

CS 112

Iterator

Iterator Design Pattern

Iterator Design Pattern

- The Iterator Pattern allows traversal *of the elements of an collection* without exposing the underlying implementation.

Iterator Design Pattern

- The Iterator Pattern allows traversal of the elements of an collection without exposing the underlying implementation.

Iterator Design Pattern

- The Iterator Pattern allows traversal of the elements of an collection without exposing the underlying implementation.
- Lets understand the Iterator design pattern by working through a design question.

Lets create a breakfast menu

Menu

<i>EGGS</i>		<i>OMELETES</i>	
Two any style	2.75	House	6.25
with bacon,ham,links	4.95	Spinach, tomatoes & feta cheese	
with Italian or kielbasa	5.95	Veggie	7.75
<i>served with homefries & toast</i>		Brosccoli, tomato, pepper, onion, mushroom, spinach	
DOUBLE EGG SANDWICHES		Western	7.00
white, wheat,rye,multigrain,hardroll,italian		Pappoz, onion, ham	
Two with cheese	3.25	Farmer's	7.75
with bacon, ham or links	4.75	Bacon, onion, pepper & potato	
Western	4.50	Hampton	7.75
Bagel or Englishtwo egg & cheese.....	3.50	Bacon, broccoli, tomato & cheese	
with bacon, ham or links	4.95	New Englander	8.50
Breakfast Club	7.25	Bacon, links, ham, onion, baked beans & cheddar	
<i>piled high triple decker with two eggs, , cheese</i>		Chicken Mexican	7.50
<i>tomato lettuce, , bacon, ham, mayo and homefries</i>		Chicken & cheese filled with a spicy tomato, pepper, onion sauce	
<i>add Italian or Kielbasa to any sandwich</i>	2.50	Triple Cheese	6.50
		American, swiss and provolone	
FROM THE GRIDDLE		Italian	8.25
biggest & fluffest buttermilk pancakes in the EAST		Itl. Sausage, pepper, mushroom, onion, olive, provolone	
Single	2.95	Hash & Cheese	7.50
Double	4.25	Philly Steak	7.75
Triple	5.25	Arizona	8.50
Hungry(4)	6.25	Bacon, chicken, hot peppers, tomato and cheddar	
Silver Dollars	3.50	Roasted Pepper Italian	8.50
		Ital. sausage, roasted pepper, onion, potato and provolone	
SPECIALITY CAKES		Healthy or white Omeletteadd extra	2.50
Create your own		Create your own Omelette Basic Omelette	4.25
Blueberries, Banana, Apples, Walnuts, Pecans			
Or Raisins any combination you choose		bacon	onion
Each item; each cake add		ham	green pepper
	0.95	ital. sausage	tomato
FRENCH TOAST		chicken	broccoli
Texas style.....Double	2.95	kielbasa	hot peppers
Triple	3.75	links	roasted peppers
Hungry (4)	4.50	cajun chicken	provolone
Italian Bread.....Double	3.25	potato	swiss
Triple	4.25	spinach	cheddar
Hungry (4)	4.95	olives	feta
Regular.....Double	2.50	baked beans	ricotta
Triple	3.25	mushroom	american
Hungry (4)	3.95		
TOPPINGS		Meat.....each item	1.25
Top any French Toast or Pancake		Veggie or Cheese	each item
Blueberries, strawberries or bananas		Benny Sauce.....(non-egg)	2.95
	1.95	Salsa.....	1.00
		All Omelets served with homefries & toast	

MenuItem

```
class MenuItem
```

```
{
```

```
}
```

```
class MenuItem
{
    private String name;
    private String description;
    private boolean isVegetarian;
    private double price;
}
```



```
class MenuItem
{
    private String name;
    private String description;
    private boolean isVegetarian;
    private double price;

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

MenuItem cont...

```
class MenuItem  
{
```

MenuItem cont...

```
class MenuItem
```

```
{
```

```
    •
```

```
    •
```

MenuItem cont...

```
class MenuItem
{
    .
    .
    public String getName()
    {
        return name;
    }
    public String getDescription()
    {
        return description;
    }
    public double getPrice()
    {
        return price;
    }
    public boolean isVegetarian()
    {
        return isVegetarian;
    }
}
```

Menu Interface

```
interface Menu
```

```
{
```

```
}
```

Menu Interface

```
interface Menu
{
    public void addItem(String name,
                        String description,
                        boolean veg,
                        double price);
}
```

```
public class BreakfastMenu .  
{
```

```
}
```

```
public class BreakfastMenu implements Menu  
{
```

```
}
```



```
public class BreakfastMenu implements Menu
{
```

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

    }
```

```
}
```

```
public class BreakfastMenu implements Menu
```

```
{  
    private List menuItems;
```

```
    public void addItem(String name, String description, boolean veg, double price)
```

```
{
```

```
}
```

```
}
```

```
public class BreakfastMenu implements Menu
{
    private List menuItems;
    public BreakfastMenu()
    {
        menuItems=new ArrayList();
        this.addItem("Pancakes!", "Pancakes with cream!!!",true,2.99);
        this.addItem("Bagels", "Bagels with butter!!!",true,2.99);
        this.addItem("Fresh Juice", "Orange Juice!!!",true,2.99);
        this.addItem("Fresh Tea", "Sweet Tea!!!",true,2.99);
    }

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

    }

}
```

```
public class BreakfastMenu implements Menu
{
    private List menuItems;
    public BreakfastMenu()
    {
        menuItems=new ArrayList();
        this.addItem("Pancakes!", "Pancakes with cream!!!",true,2.99);
        this.addItem("Bagels", "Bagels with butter!!!",true,2.99);
        this.addItem("Fresh Juice", "Orange Juice!!!",true,2.99);
        this.addItem("Fresh Tea", "Sweet Tea!!!",true,2.99);
    }

    public void addItem(String name, String description, boolean veg, double price)
    {
        MenuItem item=new MenuItem(name,description,veg,price);
        menuItems.add(item);
    }

}
```

```
public class BreakfastMenu implements Menu
{
    private List menuItems;
    public BreakfastMenu()
    {
        menuItems=new ArrayList();
        this.addItem("Pancakes!", "Pancakes with cream!!!",true,2.99);
        this.addItem("Bagels", "Bagels with butter!!!",true,2.99);
        this.addItem("Fresh Juice", "Orange Juice!!!",true,2.99);
        this.addItem("Fresh Tea", "Sweet Tea!!!",true,2.99);
    }


    public void addItem(String name, String description, boolean veg, double price)
    {
        MenuItem item=new MenuItem(name,description,veg,price);
        menuItems.add(item);
    }

    public List getMenuItems()
    {
        return menuItems;
    }
}
```

Lets create a dinner menu

Menu

MenuItem

17/12/11 *Wai Yee*  *Wi Siong*

良辰美景喜临门
Four Hot and Cold Combination Platter
干贝竹笙土鸡汤
Double-Boiled Village Chicken Soup
出地野野响

Roasted Pi-Pah Duck
煎猫耳松姑蒸龙足仔

Steamed Estuary Grouper with Black Fungus
金丝虾卷拼沙津草虾片

Crystal Prawn and Tiger Prawn in Two Varieties Style
杏片发财开心炒素果

Stir Fried Vegetable with "Fatt Choy"&Slices Almond
粽叶珍珠糯米包

H.K Glutinous Rice In Bamboo Leaf per-serving
冰糖蔗汁马蹄條

Cold Cut Waten Chessnut Stick
冰心玉露海燕丝

Sweetened Cold Longan with Sea Coconut&Sea weed

```
public class DinnerMenu  
{
```

```
}
```

```
public class DinnerMenu implements Menu  
{
```

```
}
```



```
public class DinnerMenu implements Menu  
{
```

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

```
    }
```

```
}
```

```
public class DinnerMenu implements Menu
{
    private static final int NUMBER_OF_ITEMS=6;
    int counter=0;
```

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

```
}
```

```
}
```

```
public class DinnerMenu implements Menu
{
    private static final int NUMBER_OF_ITEMS=6;
    int counter=0;
    private MenuItem[] dinnerItems;
    public DinnerMenu()
    {
        dinnerItems=new MenuItem[NUMBER_OF_ITEMS];
        this.addItem("Shawarma", "Chicken!!",true,2.99);
        this.addItem("Biryani", "Chicken!!",true,2.99);
        this.addItem("Curry", "ButterChicken",true,2.99);
        this.addItem("Naan", "Butter Naan",true,2.99);
    }

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

    }

}
```

```
public class DinnerMenu implements Menu
{
    private static final int NUMBER_OF_ITEMS=6;
    int counter=0;
    private MenuItem[] dinnerItems;
    public DinnerMenu()
    {
        dinnerItems=new MenuItem[NUMBER_OF_ITEMS];
        this.addItem("Shawarma", "Chicken!!",true,2.99);
        this.addItem("Biryani", "Chicken!!",true,2.99);
        this.addItem("Curry", "ButterChicken",true,2.99);
        this.addItem("Naan", "Butter Naan",true,2.99);
    }

    public void addItem(String name, String description, boolean veg, double price)
    {
        MenuItem item=new MenuItem(name,description,veg,price);
        if (counter>=NUMBER_OF_ITEMS)
        {
            System.err.println("Error, array is full");
        }
        else
        {
            dinnerItems[counter]=item;
            counter++;
        }
    }
}
```

```
public class DinnerMenu implements Menu
{
    private static final int NUMBER_OF_ITEMS=6;
    int counter=0;
    private MenuItem[] dinnerItems;
    public DinnerMenu()
    {
        dinnerItems=new MenuItem[NUMBER_OF_ITEMS];
        this.addItem("Shawarma", "Chicken!!",true,2.99);
        this.addItem("Biryani", "Chicken!!",true,2.99);
        this.addItem("Curry", "ButterChicken",true,2.99);
        this.addItem("Naan", "Butter Naan",true,2.99);
    }

    public void addItem(String name, String description, boolean veg, double price)
    {
        MenuItem item=new MenuItem(name,description,veg,price);
        if (counter>=NUMBER_OF_ITEMS)
        {
            System.err.println("Error, array is full");
        }
        else
        {
            dinnerItems[counter]=item;
            counter++;
        }
    }

    public MenuItem[] getMenuItems()
    {
        return dinnerItems;
    }
}
```

Problem:

- Ok, so now you are asked to print out the two menus on the console.

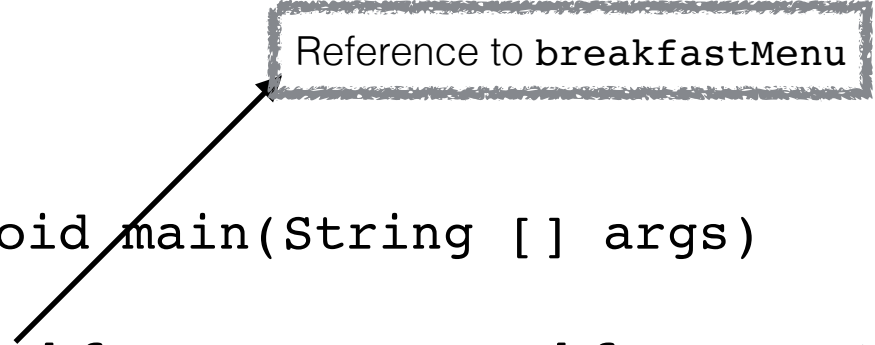
```
class PrintMenus
{
    public static void main(String [] args)
    {

    }
}
```

```
class PrintMenus
{
    public static void main(String [] args)
    {
        Menu breakfastM= new BreakfastMenu();
        ArrayList breakfastItems=((BreakfastMenu)breakfastM).getMenuItems();
    }
}
```



```
class PrintMenus
{
    public static void main(String [] args)
    {
        Menu breakfastM= new BreakfastMenu();
        ArrayList breakfastItems=((BreakfastMenu)breakfastM).getMenuItems();
    }
}
```



Reference to breakfastMenu

```
class PrintMenus
{
    public static void main(String [] args)
    {
        Menu breakfastM= new BreakfastMenu();
        ArrayList breakfastItems=((BreakfastMenu)breakfastM).getMenuItems();
    }
}
```

Reference to breakfastMenu

downcasting from Menu to BreakfastMenu. Why?

```
class PrintMenus
{
    public static void main(String [] args)
    {
        Menu breakfastM= new BreakfastMenu();
        ArrayList breakfastItems=((BreakfastMenu)breakfastM).getMenuItems();

        Menu dinnerM= new DinnerMenu();
        MenuItem[] dinnerItems=((DinnerMenu)dinnerM).getMenuItems();
        .
        .
        .
    }
}
```

Reference to breakfastMenu

downcasting from Menu to BreakfastMenu. Why?

Questions...

Questions...

- Are the two menus implemented differently?

Questions...

- Are the two menus implemented differently?

Questions...

- Are the two menus implemented differently?
- What kind of data structure did we use to implement the `breakfastMenu`?

Questions...

- Are the two menus implemented differently?
 - What kind of data structure did we use to implement the `breakfastMenu`?
 - What kind of data structure did we use to implement the `dinnerMenu`?


```
class PrintMenus
{
    public static void main(String [] args)
    {
        Menu breakfastM= new BreakfastMenu();
        ArrayList breakfastItems=((BreakfastMenu)breakfastM).getMenuItems();

    }
}
```

```
class PrintMenus
{
    public static void main(String [] args)
    {
        Menu breakfastM= new BreakfastMenu();
        ArrayList breakfastItems=((BreakfastMenu)breakfastM).getMenuItems();


        for (int i=0;i<breakfastItems.size();i++)
        {
            MenuItem menuItem=breakfastItems.get(i);
            System.out.println(menuItem.getName());
            System.out.println(menuItem.getPrice());
            System.out.println(menuItem.getDescription());
        }

        .
        .
        .
    }
}
```

```
class PrintMenus
{
    public static void main(String [] args)
    {
        Menu breakfastM= new BreakfastMenu();
        ArrayList breakfastItems=((BreakfastMenu)breakfastM).getMenuItems();

        for (int i=0;i<breakfastItems.size();i++)
        {
            MenuItem menuItem=breakfastItems.get(i);
            System.out.println(menuItem.getName());
            System.out.println(menuItem.getPrice());
            System.out.println(menuItem.getDescription());
        }

        .
        .
        .
    }
}
```



look @ JavaDoc of list for size and get

```
class PrintMenus
{
    public static void main(String [] args)
    {
        Menu dinnerM= new DinnerMenu();
        MenuItem[] dinnerItems=((DinnerMenu)dinnerM).getMenuItems();

    }
}
```

```
class PrintMenus
{
    public static void main(String [] args)
    {
        Menu dinnerM= new DinnerMenu();
        MenuItem[] dinnerItems=((DinnerMenu)dinnerM).getMenuItems();

        for (int i=0;i<dinnerItems.length;i++)
        {
            MenuItem menuItem=dinnerItems[i];
            System.out.println(menuItem.getName());
            System.out.println(menuItem.getPrice());
            System.out.println(menuItem.getDescription());
        }

        .
        .
        .
    }
}
```

```
class PrintMenus
{
    public static void main(String [] args)
    {
        Menu dinnerM= new DinnerMenu();
        MenuItem[] dinnerItems=((DinnerMenu)dinnerM).getMenuItems();

        for (int i=0;i<dinnerItems.length;i++)
        {
            MenuItem menuItem=dinnerItems[i];
            System.out.println(menuItem.getName());
            System.out.println(menuItem.getPrice());
            System.out.println(menuItem.getDescription());
        }

        .
        .
        .
    }
}
```

The diagram consists of a rectangular box containing the text "look @ JavaDoc of array for length and []". Two arrows originate from this box: one points to the ".length" property access in the for loop condition, and the other points to the "[i]" index access in the array assignment statement.

What are some issues here?

What are some issues here?

- Is the code that is printing the menus required to know the inner details of the `BreakfastMenu` and `DinnerMenu`?

What are some issues here?

- Is the code that is printing the menus required to know the inner details of the `BreakfastMenu` and `DinnerMenu`?
 - Encapsulation?

What are some issues here?

- Is the code that is printing the menus required to know the inner details of the `BreakfastMenu` and `DinnerMenu`?
 - Encapsulation?
 - What was the whole point of Encapsulation?

What are some issues here?

- Is the code that is printing the menus required to know the inner details of the `BreakfastMenu` and `DinnerMenu`?
 - Encapsulation?
 - What was the whole point of Encapsulation?

What are some issues here?

- Is the code that is printing the menus required to know the inner details of the `BreakfastMenu` and `DinnerMenu`?
 - Encapsulation?
 - What was the whole point of Encapsulation?
- Do we really care or are concerned about the inner collection used in the `Menu` when printing `MenuItem`?

What are some issues here?

- We were earlier printing the menu to the console. But assume now, that we are also:
 - printing the menu to a file.
 - printing the menu to a web page.
 - printing the menu and sending an email.
- Now, if `breakfastMenu` and `dinnerMenu` decide to use a different collection (`tree`, `set`, `linked list`, etc etc) does that mean the following code must be modified?
 - the code for printing to a file.
 - the code for printing to a web page
 - the code for printing the menu and sending an email
 - the code for printing to the console

Lets use Iterator Design Pattern

```
interface Menu
```

```
{
```

```
    public void addItem(String name, String description, boolean  
    veg, double price);
```

```
}
```

Lets use Iterator Design Pattern

```
interface Menu  
  
{  
  
    public void addItem(String name, String description, boolean  
veg, double price);  
  
    public Iterator createIterator();  
  
}
```

What is an Iterator in Java JDK?

- An `Iterator` is an interface in the Java JDK. The interface contains the following three abstract methods:

Method Summary	
boolean	<code>hasNext()</code> Returns <code>true</code> if the iteration has more elements.
<u>E</u>	<code>next()</code> Returns the next element in the iteration.
void	<code>remove()</code> Removes from the underlying collection the last element returned by the iterator (optional operation).


```
class PrintMenus
{
    public static void main(String [] args)
    {
        Menu dinnerM= new DinnerMenu();
    }
}
```

```
class PrintMenus
{
    public static void main(String [] args)
    {
        Menu dinnerM= new DinnerMenu();
        Iterator iterator=dinnerM.createIterator();

    }
}
```

```
class PrintMenus
{
    public static void main(String [] args)
    {
        Menu dinnerM= new DinnerMenu();
        Iterator iterator=dinnerM.createIterator();

        while (iterator.hasNext())
        {

            MenuItem item = (MenuItem)iterator.next();
            System.out.println(menuItem.getName());
            System.out.println(menuItem.getPrice());
            System.out.println(menuItem.getDescription());
        }

        .
        .
        .

    }
}
```

```

class PrintMenus
{
    public static void main(String [] args)
    {
        Menu dinnerM= new DinnerMenu();
        Iterator iterator=dinnerM.createIterator();

        while (iterator.hasNext())
        {
            MenuItem item = (MenuItem)iterator.next();
            System.out.println(menuItem.getName());
            System.out.println(menuItem.getPrice());
            System.out.println(menuItem.getDescription());
        }
        .
        .
        .
    }
}

```

Iterator

Method Summary	
boolean	hasNext() Returns true if the iteration has more elements.
MenuItem	next() Returns the next element in the iteration.
void	remove() Removes from the underlying collection the last element returned by the iterator (optional operation).

```
class PrintMenus
{
    public static void main(String [] args)
    {
        Menu breakfastM= new BreakfastMenu();
    }
}
```

```
class PrintMenus
{
    public static void main(String [] args)
    {
        Menu breakfastM= new BreakfastMenu();
        Iterator iterator=breakfastM.createIterator();

    }
}
```

```
class PrintMenus
{
    public static void main(String [] args)
    {
        Menu breakfastM= new BreakfastMenu();
        Iterator iterator=breakfastM.createIterator();

        while (iterator.hasNext())
        {

            MenuItem item = (MenuItem)iterator.next();
            System.out.println(menuItem.getName());
            System.out.println(menuItem.getPrice());
            System.out.println(menuItem.getDescription());
        }

        .
        .
        .

    }
}
```

```

class PrintMenus
{
    public static void main(String [] args)
    {
        Menu breakfastM= new BreakfastMenu();
        Iterator iterator=breakfastM.createIterator();

        while (iterator.hasNext())
        {
            MenuItem item = (MenuItem)iterator.next();
            System.out.println(menuItem.getName());
            System.out.println(menuItem.getPrice());
            System.out.println(menuItem.getDescription());
        }
        .
        .
        .
    }
}

```

Iterator

Method Summary	
boolean	hasNext() Returns true if the iteration has more elements.
MenuItem	next() Returns the next element in the iteration.
void	remove() Removes from the underlying collection the last element returned by the iterator (optional operation).


```
public class DinnerMenu implements Menu
{
    private static final int NUMBER_OF_ITEMS=6;
    int counter=0;
    private MenuItem[] dinnerItems;
    public DinnerMenu()
    {
        dinnerItems=new MenuItem[NUMBER_OF_ITEMS];
        this.addItem("Shawarma", "Chicken!!",true,2.99);
        this.addItem("Biryani", "Chicken!!",true,2.99);
        this.addItem("Curry", "ButterChicken",true,2.99);
        this.addItem("Naan", "Butter Naan",true,2.99);
    }

    public void addItem(String name, String description, boolean veg, double price)
    {
        MenuItem item=new MenuItem(name,description,veg,price);
        if (counter>=NUMBER_OF_ITEMS)
        {
            System.err.println("Error, array is full");
        }
        else
        {
            dinnerItems[counter]=item;
            counter++;
        }
    }

    public Iterator createIterator()
    {
    }
}
}
```

```

public class DinnerMenu implements Menu
{
    private static final int NUMBER_OF_ITEMS=6;
    int counter=0;
    private MenuItem[] dinnerItems;
    public DinnerMenu()
    {
        dinnerItems=new MenuItem[NUMBER_OF_ITEMS];
        this.addItem("Shawarma", "Chicken!!",true,2.99);
        this.addItem("Biryani", "Chicken!!",true,2.99);
        this.addItem("Curry", "ButterChicken",true,2.99);
        this.addItem("Naan", "Butter Naan",true,2.99);
    }

    public void addItem(String name, String description, boolean veg, double price)
    {
        MenuItem item=new MenuItem(name,description,veg,price);
        if (counter>=NUMBER_OF_ITEMS)
        {
            System.err.println("Error, array is full");
        }
        else
        {
            dinnerItems[counter]=item;
            counter++;
        }
    }

    public Iterator createIterator()
    {
        ???
    }
}

```

```
class DinnerMenuIterator
```

```
{
```

```
}
```

```
class DinnerMenuIterator implements Iterator
```

```
{
```

```
}
```

```
class DinnerMenuIterator implements Iterator
{

    public Object next()
    {

    }
}
```

```
class DinnerMenuIterator implements Iterator
{

    public Object next()
    {

    }

}
```

```
class DinnerMenuIterator implements Iterator
{

    public Object next()
    {

    }

    public boolean hasNext()
    {

    }

}
```

```
class DinnerMenuIterator implements Iterator
{
    private MenuItem[] items;
    int counter;

    public Object next()
    {

    }

    public boolean hasNext()
    {

    }
}
```



```
class DinnerMenuIterator implements Iterator
{
    private MenuItem[] items;
    int counter;

    public DinnerMenuIterator(MenuItem[] items)
    {
        this.items=items;
        counter=0;
    }

    public Object next()
    {

    }

    public boolean hasNext()
    {

    }

}
```

```
class DinnerMenuIterator implements Iterator
{
    private MenuItem[] items;
    int counter;

    public DinnerMenuIterator(MenuItem[] items)
    {
        this.items=items;
        counter=0;
    }

    public Object next()
    {
        MenuItem item=items[counter];
        counter++;
        return item;
    }

    public boolean hasNext()
    {

```

```
class DinnerMenuIterator implements Iterator
{
    private MenuItem[] items;
    int counter;

    public DinnerMenuIterator(MenuItem[] items)
    {
        this.items=items;
        counter=0;
    }

    public Object next()
    {
        MenuItem item=items[counter];
        counter++;
        return item;
    }

    public boolean hasNext()
    {
        if (counter>=items.length || items[counter]==null)
        {
            return false;
        }
        else
        {
            return true;
        }
    }
}
```

Revist DinnerMenu Class

```
public class DinnerMenu implements Menu
{
    private static final int NUMBER_OF_ITEMS=6;
    int counter=0;
    private MenuItem[] dinnerItems;
    public DinnerMenu()
    {
        dinnerItems=new MenuItem[NUMBER_OF_ITEMS];
        this.addItem("Shawarma", "Chicken!!",true,2.99);
        this.addItem("Biryani", "Chicken!!",true,2.99);
        this.addItem("Curry", "ButterChicken",true,2.99);
        this.addItem("Naan", "Butter Naan",true,2.99);
    }

    public void addItem(String name, String description, boolean veg, double price)
    {
        MenuItem item=new MenuItem(name,description,veg,price);
        if (counter>=NUMBER_OF_ITEMS)
        {
            System.err.println("Error, array is full");
        }
        else
        {
            dinnerItems[counter]=item;
            counter++;
        }
    }

    public Iterator createIterator()
    {
    }
}
}
```

Revist DinnerMenu Class

```
public class DinnerMenu implements Menu
{
    private static final int NUMBER_OF_ITEMS=6;
    int counter=0;
    private MenuItem[] dinnerItems;
    public DinnerMenu()
    {
        dinnerItems=new MenuItem[NUMBER_OF_ITEMS];
        this.addItem("Shawarma", "Chicken!!",true,2.99);
        this.addItem("Biryani", "Chicken!!",true,2.99);
        this.addItem("Curry", "ButterChicken",true,2.99);
        this.addItem("Naan", "Butter Naan",true,2.99);
    }

    public void addItem(String name, String description, boolean veg, double price)
    {
        MenuItem item=new MenuItem(name,description,veg,price);
        if (counter>=NUMBER_OF_ITEMS)
        {
            System.err.println("Error, array is full");
        }
        else
        {
            dinnerItems[counter]=item;
            counter++;
        }
    }

    public Iterator createIterator()
    {
        return new DinnerMenuIterator(menuItems);
    }
}
```

Ok.. how do we fix our
`breakfastMenu?`

Ok.. how do we fix our breakfastMenu?

- Our `BreakfastMenu` uses an `ArrayList` as a collection to hold the `MenuItems`.

Ok.. how do we fix our **breakfastMenu?**

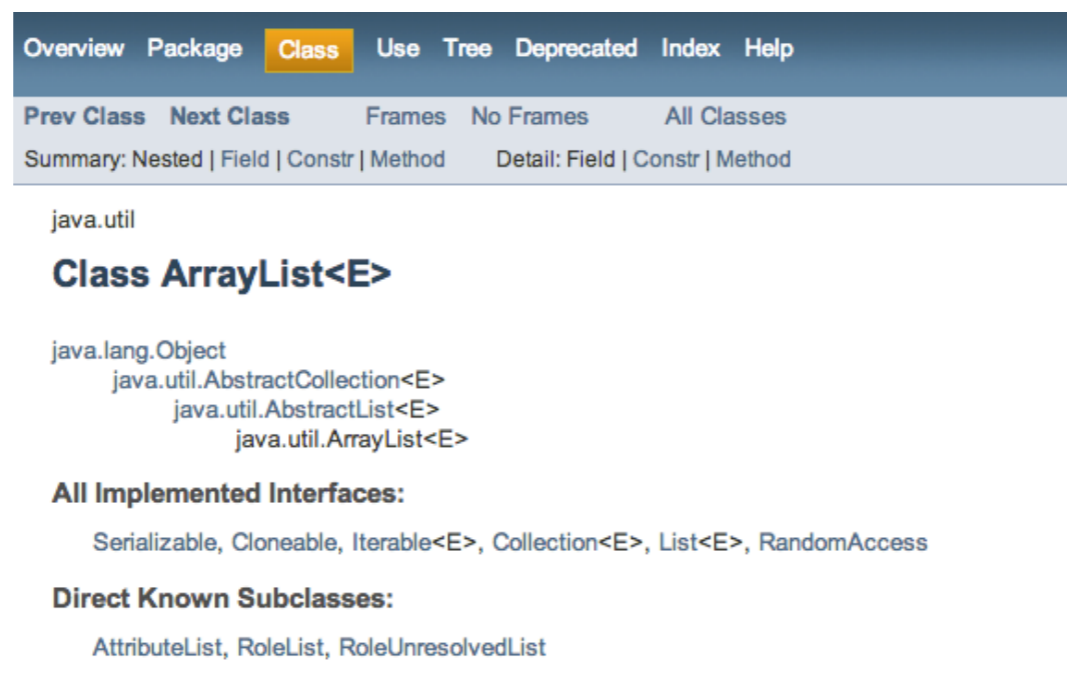
- Our `BreakfastMenu` uses an `ArrayList` as a collection to hold the `MenuItems`.

Ok.. how do we fix our `breakfastMenu`?

- Our `BreakfastMenu` uses an `ArrayList` as a collection to hold the `MenuItems`.
- Here is a snapshot of the JavaDoc for an `ArrayList`:

Ok.. how do we fix our breakfastMenu?

- Our BreakfastMenu uses an ArrayList as a collection to hold the MenuItems.
- Here is a snapshot of the JavaDoc for an ArrayList:



The screenshot shows the JavaDoc for the ArrayList class. At the top, there is a navigation bar with tabs for Overview, Package, Class (selected), Use, Tree, Deprecated, Index, and Help. Below this is another navigation bar with links for Prev Class, Next Class, Frames, No Frames, and All Classes. A summary bar indicates the current view is 'Summary: Nested | Field | Constr | Method' and provides links for 'Detail: Field | Constr | Method'. The main content area shows the package 'java.util', the class name 'Class ArrayList<E>', and a class hierarchy diagram showing 'java.lang.Object' as the superclass, with 'java.util.AbstractCollection<E>' and 'java.util.ArrayList<E>' as subclasses. Below the hierarchy, it lists 'All Implemented Interfaces: Serializable, Cloneable, Iterable<E>, Collection<E>, List<E>, RandomAccess' and 'Direct Known Subclasses: AttributeList, RoleList, RoleUnresolvedList'.

- `ArrayList` IS A `List`.
- The `List` has the following abstract instance method:

iterator

```
public Iterator<E> iterator()
```

Returns an iterator over the elements in this list in proper sequence.

- This implies that the `ArrayList` has implemented this method from the `List` interface.

```
public class BreakfastMenu implements Menu
{
    private List menuItems;
    public BreakfastMenu()
    {
        menuItems=new ArrayList();
        this.addItem("Pancakes!", "Pancakes with cream!!!",true,2.99);
        this.addItem("Bagels", "Bagels with butter!!!",true,2.99);
        this.addItem("Fresh Juice", "Orange Juice!!!",true,2.99);
        this.addItem("Fresh Tea", "Sweet Tea!!!",true,2.99);
    }

    public void addItem(String name, String description, boolean veg, double price)
    {
        MenuItem item=new MenuItem(name,description,veg,price);
        menuItems.add(item);
    }

    public List getMenuItems()
    {
        return menuItems;
    }

    public Iterator createIterator()
    {
    }
}
}
```

```
public class BreakfastMenu implements Menu
{
    private List menuItems;
    public BreakfastMenu()
    {
        menuItems=new ArrayList();
        this.addItem("Pancakes!", "Pancakes with cream!!!",true,2.99);
        this.addItem("Bagels", "Bagels with butter!!!",true,2.99);
        this.addItem("Fresh Juice", "Orange Juice!!!",true,2.99);
        this.addItem("Fresh Tea", "Sweet Tea!!!",true,2.99);
    }

    public void addItem(String name, String description, boolean veg, double price)
    {
        MenuItem item=new MenuItem(name,description,veg,price);
        menuItems.add(item);
    }

    public List getMenuItems()
    {
        return menuItems;
    }

    public Iterator createIterator()
    {
        return menuItems.iterator();
    }
}
```

```
class PrintMenus
{
    public static void main(String [] args)
    {

    }
}
```

i

}

```
class PrintMenus
{
    public static void main(String [] args)
    {
        Menu dinnerM= new DinnerMenu();
        Menu breakfastM = new BreakfastMenu();
        Iterator iterator=dinnerM.createIterator();
        Iterator iterator1=breakfastM.createIterator();
        print(iterator);
        print(iterator1);
    }
}
```

;

}

```
class PrintMenus
{
    public static void main(String [] args)
    {
        Menu dinnerM= new DinnerMenu();
        Menu breakfastM = new BreakfastMenu();
        Iterator iterator=dinnerM.createIterator();
        Iterator iterator1=breakfastM.createIterator();
        print(iterator);
        print(iterator1);
    }
    public static void print(Iterator itr)
    {
        while (itr.hasNext())
        {
            MenuItem item = (MenuItem)iterator.next();
            System.out.println(menuItem.getName());
            System.out.println(menuItem.getPrice());
            System.out.println(menuItem.getDescription());
        }
    }
}
```


Lets ask the same questions now that we have used the Iterator Design pattern.

Lets ask the same questions now that we have used the Iterator Design pattern.

- We were earlier printing the menu to the console. But assume now, that we are also:

Lets ask the same questions now that we have used the Iterator Design pattern.

- We were earlier printing the menu to the console. But assume now, that we are also:
 - printing the menu to a file.

Lets ask the same questions now that we have used the Iterator Design pattern.

- We were earlier printing the menu to the console. But assume now, that we are also:
 - printing the menu to a file.
 - printing the menu to a web page.

Lets ask the same questions now that we have used the Iterator Design pattern.

- We were earlier printing the menu to the console. But assume now, that we are also:
 - printing the menu to a file.
 - printing the menu to a web page.
 - printing the menu and sending an email.

Lets ask the same questions now that we have used the Iterator Design pattern.

- We were earlier printing the menu to the console. But assume now, that we are also:
 - printing the menu to a file.
 - printing the menu to a web page.
 - printing the menu and sending an email.

Lets ask the same questions now that we have used the Iterator Design pattern.

- We were earlier printing the menu to the console. But assume now, that we are also:
 - printing the menu to a file.
 - printing the menu to a web page.
 - printing the menu and sending an email.
- Now, if `breakfastMenu` and `dinnerMenu` decide to use a different collection (`tree`, `set`, `linked list`, etc etc) does that mean the following code must be modified?

Lets ask the same questions now that we have used the Iterator Design pattern.

- We were earlier printing the menu to the console. But assume now, that we are also:
 - printing the menu to a file.
 - printing the menu to a web page.
 - printing the menu and sending an email.
- Now, if `breakfastMenu` and `dinnerMenu` decide to use a different collection (`tree`, `set`, `linked list`, etc etc) does that mean the following code must be modified?
 - the code for printing to a file.

Lets ask the same questions now that we have used the Iterator Design pattern.

- We were earlier printing the menu to the console. But assume now, that we are also:
 - printing the menu to a file.
 - printing the menu to a web page.
 - printing the menu and sending an email.
- Now, if `breakfastMenu` and `dinnerMenu` decide to use a different collection (`tree`, `set`, `linked list`, etc etc) does that mean the following code must be modified?
 - the code for printing to a file.
 - the code for printing to a web page

Lets ask the same questions now that we have used the Iterator Design pattern.

- We were earlier printing the menu to the console. But assume now, that we are also:
 - printing the menu to a file.
 - printing the menu to a web page.
 - printing the menu and sending an email.
- Now, if `breakfastMenu` and `dinnerMenu` decide to use a different collection (`tree`, `set`, `linked list`, etc etc) does that mean the following code must be modified?
 - the code for printing to a file.
 - the code for printing to a web page
 - the code for printing the menu and sending an email

Lets ask the same questions now that we have used the Iterator Design pattern.

- We were earlier printing the menu to the console. But assume now, that we are also:
 - printing the menu to a file.
 - printing the menu to a web page.
 - printing the menu and sending an email.
- Now, if `breakfastMenu` and `dinnerMenu` decide to use a different collection (`tree`, `set`, `linked list`, etc etc) does that mean the following code must be modified?
 - the code for printing to a file.
 - the code for printing to a web page
 - the code for printing the menu and sending an email
 - the code for printing to the console