

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

a bat is both a kind of mammal and a kind of animal		0 %	
a bat is both a kind of mammal and a kind of flying thing	7 respondents	78 %	✓
both bats and swallows are kinds of flying things	1 respondent	11 %	
some mammals and most birds are kinds of flying things		0 %	
"bat" can refer to a kind of flying mammal or a piece of equipment used in the game of baseball	1 respondent	11 %	

```
public class Bat
extends FlyingThing {
    ...
}
public class Swallow
extends FlyingThing {
    ...
}
```

```
public class Bat
extends Mammal, FlyingThing {
    ...
}
```

```
public class AnimalBat {
    ...
}
public class BaseballBat {
    ...
}
```

Multiple Inheritance

something may belong to more than one category – a bat (the creature) is both a kind of mammal and a kind of flying thing

to reflect this, we might want

```
public class Bat
extends Mammal, FlyingThing {
    ...
}
```

but the specific semantics of `extends` creates some problems because classes, even abstract classes, can have instance variables and method bodies –

what if `Mammal` has an instance variable `String type_` and `FlyingThing` has an instance variable `int type_`? `Bat` inherits both, but you can't have two variables with the same name and different types

Java avoids this problem by providing *interfaces* and not allowing multiple inheritance

Interfaces

If a type is defined only by the existence of certain method headers – no storage of information, no method bodies – you can define an interface instead of an abstract class.

Why an interface instead of an abstract class?

- a class can implement multiple interfaces but can only extend one class
 - this avoids ambiguity about which method body to use should different bodies for the same method header be inherited from different places

Interfaces – Defining

- *interfaces* address the desire for multiple inheritance
 - supports polymorphism but not code reuse
- 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
 - no `new ...` for an interface

Interfaces – Implementing

- classes *implement* interfaces
- syntax
 - `public class ClassName implements Interface1, Interface2, ... { ... }`
 - a class can implement any number of interfaces (it can also extend another class)
 - 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 vs Interfaces

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

Example

Things that are like eagles can fly and hunt.
Things that are like lions can run and hunt.

Eagles are, of course, eagle-like.

Lions are, of course, lion-like. They can also roar, which is not necessarily true of everything which is like a lion.

A griffin is like both eagles and lions. It can also guard treasure.

- define types (classes, abstract classes, and/or interfaces) to capture the relationships and behavior described



