More on Abstraction in Java

More on Interfaces

Produced by: Ms. Mairead Meagher
Dr. Siobhán Drohan

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 V7.
- 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;

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

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.

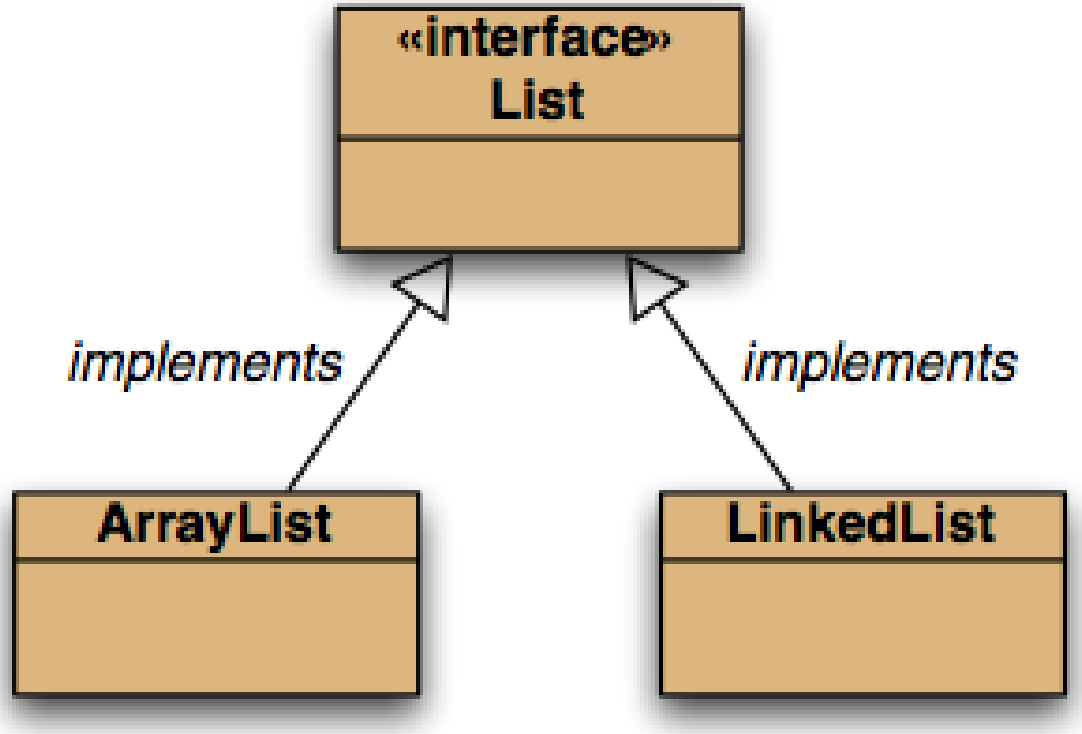
  dog = new Mammal();

```
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");
   }
}
```

https://docs.oracle.com/javase/tutorial/java/IandI/interfaceAsType.html

Interfaces in Collections Framework



Interfaces in Collections Framework

- ArrayList implements the List interface.

OVERVIEW MODULE PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Module java.base

Package java.util

Class ArrayList<E>

java.lang.Object
  java.util.AbstractCollection<E>
    java.util.AbstractList<E>
      java.util.ArrayList<E>

Type Parameters:

E - the type of elements in this list

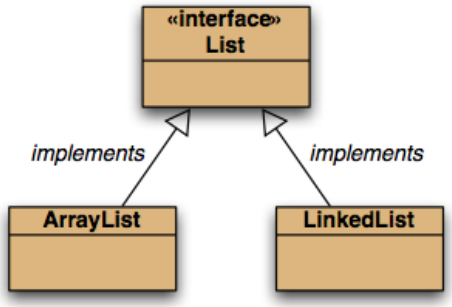
All Implemented Interfaces:

Serializable, Cloneable, Iterable<E>, Collection<E>, List<E>, RandomAccess

Direct Known Subclasses:

AttributeList, RoleList, RoleUnresolvedList

```
public class ArrayList<E>
extends AbstractList<E>
implements List<E>, RandomAccess, Cloneable, Serializable
```



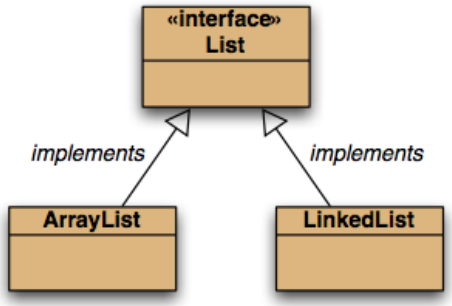
Interfaces in Collections Framework

- 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>();
```

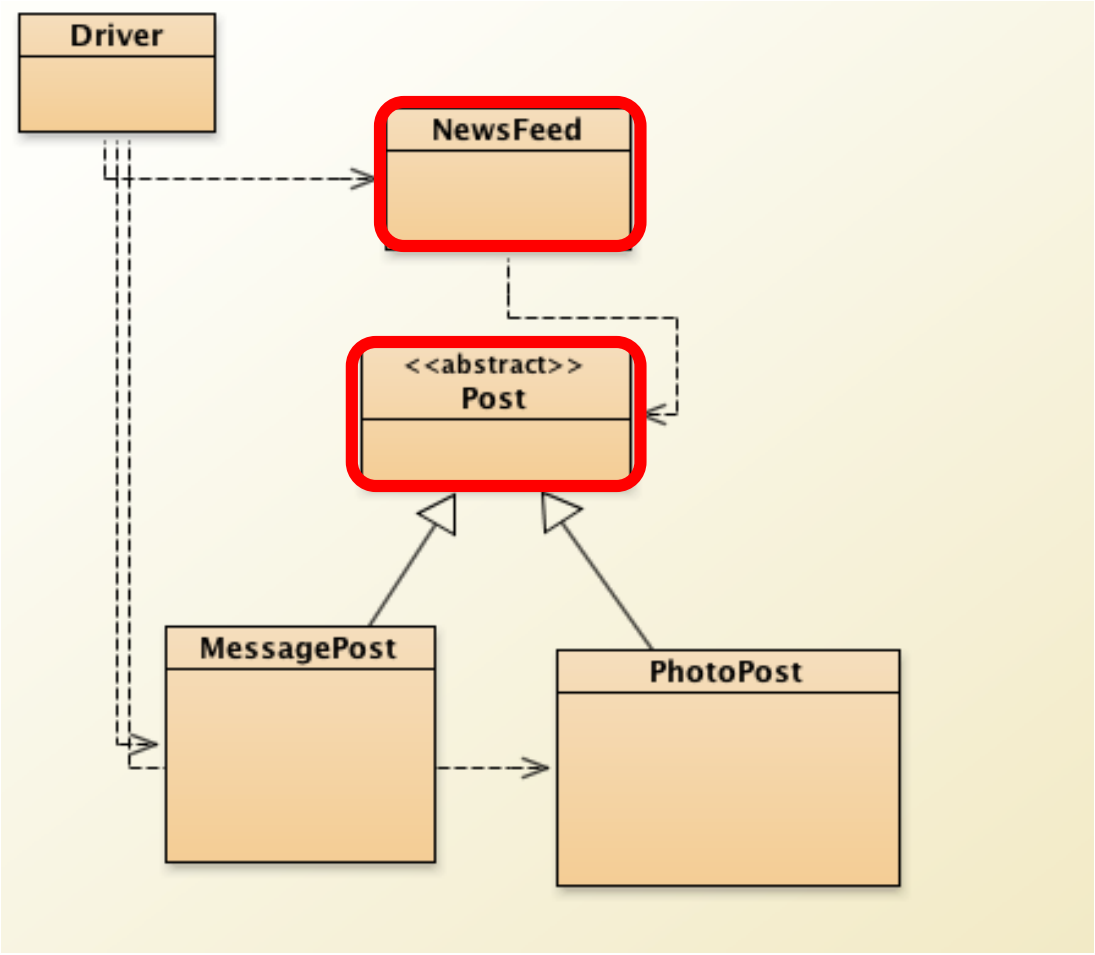


Recap - Network-V5

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

ArrayLists are in these classes:

- Post
- NewsFeed



Recap - Network-V5 (Post.Java)

```
import java.util.ArrayList;

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

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

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

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

Recap - Network-V5 (NewsFeed.Java)

```
import java.util.ArrayList;

public class NewsFeed
{
    private ArrayList<Post> posts;

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

//more code
}
```

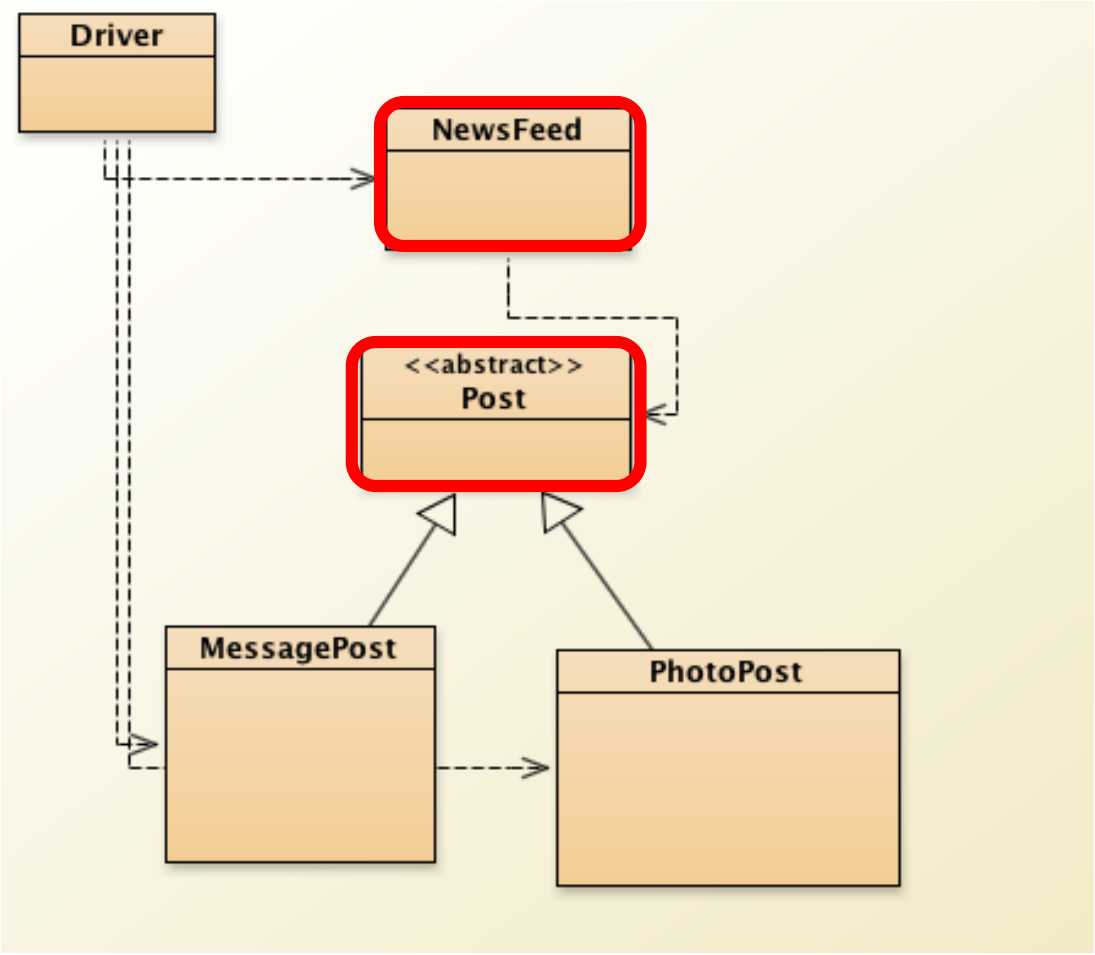
We will now refactor our V5 system so that ArrayLists are defined using interfaces instead of concrete classes.

Network-V6

```
Posts
1) Add a Text Post
2) Add a Photo Post
3) List all Posts
4) List an extract of 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;
}
```

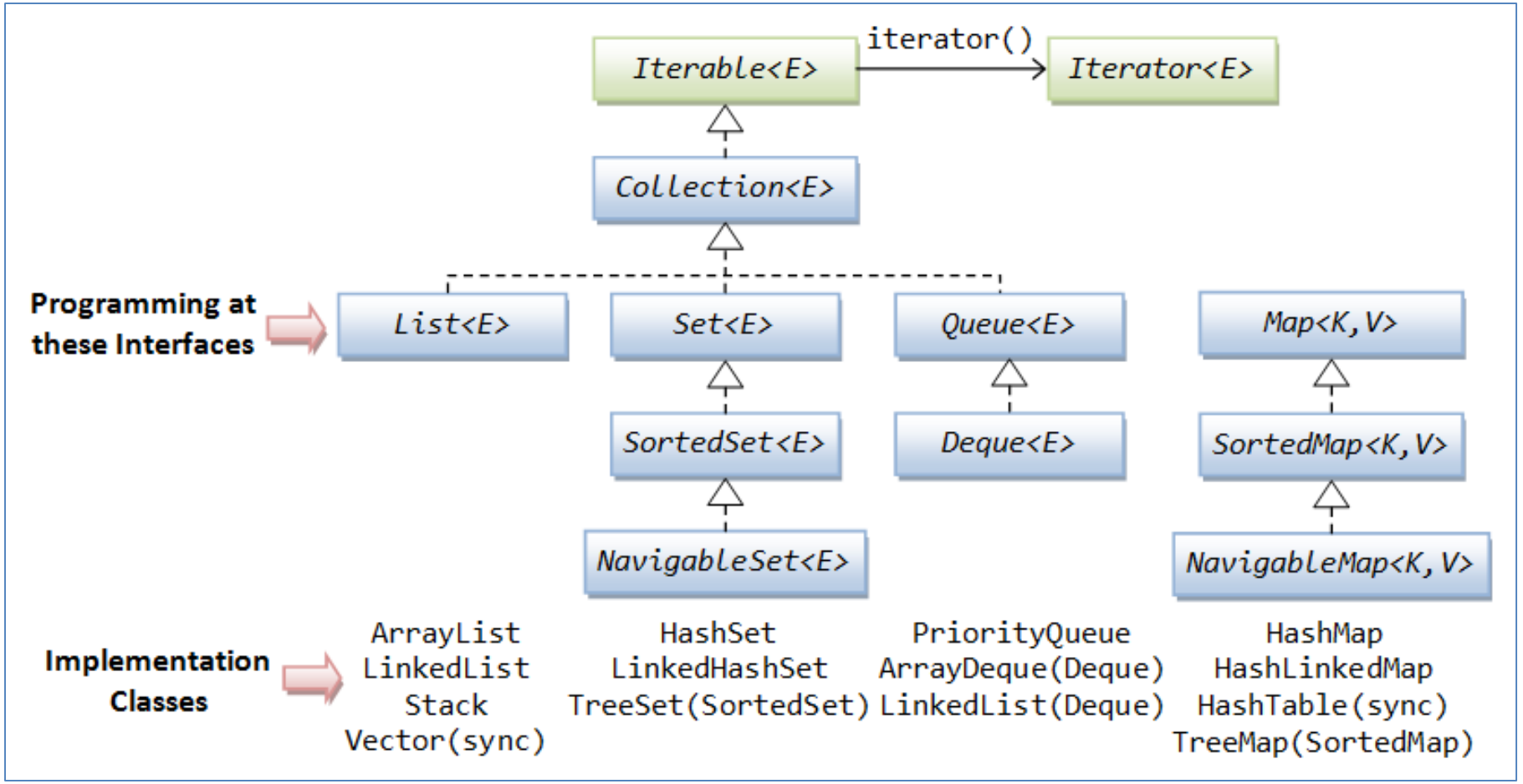
ArrayLists are now defined using interfaces instead of concrete classes.

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
}
```

ArrayLists are now defined using interfaces instead of concrete classes.

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>();
```

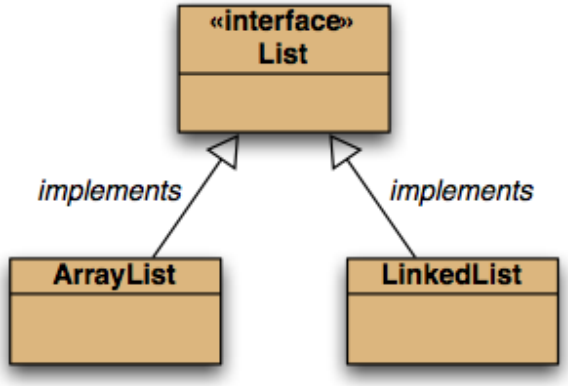
Answer: if we decided, at a later date, that we wanted to use a LinkedList instead of an ArrayList, we only have to make minor changes in the class i.e.

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

becomes

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

and `import java.util.LinkedList;`



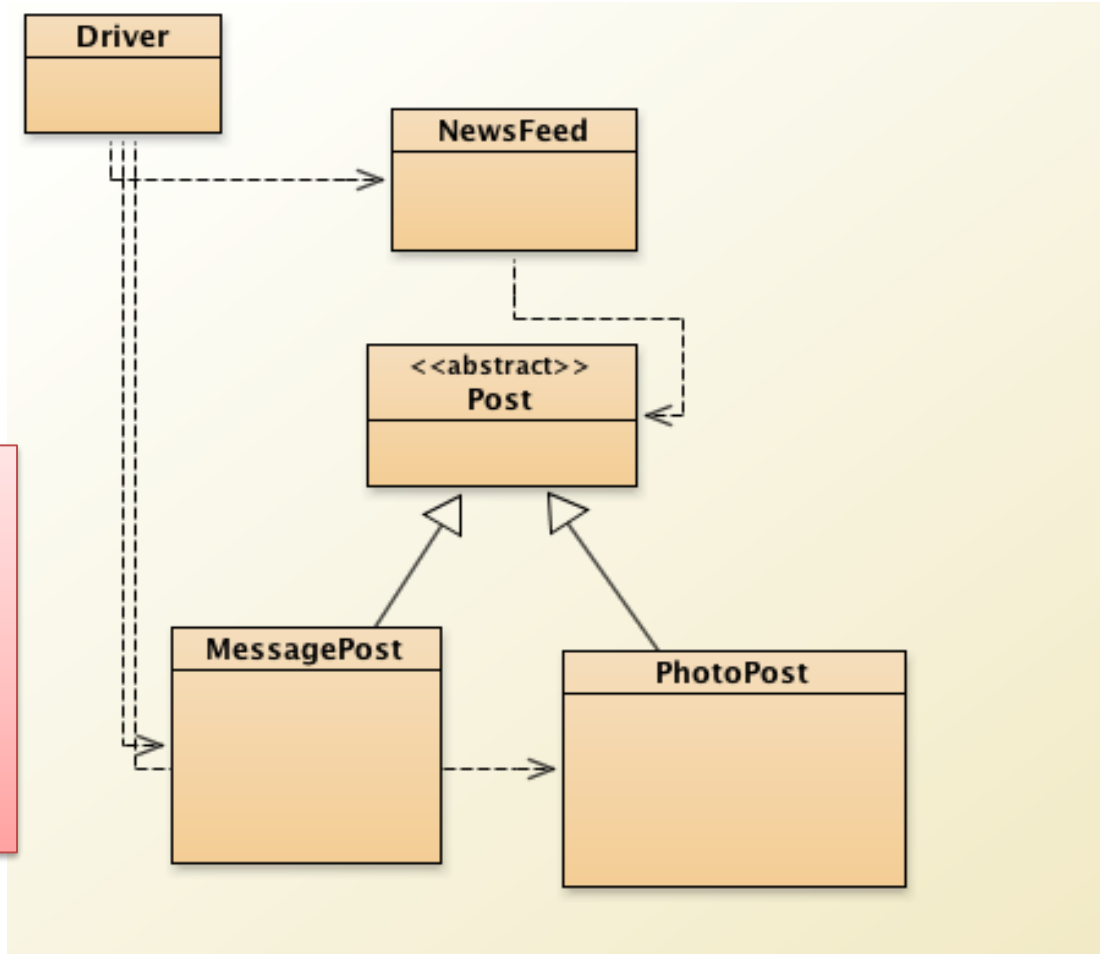
Topic List

- Interfaces define Types.
- An Interface Example – Network V7.
- Naming Conventions for Interfaces.

Network-V6

```
Posts
1) Add a Text Post
2) Add a Photo Post
3) List all Posts
4) List an extract of 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) List an extract of all Posts
5) 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 interface INewsFeed
{
    void addPost(Post post);
    void deletePost(int index);
    String show();
}
```

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

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

   public void deletePost(int index) {
       if ((index >=0 )  && (index < posts.size())) {
           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) List an Extract of all Posts");
  System.out.println("5) 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:
            showExtractPosts();
            break;
        case 5:
            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 V7.

– Naming Conventions for Interfaces.

Common Naming Conventions for 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?