lecture 22
Java interfaces
- List
- Comparable
- Iterator/Iterable
- enhanced for loop

Abstract Data Types (ADTs)
- list
- tree
- stack
- binary search tree
- queue

They allow us to ignore implementation details.

"Java API"
(application program interface)
- We are not given "the" source code for Java classes such as LinkedList, Math, etc.
- Rather, we are only given "signatures", some fields, and comments about what the methods do.

Concept of an interface

interface List<T> {
    void add(T)
    void add(int, T)
    T remove(int)
    boolean isEmpty()
    T get(int)
    int size()
}

Class ArrayList<T> implements List {
    void add(T t) {
    }

    // each of the methods of List must be implemented
}
Class LinkedList\(<T>\) implements List{
    void add(T t) {
    }
  // each of the methods of List must be implemented
}

Why is this useful?
Sometimes you want your code to be as general as possible, e.g. to allow a LinkedList or ArrayList.

Example
void mergesort(List<\(<T>\) L) {
    ...
}

Example
interface Shape {
    float getArea();
    float getPerimeter();
}

Class Rectangle implements Shape {
    float height, width;
    Rectangle (float h, float w) {
        height = h; weight = w;
    }
    float getArea() {
        return height * width;
    }
    float getPerimeter() {
        return 2*(height + width);
    }
}

Class Circle implements Shape {
    float radius;
    Circle (float rad) {
        radius = rad;
    }
    float getArea() {
        return Math.PI * radius * radius;
    }
    float getPerimeter() {
        return 2 * Math.PI * radius;
    }
}

Class Rectangle implements Shape {
    float height, width;
    Rectangle (float h, float w) {
        height = h; weight = w;
    }
    float getArea() {
        return height * width;
    }
    float getPerimeter() {
        return 2*(height + width);
    }
}

Shape s = new Rectangle(30,40);
...
s = new Circle(2.5);
...

More about this in the next few lectures.
interface Comparable<T> { 
    int compareTo(T t) 
}

// recall binary search trees 

t1. compareTo(t2) 

returns { 
    0, t1.equals(t2) 
    +, t1 > t2 
    -, t1 < t2 
}

we could have compared Rectangles instead by: 
- perimeter 
- length of longest side 
- etc

Each of these would require a different implementation of compareTo().

Class Rectangle implements Shape, Comparable { 
    
    int compareTo(Rectangle r) { 
        float diff = this.getArea() - r.getArea(); 
        if (diff > 0) 
            return 1; 
        else if (diff == 0.0) 
            return 0; 
        else return -1; 
    } 

} 

see lecture notes for discussion.

interface Iterator<T> { 
    boolean hasNext(); 
    T next(); 
}

Q: What do Iterator objects do? 
A: They use the above methods to "step through"/visit/iterate over the objects in some collection.
Where do iterator objects come from? Which objects construct them?

Linked List, ArrayList, BST, ... have a method that constructs an Iterator object.

```java
interface Iterable<T> {
    Iterator<T> iterator();
}

In my opinion, iterator() should have been called makeIterator().
```

See example code:
- Lecture 4: Singly linked lists
- Lecture 21: Binary search trees

When are iterators useful?
- 'for' or 'while' loop might not be appropriate (you cannot break out and then return back inside)
- You might need several iterators at same time

```java
ArrayList<T> list = new ArrayList<T>();
Iterator<T> iter = list.iterator();

// implementation is hidden

iter

constructed by the ArrayList object

```
LinkedList<T> list = new LinkedList<T>;

Iterator<T> iter1 = list.iterator();
iter1.next();

Iterator<T> iter2 = list.iterator();
iter2.next();
iter2.next();

LinkedList<String> list = new LinkedList<String>;

...for (String s : list) {
    System.out.println(s);
}

BST<Integer> binsearchTree = new BST<Integer>;
...

for (Integer k : binsearchTree) {
    System.out.println(k);
}

Note: the enhanced for loop (implicitly) uses the iterator method

Java "enhanced for loop"
Java language feature (1.5+)
that allows you to step through the elements of any collection class that implements Iterable.