Next Owl assignment up, due Friday (it’s short!)
Programming assignment due this morning
Next programming assignment up (probably) tomorrow, due next week.

Preregistration advice: More computing? Take 191B, or - if you’re really thriving in this class - 187 (come see me about this)
We’ve reached a new level of complexity in the applications we’ve studied -

There were several examples in chapter 9 that involved five classes, all coordinated by main.

So the class / object model as a tool for modularity is starting to pay off
However: the object / Class model, as we’ve seen it so far, is brittle:

If one class is only a slight variant of another -- e.g. a class like

WeighedInfant

(like Infant, but with new field: weight)

We need to start over!
Today: Inheritance: a class level concept

If classA’ is a variant of an existing class - class A - we’d like to reuse A as much as we can for the implementation of A’.

When the new class IS-A (an example of) the original class, we call this process Inheritance.

Sometime this extension is an obvious one, and involves only new attributes (note: we’ll see subtler ones..):

Person -> Student (add a gpa, year at school, student id, etc)

Rectangle -> colored Rectangle (add a color)

Vehicle -> motorized vehicle (yup: add a motor)
More concretely..

Suppose you have a Person class, with these attributes:

String firstName
String lastName
String SSN
int age
boolean female

-----

Now you want to create an Employee class, that’s just like Person, only an Employee has: an int EmployeeNumber and a boolean tbShot
I don’t want to touch the Person part of the class - it’s been debugged, tested, etc.

But: I DO want to retain the attributes of Person in the new class - HOW

But: I DO want to retain methods of Person - can I use them .. ? - HOW

Can I piggy-back on Employee constructor? - HOW

The big, big picture: *programming is theft*. A large OO program, say in Java, might use hundreds or even thousands of classes, a great many of which have been previously created, or are variants of classes that have been previously created.
class Book{
    int pages = 1500;

    // Prints a message about the pages of this book.
    public void pageMessage()
    {
        System.out.println("Number of pages: " + pages);
    }
}

// so: one attribute, one method
class Dictionary extends Book{
    int definitions = 52500;

    // Prints message using both local & inherited vals
    public void definitionMessage ()
    {
        System.out.println("Number of definitions: " +
                           definitions);

        System.out.println("Definitions per page: " +
                           definitions/pages);
    }
}
Terminology:

Book is the **base** class, Dictionary is the **derived** class

Book is the **super** class, Dictionary is the **sub** class

Dictionary **specializes** Book

---

Note: if we had said:

```plaintext
private int pages = 1500;
```

We would have to write

```plaintext
definitions / getPages()
```
public class Words
{

    public static void main (String[] args)
    {
        Dictionary webster = new Dictionary ();

        webster.pageMessage();       // an inherited method
        webster.definitionMessageMessage();
    }
}

class Book2
{
    int pages;

    public Book2 (int numPages)
    {
        pages = numPages;
    }

    public void pageMessage()
    {
        System.out.println("Number of pages: " + pages);
    }
}
class Dictionary2 extends Book2{
    private int definitions;

    public Dictionary2 (int numPages, int numDefinitions){
        super (numPages);
        definitions = numDefinitions;
    }

    public void definitionMessage (){{
        System.out.println ("Number of definitions: ",
                        definitions);
        System.out.println ("Defs per page: "+
                        definitions/pages);
    }
}
public class Words2
{
    public static void main (String[] args)
    {
        Dictionary2 webster = new Dictionary2 (1500, 52500);

        webster.pageMessage(); // from base class
        webster.definitionMessage(); // in derived class
    }
}
public class Car{

    private String make; // manufacturer
    private double fuelCapacity;
    private double fuelAmount;

    public Car(String what, double cap, double amt){
        make = what;
        fuelCapacity = cap;
        fuelAmount = amt;
    }
}
// the Car methods
public String getMake(){
    return make;
}
public double getCapacity(){
    return fuelCapacity;
}
public double getFuel(){
    return fuelAmount;
}

public void setFuel(double amt){
    fuelAmount = amt;
}

public double unusedCap(){
    return (fuelCapacity - fuelAmount);
}
public class UsedCar extends Car {

    private int year; // year of manufacture

    public UsedCar(String whatMk, double cap, double amt, int yr) {
        super(whatMk, cap, amt);
        year = yr;
    }

    public int getYear() {
        return year;
    }
}
public class Apartment{

    private String owner;
    private int size; // square feet

    public Apartment(String owner, int size){
        this.owner = owner;
        this.size = size;  }

    public int getSize(){ return size;}
    public String getOwner(){return owner;}

    public void setOwner(String newOwner){
        owner = newOwner;  }

    public String toString(){return (owner + " size: " + size); }
}
public class RentalApt extends Apartment{

    private String tenant;
    private boolean rented;

    public RentalApt(String owner, int size, boolean rented, String who){
        super(owner,size);
        tenant = who;
        this.rented = rented;
    }
}
public boolean getRented(){
    return rented;
}

public String getTenant(){
    return tenant;
}

public void setRented(boolean isRented){
    rented = isRented;
}

public void setTenant(String who){
    tenant = who;
}

public String toString(){
    String s = super.toString();
    return (s + " occupied? " + rented + " tenant: " +
            tenant);
}
}
In summary: When class A extends B

B’s instance variables come along (if private: can’t use directly)

B’s constructor can do some of the work of A’s constructor

B’s methods, if public, available directly in A

B’s methods, when overriding A’s methods, can use super.method() to do some of the work of B’s method
Java.util has a class called Random

It is built around a number generator

Random r = new Random();

Gives a new Random object

r.nextInt(); -> a random int

r.nextInt(10); -> a random int drawn from 0,1,2...7,8,9

r.nextInt(2); -> random 0 or 1

1+r.nextInt(6) -> 1,2,3,4,5,5,6
import java.util.Random;
public class RanTest{
    public static void main(String[] args)
    {
        Random r = new Random();
        for(int j = 0; j < 5; j++)
            System.out.println(r.nextInt());
        //
        for(int j = 0; j < 5; j++)
            System.out.println(r.nextInt(6) + 1);
    }
}
import java.util.Random;

public class Chance extends Random{

    public int throwDie(){ return (1 + nextInt(6));}

    public int throwDice(){ return (throwDie() + throwDie());}

    public void shuffle(int[] nums){

        int swapPos, temp;
        for (int i = nums.length-1; i > 0; i--) {
            swapPos = nextInt(i+1); // pick pos from 0 -> i (i is possible)
            temp = nums[swapPos]; // swap vals at i, swapPos
            nums[swapPos] = nums[i];
            nums[i] = temp;
        }
    }
}

Shuffle works this way:
Set $i$ to length of array
Pick a random value from 0 to $i$ inclusive $\rightarrow$ swapPos
Swap contents of $i$, swapPos

Reduce $i$ by 1, do it again...
Reduce $I$ by 1, do it again...

- 

Until $i = 1$
How shuffle works…

Swap contents of cell i, cell 2

swapPos
public class ChanceTester{
    public static void main(String[] args){

        int[] nums = {1,2,3,4,5,6};
        Chance c = new Chance();
        for(int j = 0; j < nums.length; j++)
            System.out.print(nums[j]);
        System.out.println();
        c.shuffle(nums); // scramble the array
        for(int j = 0; j < nums.length; j++)
            System.out.print(nums[j]);

    }
}
}
A typical run of ChanceTester..

123456

543612

and another..

123456

523614
The Drunk at the Opera

Drunk takes over opera cloak room

Hands out umbrellas at random

Sometimes, nobody gets own umbrella; other times a few people do get their own.

In 10,000 drunken episodes, how likely is it that no one gets his or her own?????
public class Drunk extends Chance{

private int count;
private int[] patrons; // the opera goers

public Drunk(int count){
    this.count = count;
    patrons = new int[count];
}

public void initialize(){  // start: everyone getting own
    for(int pat = 0; pat < patrons.length; pat++)
        patrons[pat] = pat;
}
public boolean allGetWrong(){
    boolean allWrong = true;
    initialize();
    shuffle(patrons);
    for(int pat = 0; pat < patrons.length; pat++)
    {
        if( patrons[pat] == pat)
        {
            allWrong = false;
            break;
        }
    }
    return allWrong;
}
Object

Random

Chance

Drunk