CS 121 - Intro to Programming:Java - Lecture 3

Announcements

Introductory Survey - Please do it!

Ch 2 Embedded problems due Wed, noon

First programming assignment on OWL, due Thur at 5

Ch 2 OWL hwk due Friday at 5

Ch 3 Embedded problems due Monday, 10 AM

MyOffice hours posted, TA hours later today

http://twiki-edlab.cs.umass.edu/bin/view/Moll121/WebHome

Check CourseWork link at left for due dates

ClassDiscussions09 - They will vary tomorrow!
A typical Java program
A very, very simple application

Class G

main
public class Howto{
// a baby intro example
public static void main(String args[]) {
   System.out.println("Welcome to 121");
   System.out.println("3 + 5");
   System.out.println(3 + 5);
}
}

Prints:
Welcome to 121
3 + 5
8
Java’s Object Model

This subject will occupy us for a good deal of the next month or so!
The Object Model

We model “things” as objects

Objects have attributes, and behaviors

**Trip** (attributes: start, end, days, distance)
  (behaviors: getDays, setDays, distPerDay ..)

**Horse** (attributes: name, breed, age, height)
  (behaviors: getAge, setAge ..)

Car

House

Student

Tree
Where are all of these characteristics written down?

Answer: In a class definition

A class definition for an object is different from an object, in the same way that the blueprint for a house is different from a house.

Still -

How do you make an object?

How do you invoke its behaviors?
Infant objects

Attributes

name

age (in months)

Behaviors

getName

getAge

anotherMonth (make kid one month older)
public class InfantTester{

public static void main (String[] args)
{
    Infant myKid = new Infant("Kit", 4);
    System.out.println("name: " + myKid.getName());
    myKid.anotherMonth();
    System.out.println("my kid is now " + myKid.getAge());
}
}
Broadly, two kinds of behaviors:

**Get behaviors** -
- How old is the kid
- What’s the kid’s name
- Get behaviors do NOT change the calling object

**Mutating behaviors** - Alter calling object
- Name change
- Age change

When a mutator does what it does, the state of the calling object changes.
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 class Infant{

    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;}
}

Attributes

Constructor

public String getName(){return name;}

public int getAge(){return age;}

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

}
A crucial concept - flow of control

• Statement level flow
• Class level flow
public class Infant{
    priv String name;
    priv int age;
    public Infant(...){
        name = ..
        age = ..
    }
    pub int getName(){
        return name;
    }
}

public class InfTes{
    public static ....{
        Inf myKid =
            new Infant( .. );
        S.o.p.
            ( myKid.getName() );
    }

InfantTester.java

Infant.java
Source Code - text files Infant.java, InfantTester.java

Object Code - Infant.class, InfantTester.class - Bytecode.

Exactly one object created - how do we refer to it?

• Where is it?
• How was it created?
• What is its state when it was created?
• Does its state change? How?
• What is its final state?
• Do Infant, InfantTester communicate?
In Java all data is tagged (that is, typed)

```java
int number;

double v;

Infant kid;
```
Generally there are two broad kinds of data: objects (roughly, things - defined by classes) primitives (numbers, true/false values etc.)

Variables are associated with data

For primitives anyway, there is a cell model for variables and their values:

\[
\text{int num} = 6;
\]

\[
\text{num}
\]
Assignment Statements and Identifiers

An identifier is the name of a variable (or method, or class..)

```java
int number = 7;  // number now “holds” 7
number = 4;  // number now “holds” 4
number = number + 2; // number now “holds” 6
```

Assignment is NOT equality!

Assignment is an action operator: **Compute** the RHS, Then **copy** the result to variable named on the LHS
int num = 7;
num = num + num;
-------
int value = 4;
value = value + 1;
-------
int score = 10;
score = score + score;
score = score * score;
score = score / 2;
Class A                                   Class ADriver

Our current view of the OO programming landscape..
Primitive Data Types
• objects are Java’s main currency
• Too tedious for them to be the only currency
• Primitive data types (8): integers (4), floats(2), char, boolean.

Statement like these are fairly common:

long count = 0;     // like int, but larger range

double bigPapiAvg = 0.224; // nums with decimal pts

boolean chewsTobacco = false;

char averageGrade = 'C'; // note the single quotes
**Strings** - A very important class.

```java
String greeting;
greeting = new String("ola");
greeting2 = new String("howdy");
greeting = greeting2;
System.out.println(greeting); // prints howdy
```

Some caveats:
1) Strings are NOT primitives (unlike double, int, boolean)  String is a class in the package java.lang

2) There’s a shorthand for String creation:
   ```java
greeting = "ola";
```

3) String class comes with extensive functionality
String pupName = "spot";

int len = pupName.length(); // len assigned 4

char what = pupName.charAt(1); // what is assigned 'p'

char ch = pupName.charAt(0); // ch is assigned 's'

String huh = pupName.concat("less"); // spotless

String bigHuh = pupName.toUpperCase(); // SPOT

Where do I find out about the String class??
For Thursday:

Plenty to do: Ch 2 OWL hwk, Program 1

I will lecture on Ch 3

(put off embedded problems here because beginning of class so jammed... )