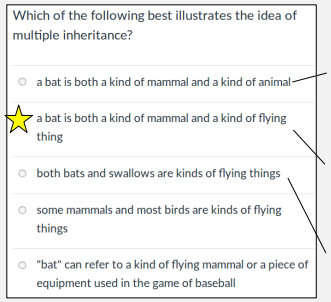
Multiple Inheritance



something may belong to more than one category

```
Bat extends Mammal
Mammal extends Animal
```

Bat has (only) the properties of Mammal – it just happens that Mammal includes the properties of Animal

```
Bat extends Mammal, FlyingThing
```

Bat has both the properties of Mammal and of FlyingThing

```
Bat extends FlyingThing
Swallow extends FlyingThing
```

Bat has only the properties of FlyingThing; it doesn't matter what else is also a flying thing

Multiple Inheritance

Establishing is-a relationships between types is important for flexible, reusable code – but the specific semantics of `extends` creates some problems.

```
Bat extends Mammal, FlyingThing
```

Bat has both the properties of Mammal and of FlyingThing

- what if `Mammal` and `FlyingThing` both…
  - have an instance variables with the same name but different types?
  - have methods with the same header but different bodies?

Interfaces

- *interfaces* address the type aspect of multiple inheritance without the code conflict problems
- syntax
  - `public interface` *`InterfaceName`* `{ … }`
  - `public` *`returntype methodname`* `(` *`paramlist`* `);`
    - like an abstract method in that no body is supplied, but `abstract` keyword is not needed
  - cannot have instance variables, constructors, or methods with bodies
- semantics
  - an interface defines a type
    - the type can be used anywhere a type is needed e.g. in variable and parameter declarations or as the base type of an array
  - it is not possible to create instances of an interface type

Interfaces

- classes *implement* interfaces
- syntax
  - `public class` *`ClassName`* `implements` *`Interface1, Interface2, … { … }`*
    - a class can implement any number of interfaces (it can also extend another class)
      - it is not legal for a class to implement two interfaces containing methods with identical headers except for the return types
    - the class must provide a body for every method in the interface(s) or else it must be declared `abstract`
- semantics
  - an object of a type implementing an interface can be used anywhere the interface type is expected
    - e.g. `Interface1 obj = new ClassName();`

Abstract Classes and Interfaces

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?

- make Animal a class, with Cow, Duck, and Horse extending Animal
- ★ make Animal an abstract class, with Cow, Duck, and Horse extending Animal
- just make Cow, Duck, and Horse classes (no Animal)
- none of these are appropriate choices

nothing is just an animal – it is always some kind of animal
the common element is having the behavior, not its implementation

→ abstract class

Which of the following best illustrates the idea of multiple inheritance?

- a bat is both a kind of mammal and a kind of animal
- ★ a bat is both a kind of mammal and a kind of flying thing
- both bats and swallows are kinds of flying things
- some mammals and most birds are kinds of flying things
- "bat" can refer to a kind of flying mammal or a piece of equipment used in the game of baseball

something may belong to more than one category

→ interface

What are the similarities and differences between an abstract class and an interface?

_____ can have methods for which no body is defined	[Choose]	both
_____ can have methods for which a body is defined	[Choose]	abstract class
_____ can be used to construct objects	[Choose]	neither
_____ can define types (and thus can be used in variable declarations, parameter declarations, return types, and base types for arrays)	[Choose]	both
_____ can be extended/implemented by more than one class	[Choose]	both
a single class can extend/implement multiple _____	[Choose]	interface