CS 121 – Intro to Programming:Java - Lecture 5
Announcements

Third programming assignment due Friday.

Ch 5 OWL exercises released soon

MidTerm: Tuesday, 10/16, 7-8:15, Bartlett 65 (covers Ch 1-5)
No Lecture next week: Tuesday: follows Monday schedule
(Also: no honors section meeting)

Sample exam at course web site available soon
Advanced note: final -> 12/17

Some OWL problems reported with textbook..
Principal theme today: methods

Methods organize (sub)jobs at the statement level

They’re the fundamental mechanism for combining elementary operations together to make reuseable, more complex operations.

You can build an entire “world” with methods: very complex chores rely on complex chores, which rely on elementary chores, which rely on primitives.. and so on.
We’re interested in creating a class that makes forming patterns of rows of stars, sharps other symbols easy - patterns like these:

```
***********  or  *
***********  **
***********  ***
***********  ****
***********  *****
***********  ******
```

or

```
Attributes: symbol, width(?)
```

Methods -??

```
#
##
###
####
```

```
```
We might do this by creating a class with attributes like:

- width (of a row)
- symbol (what’s printed)

Heart of the matter: how do we print a row?

(this is a more complex action than simply giving back or “getting” the age of a kid, or the capacity of a fuel tank)
public class Rows{
    private char sym;
    private int width;
    final char BLANK = ' '; // a constant!

    public Rows(char s, int w){
        sym = s;
        width = w;
    }
    public char getSym(){
        return sym;
    }
    public int getWidth(){
        return width;
    }
}
public void makeRow(){
    for(int j = 0; j < width; j++) // width: an attribute
        System.out.print(sym);
}

public void varyRow(int k){ // row k symbols long
    for(int j = 0; j < k; j++)
        System.out.print(sym);
}

public void newLine() {System.out.println();}
public class RowTester{  // does first pattern
    public static void main(String[] args){
        Rows r = new Rows(‘*’,11);
        for (int j = 0; j < 6; j++){
            r.makeRow();  // makes row of length 11
            r.newLine();  // goes to new line
        }
    }
}
public class RowTester { // does third pattern
    public static void main(String[] args) {
        Rows r = new Rows('#', 10); // 10 is useless
        for (int j = 1; j <= 4; j++) {
            r.varyRow(j); // does row of length j
            r.newLine();
        }
    }
}
Method mechanics

```java
public char getSym(){ // returns attribute sym
    return sym;
}

public int getWidth(){
    return width;
}

public void newline() {System.out.println();}
```
public void varyRow(int k) {
    for(int j = 0; j < k; j++){
        System.out.print(sym);
    }
}

Returns nothing (merely prints to console)

Has a parameter - k is a formal parameter

\[ f(x) = x^2 \quad g(y) = y^2 \quad h(\text{dog}) = \text{dog}^2 \]

x, y, dog are also all formal parameters
public class SimpleCoins {

    static final int HEADS = 1;
    static final int TAILS = 0;

    public int flip() {
        if (Math.random() < 0.5)
            return TAILS;
        else
            return HEADS;
    }

    public int multiFlip(int flips) {
        int total = 0;
        for (int j = 0; j < flips; j++) {
            total += flip();
        }
        return total;
    }
}

```java
public int multiFlip(int flips) {
    int total = 0;
    for (int j = 0; j < flips; j++) {
        total += flip();
    }
    return total;
}
```
public int multiFlip(int flips) {
    int total = 0;
    for(int j = 0; j < flips; j++) {
        total += flip();  // same as total = total + flip();
    }
    System.out.println( total);
}

public int multiFlip(int flips) {  
    int total = 0;
    for(int j = 0; j < flips; j++) {
        total += flip();
    }
    return “total”;  
}
We want to test if two consecutive flips are the same.

How would we add this as a method to the SimpleCoins class?

Return type?

Parameters?

Need a return statement?
We want to test if two consecutive flips are the same. How would we add this as a method to the class?

Return type? → boolean

Parameters? → none needed

Need a return statement? → yes (not void rtn)
public boolean doubleFlip() { // 2 flips: are they the same?
    int flip1 = flip(); int flip2 = flip(); // why is this legal?
    if (flip1 == flip2)
        return true;
    else
        return false;
}

public boolean doubleFlip() {
    int flip1 = flip(); int flip2 = flip();
    return (flip1 == flip2);
}

public boolean doubleFlip() {
    return (flip() == flip());
}
Suppose you throw $n$ (where $n$: some number of coins) 100,000 times, and you want to know how often (with what frequency) exactly $k$ heads comes up.

What's the return type?

What are the parameters?

Need a return statement?
Suppose you throw n (n: some number of coins) 100,000 times, and you want to know how often (with what frequency) exactly k heads comes up

What’s the return type? -> double

What are the parameters? -> n, k

Need a return statement? -> sure: non-void rtn
public double patFreq(int coinCt, int headCt){
}

public double patFreq(int coinCt, int headCt){
    double freq = 0.0;
    return freq;
}
public double patFreq(int coinCt, int headCt) {
    double freq = 0.0;
    int ctr = 0;
    for (int j = 0; j < 100000; j++) {
        if (multiFlip(coinCt) == headCt) ctr++;
    }
    freq = (double) ctr / 100000;  // note the cast
    return freq;
}
Another example: The Vowels class

One attribute: a String, str

One method: **vowelCount** (you report how many vowels there are in str.

-----------------------------

What’s the return type?

What are the parameters?

Need a return statement?

Need a helper method?
Another example: The Vowels class

One attribute: a String, str

One method: `vowelCount`

--------------------------------------------

What’s the return type? -> int

What are the parameters? -> none

Need a return statement? -> yes!

Need a helper method? -> isVowel
Until now: methods are passed, return primitives, or maybe Strings

We need to study parameter passing more carefully, look at how parameter values can, cannot change.

First up: methods that return objects
We want to make an infant’s twin - for example, Jill has a twin named Ivan.
(note: Jill, Ivan are the same age)

Infant kid = new Infant("Jill",1);

Infant kidTwin = kid.makeTwin("Ivan");

What kind of “thing” does makeTwin return???

Answer: makeTwin returns an Infant object (here: Ivan / 1)
public Infant makeTwin(String name) {
    int myAge = getAge(); // troubling!
    Infant i = new Infant(name, myAge);
    return i;
}

int myAge = this.getAge(); (replaces second line)

What’s this? this is “calling object”
Why we need this…

When we write a class definition, we haven’t created any objects from that class yet - we only have the blueprint.

But we sometimes want to refer to the to-be-created objects in the class. That’s the role of this.

Think of a house blueprint (class def), with attributes like heating, floor space, roofing, etc. The blueprint might say “this house will have a metal roof”. You’re doing the same thing: you’re referring to a characteristic of the to-be-built house before the “constructor” (the builder) is called.

This happens in natural language:

“Before she went in the house, Jill folded her umbrella.”
public class infant{
    ..
    public Infant makeTwin(String n) {
        myAge = this.age;
    }
    Or
    myAge = this.getAge();
    ..
}

main(...){
    myKid = new Infant("jill", 1);
    myTwin =
        myKid.makeTwin("fred");
}
The cell model of variables revisited:
parameter passing
Recall the cell model of Java variables:

```java
int num = 5;
```

```
5
```

```
num
```

```
num = num + 1;
```

```
5
```

```
6
```

```java
```
Parameter passing in Java

Consider this method:

```java
public void change(int x)
    {x = x + 1;}
```

-- now look at this code fragment --

```java
int a = 3;
change(a);
System.out.println(a);
```

What value is printed?
Cells and parameter passing: part I

Value of a copied to x. Copy works just in one direction!
Key idea: myKid - an Infant object name - does **not** hold the myKid object information. Instead, it holds the **address** (in computer memory) of the location where that information is stored.
Consider this statement:

```java
myKid.anotherMonth();
```

Where:

```java
public void anotherMonth()
{
    age = age + 1;
}
```

The `anotherMonth` method does not alter the `myKid` cell; it alters the information at the `location` referenced by the address in the `myKid` cell.
Recall: anotherMonth method increments age by 1

myKid.anotherMonth();

101010010110100101

class: Infant
name: Ted
age : 3 -> 4

101010010110100101

memory address
The key example:

```java
public class Flight{
    String id; // flight id
    String start; // starting airport
    String end; // ending airport
    boolean arrived; // has plane arrived yet?

    public Flight(String id, String s, String e, boolean here)
    {
        this.id = id; start = s; end = e; arrived=here;
    }

    .......

    Flight myFlight = new Flight("CE777","JFK","LAX",false);
```
The key method

Suppose there’s an Airport class, with this method:

```java
public void landFlight(Flight f) {
    f.setArrived(true);
}
```

Suppose we say:

```java
airportLAX.landFlight(myFlight);
```

This changes the state of flight f. How???
Address of myFlight object information

myFlight

101110 -> 101110

landFlight
landFlight parameter myFlight does not change - it’s the address of the Flight object information. So our calling principle is not violated. But the referenced object itself does change state: The plane has arrived.