

CS231 Fall 2013 Lecture 11

Stephanie R Taylor

September 27, 2013

1 Administrative Topics

- Go over homework.
- We take the quiz (but after we have added a method to linked lists).

2 Review Iterators

For our linked list classes (both the in-class and project classes), there are iterators. with this relationship to the class:

- The `LinkedList` implements `Iterable`, which means it has a method that creates an `Iterator` object.
- The `LinkedList` has an inner class that implements `Iterator`. This inner class has `hasNext`, `next`, and `remove` methods.

And because the linked lists have iterators, it is possible to use the `foreach` loop structure. Nice!

3 Linked Lists

We write a `addLast` method, which adds an element to the tail of the list.

If we have time, we write an `insert` method, which inserts a method at a given position in the list.

4 Generic Classes

Suppose you have a class, such as a linked list, that can be made to work with any object. Suppose further that when someone uses that class, they want to specify what type of object it should work with. Then you want a generic class. A generic class has a *type variable* that allows you to write your class for any type and allows other code to instantiate objects in your class with a specific type.

For example, the `ArrayList` class can take any object. But if you want an `ArrayList` of `Strings`, you can tell the compiler that this `ArrayList` object will hold only `Strings`. This makes writing the rest of the code simpler because

- You won't need to do as much type casting.
- Instead of doing run-time checks on the type of the object, the compiler can do it for you.

In the project, your linked list class is a general class in that the type of data it stores can be any object.

When you write

```
public class LinkedList<T> implements Iterable<T> {
```

you are telling Java that your `LinkedList` will have some aspect of it that depend on a type to be specified by the creator of the object. `T` is a variable that indicates that type.

Note: It is possible to make methods and interfaces generic, too.

5 Exceptions

An exception is an event that occurs during the execution of a program that disrupts the normal flow of instructions.

Exceptions can be thrown (when you encounter an error and want to disrupt the flow) and caught (when you know an exception might be thrown, you can catch it and handle the error as you see fit).

There are three types of exceptions: checked exceptions (such as trying to open a file that doesn't exist), errors (problems that could not be anticipated and are external to the program), and runtime exceptions (e.g. a logic error causes a null to be passed to the constructor and the constructor throws `NullPointerException`).

Checked exceptions are the sort of exceptions that your programs should recover from. E.g you should anticipate that a file might not exist. So when you open a file, you need to write code that will catch the `java.io.FileNotFoundException` exception. Your code can then write an error message and give up. Or, if it makes sense, just pretend the whole thing never happened.

Java has a rule about checked exceptions: it is called the *catch or specify requirement*. If you write a method `methNew` that calls a method `methOld` and `methOld` might throw a checked exception, then your new method must catch that exception and bow out gracefully, or it must specify that it could throw it.

E.g. Here is the constructor for Java's `FileReader`. It advertises that it might throw an exception.

```
public FileReader(String fileName)
    throws FileNotFoundException
```

So if you write code that makes a `FileReader`, you either need to catch the exception like this

```
public void doThatFileThing( String fn ) {
}
}
```

or you need to specify that you might throw it like this:

```
public void doThatFileThing( String fn ) {
    try {
    }
    catch (FileNotFoundException
}
}
```