COMP 250
Lecture 29
interfaces
Nov. 18, 2016
ADT (abstract data type)

ADT’s specify a set of operations, and allow us to ignore implementation details. Examples:

• list
• stack
• queue
• binary search tree
• priority queue (heap)
• hash map
• graph
Java API

API = application program *interface*

Gives class methods and some fields, and comments on what the methods do.  e.g.

https://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html
Java interface

• reserved word  (don’t confuse with “I” in API)

• like a class, but only the method signatures are defined
Example: List 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<T> {

void add(T) { .... }
void add(int, T) { .... }
T remove(int) { .... }
boolean isEmpty() { .... }
T get(int) { .... }
int size() { .... }
}

Each of the List methods are implemented. (In addition, other methods may be defined and implemented.)
class LinkedList<T> implements List<T> {

    void           add(T)          { .... }  
    void           add(int,  T)   {  ....  }
    T                 remove(int)   {  ....  }
    boolean  isEmpty()     {  ....  }
    T                 get( int )    {  ....  }
    int                 size()       {  ....  }

    :  

}

Each of the List methods are implemented. (In addition, other methods may be defined and implemented.)
Example: how are Java interface’s used?

List<String> list;

list = new ArrayList<String>();
list.add("hello");

list = new LinkedList<String>();
list.add( new String("hi") );
Example: how are Java interface’s used?

```java
void someUsefulMethod(List<String> list) {
    list.add("hello");
    list.remove(3);
}
```

The method can be called with a LinkedList or an ArrayList as the parameter.
Example: user defined interface

interface Shape {
    double getArea();
    double getPerimeter();
}

class Rectangle implements Shape{
    :
}

class Circle implements Shape{
    :
}
class Rectangle implements Shape{

    double height, width;

    Rectangle( double h, double w ){
        height = h;    weight = w;
    }

    double getArea(){    return height * width;   }

    double getPerimeter(){    return 2*(height + width);  }

}
class Circle implements Shape{

    double radius;

    Circle(double r) {
        radius = r;
    }

    double getArea() {
        return MATH.PI * radius * radius;
    }

    double getPerimeter() {
        return 2 * MATH.PI * radius;
    }
}

Shape \ s = \ new \ Rectangle( \ 30, \ 40 \ );

\ s = \ new \ Circle( \ 2.5 \ );

See Assignment 4 for a similar example....
Java Comparable interface

Suppose you want to define an ordering on instances of some class, and possibly allows some instances to be “equal”.

Sorting, binary search trees, priority queues (heaps) each require a well defined ordering (“ < “).
Java Comparable interface

interface Comparable<T> {
    int compareTo(T t);
}

e.g. The class String implements Comparable<String>.

Q: What does that mean?
Java Comparable interface

T implements Comparable<T>

T t1, t2;

Java API recommends that t1.compareTo(t2) returns:

\[
\begin{align*}
0, & \quad \text{if } t1.equals(t2) \text{ returns true} \\
\text{positive number}, & \quad \text{if } t1 > t2 \\
\text{negative number}, & \quad \text{if } t1 < t2
\end{align*}
\]
Example: Rectangle

Q: When are two Rectangle objects equal?
A: Their heights are equal and their widths are equal.

Q: How can we define a `compareTo()` method for ordering Rectangle objects?
class Rectangle implements Shape, Comparable{

    boolean equals( Rectangle r ) {
        return (this.height == r.height) && (this.width == r.width);
    }

    int compareTo( Rectangle r ){
        double diff = this.getArea() - r.getArea();     // arbitrary

        if (diff > 0)
            return 1;
        else if (diff == 0.0)
            return 0;
        else return -1;
    }
}
}     // not consistent with Java API recommendation
class Rectangle implements Shape, Comparable{

    boolean equals( Rectangle r ) {
        return this.getArea() == r.getArea();
    }

    int compareTo( Rectangle r ){
        double diff = this.getArea() - r.getArea();   // arbitrary

        if (diff > 0)
            return 1;
        else if (diff == 0.0)
            return 0;
        else return -1;
    }

}       // Consistent with Java API recommendation
Q: If a class implements Comparable, then why would we need an equals() method?

A1: Every class has an equals() method.

The default is that `o1.equals(o2)` if `o1` and `o2` are the same object, i.e. “==“.

A2: The equals() method is called by Java library methods, so you can’t just rely on `compareTo()`.
Java Iterator interface

Motivation 1: we often want to visit all the objects in some collection. But sometimes this is awkward to do.

e.g. LinkedList<T>.get(i)

BST_Node<T>.getSuccessor() // not covered
Java Iterator interface

Motivation 2: We sometimes want to have multiple “iterators”.

Analogy: Multiple TA’s grading a collection of exams.
Java Iterator interface

interface Iterator<T> {
    boolean hasNext();
    T next(); // returns current and advances to next
    void remove(); // optional
}
private class SLL_Iterator<T> implements Iterator<T>{  // the client doesn’t need to know the name.

    private SNode<T> cur;

    SLL_Iterator(SLinkedList<T> list){  // constructor
        cur = list.getHead();
    }

    public boolean hasNext() {
        return (cur != null);
    }

    public T next() {
        SNode<T> tmp = cur;
        cur = cur.getNext();
        return tmp.getElement();
    }
}
Java Iterator interface

Q: Who constructs the Iterator object for a collection?
A: The collection does it.

  e.g. LinkedList, ArrayList, HashSet ...

How?
If a class implements `Iterable`, then the class has an `iterator()` method.
The iterators iterate over LinkedList nodes, not Shapes. The next() method returns Shapes.
The iterator() method calls the constructor of the SLLIterator class.
ASIDE: Java enhanced for loop

For any class that implements Iterable..

Example:

```java
LinkedList<String> list = new LinkedList<String>();
...

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