Tecniche di Progettazione: Design Patterns

GoF: Iterator

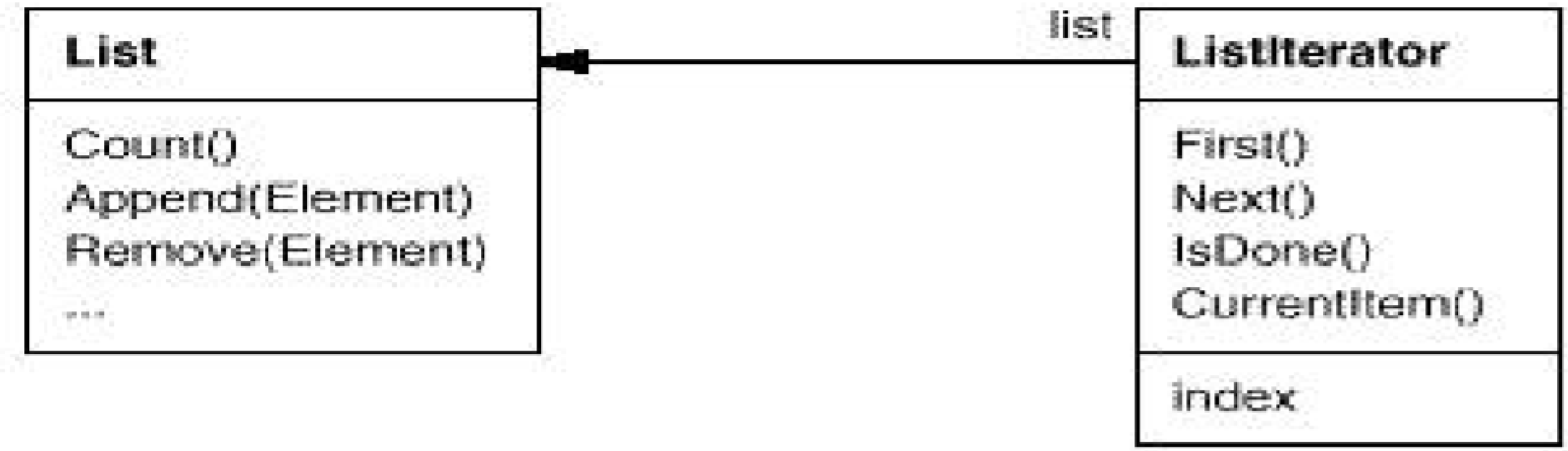
The Iterator Pattern

- Intent
  - Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation
  - An aggregate object is an object that contains other objects for the purpose of grouping those objects as a unit. It is also called a container or a collection. Examples are a linked list and a hash table.
- Also Known As
  - Cursor

The Iterator Pattern

- Motivation
  - An aggregate object such as a list should allow a way to traverse its elements without exposing its internal structure
  - It should allow different traversal methods
  - It should allow multiple traversals to be in progress concurrently
  - But, we really do not want to add all these methods to the interface for the aggregate

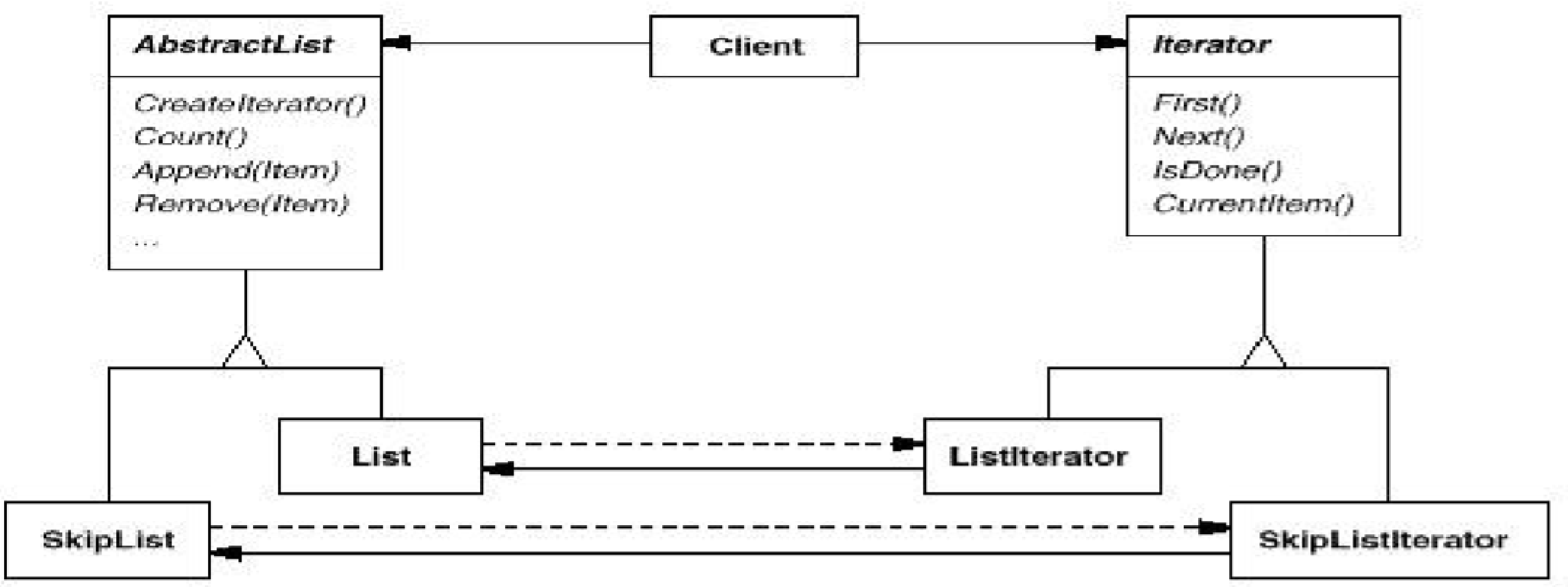
Ex: List with iterator



Typical client code

```
List list = new List();   ...
ListIterator iterator = new ListIterator(list);
iterator.First();
while (!iterator.IsDone()) {
      Object item = iterator.CurrentItem();
      // Code here to process item.
      iterator.Next();   }
...
```

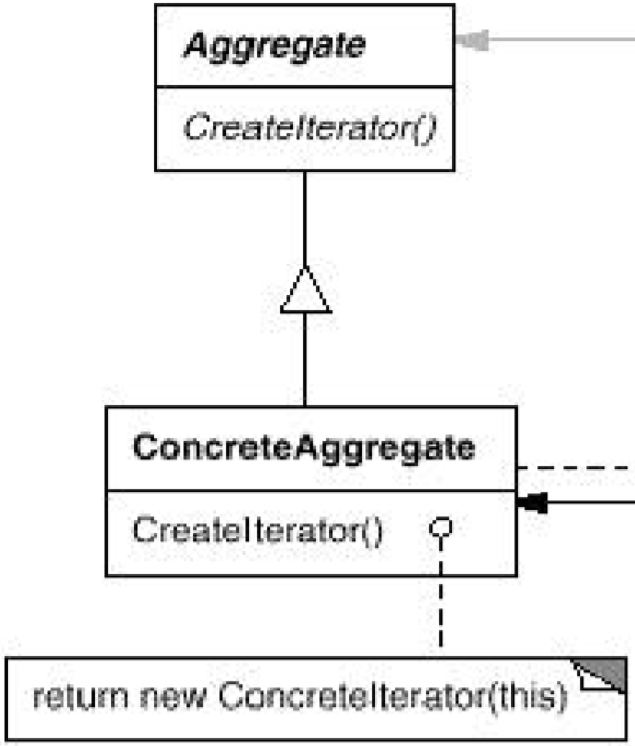
Ex: Polymorphic Iterator



Typical client code

```
List list = new List();
SkipList skipList = new SkipList();
Iterator listIterator = list.CreateIterator();
Iterator skipListIterator =
skipList.CreateIterator();
handleList(listIterator);
handleList(skipListIterator);
...
public void handleList(Iterator iterator) {
      iterator.First();
      while (!iterator.IsDone()) {
            Object item = iterator.CurrentItem();
            // Code here to process item.
            iterator.Next();      }
}
```

Structure


Aggregate
CreateIterator()
Client
Iterator
First()
Next()
IsDone()
CurrentItem()
ConcreteAggregate
CreateIterator()
ConcreteIterator
return new ConcreteIterator(this)

Participants

- Iterator
  - Defines an interface for accessing and traversing elements
- ConcreteIterator
  - Implements the Iterator interface Keeps track of the current position in the traversal
- Aggregate
  - Defines an interface for creating an Iterator object (a factory method!)
- ConcreteAggregate
  - Implements the Iterator creation interface to return an instance of the proper ConcreteIterator

Implementation Issues

- Who controls the iteration?
  - The client => more flexible(explicit iterator)
  - The iterator itself => (implicit iterator)

- Who defines the traversal algorithm?
  - The iterator => more common; easier to have variant traversal techniques
  - The aggregate => iterator only keeps state of the iteration

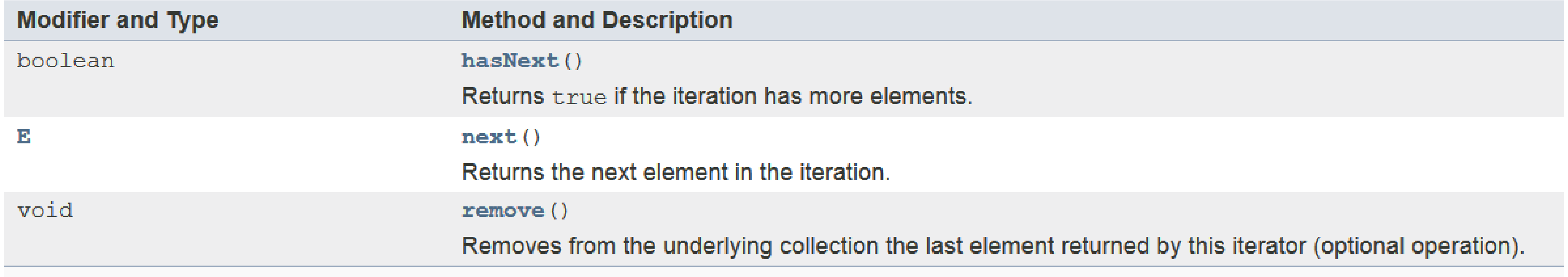
Implementation Issues cont'd

- Can the aggregate be modified while a traversal is ongoing?
  - An iterator that allows insertion and deletions without affecting the traversal and without making a copy of the aggregate is called a robust iterator.
- Should we enhance the Iterator interface with additional operations, such as previous()?

Related Patterns

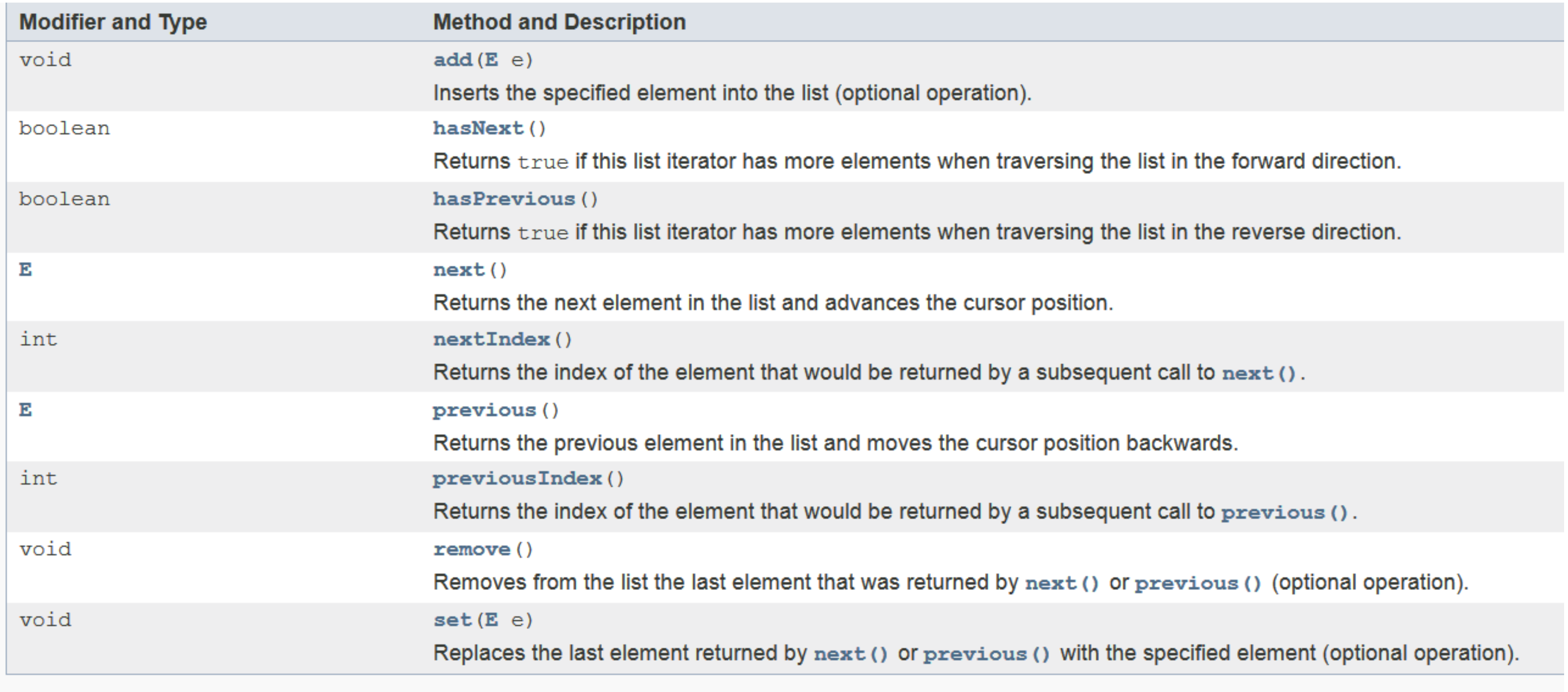
- Factory Method
  - Polymorphic iterators use factory methods to instantiate the appropriate iterator subclass
- Composite
  - Iterators are often used to recursively traverse composite structures

Java: Iterator<E> interface



Modifier and Type	Method and Description
`boolean`	`hasNext()` Returns `true` if the iteration has more elements.
`E`	`next()` Returns the next element in the iteration.
`void`	`remove()` Removes from the underlying collection the last element returned by this iterator (optional operation).

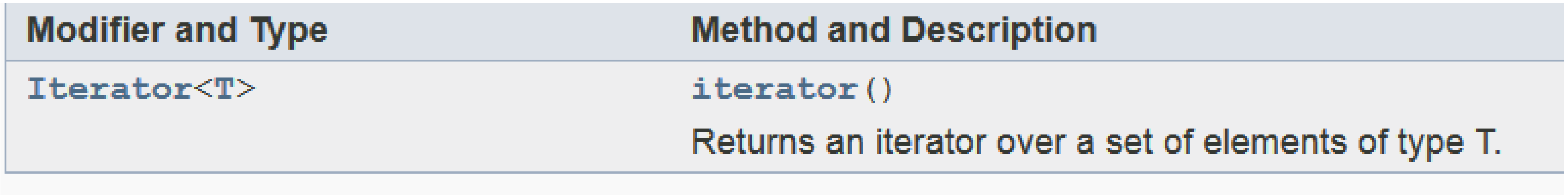
Java: ListIterator<E> extends Iterator<E>



Modifier and Type	Method and Description
`void`	`add(E e)` Inserts the specified element into the list (optional operation).
`boolean`	`hasNext()` Returns `true` if this list iterator has more elements when traversing the list in the forward direction.
`boolean`	`hasPrevious()` Returns `true` if this list iterator has more elements when traversing the list in the reverse direction.
`E`	`next()` Returns the next element in the list and advances the cursor position.
`int`	`nextIndex()` Returns the index of the element that would be returned by a subsequent call to `next()`.
`E`	`previous()` Returns the previous element in the list and moves the cursor position backwards.
`int`	`previousIndex()` Returns the index of the element that would be returned by a subsequent call to `previous()`.
`void`	`remove()` Removes from the list the last element that was returned by `next()` or `previous()` (optional operation).
`void`	`set(E e)` Replaces the last element returned by `next()` or `previous()` with the specified element (optional operation).

Java: interface Iterable<E>

- Implementing this interface allows an object to be the target of the "foreach" statement.



Modifier and Type	Method and Description
`Iterator<T>`	`iterator()` Returns an iterator over a set of elements of type T.

- Advantages and disadvantages: discussion

Diner and Pancake House Merger

Breaking News: ***Objectville Diner*** **and** ***Objectville Pancake House*** **Merge**

- Thus, both menus need to merged.
- The problem is that the menu items have been stored in an ArrayList for the pancake house and an Array for the diner.
- Neither of the owners are willing to change their implementation.

Problems

- Suppose we are required 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

- 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

- **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

- Iterating through the breakfast items:

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

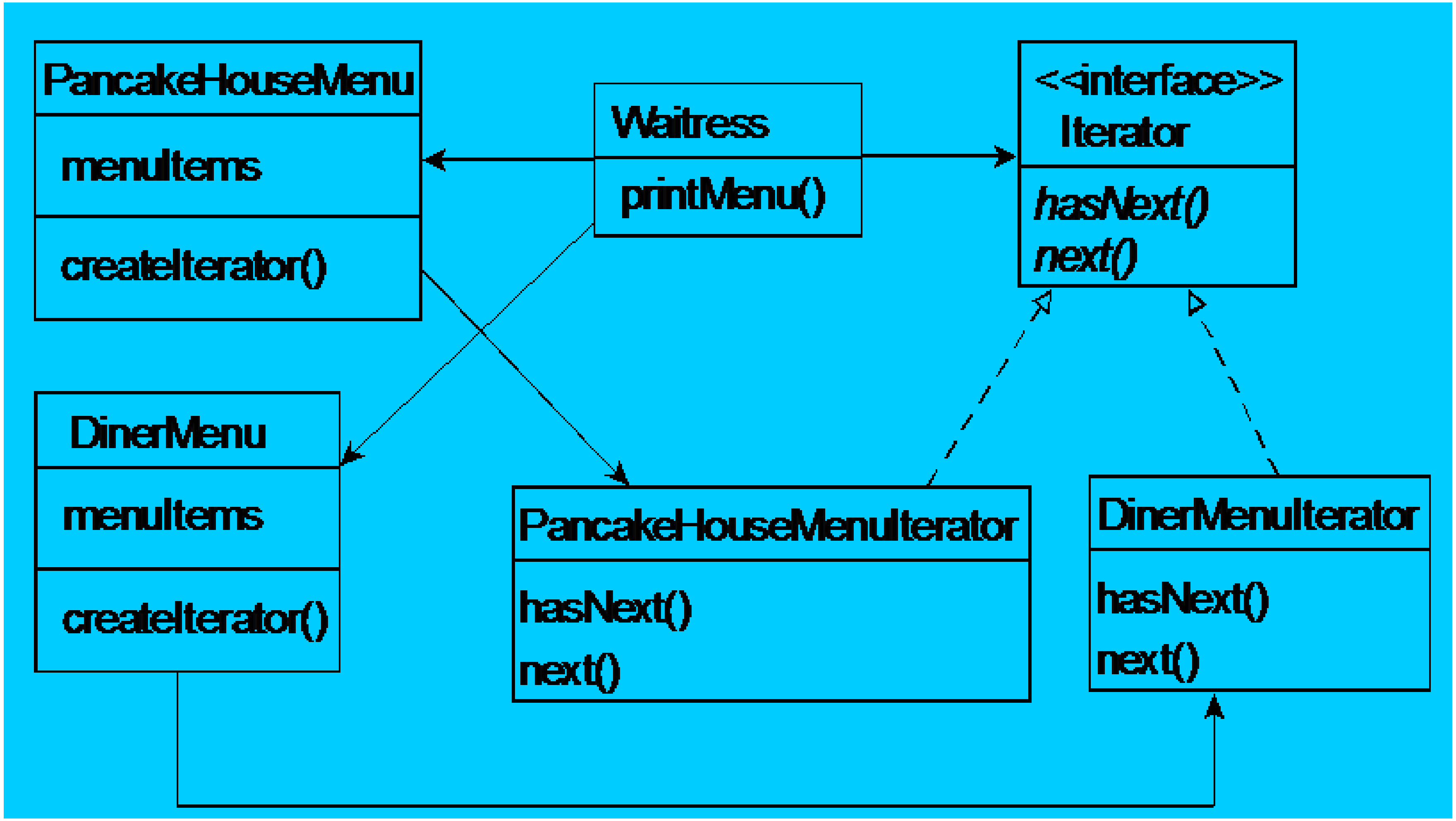
- Iterating through the lunch items:

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

Iterator Design Pattern

- The iterator pattern encapsulates iteration.
- The iterator pattern requires an interface called Iterator.
- The Iterator interface has two methods:
  - hasNext()
  - next()
- Iterators for different types of data structures are implemented from this interface.

Class Diagram for the Merged Diner



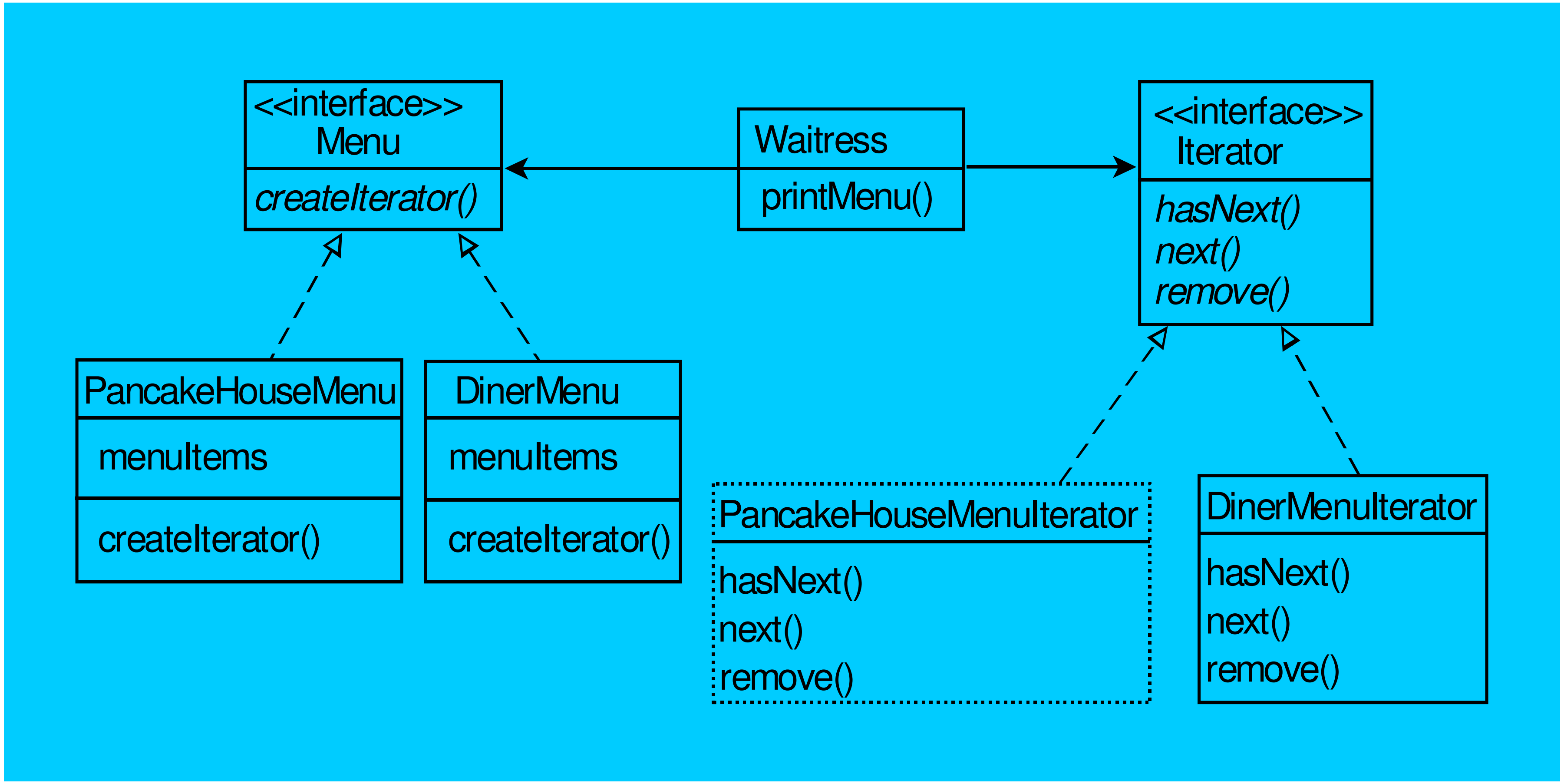
Using the Java Iterator Class

- Java has an Iterator class.
- The Iterator class has the following methods:
  - hasNext()
  - next()
  - remove()
    - Removes from the underlying collection the last element returned by the iterator (optional operation). This method can be called only once per call to next. The behavior of an iterator is unspecified if the underlying collection is modified while the iteration is in progress in any way other than by calling this method.
    - If the remove() method should not be allowed for a particular data structure, a java.lang.UnsupportedOperationException should be thrown.

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

Adding the Menu interface



Some Facts About the Iterator Pattern

- Earlier methods used by an iterator were first(), next(), isDone() and currentItem().
- Two types of iterators: internal and external.
- An iterator can iterate forward and backwards.
- Ordering of elements is dictated by the underlying collection.
- Promotes the use of "polymorphic" iteration by writing methods that take Iterators as parameters.
- Enumeration is a predecessor of Iterator.

Design Principle

- If collections have to manage themselves as well as iteration of the collection this gives the class two responsibilities instead of one.
- Every responsibility is a potential area for change.
  - More than one responsibility means more than one area of change.
- Thus, each class should be restricted to a single responsibility.
- Single responsibility: A class should have only one reason to change.
- High cohesion vs. low cohesion.

Exercise

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

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 iterated separately.

Problems with this Code? (waitress)

```
public void printMenu()
{
  Iterator pancakeIterator =
                      pancakeHouseMenu.createIterator();
  Iterator dinerIterator = dinerMenu.createIterator();
  Iterator cafeIterator = cafeMenu.createIterator();

  System.out.println("MENU\n----\nBREAKFAST");
  printMenu(pancakeIterator);
  System.out.println("\nLUNCH");
  printMenu(dinerIterator);
  System.out.println(“\nDINNER”);
  printMenu(cafeIterator);
}
```

Starts getting lenghty…..

Iterate over menus

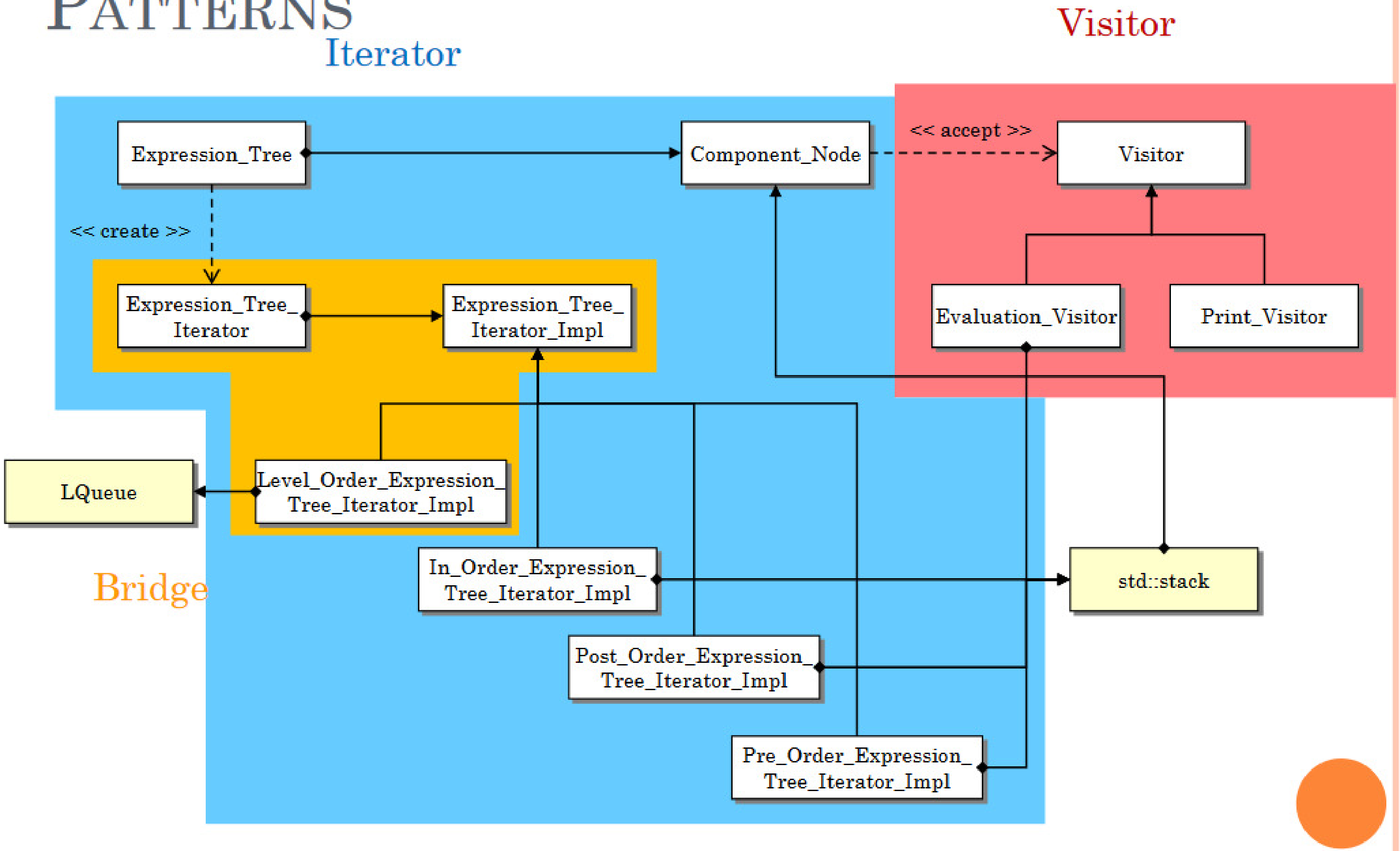
```
public class Waitress{
    ArrayList menus;
    public void printMenu(){
        Iterator menuIterator = menus.iterator();
        while (menuIterator.hasNext()){
            Menu menu = (Menu) = menuIterator .next();
            printMenu(menu.createIterator());
        }
    }
    public void printMenu(Iterator iterator);{…}
    }
```

Discussion

- Iterator vs Visitor

- (Homework in the composite slides)

SUMMARY OF TREE TRAVERSAL PATTERNS



Iterator
Visitor
Expression_Tree
Component_Node
<< accept >>
Visitor
<< create >>
Expression_Tree_Iterator
Expression_Tree_Iterator_Impl
Evaluation_Visitor
Print_Visitor
LQueue
Level_Order_Expression_Tree_Iterator_Impl
In_Order_Expression_Tree_Iterator_Impl
std::stack
Bridge
Post_Order_Expression_Tree_Iterator_Impl
Pre_Order_Expression_Tree_Iterator_Impl