Announcements

Ch 5 Embedded problems due today (10 PM)
Programming Assignment 4 up today, or Friday
Ch 4 OWL assignment due Saturday 5 PM
Online midterm coming - probably over next weekend
Old midterm at CourseAdministration link
Principal theme today: **methods**

Methods organize (sub)jobs at the statement level

They’re the fundamental mechanism for combining elementary operations together to make reusable, 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.. etc.
public int cube(int n) {
    return (n * n * n);
}

Header line / body
Parameter list
Return stmt / return type
public int cubeSum(int k){
    // sums cubes of digits
    int sum, cur, ones, tens, hundreds;
    cur = k;
    ones = cur % 10; // one's place
    cur = cur / 10;
    tens = cur % 10;
    hundreds = cur / 10;
    return (cube(ones) + cube(tens) + cube(hundreds));
}

One method in a class calls another in the same class (cubeSum calls cube)
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 = total + flip();
        }
        return total;
    }
}
public int multiFlip(int flips) {
    int total = 0;
    for(int j = 0; j < flips; j++) {
        total += flip();
    }
    return total;
}
public void greeting(String name){
    System.out.println("hello " + name);
}

void used for: object state changes; announcements
(If you’re sitting a slot machine, you pull the lever, and a message appears: “You’ve won $100” - that’s void;

    when the 400 quarters come pouring out, a return stmt is required, and your return type is int or double)
public int multiFlip(int flips) {
    int total = 0;
    for(int j = 0; j < flips; j++) {
        total += flip();
    }
    System.out.println( total);
}

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

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

Doesn’t return anything. Should be `void`.

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

Doesn’t return an `int` - returns a `String` (and not the value of `total` either)
Where does method input come from?

• parameters
• object attributes
• more global sources - Math.PI
What can methods do?

- Produce output - via return statements
- Change object state - alter instance variables
  ```java
  myKid.anotherMonth();
  ```
- Write stuff on the wall (ahem: the screen)
  ```java
  System.out.println("Hi Julie");
  ```
Suppose we want to test if 2 consecutive flips are the same. How would we add - doubleFlip - to SimpleCoins?

Return type?

Parameters?

Need a return statement?
public boolean doubleFlip(){ // 2 flips: the same?
    int flip1 = flip(); int flip2 = flip();
    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 flip n coins 100,000 times, and you want to know how often (with what frequency - a decimal fractional amount) exactly k heads comes up [for example: I flip 20 coins, with what frequency do I get 3 heads (over 100000 tosses - possible ans: 0.0087)??]

What’s the return type?

What are the parameters?

Need a return statement?
// hCt = headCount

public double patFreq(int coins, int hCt){

}
public double patFreq(int coins, int hCt){
    double freq = 0.0;

    return freq;
}
public double patFreq(int coins, int hCt) {
    double freq = 0.0;
    int ctr = 0;
    for (int j = 0; j < 100000; j++) {
        if (multiFlip(coins) == hCt) ctr++;
    }
    freq = (double)ctr / 100000;
    return freq;
}
Method `divCount`: you give it a (positive) integer, it returns the number of integers that divide the number evenly.

Example: you give it 10, it should return 3 (since 1, 2, 5 are the divisors of 10)

Write `divCount`
1) Header line
2) Return statement
3) Body
public int divCount(int k) {  // k > 0
    int ct = 0;
    for (int j = 1; j <= k/2; j++)
        if (k % j == 0) ct++;
    return ct;
}
How about this new method in **Infant** class:

It’s called `babyName`. It returns an Infant’s name, with the word “Baby” pasted on the front.

Return type?

Parameter(s)?

Return statement?
public class Infant{  // the Infant class definition

    private  String name;
    private int age;  // in months

    public Infant(String who, int months){
        name = who;
        age = months;
    }

    public String getName(){return name;}

    public int getAge(){return age;}

    public void anotherMonth(){age = age + 1;}
}
public String babyName()
{
    String realName = this.name;  //!!
    String who = ("Baby" + realName);
    return who;
}

Why this, as in this.name ??
1) How is babyName used? Answer:

   String someName = kid.babyName();

   Might appear in main somewhere

2) When the babyName method is written, a calling object (here: kid) has **not** been identified

3) So “this” is a kind of Java pronoun that stands in for the calling object to be named later(!)

4) Happens in English:

   “When **she** came in, Jill hung up her coat”
name: Jill
age: 12

```
p.c.InfantTest{
p.s.v.m.(..){
    Infant kid = ...
    ...
    s = kid.babyName();
}

p.c. Infant{..}
p.S. babyName()
    real = this.name;
```

```
main
```
Another Infant method: one that changes a kid’s name..

setName
public void setName(String newName){
    this.name = newName;
}

Usage:
myKid.setName("Ginny");

name: Virginia
age: 13
myKid
An alternate way:

```java
public void setName(String name){
    this.name = name;
}
```
Until now: methods are passed, then return primitives, or maybe Strings

Let’s look at methods that return objects (Infant objects, Car objects, etc)
Let's create a method that makes a twin of an infant (that is, a twin object of an infant object)

Remember: twins have the same age!

Infant kid = new Infant("Jill",1);
Infant kidTwin = kid.makeTwin("Ivan");
public Infant makeTwin(String name){
    int twinAge = this.getAge(); // or: this.age;
    Infant i = new Infant(name, twinAge);
    return i;
}

[aside: second line could have been:
    this.age;
Reason: makeTwin is in Infant class, so it can see private instance variables.]
public class Infant{
  ..
  public Infant makeTwin(String) {
    twinAge = this.age;
    Or
    twinAge = this.getAge();
  }
  ..

  p.s.v main(...){
    myKid = new Infant("Jill", 1);
    kidTwin =
      myKid.makeTwin("fred");
  }
Recall the cell model of Java variables:

```java
int number = 5;

number = number + 1;
```

5

number

number = number + 1;

5

6
Parameter passing in Java

Consider this method:

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

-- now this code --

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

What's the value of a? (ans: still 3)
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 of the location where that information is stored.
Consider this statement:

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

Where:

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

`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

myKid
memory address
Imagine two classes:

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

    We make a flight object, hasn't landed yet
    myFlight = new Flight("CE777", "JFK", "LAX", false);
Let's land the flight:

airportLAX.landFlight(myFlight);
   // in driver

Using method from Airport class..

public void landFlight(Flight f){
    f.setArrived(true);
}
We want the landFlight method to change an attribute of myFlight

- and we can do it, because myFlight is literally a reference to data, and we aren’t changing that reference (that address).

We’re jumping to that address and changing information there.
Address of myFlight object information

myFlight

101110

landFlight

101110
The landFlight parameter 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.