Building Java Programs

Subtype Polymorphism;
Sorting (an extended programming example)

A tricky problem

- The order of the classes is jumbled up (easy).
- The methods sometimes call other methods (tricky!!)

```
public class Lamb extends Ham {
    public void b() {
        System.out.print("Lamb b   ");
    }
}

public class Ham {
    public void a() {
        System.out.print("Ham a   ");
        b();
    }

    public void b() {
        System.out.print("Ham b   ");
    }

    public String toString() {
        return "Ham";
    }
}
```

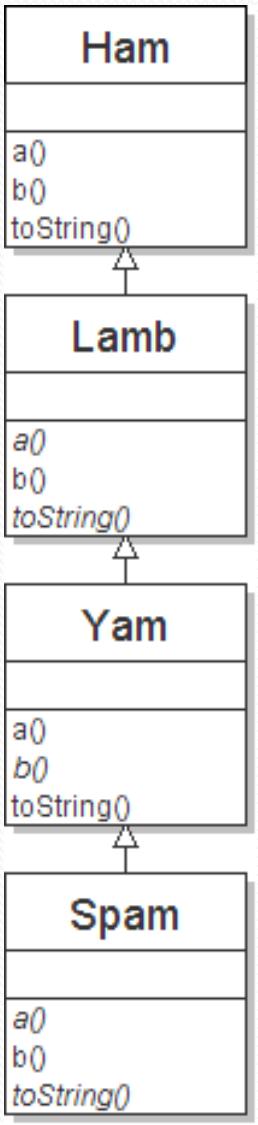
Another problem 2

```
public class Spam extends Yam {
    public void b() {
        System.out.print("Spam b   ");
    }
}
public class Yam extends Lamb {
    public void a() {
        System.out.print("Yam a   ");
    }
    public String toString() {
        return "Yam";
    }
}
```

- What would be the output of the following client code?

```
Ham[] food = {new Lamb(), new Ham(), new Spam(), new Yam()};
for (int i = 0; i < food.length; i++) {
    System.out.println(food[i]);
    food[i].a();
    System.out.println();      // to end the line of output
    food[i].b();
    System.out.println();      // to end the line of output
    System.out.println();
}
```

Class diagram



Polymorphism at work

- `Lamb` inherits `Ham`'s `a`. `a` calls `b`. But `Lamb` overrides `b`...

```
public class Ham {
    public void a() {
        System.out.print("Ham a   ");
        b();
    }

    public void b() {
        System.out.print("Ham b   ");
    }

    public String toString() {
        return "Ham";
    }
}

public class Lamb extends Ham {
    public void b() {
        System.out.print("Lamb b   ");
    }
}
```

- `Lamb`'s output from `a`:

```
Ham a   Lamb b
```

The table

method	Ham	Lamb	Yam	Spam
a	Ham a **b()**	Ham a **b()**	Yam a	Yam a
b	Ham b	Lamb b	Lamb b	Spam b
toString	Ham	Ham	Yam	Yam

The answer

```
Ham[] food = {new Lamb(), new Ham(), new Spam(), new Yam()};
for (int i = 0; i < food.length; i++) {
    System.out.println(food[i]);
    food[i].a();
    food[i].b();
    System.out.println();
}
```

- Output:

```
Ham
Ham a    Lamb b
Lamb b

Ham
Ham a    Ham b
Ham b

Yam
Yam a
Spam b

Yam
Yam a
Lamb b
```

Now what?

- Done: Almost all the Java language features we'll learn
  - There are more
  - But methods, loops, arrays, objects, files, if/else, strings, etc. are a powerful toolset (not bad for 9 weeks!)

- Remaining lectures:
  1. ArrayList library (next week)
  2. Review of references (next week; for the final)
  3. More practice with arrays
  4. More about writing larger/trickier programs

  Sorting: A great example for 3-4

Sorting

- Computers are used to *sort* a lot: Take a list of data and return all the same data "in order"
  - Example: Students by id number
  - Example: Students by exam score
  - Example: High temperatures for the last month
- "In what order" depends on what you want to do
  - For now, assume integers and lesser before greater
  - Example: 3 4 2 1 1 7 5 0 9 sorted is 0 1 1 2 3 4 5 7 9
- So far we have described the **external behavior**
  - Now we need an **algorithm**
  - Then (and only then) we can "**code it up**"

An algorithm

- Recall: An algorithm is a sequence of steps to compute an answer
  - There can be more than one algorithm for a problem!

- This is a concept bigger than programming
  - People sorted long before computers existed (of course)
  - But computer scientists invented better ways!
    - Today we'll stick to simpler not-as-good ways

Insertion sort

Pseudocode / English: Copy each input element to the output one at a time. Keep the output sorted by:

(1) finding the right place for the next element

(2) shift over all the larger numbers to make room

(3) *insert* the number in the space you just made

Selection sort

Pseudocode / English: Initially no elements have been copied to the output. **Select** the least element *that has not already been copied* and copy it to end of the output. Repeat until all elements have been copied.

Note: Have to keep track of what has already been copied.

Coding it up

- Translating an algorithm to Java is not easy
  - Defining the right arrays
  - Avoiding off-by-one bugs
  - Implementing things like "find the least"

- Want to **test** our implementations
  - Think about corner cases!
    - 0-element array
    - 1-element array
    - Negative numbers
    - Array with all the same number
    - …

More generally

- What part of our code was specific to sorting *numbers*?
  - Almost nothing – the code for something else would look almost identical
  - Example: Sort `Point` objects
    - need an order, such as distance to the origin

- Good news / bad news:
  - We ended up with offensive redundancy
  - In 142, we don't have the knowledge to avoid this
  - Okay, so there is more to learn about programming
  - At least we can recognize the algorithm is the same
    - And there "oughtta be a way" to implement the algorithm only once, even if we don't know it yet