Inheritance

The Big Picture

- object-oriented programming is meant to reflect the structure of things in the real world
  - objects correspond to individual things
  - classes correspond to kinds of things
- in the real world, different kinds of things are not always completely unrelated
  - e.g. apples and fruit – apples are a kind of fruit, though there is fruit that's not apples
  - e.g. savings accounts and checking accounts are both kinds of bank accounts (and there may be other kinds of bank accounts)
- *inheritance* is the mechanism by which we can express “is-a” relationships between classes
- *polymorphism* is the mechanism by which we can write code that works with things related by an “is-a” relationship

Preliminary Scrabble Design

class	represents	stored info	methods
HumanPlayer	one of the people playing the game	current score player's rack	take turn (choose tiles and play word – prompt user for choices)
ComputerPlayer	a computer player	current score player's rack	take turn (choose tiles and play word)

Observations –

- `HumanPlayer` and `ComputerPlayer` have the same stored info and the same methods
  - differences are only in the bodies of those methods
- `HumanPlayer` and `ComputerPlayer` are different kinds of the same sort of thing (a player)

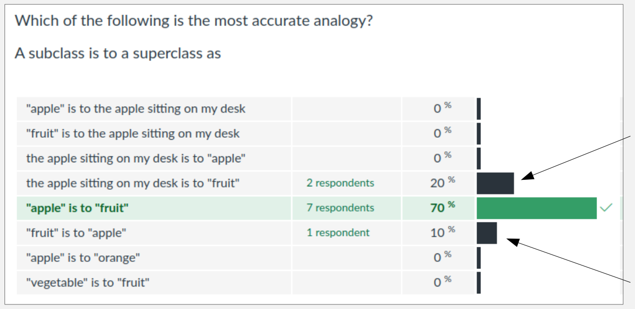
If our goal is that the program organization reflect the real-world structure, we should capture this.

Inheritance and Polymorphism

Two purposes –

- to capture an is-a relationship that is naturally present
  - human and computer players are both kinds of players, and it should be possible to treat them the same way
  - use inheritance
- to create flexible code that can work with different versions of something (even versions not yet created)
  - our Scrabble main program should care only that a player can make a move – how that move is decided on is irrelevant to the functioning of the rest of the program
  - use polymorphism – encapsulate what varies and code to the interface

Inheritance



Which of the following is the most accurate analogy?

A subclass is to a superclass as

"apple" is to the apple sitting on my desk		0 %	
"fruit" is to the apple sitting on my desk		0 %	
the apple sitting on my desk is to "apple"		0 %	
the apple sitting on my desk is to "fruit"	2 respondents	20 %	
"apple" is to "fruit"	7 respondents	**70 %**	✓
"fruit" is to "apple"	1 respondent	10 %	
"apple" is to "orange"		0 %	
"vegetable" is to "fruit"		0 %	

the apple on my desk is a specific instance while fruit is a kind of thing – comparing an object to a class

the analogy is reversed – subclass is the more specific thing (apple), superclass is the more general thing (fruit)

Inheritance

- inheritance defines an “is-a” relationship between classes

```
public class Apple extends Fruit {
  …
}
```

  - an apple is a (kind of) fruit

- subclasses inherit everything – instance variables and methods – *except* constructors
  - even `private` things, though they cannot be accessed directly
  - new access modifier: `protected` allows only the class and its subclasses to access

Inheritance

Subclasses –

- *can* add new elements (instance variables and methods)
  - a new method has a different header (name and/or number/type of parameters)
- *can* redefine (override) or extend methods
  - same header, new body
  - to extend, also invoke superclass version
- *must* define one or more constructors (in most cases)
  - constructor should first call superclass constructor, then initialize only the instance variables for its own class
- *cannot* redefine instance variables
- *cannot* remove instance variables or methods already defined