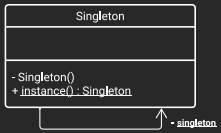
CS 2263

Introducing Singleton

Idaho State University | Computer Science

- In many situations we want to ensure that there is just one object of a certain class
- To do this we need a few things
  - A private default constructor
  - An attribute to hold the instance
  - A method to create, if not already created, and return the instance

Singleton Structure



Basic Singleton Implementation

```
public class B {
    private static B singleton;
    private B() {}
    public static B instance() {
        if (singleton == null)
            singleton = new B();
        return singleton;
    }
    // rest of class
}
```

Subclassing Singletons

Idaho State University | Computer Science

- Often we need to subclass a singleton with other singletons
- Example: A machine which runs multiple servers in separate processes where each is a singleton
  - General purpose server - time, directory, file, replication, name services
  - Directory server - sophisticated directory service
  - File Server - reading and updating files
  - File Server - only reading or creating new files
- **General Problem:** We need to implement two classes `B` and `D`, where `B` is the super class of `D` and both are singletons.

Steps in Subclassing Singletons

Idaho State University | Computer Science

1. Make `B`'s constructor **protected** rather than private
2. Add logic in the constructor to prevent `B` from creating new instances of itself
3. Add another private constructor (which takes a parameter) that will be used by `B`'s `instance()` method
4. Implement `D` as a standard singleton, but make its constructor call `B`'s protected constructor

Subclassing Implementation

Idaho State University | Computer Science

Superclass Implementation

```
import java.lang.reflect.*;

public class B {
  private B singleton;
  protected B() throws Exception {
    if (getClass().getName().equals("B"))
      throw new Exception();
  }
  private B(int i) {}

  public static B instance() {
    if (singleton == null) {
      singleton = new B(1);
    }
    return singleton
  }
  // rest of class
}
```

Subclass Implementation

```
public class D extends B {
  private static D singleton;
  protected D() {
    super();
  }
  public static D instance() {
    if (singleton == null)
      singleton = new D();
    return singleton;
  }
  // rest of class
}
```

Singleton Implementation

Idaho State University | Computer Science

- We have already seen the basic approach to implementation
- But, there have been several improvements to this approach
  - Enum
  - Double-checked Locking
  - Helper class
  - and many more

Basic Approach

- Benefits
  - Simple
  - Can be subclassed (with a little work)
- Drawbacks
  - Not thread-safe
  - In languages like Java, does not guarantee a single instance

Singleton Implementation

- We have already seen the basic approach to implementation
- But, there have been several improvements to this approach
  - **Enum**
  - Double-checked Locking
  - Helper class
  - and many more

Enum Approach

```
public enum E {

    INSTANCE;

    // methods
}
```

- Benefits
  - Thread safe
  - Simple
- Drawbacks
  - Cannot be subclassed

Singleton Implementation

- We have already seen the basic approach to implementation
- But, there have been several improvements to this approach
  - Enum
  - **Double-checked Locking**
  - Helper class
  - and many more
- Benefits
  - Thread safe
  - Can be subclassed
- Drawbacks
  - Convoluted

Double-checked Locking Approach

```
public class D {

  private static volatile D singleton;

  private D() {}
  public D instance() {
    D local = singleton;
    if (local == null) {
      synchronized (this) {
        local = singleton;
        if (local == null) {
          singleton = local = new D();
        }
      }
    }

    return singleton;
  }
```

Singleton Implementation

Idaho State University | Computer Science

- We have already seen the basic approach to implementation
- But, there have been several improvements to this approach
  - Enum
  - Double-checked Locking
  - **Helper class**
  - and many more

Helper Class Approach

```
public class H {

  private H() {}

  private static final class Helper() {
    private static final H INSTANCE = new H();
  }

  public static instance() {
    return Helper.INSTANCE;
  }
  // methods
}
```

- Benefits
  - Thread safe
  - Simpler than DCL
  - Allows for subclassing
  - Lazy Loading

Let’s See the Code!

Adapter Pattern

CS 2263

Introducing Adapter

- If we have an interface to which we need an implementation, but do not wish to undergo the labor intensive implement
- The, perhaps we can instead utilize an existing class to achieve this implementation via delegation
- We could then reach our desired functionality

Adapter Motivation

Idaho State University | Computer Science

- The General Problem:
  - Given an interface `I`
  - A existing class `C` which contains the set of methods `Mc`
  - We wish to create a class `A` which implements `I`
  - Such that each implemented method in `Mi` is realized via a combination of calls to methods in `Mc`

Adapter Types

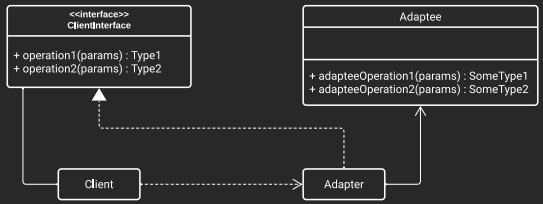
Idaho State University | Computer Science

- There are two types of adapters both of which will implement our strategy
  - **Object Adapters:** Creates an adapter class that implements a given interface using an instance of an existing class (adaptee)
    - This requires that the adapter class has an attribute of type `Adaptee` to which the work is delegated
    - **Downside** of this approach is that it will introduce another object to the system
    - **Upside** is that the choice of adaptee can be postponed to run-time
  - **Class Adapters:** Creates an adapter class that implements a given interface by extending an existing class (adaptee)
    - Less flexible than Object adapters due to the inheritance relationship
    - Choice of adaptee is forced to be made at compile-time
    - All public methods of the extended class will be exposed to the client

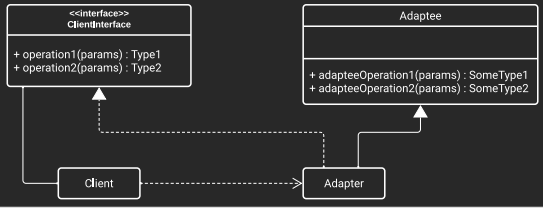
Adapter Structure

Idaho State University | Computer Science

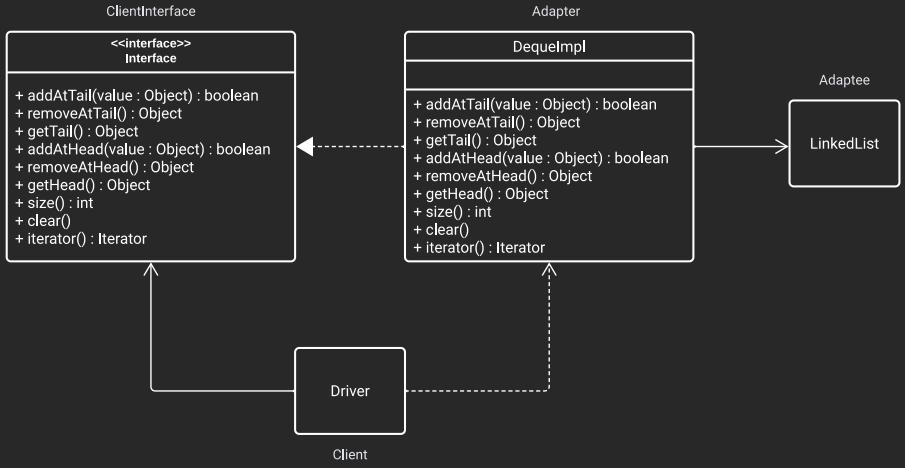
Object Adapter



Class Adapter



Adapter Implementation



Let’s See the Code!

For Next Time

- Review Chapter 5
- Review this Lecture
- Come to Class
- **Complete Project Part 1**
- Read Chapter 6

Are there any questions?