CS 121 - Intro to Programming:Java - Lecture 8

Announcements

Ch 5 Embedded problems due 10/5 (Mon) 10 AM

Programming Assignment 3 due Friday, 5 PM

Ch 4 OWL assignment due Thursday, 5 PM

Grading not complete on prog 1, 2

Midterm posted- W evening 10/14. 7 - 8:15 PM, Thompson 102, 104. Old midterm at CourseAdmin link

TA Office hours:

  M 1-4;  TU 3:45-5;  W 4-6 (4-5:15 until 10/10);  TH 2 - 5;
  F 12:15 - 3:15;  - LGRT 223
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);
}
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));
}
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
  myKid.anotherMonth();
• Write stuff on the wall (ahem: the screen)
  System.out.println("Hi Julie");
Suppose we're interested in testing if two consecutive flips are the same. How would we add this 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.getName();
    String who = ("Baby" + realName);
}

return who;
}

Why this, in: this.getName();
1) How is babyName used? Answer:

    String someName = jill.babyName();

    This might appear in main somewhere

2) When the babyName method is written, a calling object (here: jill) 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”
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
Until now: methods are passed, then return primitives, or maybe Strings

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

Infant kid = new Infant("Jill",1);
Infant kidTwin = kid.makeTwin("Ivan");

(note: if Ivan is Jill’s twin, then they have the same age in months).
public Infant makeTwin(String name) {
    int twinAge = this.getAge();
    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 t)

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;
```

![Diagram](image)
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!
Cells and Objects

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
```

```
101010010110100101
```

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

---

```
myKid
```

```
memory address
```

```
101010010110100101
```

```
101010010110100101
```

---
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;
}

myFlight = new Flight("CE777","JFK","LAX",false);
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

101110

myFlight

landFlight
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.