Abstract Classes

Sometimes two things have a method header in common but not its body.

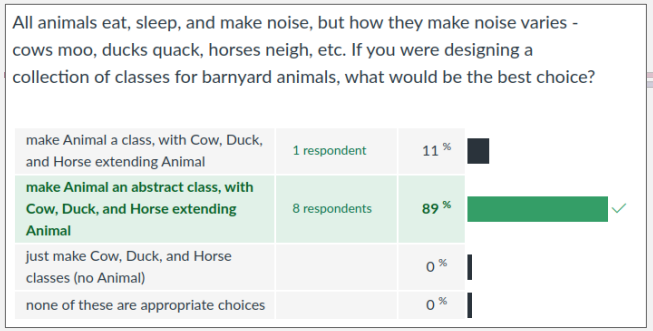
- e.g. `ArrayList` and `LinkedList` both have an `add(elt)` method because the add operation applies to both, but how they actually add the element to the list is different
- e.g. `getMove` in Scrabble Player

Abstract Classes

an abstract class can be used as a type for declaring parameters and variables
an abstract class can be extended
an abstract class cannot be instantiated

- *abstract classes* handle this situation
- syntax
  - `public abstract class ClassName { … }`
    - `abstract` means that it is not possible to create instances of `ClassName` – "nothing is just a `ClassName`"
    - `abstract` is required if there is at least one abstract method
  - `public abstract returntype methodname ( paramlist );`
    - `abstract` means that no body is supplied – "no way to write a body that makes sense for all"
      - must be overridden in a subclass or else the subclass is also abstract
    - note the difference between `;` for an abstract method (with no body) and `{}` for a (not abstract) method with an empty body
  - abstract classes must still have one or more constructors
    - can't use directly to create a standalone object, but subclass constructors still need to be able to create the core of the onion
    - can (should) be `protected` because only subclasses will use them
  - can also have instance variables and methods with bodies

All animals eat, sleep, and make noise, but how they make noise varies - cows moo, ducks quack, horses neigh, etc. If you were designing a collection of classes for barnyard animals, what would be the best choice?



nothing is just an animal – it is always some kind of animal (cow, duck, horse, …), so `Animal` as a concrete class that you can create instances of is not appropriate

the common element is having the behavior (making noise), not its implementation (moo, quack, neigh, …) – there's no way to write a body for `makeNoise` that makes sense for (all) animals

`Animal` as an abstract class allows for reuse of code common to all kinds of animals as well as using `Animal` as a type (allows coding to the interface), capturing the commonality of all animals without the awkwardness of being able to create things that don't exist

Deciding on Abstract Classes

Class or abstract class?

Some language cues to help you decide –
(though you always have to be careful to think about the meaning behind the language and not be too literal about the specific words)

- the unifying concept isn't talked about as its own thing → abstract class
  - compare –
    - "A bank account has an account number and an owner, a checking account has …, a savings account has ..."
      - a bank account is talked about as a thing separate from checking and savings accounts → class
    - "All kinds of bank accounts have account numbers and owners; a checking account also has …, a savings account also has ..."
      - only mentions kinds of bank accounts, but account numbers and owners apply to things that are bank accounts → abstract class

Deciding on Abstract Methods

Method or abstract method?

Some language cues to help you decide –

(though you always have to be careful to think about the meaning behind the language and not be too literal about the specific words)

- there is a common operation, but how it works depends on the kind of thing → abstract method
  - compare –
    - "All tickets have a price. The price of a walkup ticket is …, the price of an advance ticket is …, "
      - different things have different values → instance variable
    - "The price of a ticket depends on the type of ticket and potentially other properties of the ticket."
      - there's a shared notion of being able to compute a price but the specific process depends on the kind of ticket → abstract `getPrice()` method
    - "Tickets cost $20. Advance tickets cost 10% less and student tickets are half price."
      - there is a procedure applicable to all tickets, though specific kinds of tickets may modify that procedure or have a different version → (non-abstract) `getPrice()` method

Example

All kinds of tickets have a serial number, a price, and can be printed (which shows the ticket number, price, and type). The price is derived from the type of the ticket and potentially other properties of the ticket.

Walk-up tickets are purchased the day of the event and cost $50.

Advance tickets purchased 10 or more days before the event cost $30, purchased less than 10 days before cost $40.

Student advance tickets are half the price of regular advance tickets (no student prices for walk-up tickets). When student tickets are printed, the message "ID required" is also displayed.

- write classes `Ticket`, `WalkupTicket`, `AdvanceTicket`, and `StudentAdvanceTicket` with the elements and functionality specified, utilizing inheritance and abstract classes/methods as appropriate