CS 342 Software Design

Java Interfaces

Java Interfaces

- A reference type similar to a class
- Can only contain constants(static and final), method signatures, default methods, static methods and nested types.
- An interface can extend one or many other interfaces
- A class can implement many interfaces; must define all of the interfaces methods unless the class is abstract

Most if not all design patterns make use of interfaces!!!

Differences Between Class and Interface

- You cannot instantiate an interface.
- An interface does not contain any constructors.
- All of the methods in an interface are abstract, static or default.
- An interface is not extended by a class; it is implemented by a class.
- An interface is implicitly abstract and methods are public

Differences Between a Abstract Class and Interface

- Abstract class can have fields that are not static and final.
- Abstract class can define public, protected and private concrete methods.

When to use an Abstract Class or an Interface

- Use abstract class with several ***closely related*** classes that share code, have common methods and fields or need access modifiers other than public.

- Use an interface if you expect ***unrelated classes*** to implement it, implement behavior of a data type but not concerned with who implements it or want multiple inheritance.

Why use an Interface?

If we were building a restaurant reservation application:

We would keep the reservation list in a data structure, but which one?

Linked List, Array, ArrayList….other?

At some point you would want to iterate through the list….

Depending on the data structure, the code changes to print out the list

Linked List: while(head != null){ Customer c = head; head = head.next;

Array: for(int i = 0; i < resList.length(); i++){ Customer c = resList[i]; }

ArrayList: for(int i = 0; i < resList.size(); i++){ Customer c = resList.get(i);}

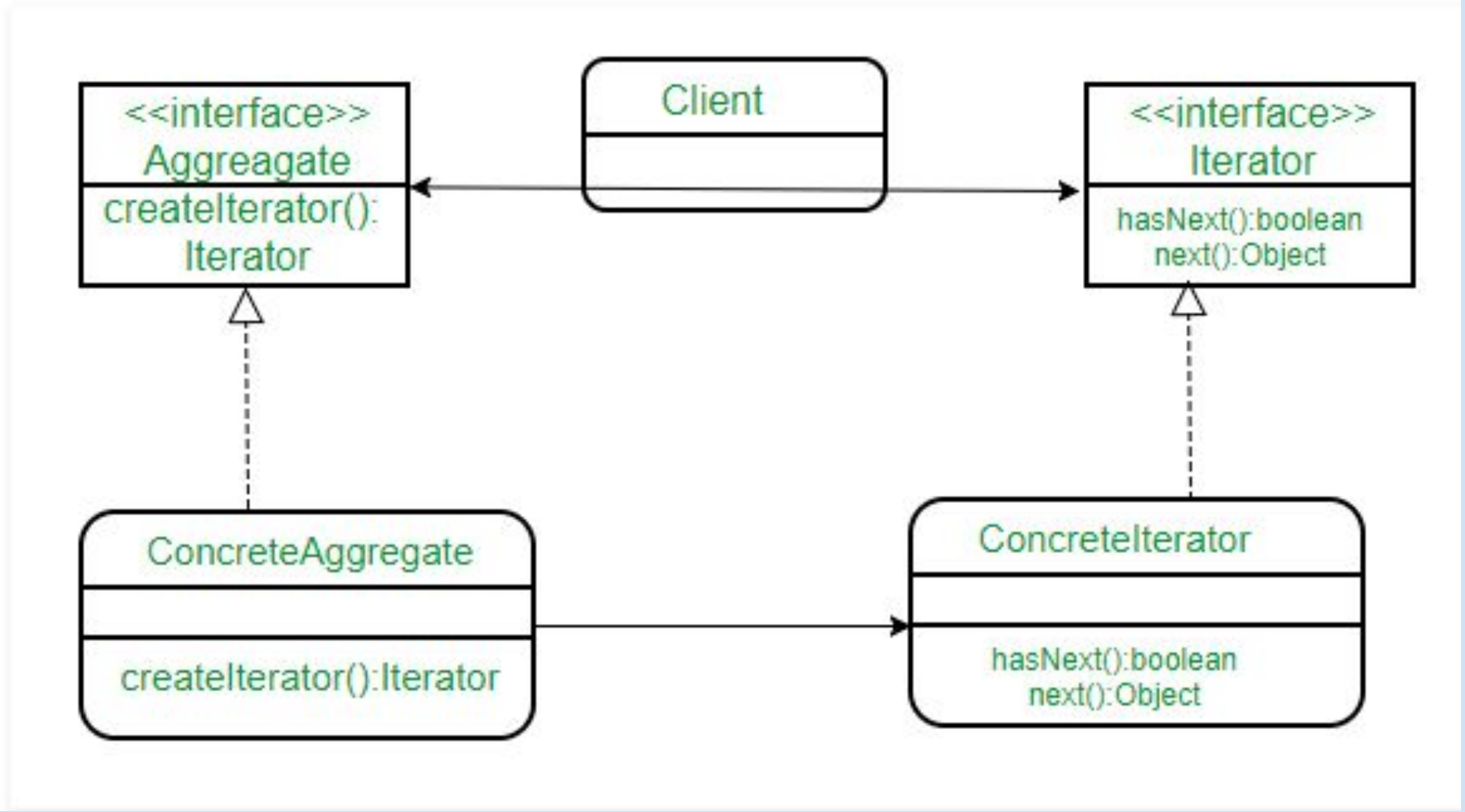
If we change the underlying data structure; we have to change everything!

We should give the client a generic way of iterating over the list that is independent of the type of list:

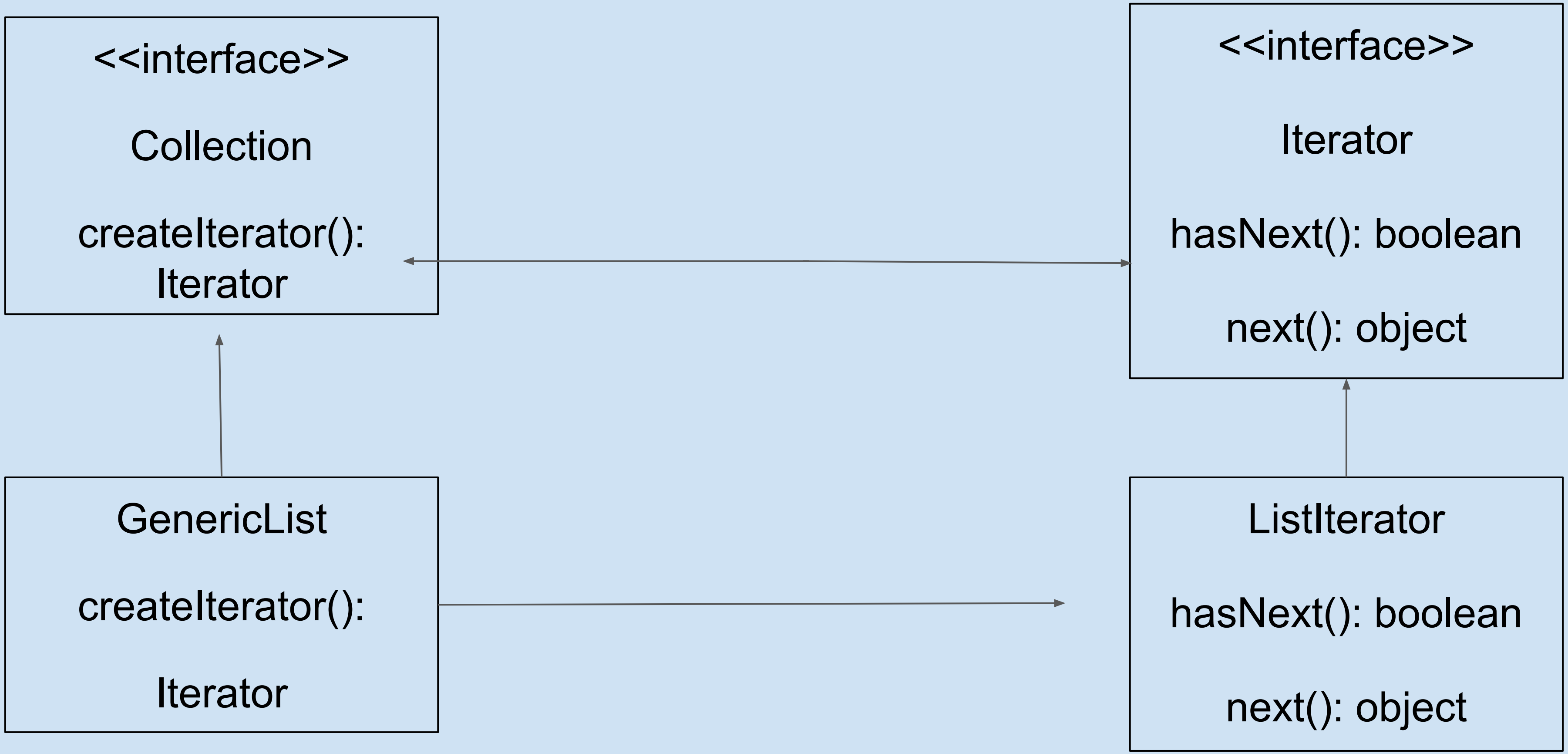
```
Iterator myIterator = reList.createIterator()
while(myIterator.hasNext()){ Customer c = myIterator.next(); }
```

Design Pattern: Iterator

Class Diagram:



Project #1 Iterator:



In class coding: