Tecniche di Progettazione: Design Patterns

GoF: Iterator

Diner and Pancake House Merger

- Objectville diner and Objectville pancake house are merging into one entity.
- 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();
}
```

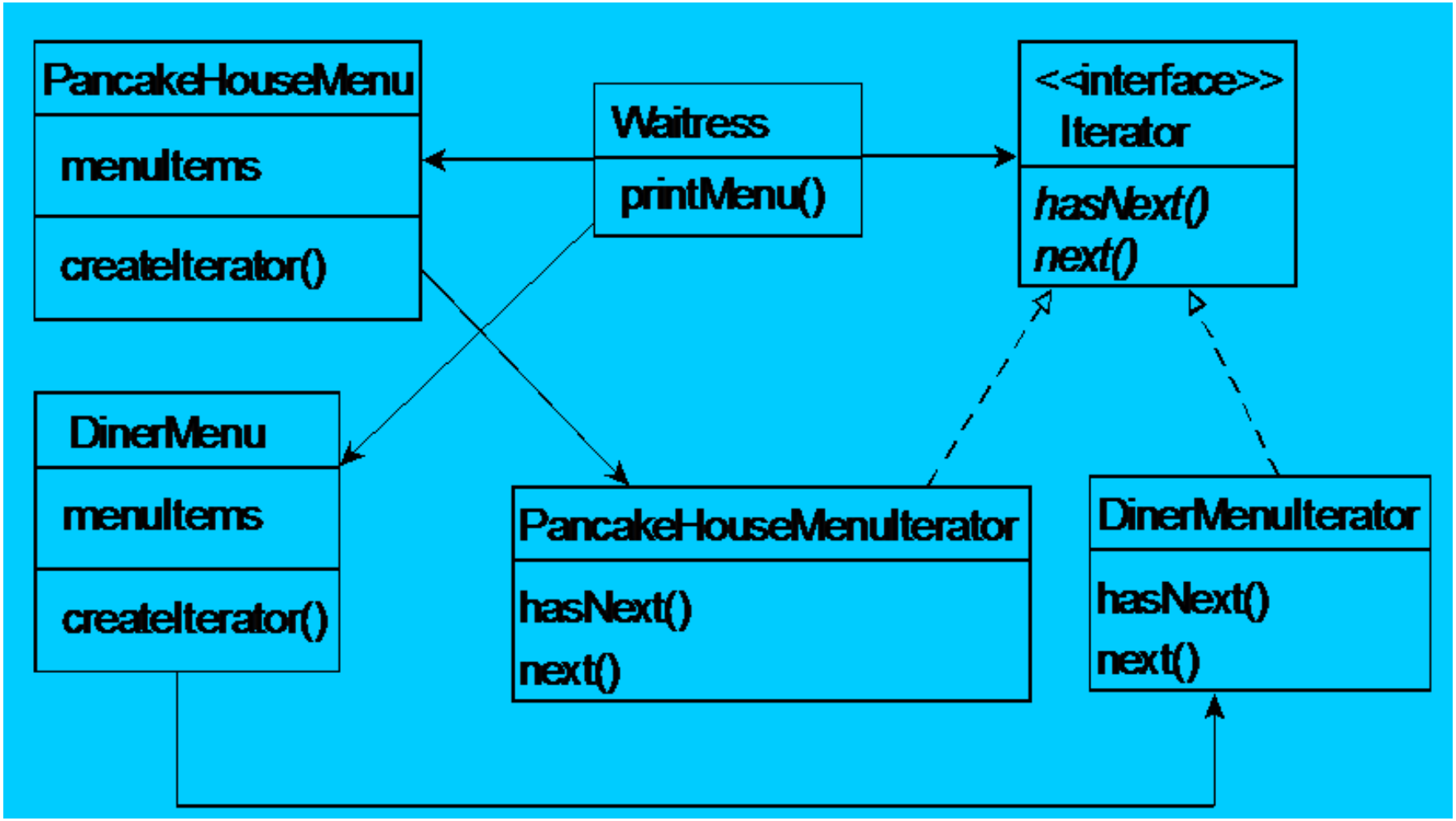
- Iterating through the lunch items:

```
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



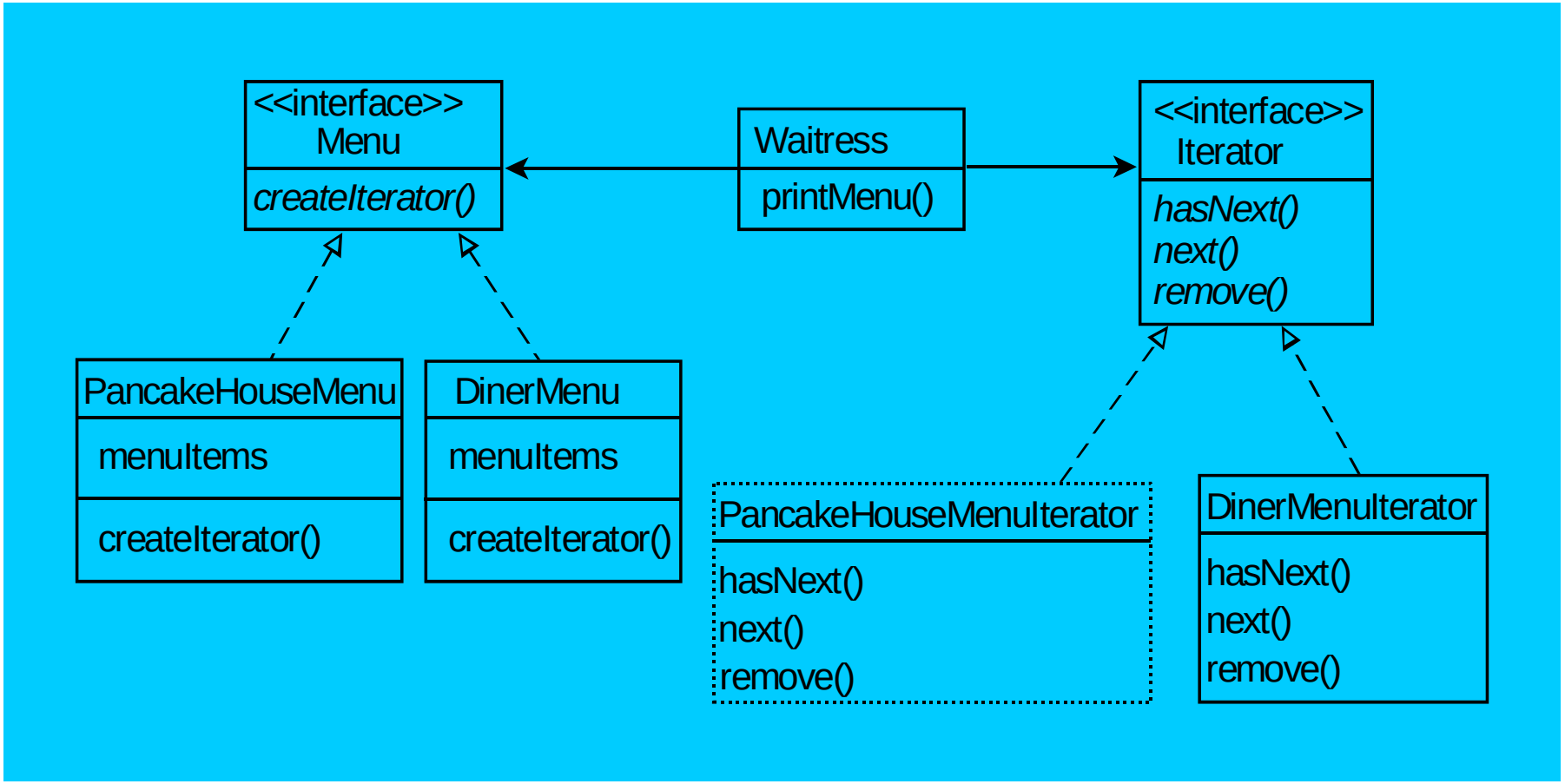
Using the Java Iterator Class

- 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

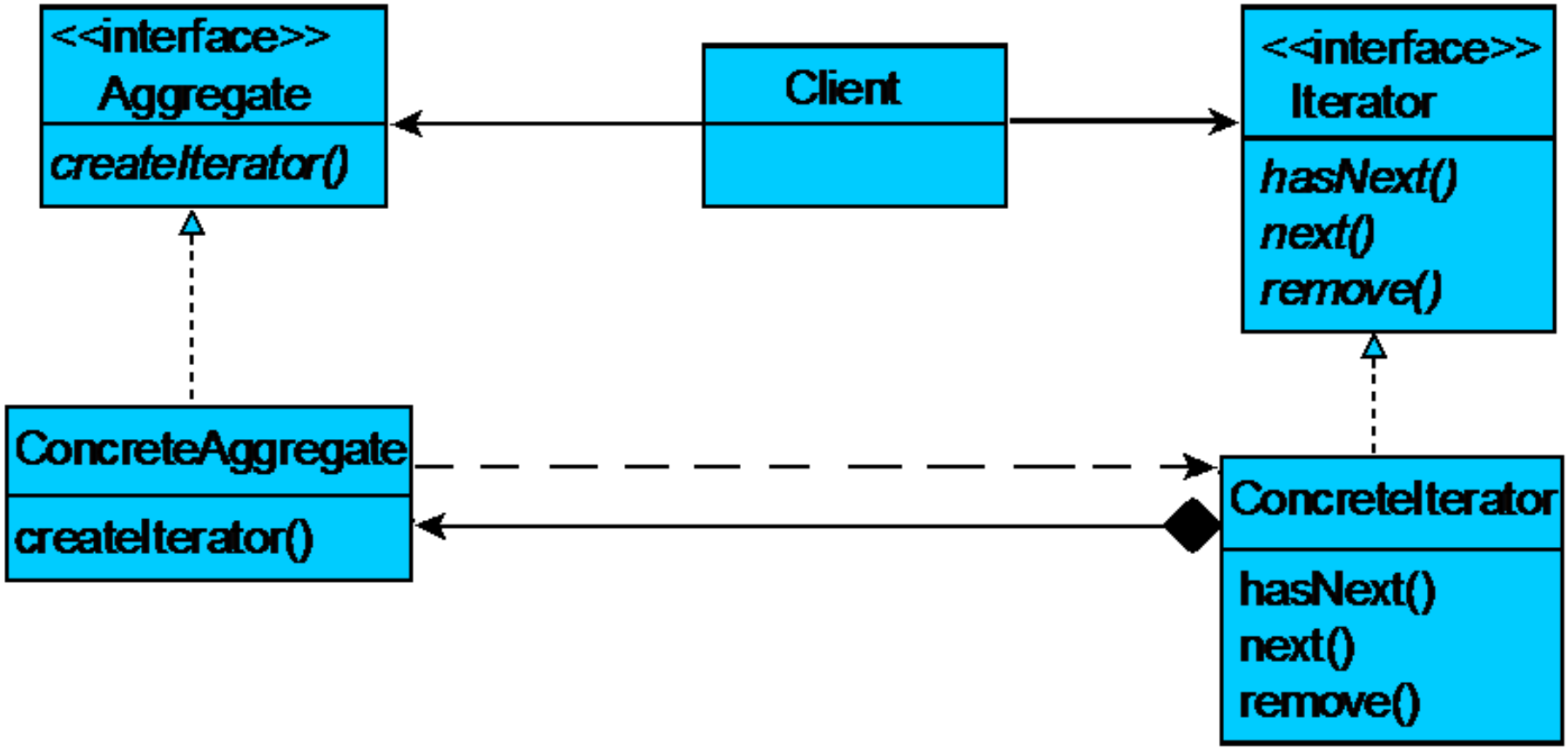
Adding the Menu interface



Iterator Pattern Definition

- Allows the traversal of the elements of a collection without exposing the underlying implementation.

Iterator Pattern Class Diagram



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

- Both will give successive elements, but Iterator is improved in such a way so:
  - Iterators allow the caller to remove elements.
  - Method names have been improved (shorter).

Enumeration	Iterator
hasMoreElement()	hasNext()
nextElement()	next()
N/A	remove()

- As also mentioned in the Java API Specifications, for newer programs, Iterator should be preferred over Enumeration, as "Iterator takes the place of Enumeration in the Java collections framework."

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 dinner menu from Objectville café.
- The program 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

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);
}
```

Comincia a diventare prolisso…..

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);{…}
   }
```