Lecture 9

Daniela Oliveira

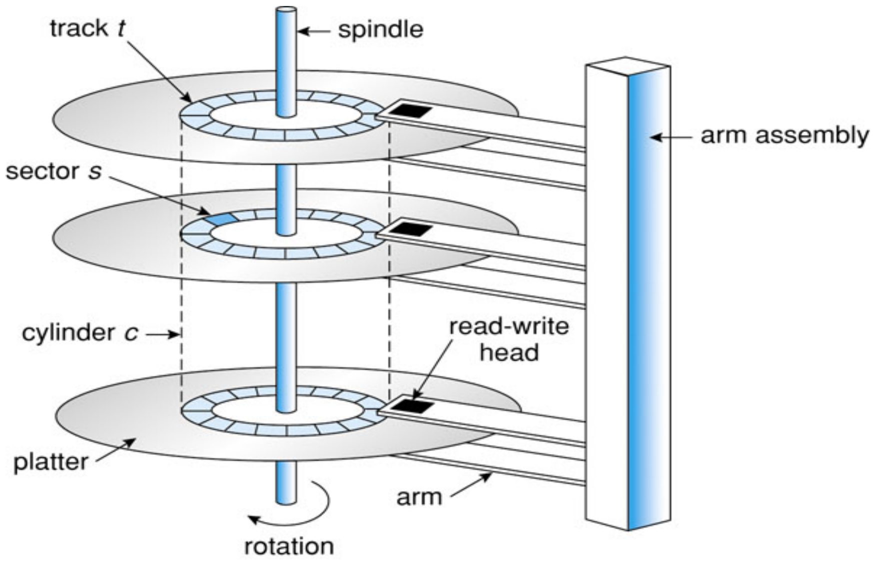
Assignment

Massive Datasets

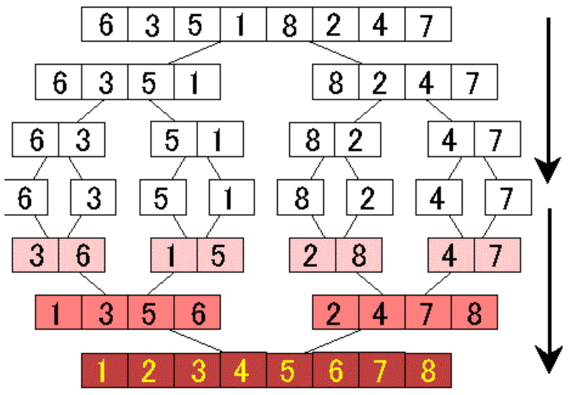
Assignment

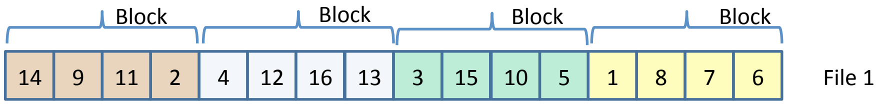
Implement an on-disk sorting algorithm for large data sets

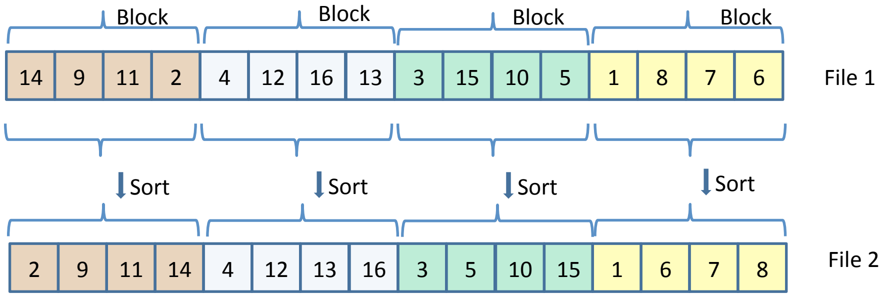
Magnetic Disks

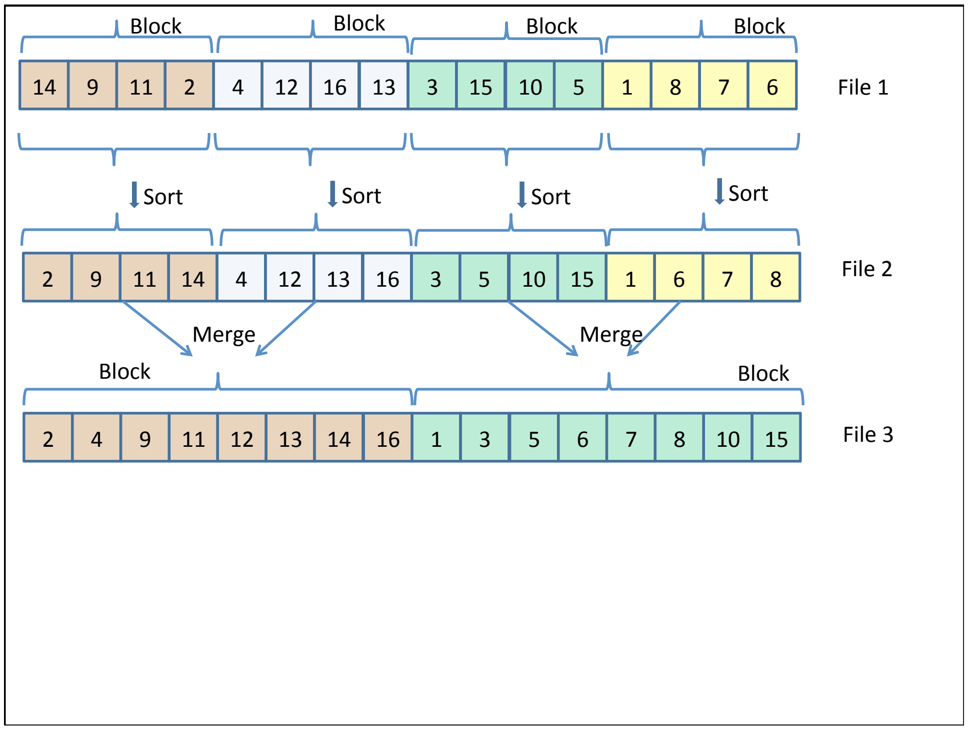


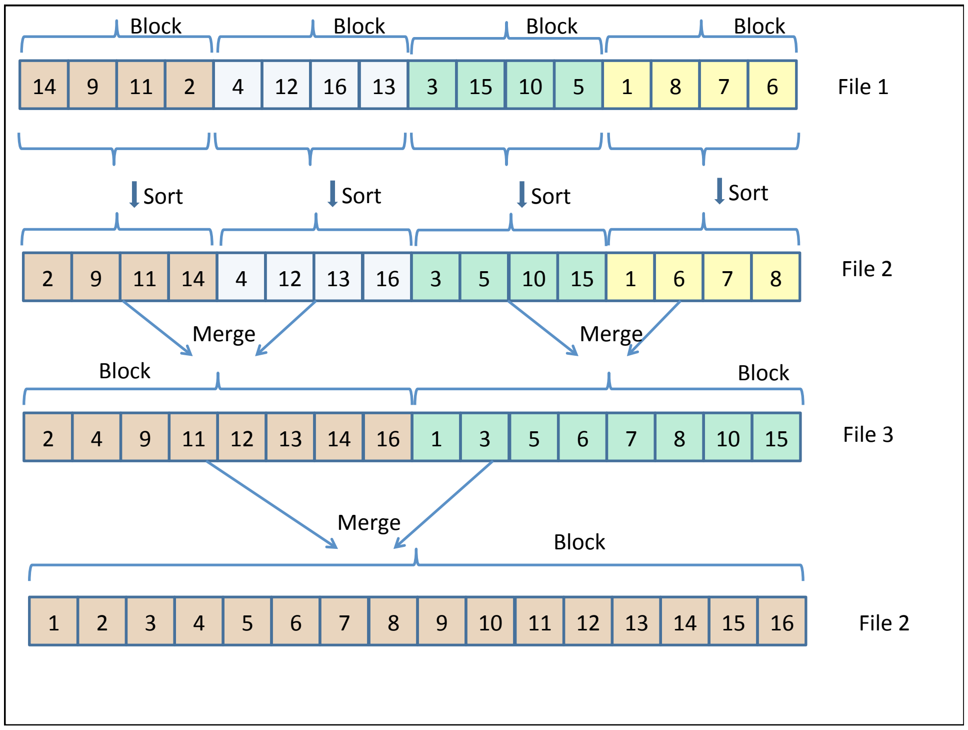
Mergesort




Block
Block
Block
Block
14 9 11 2 4 12 16 13 3 15 10 5 1 8 7 6
File 1


Block
Block
Block
Block
14 9 11 2 4 12 16 13 3 15 10 5 1 8 7 6
File 1
Sort
Sort
Sort
Sort
2 9 11 14 4 12 13 16 3 5 10 15 1 6 7 8
File 2


Block
Block
Block
Block
14 9 11 2 4 12 16 13 3 15 10 5 1 8 7 6
File 1
Sort
Sort
Sort
Sort
2 9 11 14 4 12 13 16 3 5 10 15 1 6 7 8
File 2
Merge
Merge
Block
Block
2 4 9 11 12 13 14 16 1 3 5 6 7 8 10 15
File 3


Block
Block
Block
Block
14 9 11 2 4 12 16 13 3 15 10 5 1 8 7 6
File 1
Sort
Sort
Sort
Sort
2 9 11 14 4 12 13 16 3 5 10 15 1 6 7 8
File 2
Merge
Merge
Block
Block
2 4 9 11 12 13 14 16 1 3 5 6 7 8 10 15
File 3
Merge
Block
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
File 2

Iterators

Iterators

- Simple and elegant way for accessing the elements of a collection

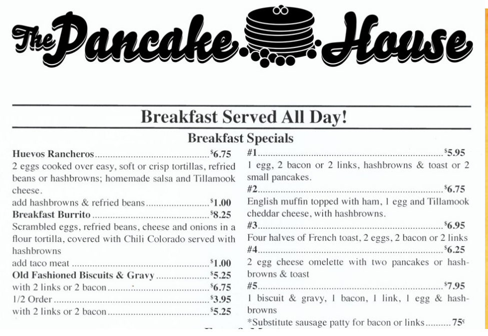
Motivational Example

Story: The merging of a Pancake House and a Diner

+

The Menus

MenuItems are similar: name, description and price



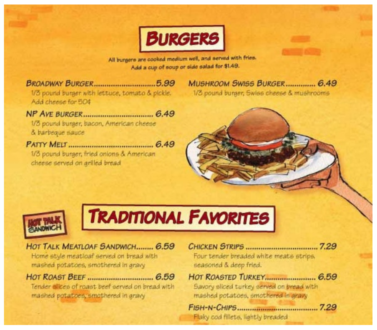
The Pancake House

Breakfast Served All Day!

Breakfast Specials

Huevos Rancheros $6.75
2 eggs cooked over easy, soft or crisp tortillas, refried beans or hashbrowns; homemade salsa and Tillamook cheese.
add hashbrowns & refried beans $1.00
Breakfast Burrito $8.25
Scrambled eggs, refried beans, cheese and onions in a flour tortilla, covered with Chili Colorado served with hashbrowns
add taco meat $1.00
Old Fashioned Biscuits & Gravy $5.25
with 2 links or 2 bacon $6.75
1/2 Order $3.95
with 2 links or 2 bacon $5.25

#1 .... $5.95
1 egg, 2 bacon or 2 links, hashbrowns & toast or 2 small pancakes.
#2 .... $6.75
English muffin topped with ham, 1 egg and Tillamook cheddar cheese, with hashbrowns.
#3 .... $6.95
Four halves of French toast, 2 eggs, 2 bacon or 2 links
#4 .... $6.25
2 egg cheese omelette with two pancakes or hash-browns & toast
#5 .... $7.95
1 biscuit & gravy, 1 bacon, 1 link, 1 egg & hash-browns
*Substitute sausage patty for bacon or links .... 75¢



BURGERS

All burgers are cooked medium well, and served with fries.
Add a cup of soup or side salad for $1.49.

BROADWAY BURGER 5.99
1/3 pound burger with lettuce, tomato & pickle. Add cheese for 50¢
NP AVE BURGER 6.49
1/3 pound burger, bacon, American cheese & barbeque sauce
PATTY MELT 6.49
1/3 pound burger, fried onions & American cheese served on grilled bread
MUSHROOM SWISS BURGER 6.49
1/3 pound burger, Swiss cheese & mushrooms

HOT TALK SANDWICH

TRADITIONAL FAVORITES

HOT TALK MEATLOAF SANDWICH 6.59
Home style meatloaf served on bread with mashed potatoes, smothered in gravy
HOT ROAST BEEF 6.59
Tender slices of roast beef served on bread with mashed potatoes, smothered in gravy
CHICKEN STRIPS 7.29
Four tender breaded white meats strips, seasoned & deep fried.
HOT ROASTED TURKEY 6.59
Savory sliced turkey served on bread with mashed potatoes, smothered in gravy
FISH-N-CHIPS 7.29
Flaky cod fillets, lightly breaded

The MenuItem Class

```
public class MenuItem {


      String name;
      String description;
      double price;

      public MenuItem(String name, String description, double price) {
              this.name = name;
              this.description = description;
              this.price = price;
      }

      public String getName() {
              return name;
      }

      public String getDescription( ){
              return description;
      }

      public double getPrice() {
              return price;
      }

      public void print() {
              System.out.println(name + ": " + description + " - " +
                              price);
      }
}
```

DinerMenu

```
public class DinerMenu {


      static final int MAX_ITEMS = 2;
      int numberOfItems = 0;
      MenuItem[] menuItems;

      public DinerMenu() {

              menuItems = new MenuItem[MAX_ITEMS];
              addItem("Hotdog", "The classic American hotdog !", 3.99);
              addItem("Soup of the day", "The soup of the day with a side of potato salad", 2.99);

      }
      public void addItem(String name, String description, double price) {

              MenuItem menuItem = new MenuItem(name, description, price);
              if (numberOfItems >= MAX_ITEMS) {
                      System.out.println("Can't add item! Menu is full!");
              }
              else {
                      menuItems[numberOfItems] = menuItem;
                      numberOfItems++;
              }
      }

      public MenuItem[] getMenuItems() {

              return menuItems;
      }

      // other menu methods
}
```

PancakeHouseMenu

```
public class PancakeHouseMenu {

     ArrayList menuItems;

     public PancakeHouseMenu() {

             menuItems = new ArrayList();
             addItem("Pancake Breakfast", "Pancakes with scrambled eggs, and toast", 2.99);
             addItem("Waffles", "With strawberry or blueberry", 3.99);

     }
     public void addItem(String name, String description, double price) {

             MenuItem menuItem = new MenuItem(name, description, price);
             menuItems.add(menuItem);
     }

     public ArrayList getMenuItems() {
             return menuItems;
     }

     // other menu methods
}
```

The Waitress Class

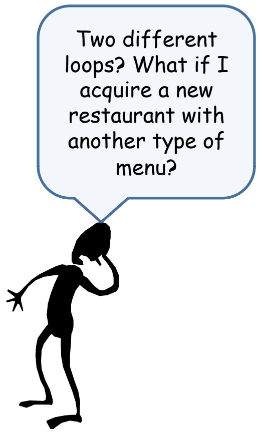
Prints a custom menu for customers on demand

```
public class Waitress {

      PancakeHouseMenu pancakeMenu;
      ArrayList breakfastItems;

      DinerMenu dinerMenu;
      MenuItem[] lunchItems;

      public Waitress() {

              pancakeMenu = new PancakeHouseMenu();
              breakfastItems = pancakeMenu.getMenuItems();

              dinerMenu = new DinerMenu();
              lunchItems = dinerMenu.getMenuItems();

      }

      public void printMenu() {

              // Print Breakfast Items
              System.out.println("Breakfast Menu:");
              for (int i=0; i<breakfastItems.size(); i++) {
                      MenuItem menuItem = (MenuItem) breakfastItems.get(i);
                      menuItem.print();
              }

              // Print Lunch Items
              System.out.println("\nLunch Menu:");
              for (int i=0; i<lunchItems.length; i++) {
                      MenuItem menuItem = lunchItems[i];
                      menuItem.print();
              }
      }
}
```

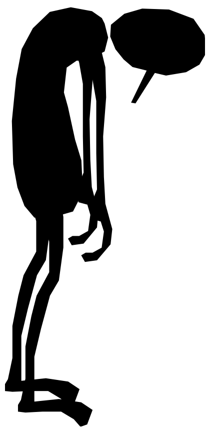
The Waitress Class

Prints a custom menu for customers on demand

```
public class Waitress {

      PancakeHouseMenu pancakeMenu;
      ArrayList breakfastItems;

      DinerMenu dinerMenu;
      MenuItem[] lunchItems;

      public Waitress() {

              pancakeMenu = new PancakeHouseMenu();
              breakfastItems = pancakeMenu.getMenuItems();

              dinerMenu = new DinerMenu();
              lunchItems = dinerMenu.getMenuItems();

      }

      public void printMenu() {

              // Print Breakfast Items
              System.out.println("Breakfast Menu:");
              for (int i=0; i<breakfastItems.size(); i++) {
                      MenuItem menuItem = (MenuItem) breakfastItems.get(i);
                      menuItem.print();
              }

              System.out.println("\nLunch Menu:");
              for (int i=0; i<lunchItems.length; i++) {
                      MenuItem menuItem = lunchItems[i];
                      menuItem.print();
              }
      }
```


Two different loops? What if I acquire a new restaurant with another type of menu?

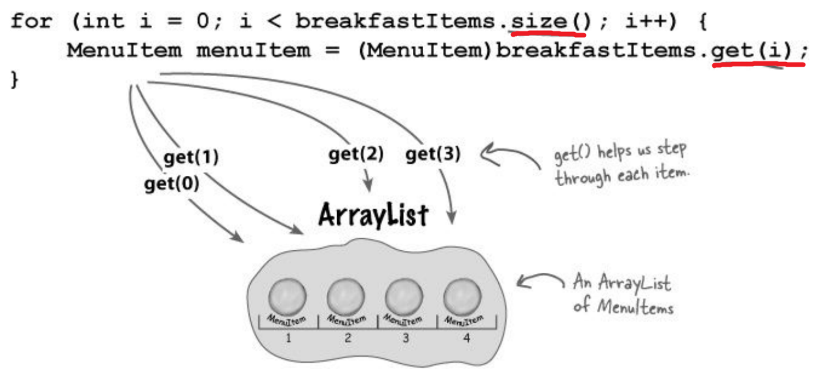
This doesn't look good…



Interface Example

```
public interface Measurable {

      public double getPerimeter();

      public double getArea();

}
```

Can We Encapsulate the Iteration?



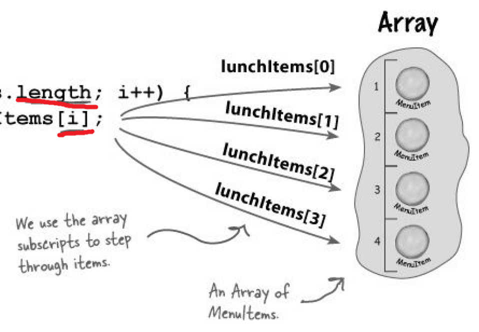
```
for (int i = 0; i < breakfastItems.size(); i++) {
    MenuItem menuItem = (MenuItem)breakfastItems.get(i);
}
```

get(1)
get(0)
get(2)
get(3)
get() helps us step through each item.
ArrayList
An ArrayList of MenuItems
MenuItem
1
2
3
4

Source: Head First Design Patterns – Eric and Elizabeth Freeman. O'Reilly 2004

Can We Encapsulate the Iteration?

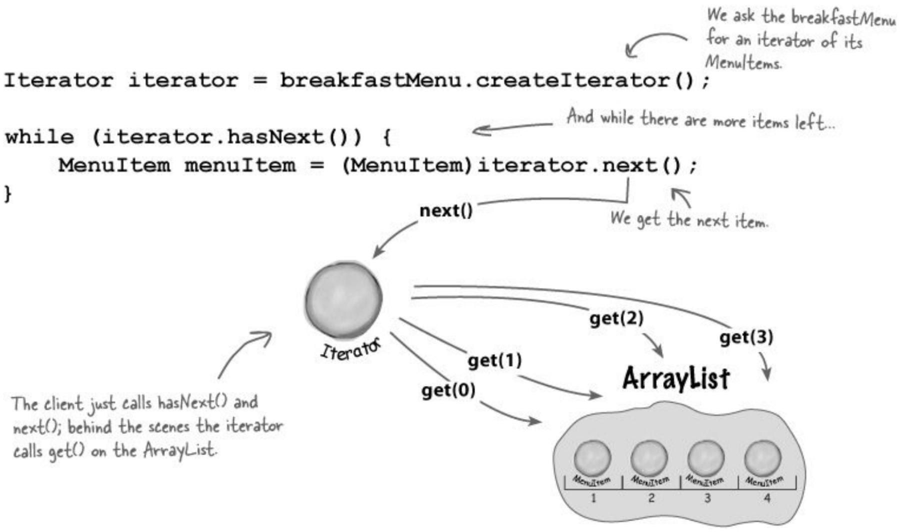
```
for (int i = 0; i < lunchItems.length; i++) {
    MenuItem menuItem = lunchItems[i];
}
```



Source: Head First Design Patterns – Eric and Elizabeth Freeman. O'Reilly 2004

An Iterator for an ArrayList

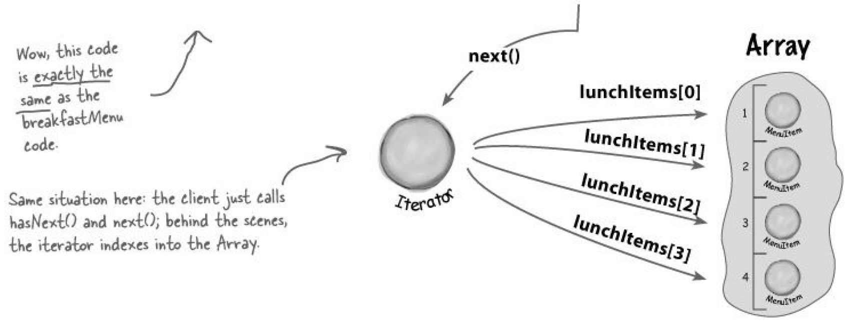
```
Iterator iterator = breakfastMenu.createIterator();

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


We ask the breakfastMenu for an iterator of its MenuItems.
And while there are more items left...
next()
We get the next item.
Iterator
get(0)
get(1)
get(2)
get(3)
ArrayList
MenuItem
MenuItem
MenuItem
MenuItem
1
2
3
4
The client just calls hasNext() and next(); behind the scenes the iterator calls get() on the ArrayList.

Source: Head First Design Patterns – Eric and Elizabeth Freeman. O'Reilly 2004

An Iterator for an Array

```
Iterator iterator = lunchMenu.createIterator();

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


Wow, this code is exactly the same as the breakfastMenu code.
Same situation here: the client just calls hasNext() and next(); behind the scenes, the iterator indexes into the Array.
next()
Iterator
lunchItems[0]
lunchItems[1]
lunchItems[2]
lunchItems[3]
Array
1
2
3
4
MenuItem

Source: Head First Design Patterns – Eric and Elizabeth Freeman. O'Reilly 2004

An Iterator Interface

```
public interface Iterator<E> {

      // Returns true if the iteration has more elements.
      public boolean hasNext();

      // Returns the next element in the iteration.
      public E next();

      // Removes from the underlying collection the last
      // element returned by this iterator (optional
      // operation).

      public void remove();
}
```

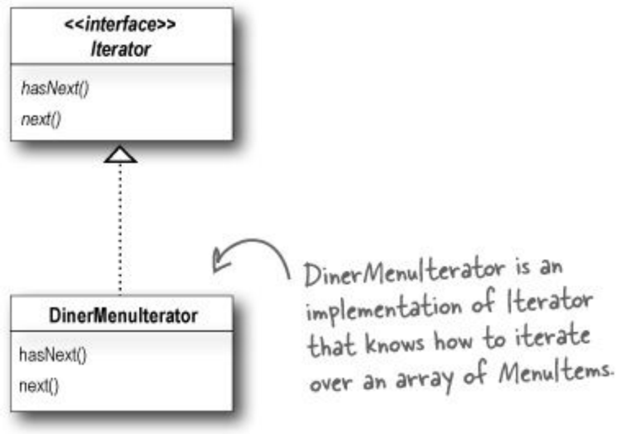
Generic Types

```
public class Box {

      private Object object;

      public void set (Object object){
             this.object = object;
      }

      public Object get() {
             return object;
      }
}
```

Generic Types

```
public class Box<T> {

      private T t;

      public void set (T t){
             this.t = t;
      }

      public T get(){
             return t;
      }
}
```

DinerMenuIterator


<<interface>>
Iterator
hasNext()
next()
DinerMenuIterator
hasNext()
next()
DinerMenuIterator is an implementation of Iterator that knows how to iterate over an array of MenuItems.

Source: Head First Design Patterns – Eric and Elizabeth Freeman. O'Reilly 2004

DinerMenuIterator

```
import java.util.*;

public class DinerMenuIterator implements Iterator<MenuItem> {

      MenuItem[] items;
      int position = 0;

      public DinerMenuIterator(MenuItem[] items) {
              this.items = items;
      }

      public MenuItem next() {

              MenuItem menuItem = items[position];
              position++;
              return menuItem;
      }

      public boolean hasNext() {

              if (position >= items.length || items[position] == null) {
                      return false;
              }
              else return true;
      }

      public void remove() {
              return; // does nothing...
      }
 }
```

Reworking DinerMenu

```
public class DinerMenu {

    static final int MAX_ITEMS = 2;
    int numberOfItems = 0;
    MenuItem[] menuItems;

    public DinerMenu() {

        menuItems = new MenuItem[MAX_ITEMS];
        addItem("Hotdog", "The classic American hotdog !", 3.99);
        addItem("Soup of the day", "The soup of the day with a side of potato salad", 2.99);
    }

    public void addItem(String name, String description, double price) {

        MenuItem menuItem = new MenuItem(name, description, price);
        if (numberOfItems >= MAX_ITEMS) {
            System.out.println("Can't add item! Menu is full!");
        }
        else {
            menuItems[numberOfItems] = menuItem;
            numberOfItems++;
        }
    }

    ~~public MenuItem[] getMenuItems() {~~

        ~~return menuItems;~~
    }

    public Iterator createIterator() {

        DinerMenuIterator iterator = new DinerMenuIterator(menuItems);
        return (iterator);
    }
}
```

Reworking PancakeHouseMenu

```
import java.util.*;

public class PancakeHouseMenu {

    ArrayList<MenuItem> menuItems;

    public PancakeHouseMenu() {

        menuItems = new ArrayList<MenuItem>();
        addItem("Pancake Breakfast", "Pancakes with scrambled eggs, and toast", 2.99);
        addItem("Waffles", "With strawberry or blueberry", 3.99);

    }

    public void addItem(String name, String description, double price) {

        MenuItem menuItem = new MenuItem(name, description, price);
        menuItems.add(menuItem);
    }

    ~~public ArrayList getMenuItems() {~~

        ~~return menuItems;~~
    }

    public Iterator createIterator() {

        return menuItems.iterator();
    }

}
```

Fixing the Waitress Code

```
public class Waitress {

      PancakeHouseMenu pancakeMenu;
      DinerMenu dinerMenu;

      public Waitress() {

              pancakeMenu = new PancakeHouseMenu();
              dinerMenu = new DinerMenu();

      }

      public void printMenu() {

              Iterator pancakeIterator = pancakeMenu.createIterator();
              Iterator dinerIterator = dinerMenu.createIterator();

              System.out.println("Breakfast Menu:");
              printItems(pancakeIterator);


              System.out.println("\nLunch Menu:");
              printItems(dinerIterator);
      }

      private void printItems(Iterator iterator) {

              while(iterator.hasNext()) {
                      MenuItem menuItem = (MenuItem) iterator.next();
                      menuItem.print();
              }
      }
}
```

```
public class ArrayList<E> implements Iterable<E> {

    private E[] array;
    private int capacity = 16; // initial length of array elts
    private int numElements = 0; // number of elements stored in the indexed list

    //contructors


     // Returns the element at the specified position in this list
     public  E get(int index) {
              checkIndex(index);
              return array[index];

     }
     // other methods...

     public Iterator<E> iterator() {
              return new ArrayListIterator();
     }


     class ArrayListIterator implements Iterator<E>{

              private int currentIndex = -1;

              public boolean hasNext() {

                      return (currentIndex < numElements-1);
              }

              public E next() {
                      currentIndex++;
                      return array[currentIndex];
              }

              public void remove() throws UnsupportedOperationException {
                      throw new UnsupportedOperationException("remove method not implemented");
              }
      }
}
```

References

- Kim Bruce and America Chamber's notes
- Head First Design Patterns – Eric and Elizabeth Freeman. O'Reilly 2004.
- Design Patterns – Elements of Reusable Object-Oriented Software. Erich Gamma *et al.* Addison-Wesley 1995.
- Data Structures and Algorithms in Java. Robert Lafore. 2nd Edition SAMS 2003.
- Java - An Introduction to Problem Solving and Programming. 6th Edition Walter Savitch Prentice Hall – Pearson 2012