Intro to Data Structures

Lecture #11 – Implementing a generic Linked List
September 23, 2014

Mark Stehlik

Outline for Today

- Return & discuss HW2
- Handout midterm on Thursday
- Implementing a generic Linked List class

HW2 debrief

- What about 50? (check bounds/"magic number")
- Helper methods and separation of concerns…
- Issues with removePerson?

```
for (int i = index; i < numContacts-1; i++)
        contacts[i] = contacts[i+1];
```

  No error message
- Error messages in general
- Using booleans as flags, not ints…

Booleans for flags, not ints

```
public void changeNumber(String id, int newNumber)
{
    int a = 120;
    for(int i = 0; i<numPeople; i++)
    {
        if(contacts[i].getID().equals(id))
        {
            contacts[i].updateNumber(newNumber);
            a = 100;
        }
    }
    if (a= =120)
    {
      System.out.println("This person is not in your contacts");
    }
}
```

Implementing a generic Linked List class

- What do we need?
  - in general, what is in a class?
    - fields (data)
    - methods (things that operate on/use/display the data)
  - so which should I think about first…
    - fields - what does an object of this class need to store?
    - then methods
      - first the constructor(s)
      - then toString (why?)
      - then the rest (start with the easy ones)

Implementing a generic Linked List class

- A reminder about visibility
  - visibility determines who can access what
  - 4 levels of visibility in Java (3 explicit, plus default):
    - public - everyone can see/use
    - protected - inside class, inside package, inside subclass
    - *package-private* (default, <u>no modifier used</u>) - inside class, inside package
    - private - inside class
  - Classes can either be public or *package-private*
  - fields/methods can be all 4 (fields usually private; methods public)

Implementing a generic Linked List class

- OK, so for a Linked List, what do I need to store?
  - A reference to the first (front, initial) Node
  - And what's in a node?
    - data
    - a reference to the next node in the list
  - So we could use the ListNode class from Lecture10, but…
  - Isn't it the case that the ListNode is an inherent, essential (and internal) part of a Linked List?

Implementing a generic Linked List class

- So what if we actually did that – made it an internal part of the Linked List class? What would that mean?
  - where should the Node class be declared and what is its visibility?
  - and what type should its data be?
    - Any type!!
    - but how to do that???

Implementing a generic Linked List class

- We will implement most of the methods we saw from the ArrayList (and a few others):
  - boolean isEmpty(); //true if *list* is empty; false if not
  - void addFirst(value); //adds at front
  - String toString(); //obvious (use StringBuilder instead of String; why?)
  - int size(); //returns the number of elements in *list*
  - void clear(); //removes all elements, *list* is empty
  - boolean contains(value); //true if value in *list*; false if not
  - AnyType get(index); //returns the element at index
  - boolean add(value); //adds value at end of *list*
  - int indexOf(value); //returns first index of value; -1 if not
  - boolean remove(value); //removes first occurrence of value