Linked Lists Part 2

Linked List Implementation

Checkout *SinglyLinkedList* project from SVN (Homework)
Checkout LinkedListSimpleGeneric
Checkout CoolPair

Let’s modify our simple linked list to take arbitrary objects!

- Two ways:
  - Object
  - Generics

What if we just use object?

```
LinkedList objectList = new LinkedList();
objectList.addAtBeginning(new Dog("Max", 15));
objectList.addAtBeginning(new Dog("Sammy", 9));
objectList.addAtBeginning(new Dog("Gracie", 4));

System.out.println("Average age is: " +
getAverageAge(objectList));
```

Output: Average age is: 9.333333333333334

The problem with Object

```
//But what happens if we add a car to that list?
objectList.addAtEnd(new Car("Toyota", "Camry"));

System.out.println("Average age is: " + getAverageAge(objectList));
```

Java allows us to add a Car to a list of Dogs, because it only knows the Node values are stored as objects

Output:

Exception in thread "main" java.lang.ClassCastException:
withObject.Car cannot be cast to withObject.Dog

```
public static double getAverageAge(LinkedList objectList) {
        double totalAge = 0;
        int count = 0;
        for (Object o : objectList) {
                Dog d = (Dog)o;
                totalAge += d.getAge();
                count++;
        }
        return totalAge/count;
}
```

This cast is what causes the previous code to fail (when it tries to cast a Car to a Dog). But we must have the cast to get the age field of the Dog objects.

Generics Prevent Type Errors

```
LinkedList<Dog> dogList = new LinkedList<Dog>();
dogList.addAtBeginning(new Dog("Max", 15));
dogList.addAtBeginning(new Dog("Sammy", 9));
dogList.addAtBeginning(new Dog("Gracie", 4));
//But what happens if we add a car to that list?
dogList.addAtEnd(new Car("Toyota", "Camry"));
```

Attempting to insert an object that IS NOT a Dog into the list causes a **compilation** error (better since we'd rather it crash for us and not our clients!).

dogList is declared as a generic list of Dog objects, so only Dog objects (and objects that inherit from Dog) can be put in this list.

```
public static double getAverageAge(LinkedList<Dog> dogList) {
        double totalAge = 0;
        int count = 0;
        for (Dog d : dogList) {
                totalAge += d.getAge();
                count++;
        }
        return totalAge/count;
}
```

The enhanced for loop no longer needs a cast, because it knows that the objects in the list are Dog objects. No possibility for a runtime error.

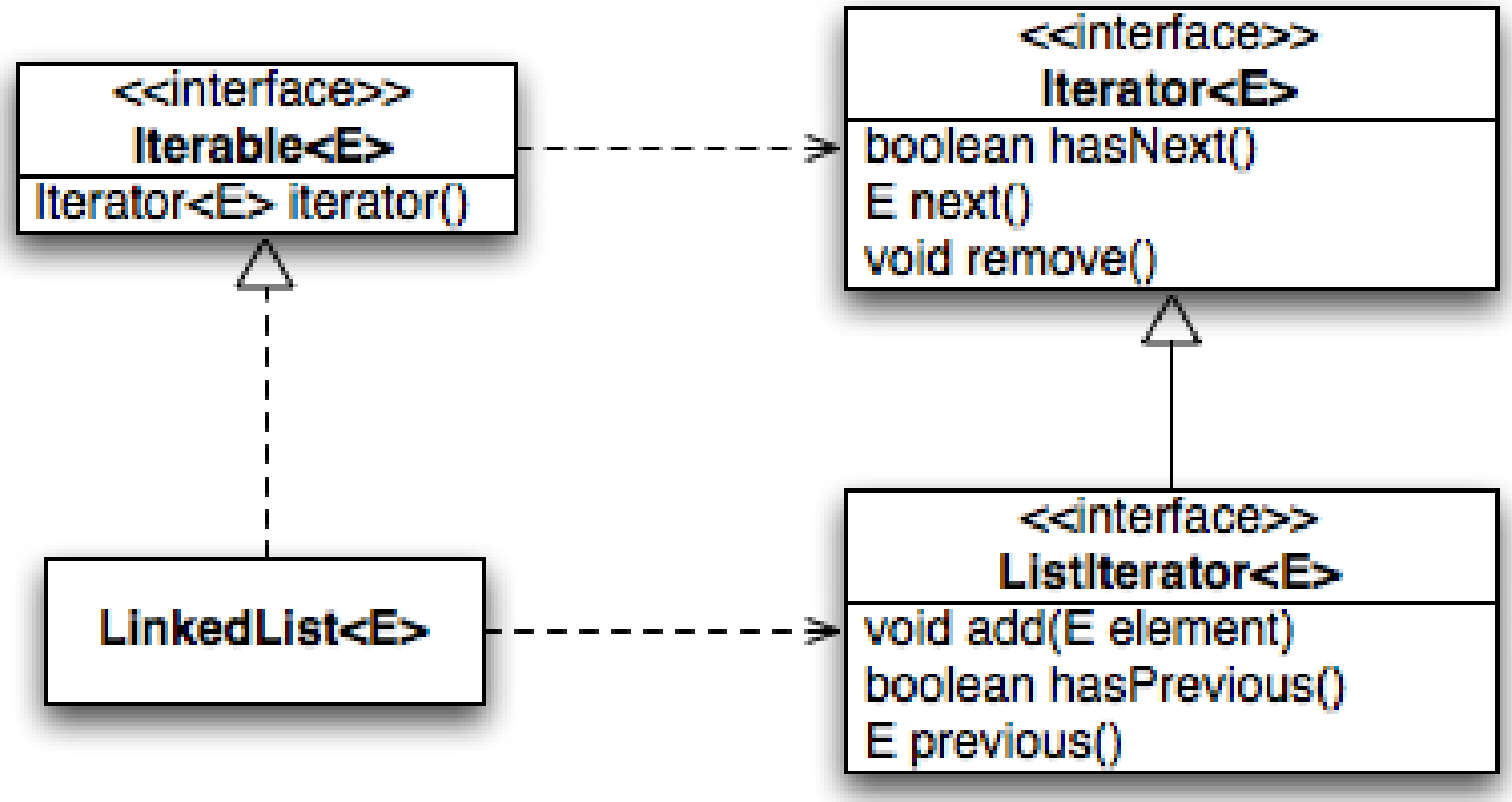
Generics Advanced

- Type parameters:
  - `class DLList<E>`
- Bounds:
  - `class DLList<E extends Comparable>`
  - `class DLList<E extends Comparable<E>>`
  - `class DLList<E extends Comparable<? super E>>`
- Generic methods:
  - `public static <T> void shuffle(T[ ] array)`
- http://docs.oracle.com/javase/tutorial/java/generics/index.html

What are iterators and why do they exist?

- Iterators are objects designed to encapsulate a position in a data structure – in the case, a pointer to a current (and previous) node in a list
- Your textbook has a detailed discussion of the operation of linked list iterators, including lots of sample code

Accessing the Middle of a LinkedList



Why iterators?
They let you write nice for loops!

Enhanced For Loop

```
for (String s : list) {
  // do something
}
```

What Compiler Generates

```
Iterator<String> iter =
    list.iterator();

while (iter.hasNext()) {
  String s = iter.next();
  // do something
}
```

Practice

- Weird warmup: Add an iterator to CoolPair<T>
  - Weird: why iterate over a Pair? Oh well.

- Make LinkedListGeneric generic and add an iterator to it. Notes:
  - T could be any object. So will need to change == to .equals() when comparing things of type T.
    - But still use == for Nodes: if (this.current == null) { …}
  - When adding <Integer> to tests, also need to change the int[] array passed in to Integer[] to match.
  - You can test your iterator using a foreach loop in main
  - Get help! This is practice for the next assignment.

Homework: Implementing SinglyLinkedList

- Just a step up from the ones we've written, but more focused on implementing the essentials from the java.util.List interface

- Will have the usual linked list behavior
  - Fast insertion and removal of elements
    - Once we know where they go using an iterator
  - Slow random access

TEAM PROJECT WORK TIME