Announcements

Program 5 up, SEE WEBSITE
Next OWL assignment up
Extra Credit

Preregistration:

123 (191B)
187 (a “theme” course)
We’ve reached a new level of complexity in the applications we’ve studied -

There were 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. WeighedInfant (like Infant, but with new field: weight)

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

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

Note: Java (OO) shines in the machinery it provides for seamless, hygienic code recycling.
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 (we’ll see subtler ones examples shortly):

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

Rectangle -> colored Rectangle (add a color)

Vehicle -> motorized vehicle (yup: add a motor)
Given a Person class, with these attributes:

String firstName
String lastName
String SSN
int age
boolean female

-----

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

Retain the attributes of Person in the new class - HOW

Retain methods of Person - can I use them .. ? - HOW

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.
public class Person{
    private String name;

    public Person(String who){ name = who;}

    public void personInfo(){
        System.out.println("I am " + name);
    }
}
public class SizedPerson extends Person{
    private int height;

    public SizedPerson(String who, int ht){
        super(who);
        height = ht;
    }

    public void personInfo(){
        super.personInfo();  // personInfo from super class
        System.out.println("my height is " + height);
    }
}
public class PersonDriver{
    public static void main(String[] args){
        SizedPerson p = new SizedPerson("hilda",62);
        p.personInfo();
    }
}

----------------
Output
----------------
I am hilda
my height is 62
Person is the **base** class or the **super** class; SizedPerson is the **derived** or the **subclass** class. SizedPerson **extends** Person; SizedPerson **specializes** Person.

The **super** class is the **more general** 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 mk, double cap, double amt, int yr){
    super(mk, cap, amt);
    year = yr;
}
public int getYear(){
    return year;
}
}

-- a key idea - the public methods of the Car class are available to UsedCar objects
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 attributes come along (if private: can’t use directly) [RentedApt extends Apt]

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

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

A’s methods, when overriding B’s methods, can use
  super.method() to do some of the work of A’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 < 3; j++)
            System.out.print(r.nextInt() + " ");

        for (int j = 0; j < 3; j++)
            System.out.println((r.nextInt(6) + 1) + " ");
    }
}

---
-14612 123 4532 5 1 4
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
            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 rand value from 0 to i inclusive -> 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...

0 1 2  i

swapPos

Swap contents of cell i, cell 2
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.

How frequently does no one gets his or her own umbrella?

(answer: \( \sim 0.37 \)) (Actually: \( 1/e \))!
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()
{
    // hand out those umbrellas, then check..
    boolean allWrong = true;
    initialize();
    shuffle(patrons);
    for(int pat = 0; pat < patrons.length; pat++)
    {
        if( patrons[pat] == pat)
        {
            allWrong = false;
            break;
        }
    }
    return allWrong;
}
import java.util.Scanner;
public class DrunkTester{
    public static void main(String[] args){
        int trials = 10000;   // run many experiments..
        int allWrong = 0;
        Scanner scan = new Scanner(System.in);
        System.out.println("Enter number of patrons");
        int num = scan.nextInt();
        Drunk d = new Drunk(num);
        for(int j = 0; j < trials; j++){
            if (d.allGetWrong()) allWrong++;
        }
        System.out.println("fraction all get wrong umbrellas: " +
                        (double)allWrong/trials);
    }
}
What’s Object?
Object is the ultimate base class.
Every class is either directly or indirectly derived from Object
Object has a few methods - One is toString
Example:
Infant myKid = new Infant("jill", 23);
System.out.println(myKid.toString());
Prints: Infant@124821
Reason: we haven’t overwritten toString