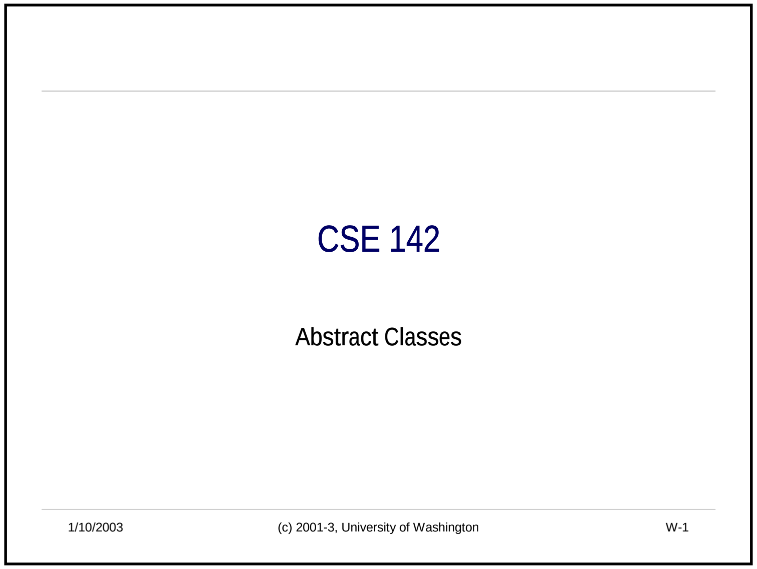
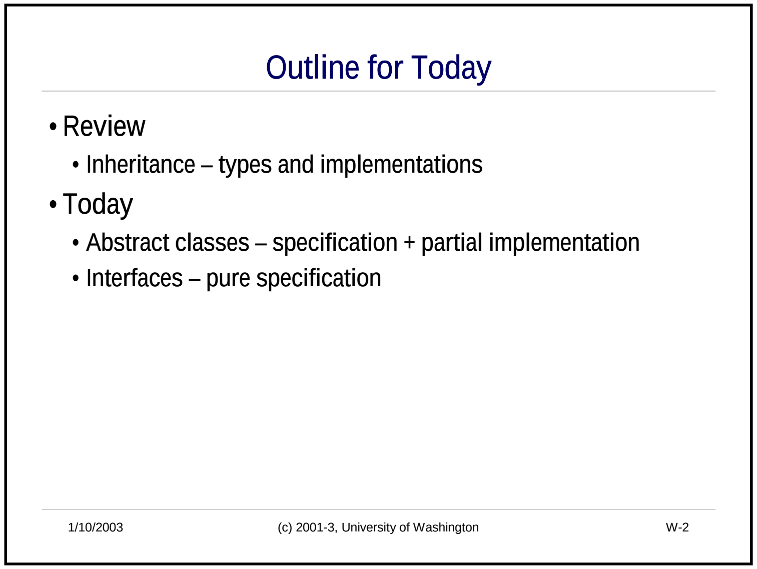


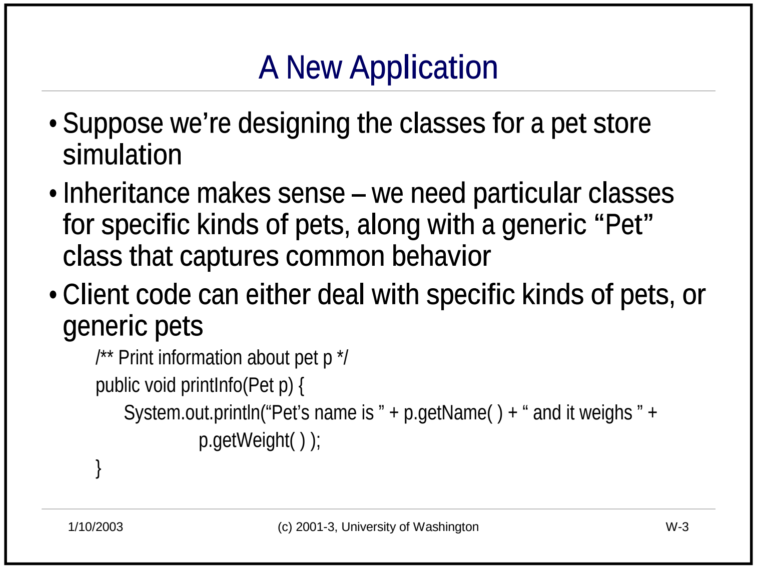
CSE 142

Abstract Classes



Outline for Today

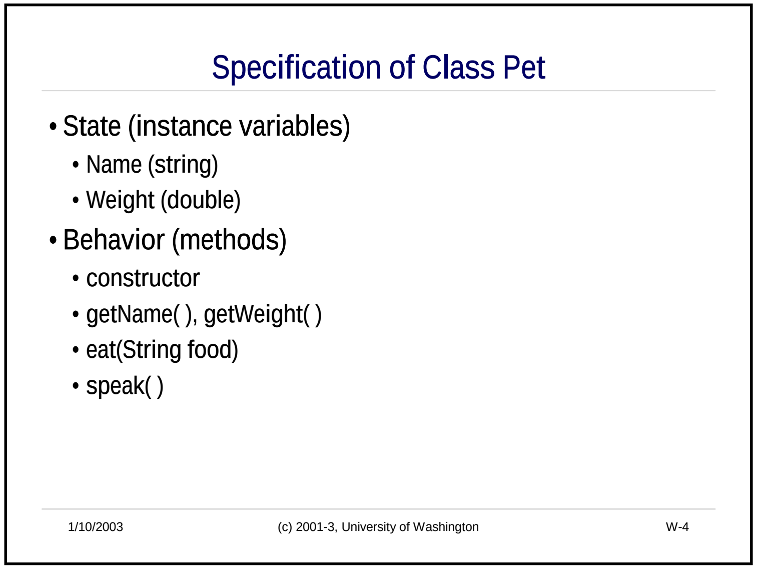
- Review
  - Inheritance – types and implementations
- Today
  - Abstract classes – specification + partial implementation
  - Interfaces – pure specification



A New Application

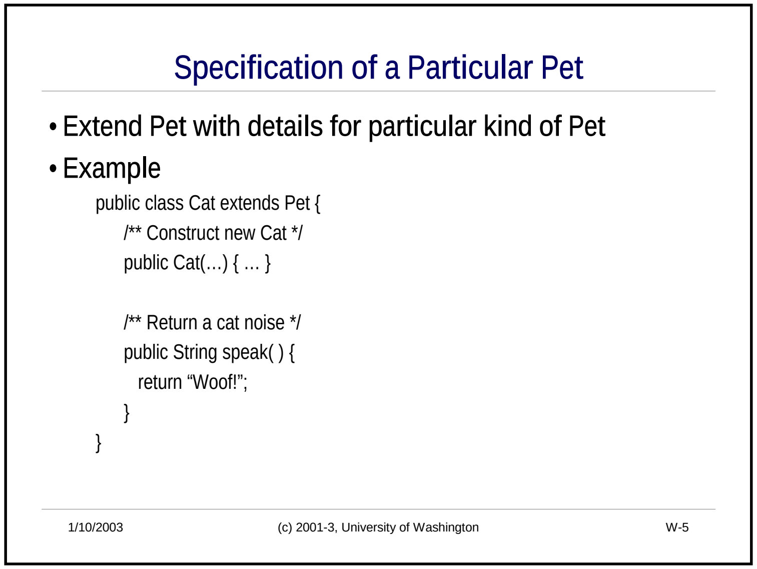
- Suppose we're designing the classes for a pet store simulation
- Inheritance makes sense – we need particular classes for specific kinds of pets, along with a generic "Pet" class that captures common behavior
- Client code can either deal with specific kinds of pets, or generic pets

```
/** Print information about pet p */
public void printInfo(Pet p) {
    System.out.println("Pet's name is " + p.getName( ) + " and it weighs " +
              p.getWeight( ) );
}
```



Specification of Class Pet

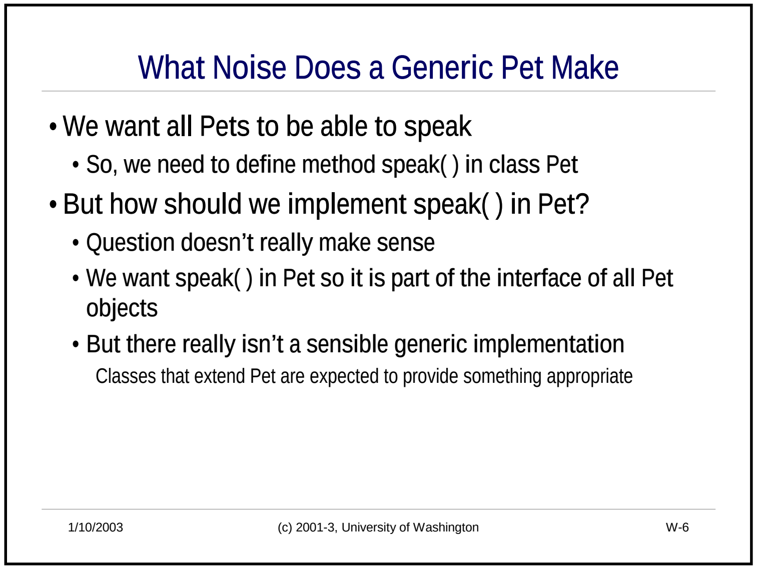
- State (instance variables)
  - Name (string)
  - Weight (double)
- Behavior (methods)
  - constructor
  - getName( ), getWeight( )
  - eat(String food)
  - speak( )



Specification of a Particular Pet

- Extend Pet with details for particular kind of Pet
- Example

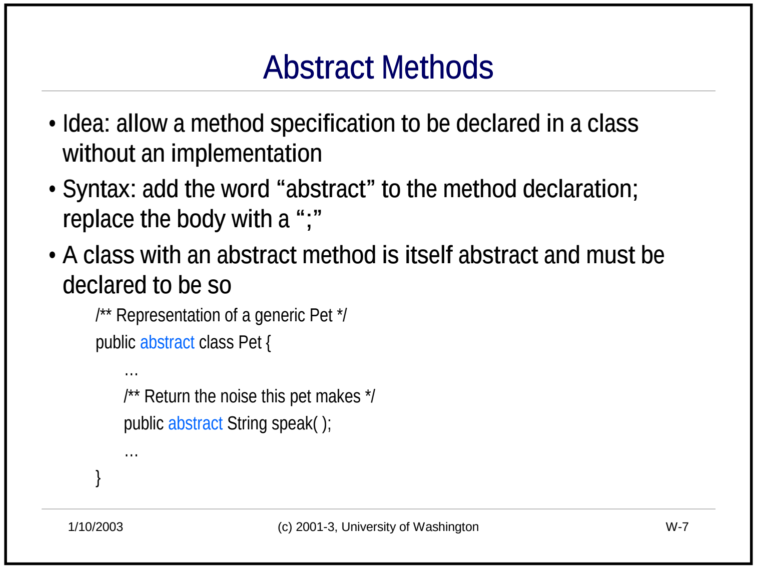
```
public class Cat extends Pet {
    /** Construct new Cat */
    public Cat(...) { ... }

    /** Return a cat noise */
    public String speak( ) {
      return “Woof!”;
    }
}
```



What Noise Does a Generic Pet Make

- We want all Pets to be able to speak
  - So, we need to define method speak( ) in class Pet
- But how should we implement speak( ) in Pet?
  - Question doesn’t really make sense
  - We want speak( ) in Pet so it is part of the interface of all Pet objects
  - But there really isn’t a sensible generic implementation

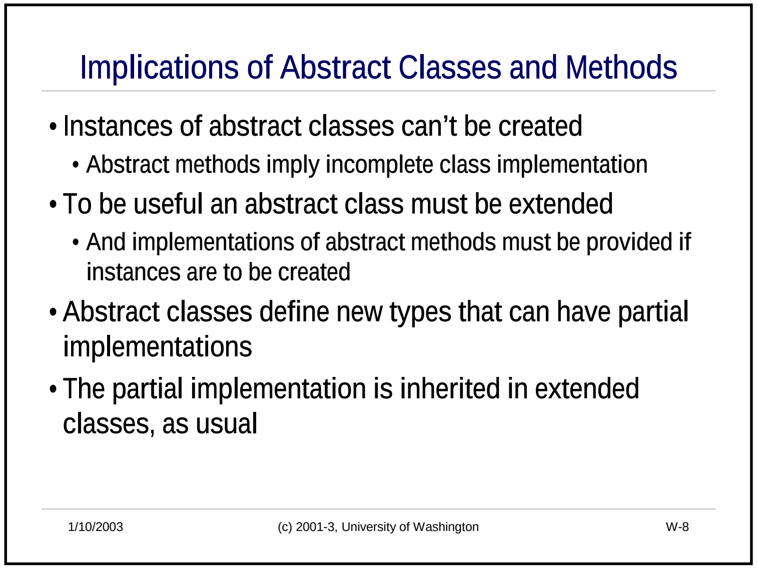
    Classes that extend Pet are expected to provide something appropriate



Abstract Methods

- Idea: allow a method specification to be declared in a class without an implementation
- Syntax: add the word “abstract” to the method declaration; replace the body with a “;”
- A class with an abstract method is itself abstract and must be declared to be so

```
/** Representation of a generic Pet */
public abstract class Pet {
    …
    /** Return the noise this pet makes */
    public abstract String speak( );
    …
}
```



Implications of Abstract Classes and Methods

- Instances of abstract classes can’t be created
  - Abstract methods imply incomplete class implementation
- To be useful an abstract class must be extended
  - And implementations of abstract methods must be provided if instances are to be created
- Abstract classes define new types that can have partial implementations
- The partial implementation is inherited in extended classes, as usual

Interfaces

- Sometimes we only want to define/describe a new type without providing any implementation at all
- We now have two choices
  - Define an abstract class with only abstract methods
  - Define an interface – pure specification with no implementation

Interface Definition (Review)

- Much like a class definition, but no method bodies and no state (except for static final constants)
- Everything is implicitly abstract, so “abstract” doesn’t need to be written

```
/** Interface filter. All the filters implement this interface */
public interface Filter {
  /** Modify the image according to the filter algorithm */
  void filter(PixelImage theImage);
}
```

Using Interfaces

- Any class can implement any interfaces that make sense
- Syntax

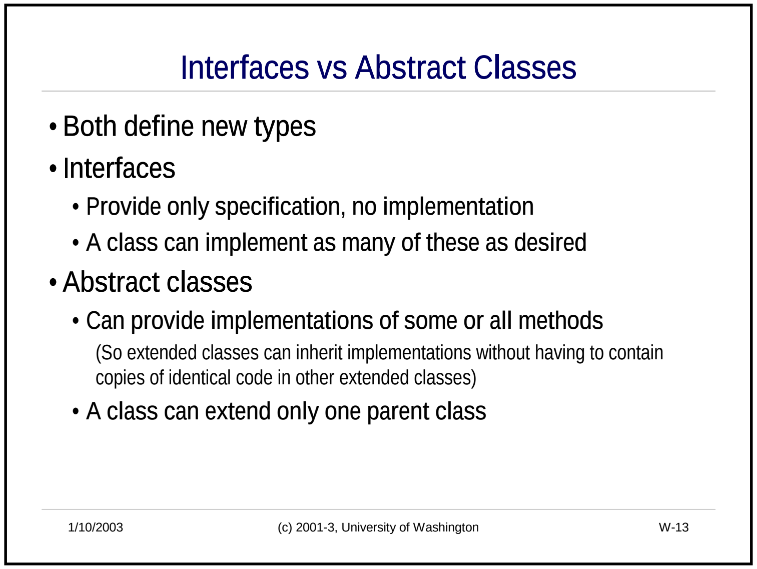
```
public class <classname> implements <interfacename> {
    …
}
```

- A class that implements an interface must provide implementations of all methods declared in the interface
  - No code is inherited from an interface
- An interface defines a new type
- Any class that implements an interface has the interface type (in addition to any others it defines or inherits)

Uses for Interfaces

- Allows a class to implement several different types
  - Can implement as many interfaces in a single class as makes sense for the application
- Allows *otherwise unrelated* classes to have common behavior
  - Example: objects in a simulation can all implement behavior that allows them to participate in the simulation, even if they have nothing else in common

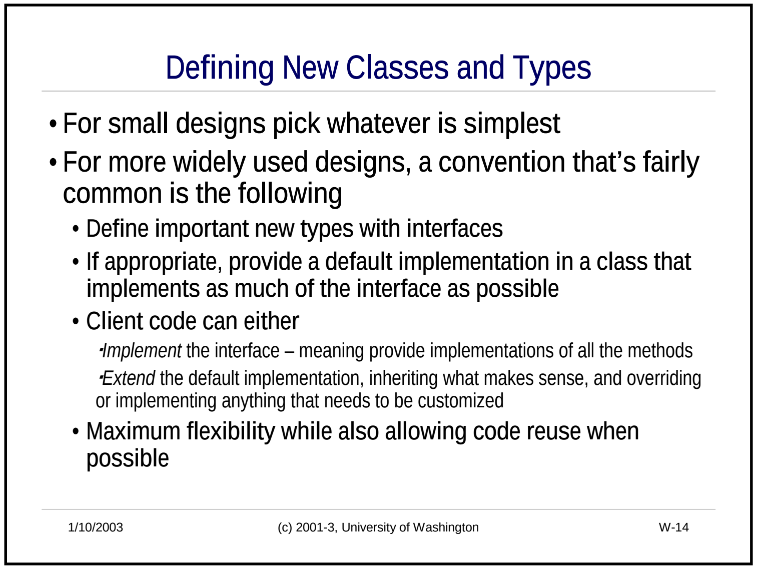
    Simulation engine only needs to know about the common interface



Interfaces vs Abstract Classes

- Both define new types
- Interfaces
  - Provide only specification, no implementation
  - A class can implement as many of these as desired
- Abstract classes
  - Can provide implementations of some or all methods

    (So extended classes can inherit implementations without having to contain copies of identical code in other extended classes)
  - A class can extend only one parent class



Defining New Classes and Types

- For small designs pick whatever is simplest
- For more widely used designs, a convention that's fairly common is the following
  - Define important new types with interfaces
  - If appropriate, provide a default implementation in a class that implements as much of the interface as possible
  - Client code can either
    - *Implement* the interface – meaning provide implementations of all the methods
    - *Extend* the default implementation, inheriting what makes sense, and overriding or implementing anything that needs to be customized
  - Maximum flexibility while also allowing code reuse when possible