

More on Abstraction in Java

More on Interfaces

Produced Mairead Meagher
by: Dr. Siobhán Drohan
 Eamonn de Leastar



Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Department of Computing and Mathematics
<http://www.wit.ie/>

Topic List

- Interfaces define Types.
- An Interface Example – Network V6.
- Naming Conventions for Interfaces.

Interfaces Define Types

- Interfaces define Types
 - They define common behaviour i.e. methods
 - Can be used to promote design to a higher level of abstraction
 - Can be used where multiple implementations of one abstraction are envisaged
 - Implementing classes are subtypes of the interface type.
- Classes can implement one or more Interfaces as appropriate i.e. have more than one type.

Interfaces Impose Types

- If a variable is declared as an Interface type:

`IMammal dog;`

- Then any instance of any class that implements that Interface can be assigned to that variable.

`IMammal dog = new Mammal();`

```
interface IMammal
{
    void eat();
    void travel();
}
```

```
public class Mammal implements IMammal{
    public void eat(){
        System.out.println("Mammal eats");
    }

    public void travel(){
        System.out.println("Mammal travels");
    }
}
```

Reference Data Type

- When you define a new interface, you are defining a new **Reference Data Type**.
- You can use interface names anywhere you can use any other data type name.
- If you define a **Reference** variable whose type is an interface e.g.
IMammal dog;
- any object you assign to it must be an instance of a class that implements the interface e.g.

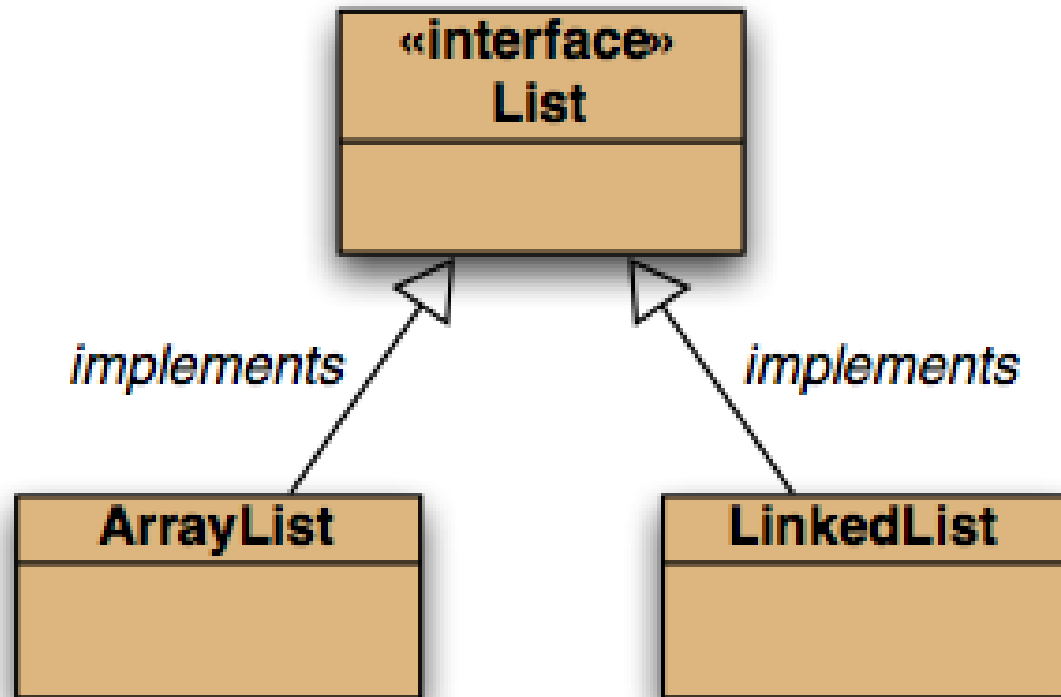
dog = new Mammal();

```
interface IMammal
{
    void eat();
    void travel();
}
```

```
public class Mammal implements IMammal{
    public void eat(){
        System.out.println("Mammal eats");
    }

    public void travel(){
        System.out.println("Mammal travels");
    }
}
```

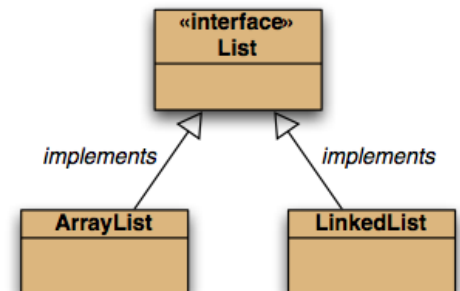
Interfaces in Collections Framework



Interfaces in Collections Framework

- ArrayList implements the List interface.
- Recall this rule:
 - If you define a reference variable whose type is an interface, any object you assign to it must be an instance of a class that implements the interface.
- Applying this rule to a List:

`List<Product> products = new ArrayList<Product>();`



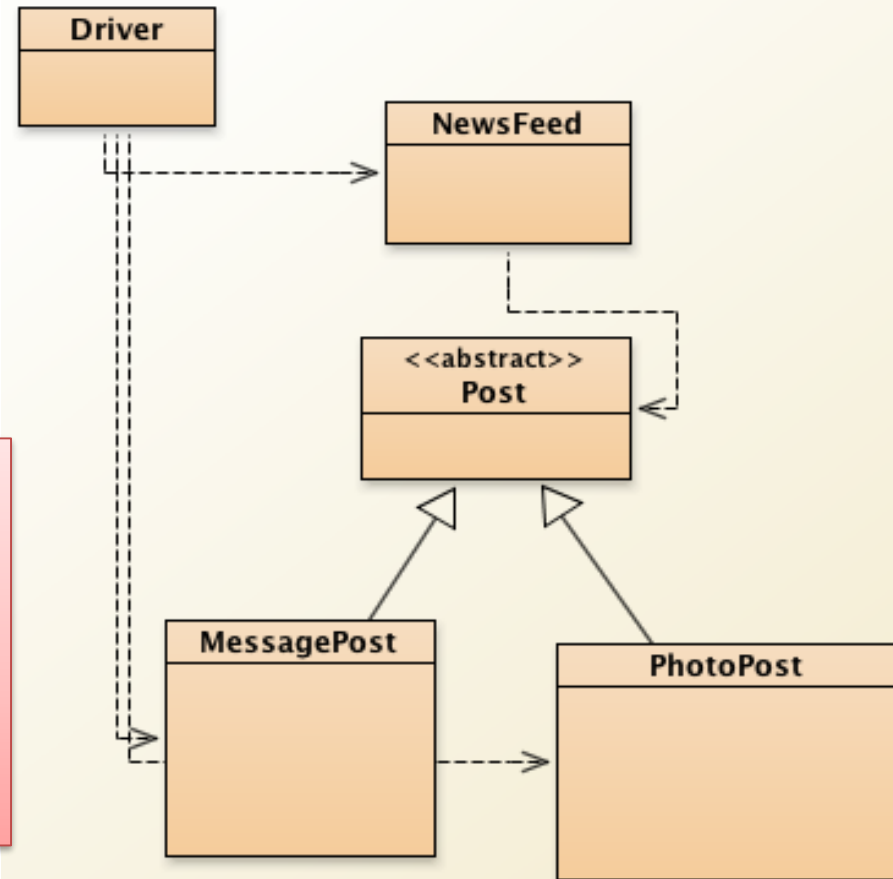
Recall Network-V5

Options

Posts

- 1) Add a Text Post
 - 2) Add a Photo Post
 - 3) List all Posts
 - 0) Exit
- ==>>

We will now refactor our system so that ArrayLists are defined using interfaces instead of concrete classes.



Network-V6

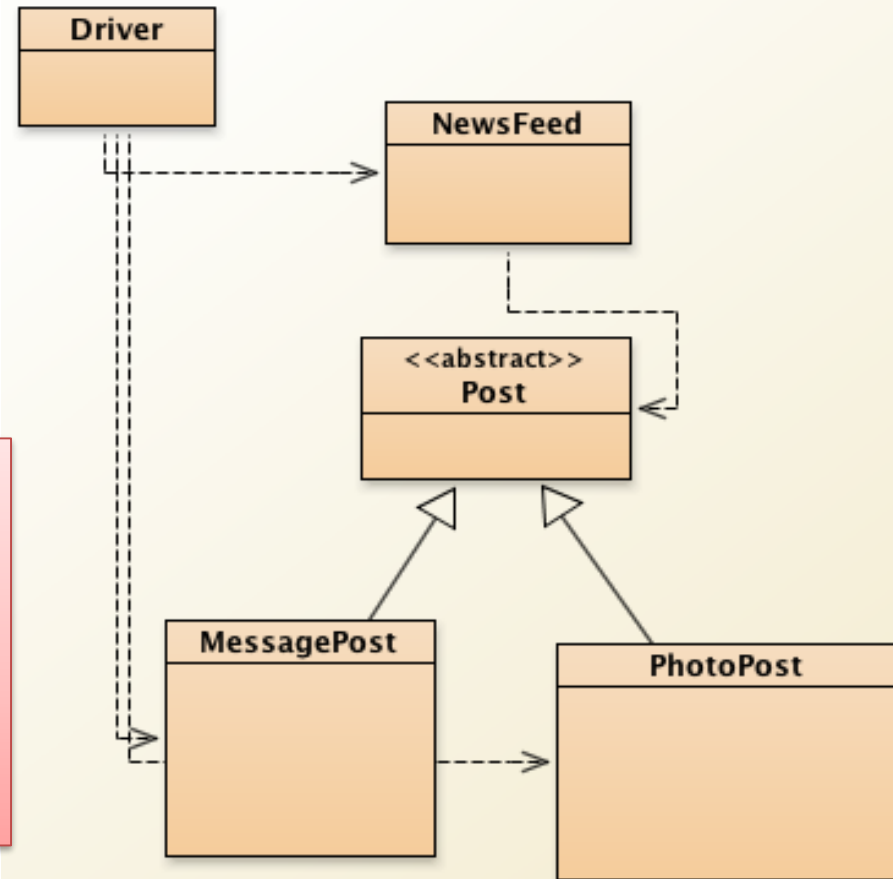
Options

Posts

```
1) Add a Text Post
2) Add a Photo Post
3) List all Posts
0) Exit
==>>
```

ArrayLists are in these classes:

- Post
- NewsFeed



Network-V6 – Post.Java

```
import java.util.ArrayList;
import java.util.List;

public abstract class Post
{
    private String username;
    private long timestamp;
    private int likes;
    private List<String> comments;
}
```

```
public Post(String author)
{
    username = author;
    timestamp = System.currentTimeMillis();
    likes = 0;
    comments = new ArrayList<String>();
}
```

```
public void setComments(List<String> comments)
{
    this.comments = comments;
}

public List<String> getComments()
{
    return comments;
}
```

Network-V6 – NewsFeed.Java

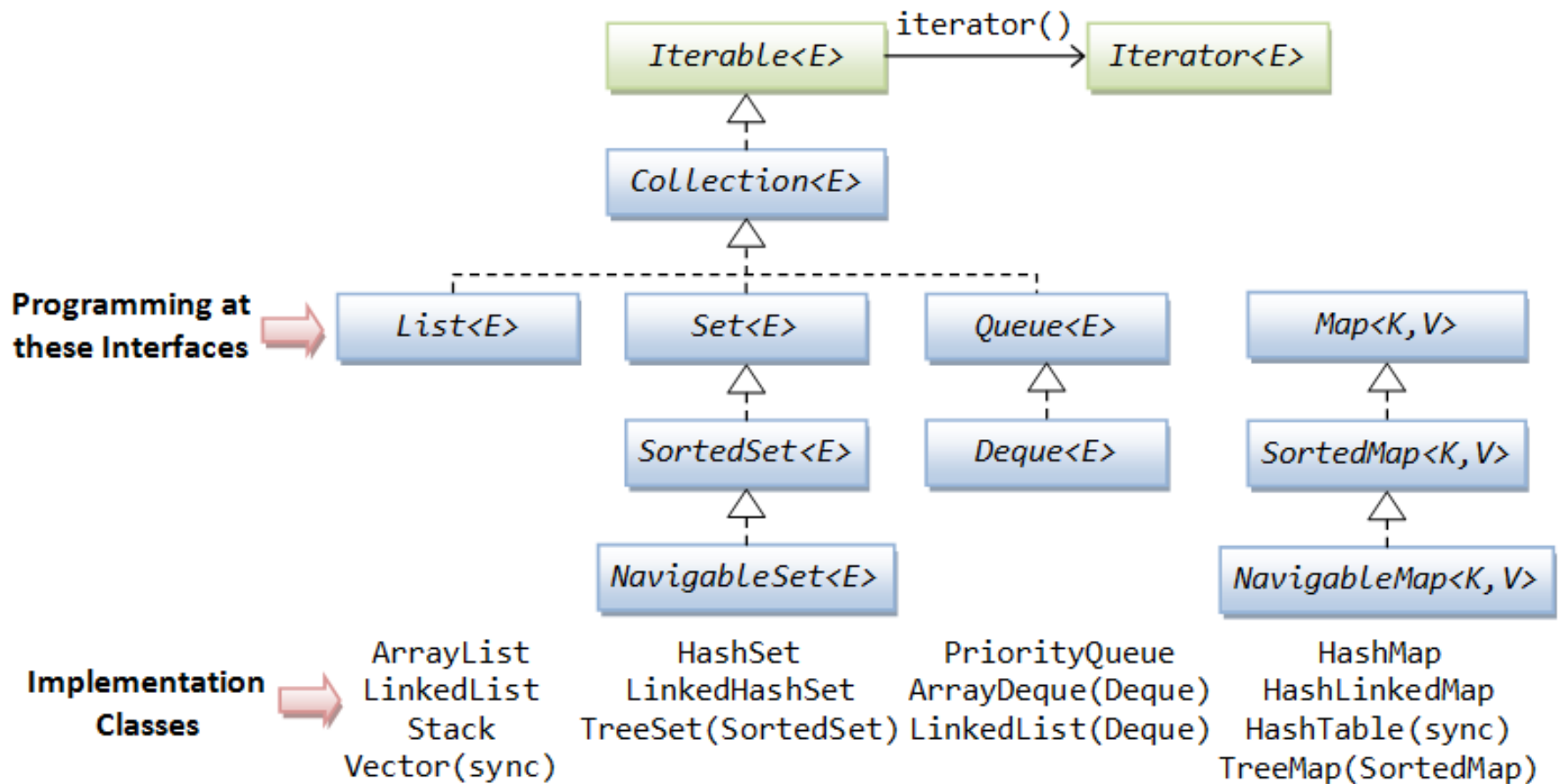
```
import java.util.ArrayList;
import java.util.List;

public class NewsFeed
{
    private List<Post> posts;

    public NewsFeed(){
        posts = new ArrayList<Post>();
    }

    //more code
}
```

Interfaces in Collections Framework



Interfaces in Collections Framework

Your code is more **maintainable** when you define collections like this:

- ArrayLists:

```
List<Product> products = new ArrayList<Product>();
```

- Maps:

```
Map<String, String> addresses = new HashMap<String, String>();
```

- Sets:

```
Set<String> words = new HashSet<String>();
```

Note: Include this approach in your Assignment 2.

Interfaces in Collections Framework

Why is code more maintainable when ArrayLists are defined like:

```
List<Product> products = new ArrayList<Product>();
```

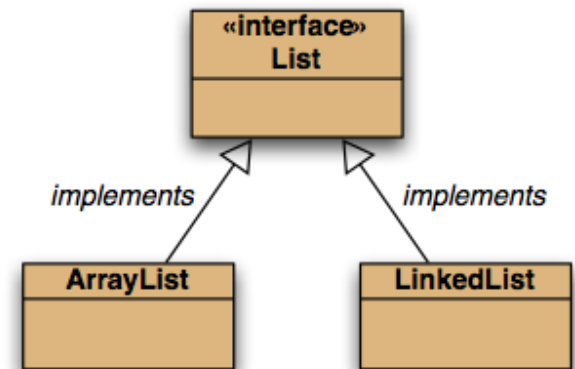
Answer: if we decided, at a later date, that we wanted to use a LinkedList instead of an ArrayList, we only have to make minor changes in the class i.e.

```
new ArrayList<Product>();
```

becomes

```
new LinkedList<Product>();
```

and `import java.util.LinkedList;`



Topic List

- Interfaces define Types.

- An Interface Example – Network V6.

- Naming Conventions for Interfaces.

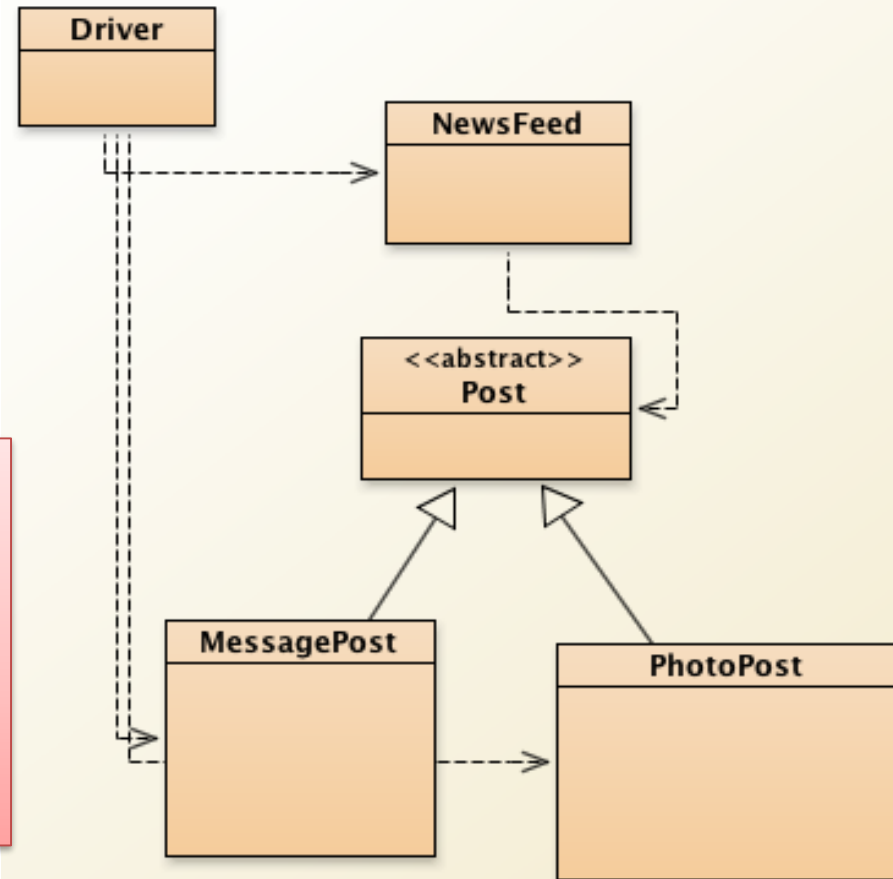
Network-V6

Options

Posts

- 1) Add a Text Post
 - 2) Add a Photo Post
 - 3) List all Posts
 - 0) Exit
- ==>>

We have used Interfaces in our code (for Collections) but we have not written an Interface ourselves.



Network-V7

```
Posts
1) Add a Text Post
2) Add a Photo Post
3) List all Posts
4) Delete a Post
0) Exit
==>>
```

We will write an Interface called **INewsFeed** that imposes a design contract on all classes implementing it.

When the **NewsFeed** class implements this interface, it will be forced to provide an implementation for the abstract methods defined in **INewsFeed**.

Network-V7 – INewsFeed.Java

```
public interface INewsFeed
{
    void addPost(Post post);
    void deletePost(int index);
    String show();
}
```

Network-V7 – current NewsFeed.Java

```
public class NewsFeed
{
    private List<Post> posts;

    public NewsFeed(){
        posts = new ArrayList<Post>();
    }

    public void addPost(Post post){
        posts.add(post);
    }

    public String show(){
        String str = "";
        for(Post post : posts) {
            str += post.display() + "\n";
        }
        return str;
    }
}
```

Network-V7 - updated NewsFeed.Java

```
public class NewsFeed implements INewsFeed
{
    //omitted code

    public void deletePost(int index) {
        posts.remove(index);
    }

    public String show() {
        String str = "";
        // display all posts with an index number
        int i = 0;
        for(Post post : posts) {
            str += i + ": " + post.display() + "\n";
            i++;
        }
        return str;
    }
}
```

Network-V7 – updated Driver.Java

```
private int mainMenu()  
{  
    System.out.println("1) Add a Text Post");  
    System.out.println("2) Add a Photo Post");  
    System.out.println("3) List all Posts");  
    System.out.println("4) Delete a Post");  
    System.out.println("0) Exit");  
    System.out.print("==>>");  
    int option = sc.nextInt();  
    return option;  
}
```

```
switch (option){  
    case 1:  
        addMessagePost();  
        break;  
    case 2:  
        addPhotoPost();  
        break;  
    case 3:  
        showPosts();  
        break;  
    case 4:  
        deletePost();  
        break;  
}
```

```
public void deletePost(){  
    showPosts();  
    System.out.print("Enter the index number for the post you wish to delete> ");  
    newsFeed.deletePost(sc.nextInt());  
}
```

Topic List

- Interfaces define Types.
- An Interface Example – Network V6.
- Naming Conventions for Interfaces.

Common Naming Conventions

- There are a few conventions when naming interfaces:
 - Suffix able is often used for interfaces
 - Cloneable, Serializable, and Transferable
 - Nouns are often used for implementing classes names, and I + noun for interfaces
 - Interfaces: IColor, ICar, and IColoredCar
 - Classes: Color, Car, and ColoredCar
 - Nouns are often used for interfaces names, and noun+Impl for implementing classes
 - Interfaces: Color, Car, and ColoredCar
 - Classes: ColorImpl, CarImpl, and ColoredCarImpl

**Any
Questions?**





Except where otherwise noted, this content is licensed under a Creative Commons Attribution-NonCommercial 3.0 License.

For more information, please see <http://creativecommons.org/licenses/by-nc/3.0/>



Waterford Institute of Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Department of Computing and Mathematics
<http://www.wit.ie/>