CS112 Lab 02, Jan 28, 31 2010

http://cs-people.bu.edu/deht/cs112_spring11/lab02/

Diane H. Theriault

deht@cs.bu.edu

http://cs-people.bu.edu/deht/

Today's Topics

- Inheritance & Interfaces
- Collections and Iterators (Queue in particular)
- Generics

References

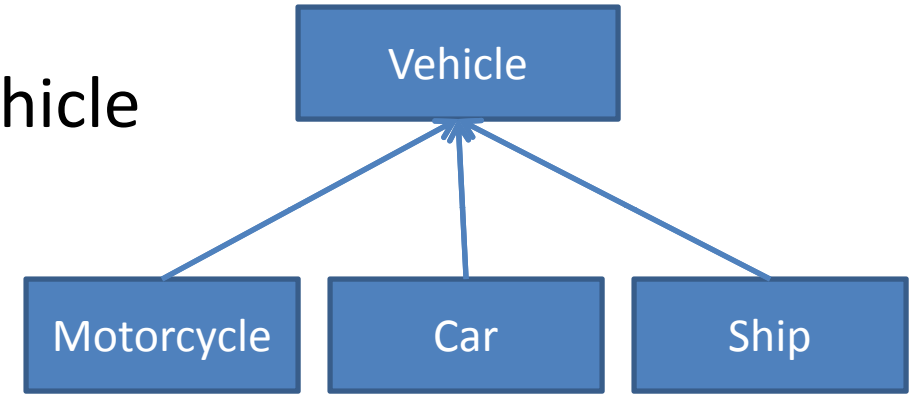
- http://download.oracle.com/javase/tutorial/java/IandI/index.html
  (Inheritance & Interfaces)

- http://download.oracle.com/javase/1.5.0/docs/api/java/util/Queue.html
- http://download.oracle.com/javase/1.5.0/docs/api/java/util/LinkedList.html
  (LinkedList and Queue API)

- http://download.oracle.com/javase/tutorial/java/generics/index.html
  Generics

Inheritance

- "is a" relationship.
  - A Motorcycle **IS A** Vehicle
  - A Car **IS A** Vehicle
  - A Ship **IS A** Vehicle



public class Vehicle { … }

public class Car extends Vehicle {…}

- *(BTW, Inheritance hierarchies can be a source of a lot of arguments in large code bases!)*

Inheritance

- When a class extends a superclass, it *inherits* all of the member variables and methods of the parent class

- A subclass can *override* or change the behavior of any of the methods of the superclass. (This is called polymorphism)

- Everything in Java inherits from the “Object” class.

Inheritance

- Assigning Subclass to Superclass reference is free.

  Object myObject = new LinkedList();

- Assigning Superclass to Subclass must be done carefully.
- Java checks for type compatibility at run-time.

  LinkedList myList = (LinkedList) myObject;

- Doing it incorrectly can cause exceptions (crashes)

  Object myObject = new String(“hello”)

  Vehicle myVehicle = (Vehicle) myObject;

  //WRONG!

Inheritance

- An “abstract” class does not need to provide a base implementation for all of its methods.

- The keyword “super” is used in subclasses to invoke the constructor of the parent (possibly with arguments)

Interface

- "acts like a" relationship
  - A Car acts like a transportation mechanism
  - A Car acts like a bright shiny object
  - A Car acts like a thing that needs maintenance
- A contract to provide certain functionality

```
public interface Drivable { public void steer(); }
public class Car extends Vehicle implements Drivable
{  …
  public void steer() {…}
}
```

Inheritance vs Interfaces

- A “base class” or “superclass” can have many “subclasses”
- In Java, a subclass can only ***extend*** one superclass.

- BUT, some objects can have many different types of behavior (***implement*** many different interfaces).

- (Interfaces can inherit from other interfaces, just like classes)

Example from Java Library

public class **LinkedList<E>**

extends AbstractSequentialList<E>

implements List<E>, Queue<E>, Cloneable, Serializable

public interface **Collection<E>**

extends Iterable<E>

(We will explain the <E> notation in a bit)

Referencing Objects

- Can refer to an object using a reference of its type, any of its parent's types, or the type of any interface it implements

```
LinkedList myList = new LinkedList();  //the type itself

Object myList = new LinkedList();
//everything in Java inherits from Object

Queue myList = new LinkedList();
//LinkedList implements the Queue interface

Drawable myList = new LinkedList();
//No! LinkedList does not implement Drawable!
```

How do you know?

- How do you know what class XXX inherits from?
- How do you know what interfaces it implements?

- The Java Doc!

http://download.oracle.com/javase/1.5.0/docs/api/

Today's Topics

- Inheritance & Interfaces
- Collections and Iterators (Queue in particular)
- Generics

Collections

- “Collection” is a Java interface.
- It is implemented by several classes

  (ArrayList, LinkedList, TreeSet, etc.)

http://download.oracle.com/javase/1.5.0/docs/api/java/util/Collection.html

Methods include: add(), remove(), size(), etc.

Collections & Iterators

- Collections can be stored in many ways (that we will learn about in CS112)
- *Iterators* provide a unified way to access the elements of a collection
- (The Collection interface extends the Iterable interface)

Why use Iterators?

- Why can't I just loop over a collection using for(int i=0; i<myCollection.size(); i++)?
- This only works for arrays!
  - Not ArrayLists, LinkedLists, or any other type of Collection.
- Even if it did work, there are many data structures (like trees) where the notion of the "i'th" element is not well-defined
- Within a structure, indexes can change as you manipulate the object.

Two ways to iterate over a Collection

- Just syntax. Semantically identical.

```
for(Iterator iter=myList.iterator(); iter.hasNext(); )
{
    Object myObject = iter.next();
    System.out.println(myObject);
}
   //confusing! The next() call both retrieves the current
   item in the collection and increments the iterator.
```

Two ways to iterate over a Collection

- Just syntax. Semantically identical.

```
for(Object myObject: myList)
{
    System.out.println(myObject);
}
//much better!
```

What is a Queue?

- First In, First Out (FIFO)
- First Come, First Serve (FCFS)

- Like being in line at the movies.

The Queue Interface

- Queue is just another Java interface, which happens to be implemented by LinkedList.
- It inherits from Collection.

```
Queue myQueue = new LinkedList();
myQueue.offer()
myQueue.peek()
myQueue.poll() / myQueue.remove()
```

http://download.oracle.com/javase/1.5.0/docs/api/java/util/Queue.html

The Queue Interface

- What's with the weird names?
- Queues can fill up.
  - Offer() can return false
  - Add() can only throw an exception
- Also, need to differentiate between the queue-like methods and the collection-like methods (which may not enforce ordering correctly).

Today's Topics

- Inheritance & Interfaces
- Collections and Iterators (Queue in particular)
- Generics

Collections and Types

- Collections don’t know what’s in them.
- Type-checking must be done at run-time.
- Exceptions and crashes may ensue.

```
LinkedList myList = new LinkedList();
myList.add(new String(“hello”));
for(Object myObject: myList)
{
    Integer value = (Integer) myObject; //exception!
    System.out.println(value);
}
```

Types and Generics

- Special “<…>” syntax allows you to promise the type of objects that the Collection holds.
- This allows compile-time type-checking.

```
LinkedList<String> myList = new LinkedList<String>();
myList.add(new String(“hello”));
for(String myObject: myList)
{
    Integer value = (Integer) myObject; //compiler error!
    System.out.println(value);
}
```

Practical Lab: DiscoveryChannel Party

- I have invited the casts of 3 Discovery Channel series to my house for a party.
- Each cast will travel in their preferred vehicle type
  - Mythbusters (Car)
  - Deadliest Catch (Ship)
  - American Chopper (Motorcycle)

Practical Lab: DiscoveryChannel Party

- Motorcycle, Car, and Ship extend Vehicle
  - Vehicle has Strings mName and mDriver
  - Vehicle has abstract methods
    - getManifest() // returns a list of Strings (passenger names)
    - Go() //returns a string describing the action of the Vehicle
- Car and Ship implement MultipleOccupancy
  - MultipleOccupancy requires an addPassenger method.

- Given the driver class, fill in the implementation of Motorcycle, Car, and Ship

Practical Lab: DiscoveryChannel Party

- The skeleton code:
  - Creates some vehicles containing the people.
  - Prints the Vehicle driver and name
  - Invokes the Go() method
  - Welcomes each of the people in each Vehicle to the party
- I have implemented the Motorcycle class for you.

Practical Lab: DiscoveryChannel Party

- Car and Ship will need some type of Collection to hold the set of passengers.

- As you know, there is a lot of drama on Deadliest Catch. You will need to use a data structure that provides first-come, first-serve access to avoid conflicts.

Practical Lab: DiscoveryChannel Party

- On Deadliest Catch, they are very proud of their boats.
- Change the implementation of the Ship constructor so that the **name** of the boat will be printed, instead of the word “boat”
  - Hint: add an additional argument to the constructor.

Practical Lab: DiscoveryChannel Party

- The caravan variable in the main is a Set, **not** a Queue.

- In the Travel() method, note that the items in caravan are **not** printed in the order that they were added.

Things you will need for HW1

- Q1: Derivative of a polynomial function:
  - $F(x) = x^n + c \rightarrow F'(x) = n * x^{n-1}$
- Q2: you do not need to discuss the complexity / run-time.
- Q3: you will need to use a data structure that implements the Queue interface.