Sunday, February 23, 2014

/*MIDTERM 1 REVIEW*/
CS61B Lecture 1

Object-Oriented Programming

Object: A repository of data
Class: Type of object
Method: procedure of function that operates on an object
    ex: addItem: adds an item to any list type object
Inheritance: a class can inherit properties from a base/ parent class
    ex: ShoppingList inherits properties from a List class, which has the property of storing a sequence of items
Polymorphism: One method works on several classes, even if the classes need different implementations (of that method)
Object-Oriented: Each object knows its own class and methods

Java

variables: you must declare them and their type
    python: x = 1   scheme: (let ((x 1)))
    Java:

```
int x;
x = 1;
```

int x does 2 things:
1. allocates memory to store integer, type "int"
2. gives the chunk of memory a variable name "x"

    x [ ]

x = 1 ======> x [1]

variables are used to reference object
1. Use one defined by someone else
2. create your own

```
string myString                    myString [ ]
```

constructor: new String()

```
myString = new String()
```

    SIGNATURE: String()
    CALL: new String()

"=" is an assignment, causes myString to reference the string object constructed
myString [ ] ——————-> [empty string object]

Java programs must be compiled before you can run them
Java Program (.java)—>java_compiler—>".class" files—>run JVM on class files
JVM: java virtual machine

Objects and Constructors:

```
string s;          //declare a string variable
s = new String();       //construct empty string, assign it to s
OR
string s = new String();       //both steps combined
```

s = “Yow!”; //old reference is replaced by new “Yow!”
if nothing else is pointing to old object, garbage collection will clean up the memory spot of the old object

string s2 = s //s2 now points at “Yow!” (same object that s points to)

s2 = new String(s) //Constructor makes copy of “YOW” (same word, diff obj)

s2 now points to a different object than s, though the objects are copies

1. Looks where s points
2. reads string object
3. constructs new string with copy of characters
4. makes s2 reference the new string just created

3 Constructors:

1. new String(): constructs empty string (zero characters)
2. “Yow!”: specifies string (quotation construction)
3. new String(s): takes a parameter s, makes copy of object that s references

constructors always have same name as their class, except the quotation constructor

Methods

s2 = s.toUppercase();

s2 [ ]——>[YOW!]

string s3 = s2.concat(“!!”);

s3 = s2 + “!!”;

Strings are immutable, once a string is made, only modified copies can be constructed, but you cannot change the original string (contents never change)

I/O classes and Objects

objects in System class for interacting with a user:
System.out is a PrintStream object that outputs to the screen
System.in is an InputStream object that reads from the keyboard
readLine is defined on BufferedReader objects
BufferedReader are constructed with an InputStreamReader
InputStreamReader is constructed with an InputStream such as System.in
(Look in JAVA API libraries— java.io)
InputSream objects reads raw data
InputStreamReader composes raw data into characters
characters in java are 2 bytes long
BufferedReader compose characters into an entire line of text

EXAMPLE JAVA PROGRAM:

```
import java.io.*;
class SimpleIO {
        public static void main (String[] arg) throws Exception {
                BufferedReader keybd =
                        new BufferedReader(new InputeStreamReader(System.in))
                System.out.println(keybd.readLine());
        }
}
```

CS61b Lecture 3

Defining Classes

Fields: variables stored in objects
also known as INSTANCE VARIABLES
does not have parentheses after
ex:
amanda.age (field/instance variable)
amanda.introduce() (method call)

CLASS DEFINITION: (with field declarations and method definition)

```
class Human {
        public static int numberOfHumans;
        public int age;
        public String name;
        public void introduce() {
                System.out.println("I'm "+name+" and I'm "+age+" years old.");
        }
        public void copy(Human original) {
                age = original.age;
                name = original.name;
        }
        public Human(String givenName) {
        age = 6;
        name = givenName;
        }
        public Human() {
                numberOfHumans++;
        }
}
```

method definition knows that NAME refers to amanda.name

Constructing a Human Object:

```
Human amanda = new Human();
amanda.age  = 6;
amanda.name = "Amanda";
amanda.introduce()
```

Using human.copy() Method: copy method require HUMAN type input

```
amanda.copy(rishi);
```

Class Constructor:

```
public Human(String givenName) {
	age = 6;
	name = givenName;
}
```

Using Constructor:

```
Human amanda = new Human("Amanda");
```

When no constructor is defined in a class, Java will automatically create a default constructor that takes no parameters. Does not initialize any fields (null, 0, etc.):

```
public Human() { }
```

Writing a new constructor will make the default constructor unusable; if you want to use it still you will need to explicitly write a new default constructor again (like above).
Constructor methods need the NEW keyword in order to work, since they are not called on an object.

THIS keyword

amanda.introduce implicitly passes the object (amanda) as a parameter called "this"

```
public void change(int age) {
	String name = "Tom";
	this.age = age;
	this.name = name;
}
```

must distinguish the field name this.age (amanda) from the local name "tom", and the field age this.age from the local parameter age (parameter of change method)
IMPORTANT: you cannot change the value of "this"

```
	this = amanda; ————> compile time error
```

STATIC keyword

a static field is a single variable charred by a whole class of objects; called a class variable

Lecture 1-29-2014 Number 4-5

Primitives

Byte: 8 bit integer -128 to 127
short : 16 bit
long: 64-bit
double: A 64-bit floating point number 18.3555

float: 32 bit
boolean: true or false
char: a character
int: 32 bit

long x = 43L;
double & float values must have a decimal point
double y = 18.0;
float f = 43.9f;

	Object Type	**Primitive Types**
Contains	reference	values
Defined	class definition	built into Java
Created	"new" keyword	"6", "3.4", "true"
Initialized	constructor	default "zero" value string: empty int: 0
Used	method	operators (+, *)

char c= 'h';

Object Types vs Primitive Types

number operators:
-x x*y x+y x/y x-y
integer operators:
x%y (remainder)
x/y truncating division (not true division)

java.lang library:
-Math class
x = Math.abs(y); x = Math.sqrt(y);
-Integer class
int x = Integer.parseInt("1984");
double d = Double.parseDouble("3.14");
Integers can be assigned to variables of longer types

```
int i = 43;
long l = 43;
l = i
i = l           //Compiler ERROR
i = (int) l;    //Cast will compile correctly
```

Boolean Values

"true" or "false"

a	b	a&&b	a‖b	!a
f	f	f	f	t
f	t	f	t	t
t	f	f	t	f
t	t	t	t	f

Boolean values can be created by comparison operators:

"==" "<" ">" "<=" ">=" "!="

```
boolean x  = 3 == 5;           //x is false
x = 4.5 >= 4.5;        //true
x =  4 != 5 -1;        //false
x = false == (3 == 0);         //true
```

Conditional Statements

```
if (boolValue) {
        statements;
}
```

Boolean value must be in parentheses

```
boolean pass = score >= 75;
if (pass) {
        output("You pass CS61b")
} else {
        output("You are such an unbelievable loser.")
}
```

if-then-else clauses can be Nested and Daisy chained

find maximum of 3 numbers

```
        if (x>y) {
                if (x > z) {
                        max = x;
                } else {
                        max = z;
                }
        } else if (y > z) {
                max = y;
        } else {
                max = z;
        )
```

Switch Statements

```
switch (month) {
case 2:
        days = 28;
        break;
case 4:
case 6
case 9
case 11:
        days = 30;
        break;
default:
        days = 31;
        break;
}
```

Return Keyword

return causes a method to end immediately. Control returns to the calling method
Print numbers from 1 to x:

```
public static void oneToX(int x) {
        if (x < 1) {
                return;
        }
        oneToX(x-1)
        System.out.println(x);
}
```

While Loops

```
public static boolean isPrime(int n) {
        int divisor = 2;
        while (divisor < n) {
                if (n % divisor == 0) {
                        return false;
                }
                divisor ++;
        }
        return true;
}
```

“divisor < n” is called the loop condition. The loop will not be executed if the condition is false right away. The code inside the braces created by the loop condition is called the loop body.
Iteration: a pass through the loop body.

For Loop

```
for (initialize; condition; next) {
        statements;
}

EQUIVALENT TO WHILE LOOP
initialize;
```

```
while (condition) {
	statements;
	next;
}


public static boolean isPrimeFor (int n) {
	for (int divisor = 2; divisor<n; divisor++) {
		if (n % divisor == 0) {
			return false;
		}
	}
	return true;
}
```

Arrays

an object storing a numbered list of variables
Each is a primitive type or reference

```
char[] c;
c = new char[4];
c[0] = ‘b’;
c[3] = ‘e’
c[4] = ’s’;	///RUNTIME ERROR
```

Field “c.length”
	cannot assign a value to length (compile time error)

Primes

Algorithm: Sieve of Eratosthenes
public static void printPrimes(int n)

Multi-dimensional Arrays

2D array: array of references to arrays

```
public static int[][] pascalTriangle(int n) {
	int[][] pt = new int[n][]
		for (int i = 0; i < n; i++) {
			pt[i] = new int[i+1];
			pt[i][0]=1;
			for (int j = 1; j<i;j++) {
				pt[i][j]=pt[i-1][j-1]+pt[i-1][j];
			}
			pt[i][i]= 1;
		}
		return pt;
	}
```

Sunday, February 23, 2014

CS61b 6

Automatic Array Construction

```
int[][] table = new int[x][y];
```

table [ ]———>[x “rows”]——>[y “columns”]
x arrays of y ints, all in one array called table
Initializers:

```
	Human[] b = {amand, rishi, new Human(“Paolo”)};
```

automatically constructs a 3 Human array

```
	int[][] c = {{7,3,2},{x},{8,5,0,0},{y+z, 3}};
```

creates non rectangular array
CANNOT construct array with {} for a variable already declared

```
	d = {1,2,3};	///COMPILER ERROR
```

Multi-dimensional array declaration:

```
int[] a, b, c;	//all reference arrays
int a[], b, c[][];		//a is 1D; c is 2D; b is not array
int[] a, b[];	//a is 1D; b is 2D
```

Array of Objects

when you construct array of objects, java does not construct objects
Objects are null until initialized

Main() parameters

You can use command line arguments in terminal when running the main method for a class. The parameter for main() is a String[] array of args

```
class Echo {
	public static void main(String[] args)
		for (int i = 0; i < args.length; i++) {
			System.out.println(args[i]);
		}
	}
}
```

```
unix prompt% java Echo kneel and worship Java
kneel
and
worship Java
```

Do Loop

do always executes loop body at least once

```
do {
	s = keybd.readline()
	process (s);
} while (s.length()>0);
```

useful for when you need to check a condition after a loop has gone through and initialized some variables

Break and Continue statements

break exits the inner most loop or “switch” enclosing the “break”
“TIME-AND-A-HALF” loop

```
s = keybd.readLine();
while (s.length() > 0) {
        process(s);
        s = keybd.readLine();
        }
OR
while (true) {
        s = keybd.readLine();
        if (s.length() == 0) {
                break;
        }
        process(s);
}
```

CONTINUE

1. only applies to loops
2. jumps to end of loop body but another iteration can happen if loop condition

for loop incrementer will still execute after a continue jump

Constants

“final” keyword used for values that can never change (a constant)
array x, x.length is a final field.

Lecture 7-8

List

Array-based lists:

fast access to each item
difficult to insert items at beginning or middle; takes time proportional to length of list

***FINISH IMPLEMENTATION AFTER LECTURE NOTES UPLOADED

```
public class List {
        int[] a;
        int lastItem;

        public List() {
                a = new int[10];
                lastItem = -1;
        }
        public void insertItem(int newItem, int location) {
                int i;
                if (lastItem+1 ==a.length) {
                        int[] b = new int[2*a.length];
```

```
for (i =0;          ;          ) {
}
```

Linked List

Recursive Data type
Made up of nodes. Each node has:
    -an item   ("car")
    -a reference to next node in list   ("cdr")

```
public class ListNode {
        //FIELDS
        public int item;
        public ListNode next;

        //CONSTRUCTORS
        public ListNode(int i, ListNode n) {
                item = i;
                next = n;
        }f
        public ListNode(int i) {
                this(i, null);
        }

        //METHODS
        public void insertAfter(int item) {
                next = new ListNode(item, next);  //////old next reference is given to new node
}


ListNode l1 = new ListNode, l2 = new ListNode(), l3 = new ListNode();
l1.item= 7;
l2.item=0;
l3.item=6;
l1.next = l2;
l2.next = l3;
l3.next = null;
OR
ListNode l1 = new ListNode(7, new ListNode(0, new ListNode(6)));

l2.insertAfter(3);
```

Inserting items into linked lists takes constant time, if you have a reference to a previous node (may not have constructed with references)
Linked Lists can keep growing until memory runs out.
Take O(n) time to find nth element in linked list

Lists of Objects

```
public class SListNode {
	public Object item;			//any object, no primitives
	public SListNode next;
```

Insert new item at beginning:
	x = new SListNode(“soap”, x);
Null

```
public class SList {
	private SListNode head;
	private int size;
	public SList() {
		head = null;
		size = 0;
	}
	public void inserFront(Object item) {
		head = new SListNode(item, head);
		size++;
	}
}
```

Public and Private

private method or field: invisible and inaccessible to other classes
	prevent data corrupting by other classes
	objects of the same class can access private variables from other objects
	improve implementation without causing other classes to fail

Software Engineering

Interface: how other classes interact with a class
	prototypes for public methods, public fields
	plus description of their behaviors

Abstract Date type: a class with a well defined interface, but implementation details are hidden

Invariant: a fact about a data structure that is always true
	ex: date abstract data type always stores a valid date
	invariants are enforced by allowing access only through method calls
Not all classes are ADTS. some classes just store data, without invariants (fields can be public)

SLIST ADT
slist enforces 2 invariants:
	1. size is always correct
	2. list is never circularly linked
these invariants are enforced because only SList methods can change the list
	1. Slist fields are private (head and size)
	2. no method of SList returns an SListNode

Doubly Linked Lists

Slist inserting and deleting at front of list is easy, but end of list takes a long time

```
class DListNode {
        Object item;
        DListNode next;
        DListNode prev;
}
class Dlist {
        private DlistNode head;
        privated DListNode tail;
        long size;
}
```

Insert and delete items at both ends in constant running time

Remove the tail node (at least 2 items in Dlist):

```
        tail.prev.next = null
        tail = tail.prev
```

Sentinel: a special node that does not represent an item

- item is null
- previous points to tail
- next is head
- sentinel circularly links lists

DList invariants (with sentinel):

1. for any DList d, d.head != null
2. for any DListNode x, x.prev !=null
3. for any DListNode x, x.next != null
4. for any DListNode x, if x.next == y, then y.prev == x
5. if x.prev = y, then y.next == x
6. size field is correct, does not include sentinel, accessible from sentinel with next

Empty DList: Sentinel's prev and next fields point to itself

```
public void removeBack() {
        if (head.prev != head) {
                head.prev = head.prev.prev;
                head.prev.bext = head;
                size- -;
```

Lecture 2-10 Number 9

Reading Notes: pp. 77, 235–239, 258–265, 663

Java is pass-by-value, meaning that methods copy the value of arguments passed

Project 1 is due February 22

The Stack ad the Heap

The heap stores all object, including all arrays and all class variables
The stack stores all local variables, including parameters.
When the JVM calls a method, it puts the call in a stack frame (aka activation record) on top of the stack, with all its local variables and parameters stored
JVM keeps an internal stack of stack frames. The top of the stack is whatever is executing at that instant in the program.

WHEN A METHOD FINISHES, ITS STACK FRAME IS ERASED. You must store internal values as an object if you want it saved.

Method “Thread.dumpStack()” useful for debugging (ie finding illegal parameter passing)

Parameter Passing

Like in scheme, Java passes all parameters by value: meaning that arguments are copied into the formal parameters, originals are not changed. You get a copy in the stack frame. Changing it won’t change the original. See IntBox{}

When the parameter is a reference, the reference is copied, but the object is shared.

Binary Search (and Recursive Stack Frames)

When a method calls itself, JVM creates multiple stack frames, but only the top stack frame is accessible.
Binary Search searches a sorted array. ONLY WORKS ON SORTED ARRAY
Recursive base cases:

1. search element is the middle element, return index
2. subarray of length zero: return Failure

```
public static final int FAILURE = -1;
private static int bsearch(int[] i, int left, int right, int findMe) {
        if (left > right) {
                return FAILURE;
        }
        int mid = (left + right)/2;
        if (findMe == i[mid]) {
                return mid;
        } else if (findMe < i[mid]) {
                return bsearch(i, left, mid - 1, findMe);
        } else {
                return bsearch(i, mid+1, right, findMe);
        }
}
public static int bsearch(int[] i, int findMe) {
        bsearch(i, 0, i.length -1, findMe);
}
```

WRAPPER FUNCTION: a method that calls another method that makes it easier to use that second method with a simpler interface. bsearch(int[], int) is a wrapper method of bsearch(int[], int, int, int)

Takes log2n bsearch calls to reduce the problem down to one base case.
Logarithms: 4.1.2 and 4.1.7 of GT

Java has enough stack space for a few thousand stack frames.

Lecture 10 and 11

Inheritance

```
public class TaiList extends SList {
//"head" and "size" inherited from SList
private SListNode tail;
}
```

TaiList is a <u>subclass</u> of SList. SList is the <u>superclass</u> of TailList.
A subclass can modify a superclass in 3 ways:

1. It can declare new fields.
2. Declare new methods.
3. override old methods with new implementations

```
public void insertEnd(Object obj){
        ///solution to lab3
   SListNode node = new SListNode(obj);
   if (head == null) {
     head = node;
     tail = node;
   } else {
     tail.next = node;
     tail = node;
     }
   size++;
}
```

isEmpty(), length(), nth(), toString() are inherited from SList.

Inheritance and Constructors

Java executes TaiList constructor, which calls SList() constructor before it executes any code.

```
public TailList() {
        // SList() sets size = 0, head = null
        tail = null;
}
public TailList(int x) {
        super(x);      ///Must be first statement in constructor
        tail = null;
}
```

Invoking Overridden Methods

```
public void insertFront(Object obj) {
        super.insertFront(obj);
        if (size == 1) {
                tail = head;
        }
}
```

“Protected” Keyword

protected variables let subclasses see superclass variables. A proceed field is visible to declaring class and all its subclasses, but no others. (Private fields are not visible to subclasses). Protected fields are READ and WRITE for subclasses.

```
public class SList{
        protected SListNode head;
        protected int size;
}
```

Class Hierarchies

Object:

1. Primitives: string, etc.
2. User Defined Classes: Worker
    - i. Proletariat
        - a. Student
        - b. TA
    - ii. Bourgeoisie
        - a. Professor

Dynamic Method Lookup

Every TailList is an SList

```
SList s = new TailList();          //works fine
TailList t = new SList();          //COMPILE TIME ERROR
```

Static type: type of a variable

Dynamic type: the class of the object that the variable references.

When we invoked overridden method, Java calls method for object’s dynamic type, regardless of static type.

```
SList s = new TailList();
s.insertEnd(obj);      ///Calls TailList.insertEnd()
s = new SList();
s.insertEnd(obj);      ////Calls SList.insertEnd()
```

Why dynamic method lookup matters?

Method that sorts an SList using only SList method calls now sorts TailList TailLists too.

Subtleties of Inheritance

1. New method in TailList - eatTail()

```
TailList t = new TailList();
t.eatTail();
Sist s = new TailList();
s.eatTail();                //Compiler Error
```

Cannot compile because not every SList has an "eatTail()" method. Java can't use dynamic method lookup on s.

2.

```
SList s;
TailList t = new TailList();
s = t;
t = s;              ///Compile time error. SList cannot be stored in the subclass TailList
t= (TailList) s;            //Cast will work
s = new SList();
t = (TailList) s;           ///RUNTIME ERROR: ClassCastException

int x = t.nth(1).intValue();    ///Compile-time error. Not ever object has an intValue()
int y =((integer)  t.nth(1)).intValue();        //casts nth object as an integer.
```

3. "instanceof" operator. Operators are in lower case. This operator tells you whether the object is of a specific class

```
if (s instanceof TailList) {
        t = (tailList) s;
}
```

false is s is null or doesn't reference a tail list. true if s is a reference to a tai list or a subclass of tail list.

Equals()

Every class has an equals() method. It is a method of the object class, so unless the equals method is user defined, it will be inherited from Object.

```
Default:        inherit Object.equals()
                r1.equals(r2)           /// same as r1==r2
                        runtime error if r1 is null
```

Many classes override equals() to compare content.

```
int i1 = 7;
int i2 = 7
(i1 == i2)              ///FALSE
i1.equals(i2)           ///returns TRUE
i3 =  i2;
(i3 == i2)              ///True
i3.equals(i2)           //True
```

Four degrees of equality:

1. Reference equality, == (do they reference the same object)
2. Shallow structural equality: two objects are .equals() when all of their fields are ==
3. Deep structural equality: two objects are .equals() if their fields are .equals()
4. Logical Equality. Ex:
    a. "Set" objects are .equals if they contain the same elements (even in different order)
    b. Fractions 1/3 and 2/6 are .equals()

equals() may test any of these 3 levels.

Deep Structural Equality for SLists (Iterative Algorithm)

Signature must be the same as Object.equals(). Don't make parameter SList!

```
public class SList {
        public boolean equals (Object other) {
                if (!(other instanceof Slist)){
                        return false;
                }
                SList o = (SList) other;
                if (size != o.size) {
                        return false;
                }
                SListNode n1 = head;
                SListNode n2 = o.head;
                while (n1 != null){
                        if (!n1.item.equals(n2.item)) {
                                return false;
                        }
                        n1 = n1.next;
                        n2 = n2.next;
                }
                return true;
        }
}
```

Only works when SList invariants are met.

For Each Loops

Iterates through array elements.

```
int[] array = {7,12,3,8,4,9}
for (int i : array) {
        System.out.print(i+" ");
```

i takes on the values of the array

Testing

1. Modular Testing- test each method separately.
    Test drivers and stubs:
    a. Test driver calls the code, check results. Must be inside the class where the methods are declared. This way, test driver can check private methods.
    b. Stubs: bits of code that are called by the code being tested.

2. Integration testing: testing all components together.
    -Define interfaces well
    - Learn to use a debugger
3. Result Verification
    a. Data structure integrity checkers.
        Inspects a data structure and verifies that all invariants are satisfied
    b. Algorithm result checkers (i.e. sort verification)

<u>assertion</u>: code that tests an invariant or a result. Boolean value

```
assert x==3;
assert list.size ==list.countLength():
    “wrong SList size: “+list.size;
```

To turn on assertions:
```
java -ea
```
Turn off: (better speed)
```
java -da
```
(countLength is never called)

Lecture 12 & 13

Abstract Classes

```
public abstract class List{
    protected int size;
    public int length() {
        return size;
    }
    public abstract void insertFront(Object item);
}
```

abstract method lacks an implementation.

```
public class SList extends List {
    //inherits “size”
    protected SListNode head;
    //inherits “length()”
    public void insertFront(Object item) {
        head  = new SListNode(item, head);
        size++;
    }
}
```

A non-abstract class may never contain an abstract method or inherit one without providing an implementation.

```
List myList = new SList():
myList.insertFront(obj);
```

Sunday, February 23, 2014

Abstract classes provide a reliable interface:
multiple classes can share
without defining any of them yet

```
public void listSort(List L) {…}
```

List L can be any subclass of List: SList, DList, TailList, TimedList, etc.…
TimedList: records time spent on operations
TransactionList: logs all changes on a disk

Java Interfaces

interface: public fields, method prototypes and behaviors.
java interface: interface keyword

Java interface is like an abstract class. 2 differences:

1. A class can inherit from only one class. You can implement (inherit) many java interfaces.
2. A java interface cannot implement any methods or include and fields except “final static” constants.

```
public interface Nukable {                    /// in Nukeable.java
        public void nuke();                   /// java assumes the method is abstract
}
```

java interface only takes abstract methods

```
public interface Comparable {          // in java.lang
        public int compareTo(Object o);
}

public class SList extends List implements Nukeable, Comparable {
        public void nuke() {
                head = null;
                size = 0;
        }
        public int compareTo(Object o) {
                [Returns a number < 0 if this < object o. Returns 0 if this.equals(o). Returns # > 0
                if this > object o.
        }
}

Nukeable n = new SList();
Comparable c = (Comparable) n;
```

Arrays class in java.util sorts arrays of Comparable objects

```
public static void sort(Object[] a)
```

A subinterface can have multiple superinterfaces.

```
public interface NukeAndCompare extends Nukeable, Comparable {}
```

WHEN A SUBINTERFACE uses superinterfaces, use the **extends** keyword. When classes use any kind of interface, they use **implements**

Java Packages (Not on midterm 1, useful for hw and projects)

packages: a collection of classes and Java interfaces and sub packages

1. packages can contain hidden classes not visible outside package
2. classes can have fields and methods visible inside package only
3. Different packages can have classes with same name.

Examples of packages:

1. java.io
2. Homework 4 uses list package containing

Using Packages: use a fully qualified name

```
java.lang.System.out.println(" ");
```

```
import java.io.*                    // imports everything in java.io package
```

every program imports java.lang.*

package is a protection level, provided by default if not declared private or protected.

A class or variable with package protection is visible to any class in the same package, but not outside (outside the directory where the package is stored.

Homework 4: DListNode class is public (not package protected). We want applications to hold DLists, but not change them. "prev" and "next" will still have package protection;

Public class must be declared in file named after class. "package" classes can appear in any .java file.

Compiling and running must be done from outside pacakage.

```
javac -g list/SList.java
java list.Slist
```

Iterators

```
public interface Iterator{
        boolean hasNext();
        Object next();
        void remove();
}
```

Sunday, February 23, 2014

An iterator is like a bookmark. Can have many iterators in the same data structure. Calling next() nth time returns nth item in sequence. Subsequent calls to next() throw an exception.

hasNext(): true iff more items to return in the list.

```
public interface Iterable {
        Iterator iterator();
}
```

DS.iterate() constructs a DSIterator for DS
“for each” loop iterates over item in data structure.

```
        for (Object o : l ) {
                System.out.println(o);
        }

        equivalent to:
        for (Iterator i = l.iterator()
```