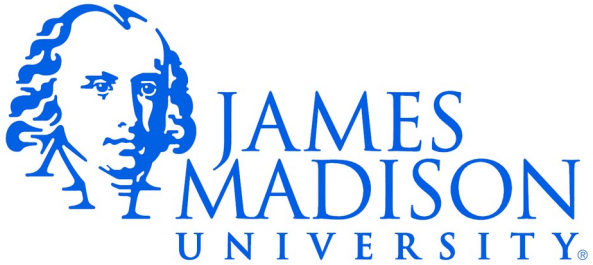
CS159



Declaring an ArrayList

- ArrayList is a “generic” type. Declaration determines the type of object it will store.
- Declaring an ArrayList to store String objects:

```
ArrayList<String> myList;
```

Instantiating an ArrayList

```
ArrayList<String> myList;  // Declaring.

myList = new ArrayList<String>();  // Instantiating.
```

OR...

```
myList = new ArrayList<>(); // Instantiating.
```

ArrayLists and Primitive Types

- ArrayLists can only store reference types

```
ArrayList<int> myList = new ArrayList<int>();
```

ArrayLists and Primitive Types: Solution

- Each primitive type has a wrapper class that can be used instead:

```
ArrayList<Integer> myInts = new ArrayList<Integer>();

myInts.add(new Integer(3));

int num = myInts.get(0).intValue();
```

- Fortunately, Java converts back and forth as needed: autoboxing

```
ArrayList<Integer> myInts = new ArrayList<Integer>();

myInts.add(3);

int num = myInts.get(0);
```

Evil of Indexed For Loops

- The following pattern is very common:

```
// Array Version:
Car curCar;
for (int i = 0; i < carArray.length; i++) {
    curCar = carArray[i];
    curCar.drive();
}

// ArrayList Version:
Car curCar;
for (int i = 0; i < carList.size(); i++) {
    curCar = carList.get(i);
    curCar.drive();
}
```

- Much of this code involves updating and monitoring the index variable i, even though we don't care about its value.

Evil of Indexed For Loops

- These loops have exactly the same functionality:

```
//Array Version:
for (Car curCar: carArray) {
    curCar.drive();
}

//ArrayList Version:
for (Car curCar: carList) {
    curCar.drive();
}
```

- Cleaner, easier to read, less error-prone.
- You should only use indexed for loops when you actually need to use the index variable.