Iterator Pattern

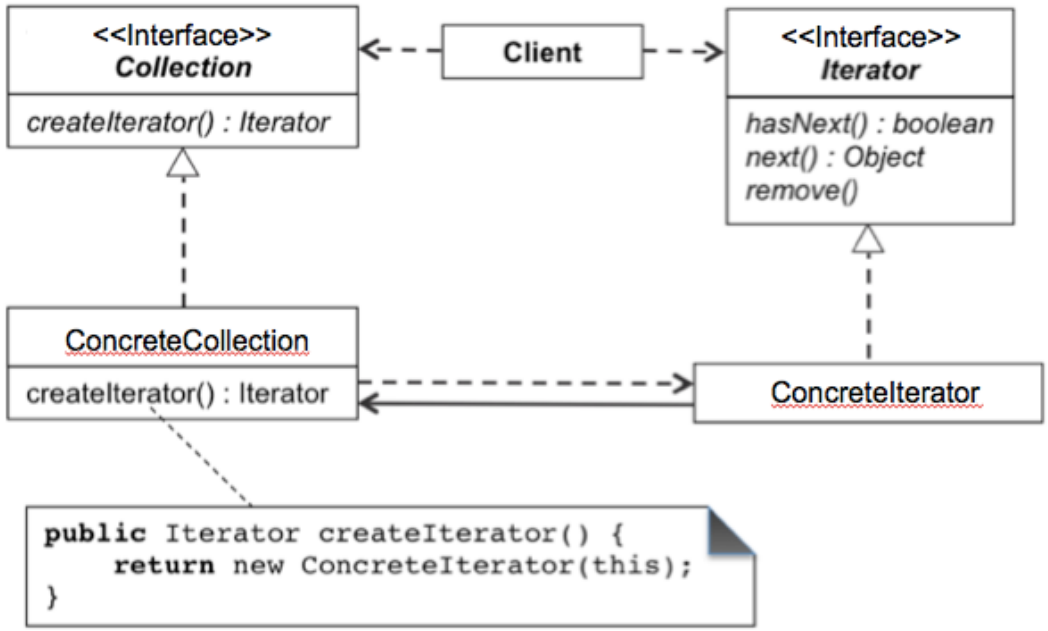
CSCI-4448 - Boese

University of Colorado **Boulder**

Problem

Problem

- You have a list of items, but do not want to change your code to walk through the list if the implementation changes from array, to ArrayList, to Vector, to LinkedList, etc.
- Regardless of the implementation, you just want to be able to walk through the items in the list!



Definition

Definition

"Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation."

-Gang of Four

Iterator

- **Recurring Problem**:
  How can you loop over all objects in any collection.
  - You don't want to change client code when the collection changes. Want the same methods
- **Solution**:
  - 1) Have each class implement an interface, and
  - 2) Have an interface that works with all collections

- **Consequences**: Can change collection class details without changing code to traverse the collection

Why

Definition

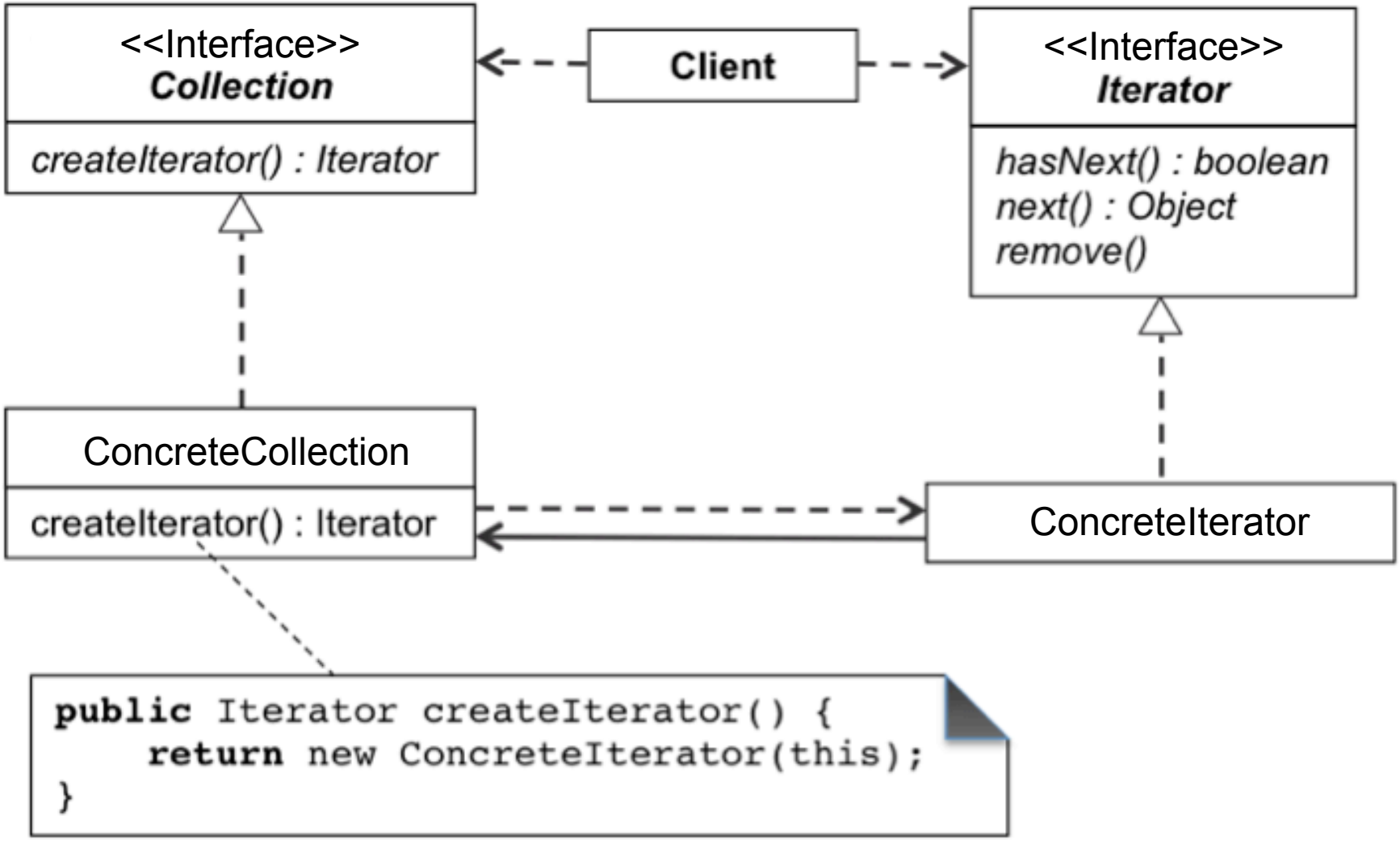
- **Name** “Iterator”
  - Uniform way of traversing.

- **Intent**
  - Access elements of an aggregate sequentially without exposing its representation
  - Provide different kind of iterators based on our requirements
  - Polymorphic traversal

- **As filters/views**
  - Return a partial list based on a filter
  - Different implementations of iterator can provide different criteria for the filter
- **Avoid**
  - Complicated loops with different strategies
  - Criteria passed as function parameters
  - Duplicate code
- Think of traversal/iteration as '**separate responsibility**'

How

Structure

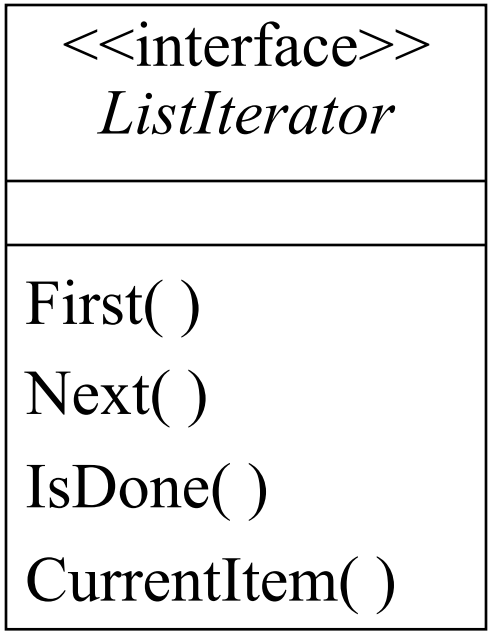

<<Interface>>
Collection
createIterator() : Iterator
Client
<<Interface>>
Iterator
hasNext() : boolean
next() : Object
remove()
ConcreteCollection
createIterator() : Iterator
ConcreteIterator
public Iterator createIterator() {
return new ConcreteIterator(this);
}

Iterator Pattern - Participants

- **Iterator**
  - Defines an interface for accessing and traversing elements
- **ConcreteIterator**
  - Implements the iterator interface
  - Keeps track of the current position in the traversal of the aggregate
- **Collection**
  - Defines an interface for defining an Iterator object
- **ConcreteCollection**
  - Implements the iterator creation interface to return an instance of the proper ConcreteIterator

Iterator Implementations

GOF



Java

```
public interface Iterator<E>
{
    boolean hasNext();
    E next();
    void remove(); //optional
}
```

- **`hasNext`** returns true if the iteration has more elements
- **`next`** returns the next element in the iteration
- **`remove`** removes the last element that was returned by **`next`** from the underlying Collection, call only once per **`next`**

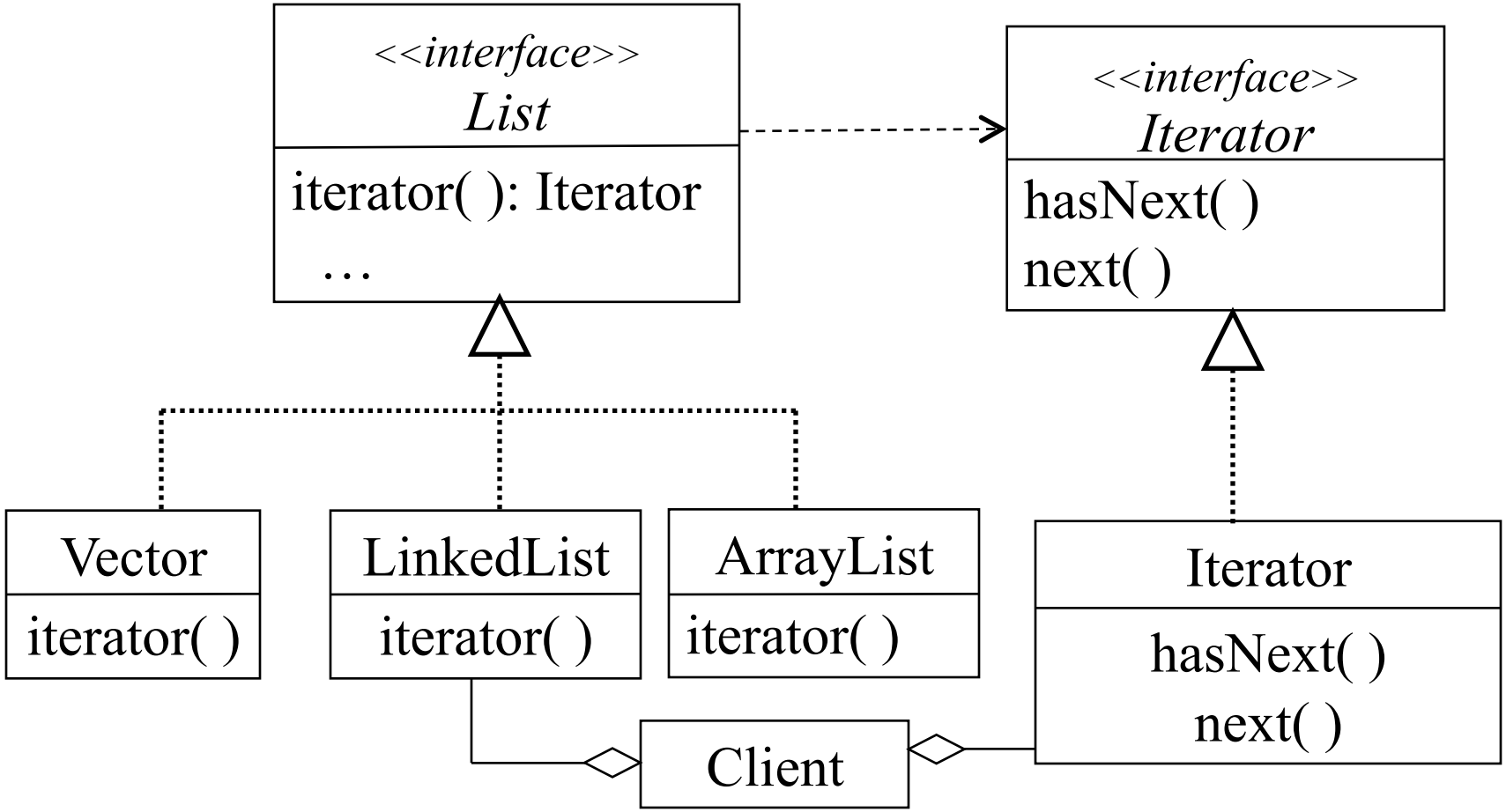
Using the Java *Iterator* Class

- Java has an *Iterator* class.
- The *Iterator* class has the following methods:
  - `hasNext()`
  - `next()`
  - `remove()`
- If the *remove()* method should not be allowed for a particular data structure, a `java.lang.UnsupportedOperationException` should be thrown.

Iterators and Collections

- In Java the data structure classes form part of the Java collections framework.
- These include the ArrayList, Vector, LinkedList, Stack and PriorityQueue classes.
- Each of these classes implements the *java.util.Collection* interface which forces all subclasses to have an *iterator()* method.
- The *Hashtable* class contains keys and values which must be iterated separately.

UML Diagram of Java's Iterator with a few Collections



http://download.oracle.com/javase/8/docs/api/java/util/List.html

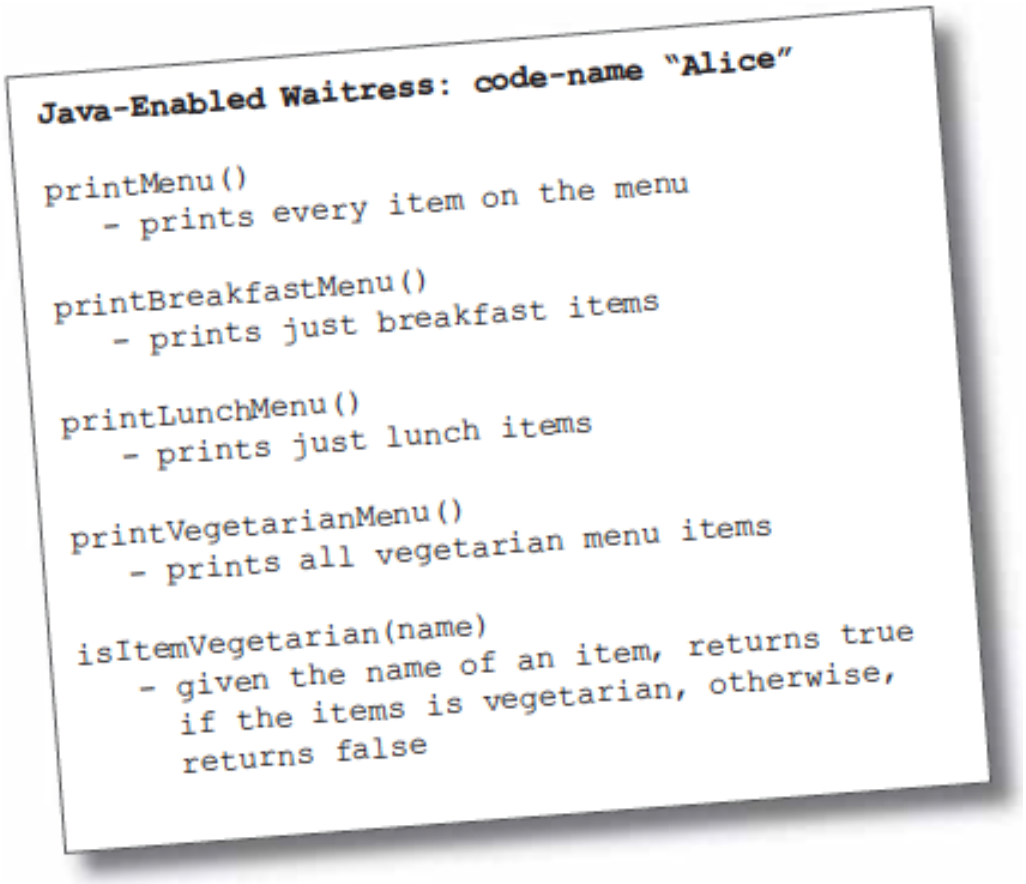
Example
Diner and Pancake House Merger

Example from Heads First Design Patterns (O'Reilly)

Diner and Pancake House Merger

Objectville diner and Objectville pancake house are merging

- Both menus need to merged.
- The problem
  - Pancake House stores items in an ArrayList
  - Diner stores items in an Array.
- Neither of the owners are willing to change their implementation.



Problems

Suppose we need to print every item on both menus.

- Two loops will be needed instead of one.
- If a third restaurant is included in the merger, three loops will be needed.
- Design principles that would be violated:
  - Coding to implementation rather than interface
  - The program implementing the joint print_menu() needs to know the internal structure of the collection of each set of menu items.
  - Duplication of code

Solution

Solution

- Encapsulate what varies, i.e. encapsulate the iteration.
- An iterator is used for this purpose.
- The *DinerMenu* class and the *PancakeMenu* class need to implement a method called *createIterator()*.
- The *Iterator* is used to iterate through each collection without knowing its type (i.e. Array or ArrayList)

Original Iteration

Do it the hard way

- Getting the menu items:

```
PancakeHouseMenu pancakeHouseMenu = new PancakeHouseMenu();
ArrayList breakfastItems = pancakeHouseMenu.getMenuItems();
DinerMenu dinerMenu = new DinerMenu();
MenuItems[] lunchItems = dinerMenu.getMenuItems():
```

- Iterating through the breakfast items:

```
for(int i=0; i < breakfastItems.size(); ++i)
{
  MenuItem menuItem =(MenuItem) breakfastItems.get(i);
}
```

- Iterating through the lunch items:

```
for(int i=0; I < lunchItems.length; i++)
{
    MenuItem menuItem = lunchItems[i];
}
```

Using an Iterator

Use an iterator

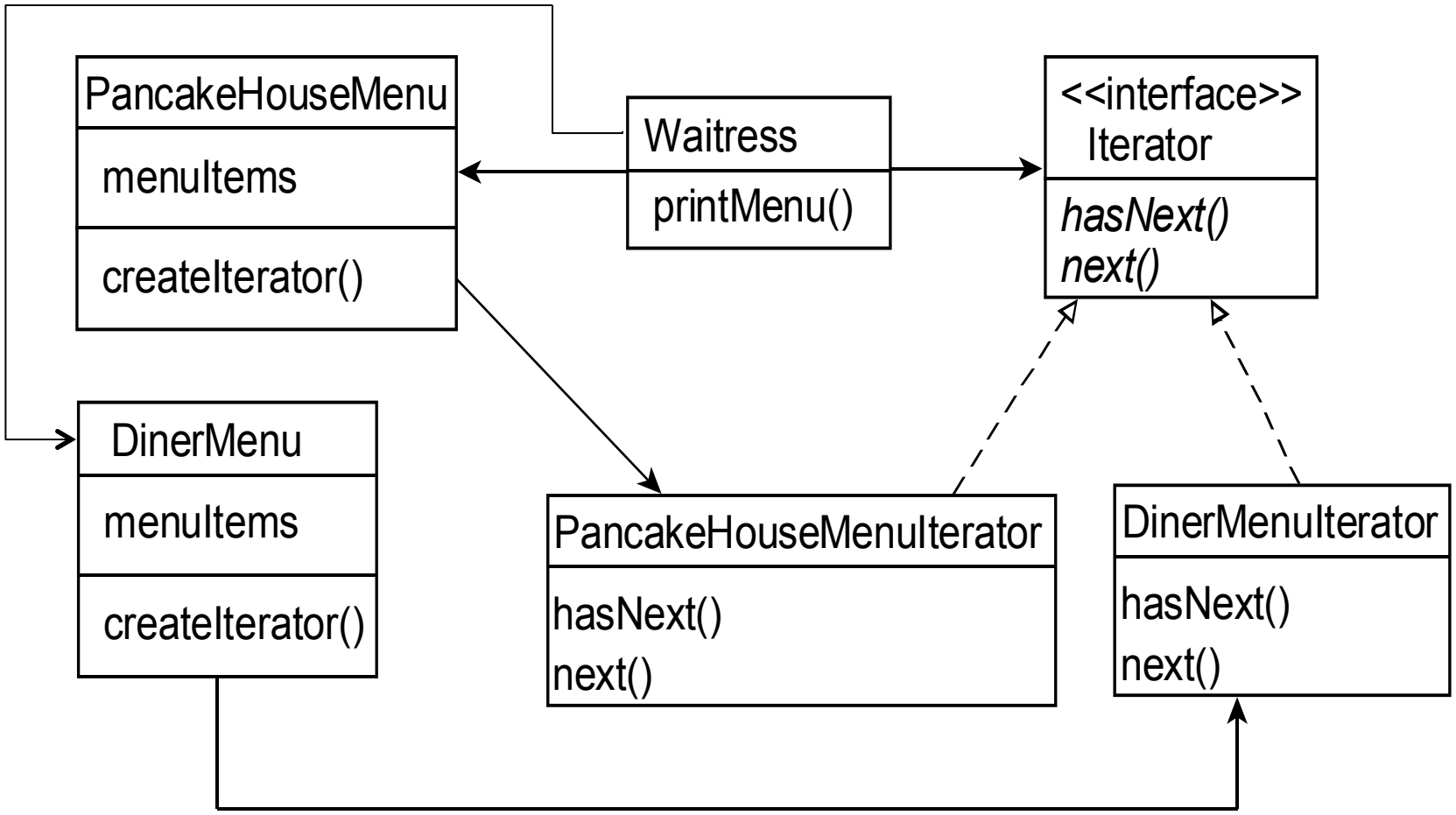
- Iterating through the breakfast items:

```
Iterator iterator = breakfastMenu.createInstance();
while(iterator.hasNext())
{
    MenuItem menuItem = (MenuItem)iterator.next();
}
```

- Iterating through the lunch items:

```
Iterator iterator = lunchMenu.createIterator();
while(iterator.hasNext())
{
    MenuItem menuItem = (MenuItem)iterator.next();
}
```

Class Diagram for the Merged Diner



Improving the Diner Code

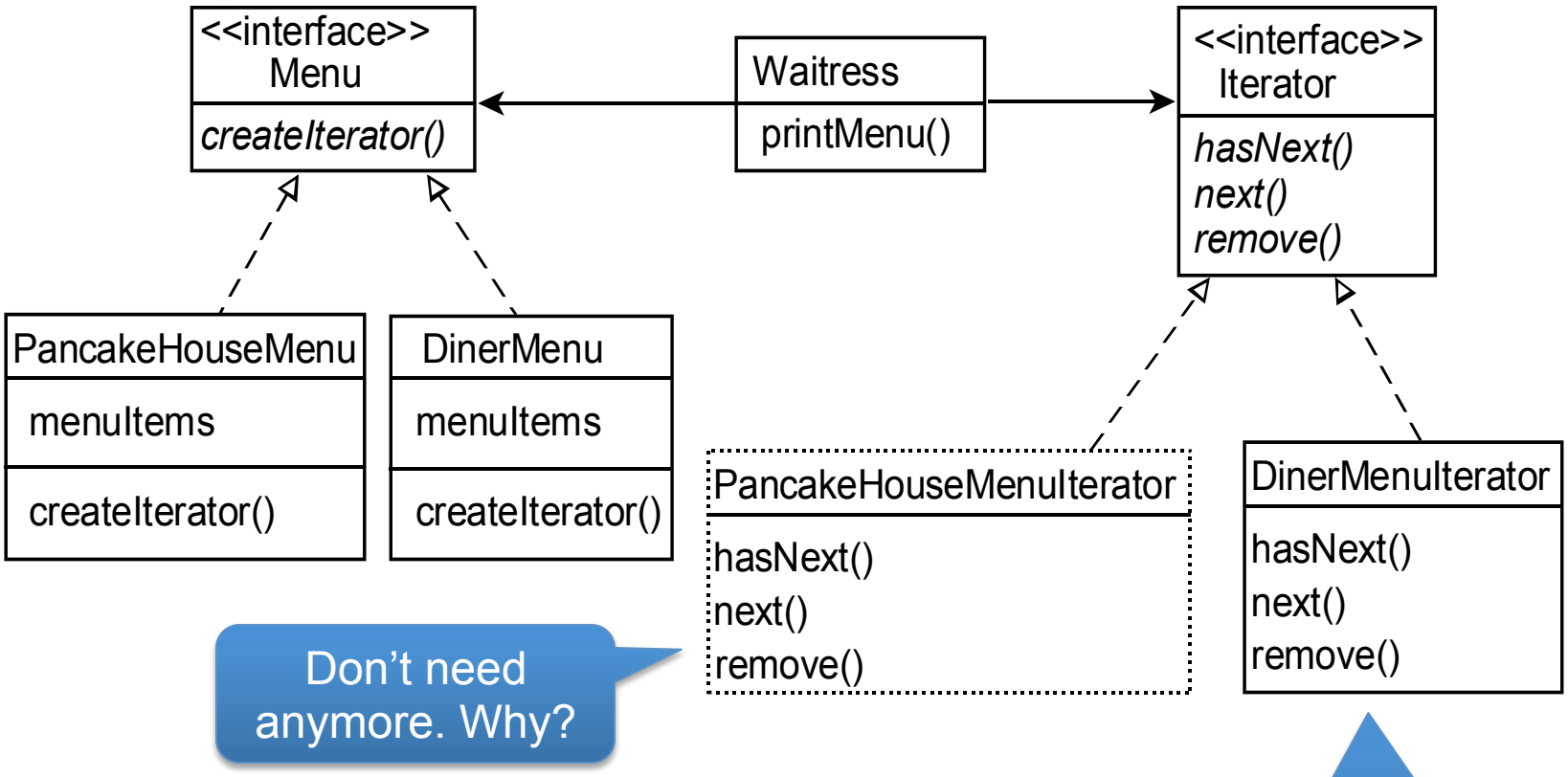
- Changing the code to use *`java.util.Iterator`*:
  - Delete the PancakeHouseIterator as the ArrayList class has a method to return a Java Iterator.
  - Change the *DinerMenuIterator* to implement the Java *Iterator*.

- Another problem - all menus should have the same interface.
  - Include a *Menu* interface

Class Diagram Include *Menu*

Simplified Client!

Adding the *Menu* interface



Exercise

Extend the current restaurant system to include a dinner menu from Objectville café.

The program that for the café stores the menu items in *Hashtable*. Examine and change the code to integrate the code into the current system.

Changes

- The *CafeMenu* class must implement the *Menu* interface.
- Delete the *getItems()* method from the *CafeMenu* class.
- Add a *createIterator ()*method to the *CafeMenu* class.
- Changes to the *Waitress* class
  - Declare an instance of *Menu* for the *CafeMenu*.
  - Allocate the *CafeMenu* instance in the constructor.
  - Change the *printMenu()* method to get the iterator for the *CafeMenu* and print the menu.
- Test the changes