CS 1114

Introduction to Software Design

Spring 2017 - Michael Irwin

How's Program 3?

Introducing Interfaces

- An **interface** is NOT a class - only defines required methods
- Contains *only method signatures and constants*
  - No constructors. No method bodies
- Any class that **implements** an interface is required to define the method bodies of the interface
- You cannot create an instance of an interface - only classes that implement the interface

Interface Example

You don't need to know how to create interfaces yet, but here's an example...

```
public interface Paintable {
    void paint(Color color);
}
```

And two implementations...

```
public class Cube implements Paintable {
    public void paint(Color color) {
        this.left = color;
        this.right = color;
        this.back = color;
        this.front = color;
        this.top = color;
        this.bottom = color;
    }
}
```

```
public class Table implements Paintable {
    public void paint(Color color) {
         this.tableTop = color;
         this.legs = color;
    }
}
```

Java's collection interfaces

- List (`java.util.List`)
  - Keeps items in a sequential order - allows you to get the `nth` item
  - Allows multiple entries of the same item
- Set (`java.util.Set`)
  - Generally, not kept in sequential order
  - Does not allow duplicates
  - Not going to cover in this course
- Map (`java.util.Map`)
  - Key-value collection - items are stored with a name. Allows retrieval using the name
  - Duplicate values allowed, but each name exists at most once
  - Will cover in a few weeks

Lists

- Keeps a group of zero or more objects as if they were arranged in a line
- The group can grow or shrink as you **add** and **remove**
- Positions start at index 0
- Main methods introduced by List interface
  - `boolean add(Object o);`
  - `int size();`
  - `Object get(int index);`
  - `Object remove(int index);`
- The "go to" implementation is `ArrayList`

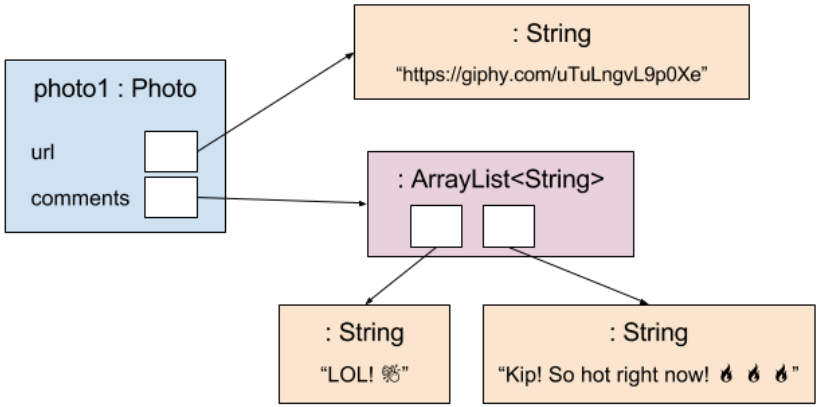
Example

```
public class Photo {
    private String url;
    private List<String> comments;

    public Photo() {
        this.comments = new ArrayList<String>();
    }
}
```

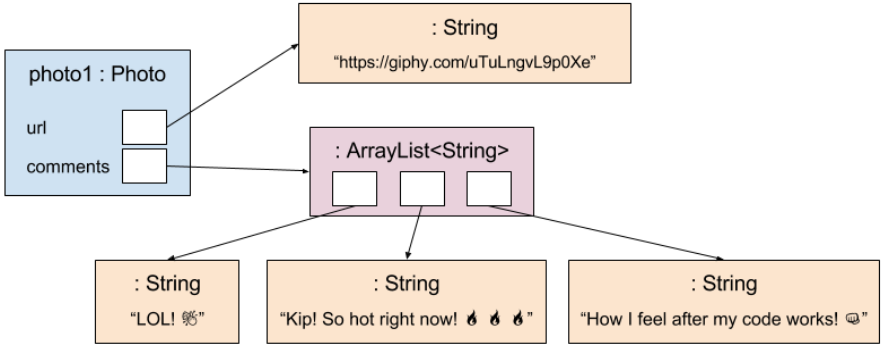
- Notice that comments is using a `List`, which is an interface
- `<String>` is using a feature of Java called generics
  - Simply says that things going in and out of the list are of type `String`
  - Won't dive much into generics in this course
- `new ArrayList<String>()` creates the actual list and assigns it to `comments`

Object References with an ArrayList



Adding a third file

```
comments.add("How I feel after my code works!   ");
```



Updating the `Photo` class

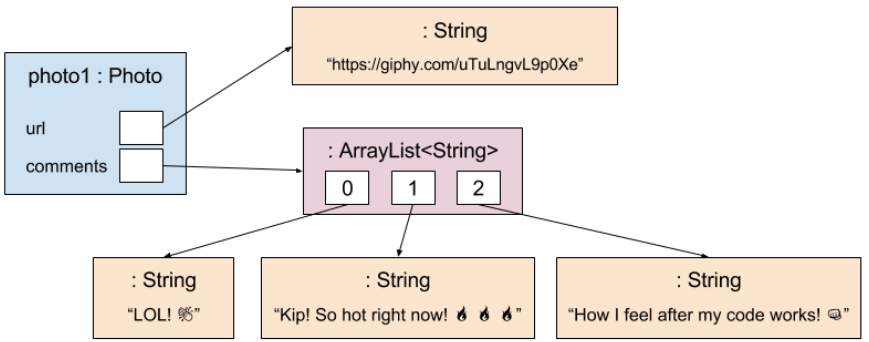
```
public class Photo {
    private List<String> comments;
    ...

    public void addComment(String comment) {
        comments.add(comment);
    }

    public int getNumberOfComments() {
        return comments.size();
    }
}
```

Index Numbering

- Accessing elements in a list uses indexes
- First item is found at index 0 (not 1!)
- Think of indexes as offsets from the beginning
  - To get second item, you need the item offset 1 from the beginning
  - `comments.get(1);`



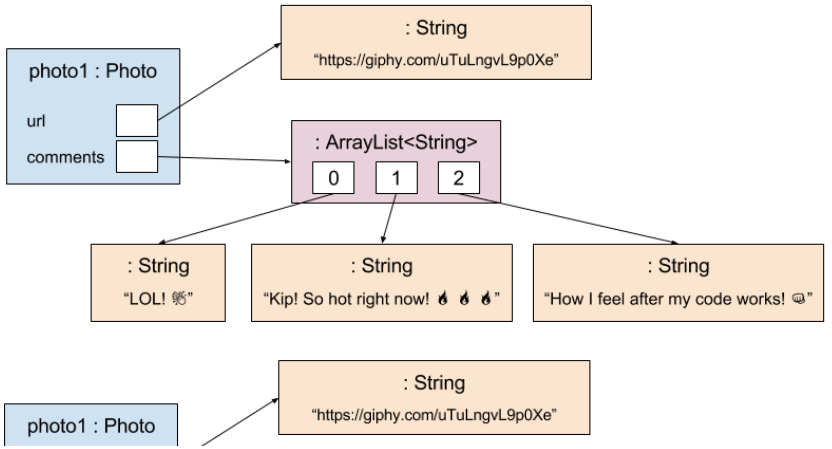
Retrieving an item from the list

- `Object get(index)` - returns the item found at the specified index
- **CAUTION** - ensure that index is a valid index number
  - If the number is not valid, you will get an exception (most likely `ArrayIndexOutOfBoundsException`)
  - Valid numbers range from 0 to list size - 1

Removing items from list

- **Important** - removing renumbers all elements after removed item

```
comments.remove(1);
```



Iterating through collections using for-each loops

- There are multiple ways to iterate through collections
- The for-each loop is most frequently seen
- Pros
  - Easiest to write
  - Terminates automatically (no infinite loops)
- Cons
  - Can't modify list while looping (`ConcurrentModificationException`)
  - Index not available

```
public void listAllComments() {
    for (String comment : comments) {
        System.out.println(comment);
    }
}
```

Searching within a list

- Two approaches to find an element - immediate return or breaking

```
public String findComment(String searchString) {
    for (String comment : comments) {
        if (comment.contains(searchString)) {
            return comment;
        }
    }
    return null;
}
```

```
public String findComment(String searchString) {
    String target = null;
    for (String comment : comments) {
        if (comment.contains(searchString)) {
            target = comment;
            break;
        }
    }
    return target;
}
```