Linked Lists
Abstract Data Types

- An Abstract Data Type (ADT) is:
  - a set of values
  - a set of operations
- Sounds familiar, right? I gave a similar definition for a data structure.
Abstract Data Types

- Abstract Data Types are a taxonomy of data structures.
- Moreover, an abstract data type is an abstraction.
  - A *contract* for a type of data structure.
- There are typically many data structures for a given ADT
  - Ex A *Tree* is an ADT
    - *Binary Search Tree* is a concrete data structure
    - *Red-black Tree* is another.
Abstract Data Type: List

- A List is an abstract data type that represents an ordered sequence of values, where the same value may occur more than once.

- Some operation it may provide:
  - a constructor for creating an empty list;
  - an operation for testing whether or not a list is empty;
  - an operation for appending an entity to a list
  - …

- Does this sound familiar?
ArrayList

- An ArrayList is an example of an implementation of the List ADT
  - (In fact, there is a List interface which ArrayList implements)
- So as we know an ArrayList is backed by an array
- However, that's not the only data structure that is part of the class of data structures known as Lists
A linked list is a series of connected nodes.

Each node contains at least:

- A piece of data
- ‘Reference’ (pointer) to the next node in the list

Head: pointer to the first node

The last node points to NULL.
Linked List

- In order to traverse the list, you follow the pointer from each node to the next.
  - Note that the nodes are not contiguous in memory!
- We create a new Node very time we add something to the List
  - Prepending to a list is very fast
  - Inserting into a sorted list is very fast
Recursive Data Type

- A recursive data type is a class that may contain other values of the same type.

- This is the other common use case of recursion.

- Note that each Node has a reference to another Node. How would this look in code?

- A recursive data type is a class that may contain other values of the same type.

```java
public class Node {
    private int value;
    private Node next;
}
```
Versus Arrays: Advantages

- They are dynamically sized: a linked list can easily grow and shrink in size without expensive memory allocations.

- With arrays we have to declare a fixed size, and resizes are costly!

- Easy and fast insertions and deletions.

- To insert or delete an element in an array we may need to move every other element in the array to accommodate. With a linked list, no need to move other nodes, only adjust some pointers.
Versus Arrays: Disadvantages

- Access to any particular node in a linked data structure requires following a chain of references that stored in it.

- No way to ‘randomly’ access a node in the middle of the array.

- For example, A HashMap could not be implemented efficiently using a linked list.
There is a LinkedList in the Java Collections Library that is a sibling to the ArrayList.

They both implement the List interface (which represents the list ADT)

http://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html
Programming Example

- We will look at a simplified linked list with only a few operations:
  - `isEmpty`: determine whether or not the list is empty
  - `insert`: insert a new node *at the end* of the list
  - `print`: print the entire contents of the linked list
- See `linkedlist/LinkedListTest.java`
Variation: Doubly Linked List

What we’ve see before is what’s known as a ‘singly linked list’

There are others, for example, the ‘doubly linked list’

Each node points to not only successor but the predecessor

There are two NULL: at the first and last nodes in the list

Advantage: given a node, it is easy to visit its predecessor. Convenient to traverse lists backwards
Generics
There are many types of polymorphism. So far in this course we’ve seen..

- Method overloading
- Using a subclass as its superclass in arrays or with methods
- There is another kind called ‘parametric polymorphism’
- You’ve actually see this already too, though I don’t think I used this name.
Parametric Polymorphism

- Parametric polymorphism is a way to make a language more expressive, while still maintaining static type-safety.
  - Remember, we have a static type system in Java to help write more correct code.
  - A function or a class can be written such that it can handle values identically without depending on their type.
  - Such functions and data types are called generic and form the basis of generic programming.
Generic Programming

- Generic programming is a style of programming in which algorithms are written in terms of \textit{types to-be-specified-later}

- Moreover, generics allow you to abstract over types.

- A common case for generic programming is container data structures, such as a List or a Map.
Generic Programming

- Those last few slides had words on them.
Generics in Java

- Java Generics is an example of a generic programming language construct.
- Common examples are found in the Collections library.
- Do you recall seeing this before?
  ```java
  List<String> list = new ArrayList<String>();
  ```
- What does this mean?
Generics in Java

- Generics were introduced in Java 5, which was released in 2004.
- Therefore you can use Java without them, but there are many benefits to using them.
- Prior to Java 5, all Collection types worked by Type-Erasure
- That means that any reference type put into any collection was treated as an Object!
  - (Remember Object is the root super class for all classes in Java, every reference type is-a Object)
Use in Java Collections

```java
// Without Generics
ArrayList l1 = new ArrayList();
l1.add("hello");
String s1 = (String) l1.get(0); // Note the cast

// With Generics
ArrayList<String> l2 = new ArrayList<String>();
l2.add("hello");
String s2 = l2.get(0);
```

- Note the cast on line 4
- The compiler can only guarantee that an Object will be returned.
- To ensure the assignment to a variable of type String is type-safe, the cast is required.
- There is the possibility of an error, since the programmer may be mistaken.
Use in Java Collections

- Enter Generics. Notice the type declaration on line 7.

- It specifies that this is a List of of String, written ArrayList<String>.

- Why? Type-safety! The compiler can now say for certain that there are only strings in the ArrayList.

- Every place you try and add to the list, the compiler will check to see if its String
Use in your code

- Great, so we can use Generics to have more reliable code when we use the Java Collection Library.
- Wouldn’t it be useful to be able to use Generics in our own classes?
- Remember our StackOfCharacters.java earlier in the semester?
- Wouldn’t it be annoying if we had to write a stack implementation for every type or class we want to store in a stack??
Use in your code

- We can!

- Here is a partial Stack data structure implementation using Generics.

- By changing our class to have ‘Type Parameters’ we can let the user of our class specify what type our stack will contain, just like ArrayList or HashMap

```java
class GenericStack<E> {
    private ArrayList<E> contents = new ArrayList<E>();

    public void push(E element) {
        contents.add(element);
    }

    public E pop() {
        int top = contents.size() - 1;
        E result = contents.get(top);
        contents.remove(top);
        return result;
    }
}
```
- Use in your code

- Note the <E> syntax in the class definition.

- This is saying “This class takes a type parameter”

- Moreover, “if you use this class, you must tell me what type it will contain”

```java
class GenericStack<E> {
    private ArrayList<E> contents = new ArrayList<E>();

    public void push(E element) {
        contents.add(element);
    }

    public E pop() {
        int top = contents.size() - 1;
        E result = contents.get(top);
        contents.remove(top);
        return result;
    }
}
```
Use in your code

- Note on line 2, we can use this type parameter, to a limited extent, as if it were a class.

- We can pass it to the ArrayList as its type parameter.

- In other words, <E> says “There exists a type E of which this class and the ArrayList will contain”
Use in your code

- Throughout the rest of the class we can use E as if it were a class (but we cannot call any methods on it, since it could be anything)

```java
public class GenericStack<E> {
    private ArrayList<E> contents = new ArrayList<E>();

    public void push(E element) {
        contents.add(element);
    }

    public E pop() {
        int top = contents.size() - 1;
        E result = contents.get(top);
        contents.remove(top);
        return result;
    }
}
```
Use in your code

- Note line 19

- We define an instance of our GenericStack and give a “type parameter of String
Use in your code

- Note line 20
- We ‘push’ a String onto our stack.
  All is well.
+ Use in your code

- Note line 21
- We attempt to ‘push’ an int onto our GenericStack<String>
- What happens?

```java
class GenericStack<E> {
    private ArrayList<E> contents = new ArrayList<E>();

    public void push(E element) {
        contents.add(element);
    }

    public E pop() {
        int top = contents.size() - 1;
        E result = contents.get(top);
        contents.remove(top);
        return result;
    }
}

// Somewhere else in the codebase...

// This is a compile error
GenericStack<String> gStack = new GenericStack<String>();
gStack.push("String");
gStack.push(123); // <-- here
String str2 = (String) ngStack.pop();
```
Use in your code

- Note line 21
- We attempt to ‘push’ an int onto our GenericStack<String>
- What happens?
  - Compile error.
- It is impossible to the wrong type in our Stack!

```java
class GenericStack<E> {
    private ArrayList<E> contents = new ArrayList<E>();

    public void push(E element) {
        contents.add(element);
    }

    public E pop() {
        int top = contents.size()-1;
        E result = contents.get(top);
        contents.remove(top);
        return result;
    }
}

// Somewhere else in the codebase...

// This is a compile error
GenericStack<String> gStack = new GenericStack<String>();
gStack.push("String");
gStack.push(123); // --- here
String str2 = (String) ngStack.pop();
```
Programming Example

- Lets convert our StackOfCharacters to a Stack that can be used for any type.
- See the stack package
Programming Example

- Lets convert our LinkedList of String to a LinkedList that can be used for any type.
- See the linkedlist package