CSE 143 Java

Abstract Classes and Frameworks

Reading: Ch. 15

What is a generic Animal?

- Example: class Animal (base class for Dog and Cat)
  - What noise should a generic Animal make?
  - Answer: class Animal doesn't have enough information to know!
- Purpose of class Animal
  - provide common specification for all Animals
  - provides implementation for some methods
  - intended to be extended, not used directly to create instances

Abstract Classes

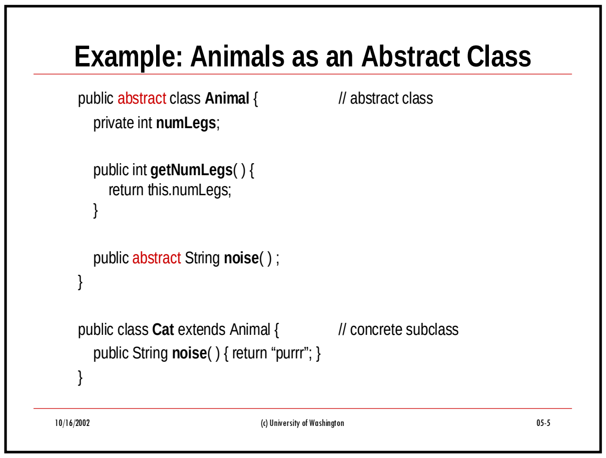
- Main idea: methods may be declared **abstract**, and left unimplemented

  public abstract myMethod( ) **;**
- If a class contains an abstract method, it must be declared as an **abstract** class with the ***abstract*** keyword

  public abstract class MyClass {...}
- Compare and contrast:
  - Interface
  - Abstract class
  - Concrete class

Abstract vs Concrete

- Cannot instantiate an abstract class (no `new`)
  - Like an interface
- A class that extends an abstract class can override methods (including abstract methods) as usual
- A class that provides implementations for all abstract methods it inherits is said to be **concrete**
  - If a class inherits an abstract method and doesn't override it, it is still abstract
  - An error message is reported if a non-abstract class doesn't implement all inherited abstract methods



Example: Animals as an Abstract Class

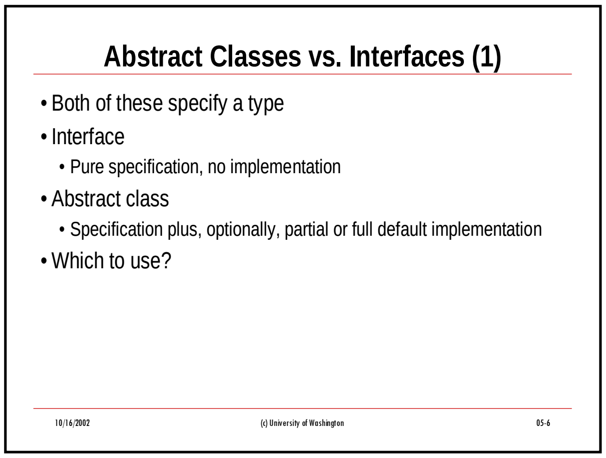
```
public abstract class Animal {                    // abstract class
    private int numLegs;

    public int getNumLegs( ) {
        return this.numLegs;
    }

    public abstract String noise( ) ;
}

public class Cat extends Animal {                 // concrete subclass
    public String noise( ) { return “purrr”; }
}
```

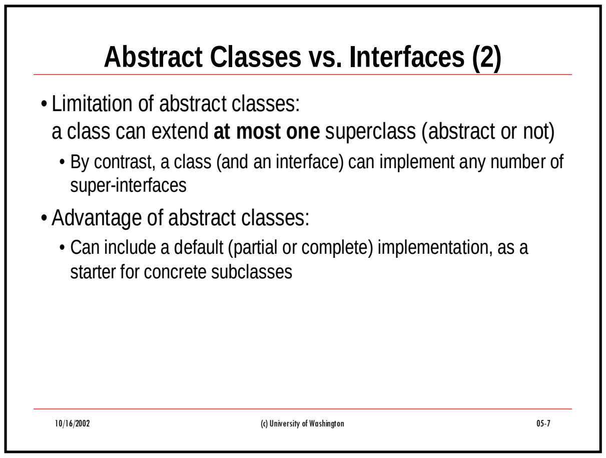
10/16/2002 (c) University of Washington 05-5



Abstract Classes vs. Interfaces (1)

- Both of these specify a type
- Interface
  - Pure specification, no implementation
- Abstract class
  - Specification plus, optionally, partial or full default implementation
- Which to use?

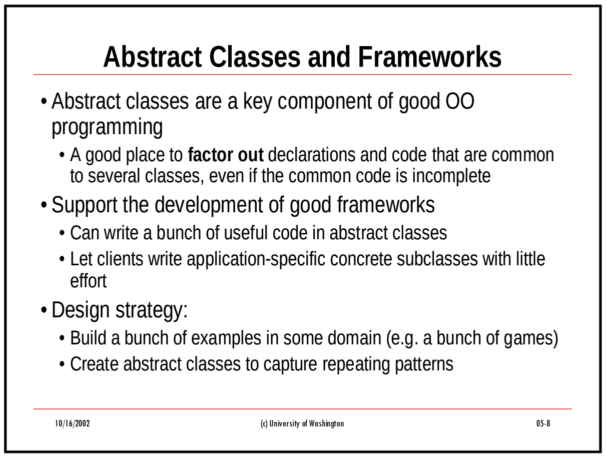
10/16/2002 (c) University of Washington 05-6



Abstract Classes vs. Interfaces (2)

- Limitation of abstract classes:
  a class can extend **at most one** superclass (abstract or not)
  - By contrast, a class (and an interface) can implement any number of super-interfaces
- Advantage of abstract classes:
  - Can include a default (partial or complete) implementation, as a starter for concrete subclasses

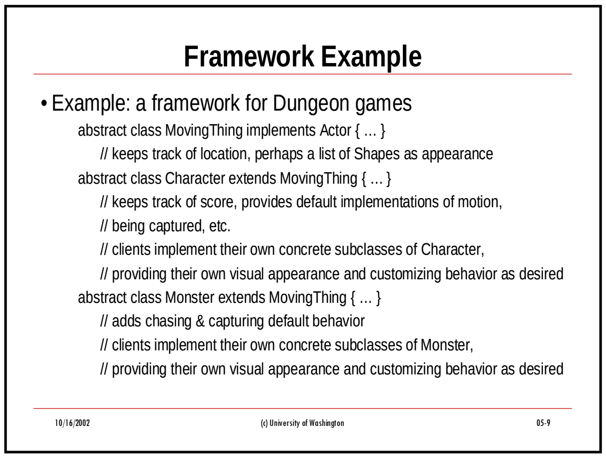
10/16/2002 (c) University of Washington 05-7



Abstract Classes and Frameworks

- Abstract classes are a key component of good OO programming
  - A good place to **factor out** declarations and code that are common to several classes, even if the common code is incomplete
- Support the development of good frameworks
  - Can write a bunch of useful code in abstract classes
  - Let clients write application-specific concrete subclasses with little effort
- Design strategy:
  - Build a bunch of examples in some domain (e.g. a bunch of games)
  - Create abstract classes to capture repeating patterns

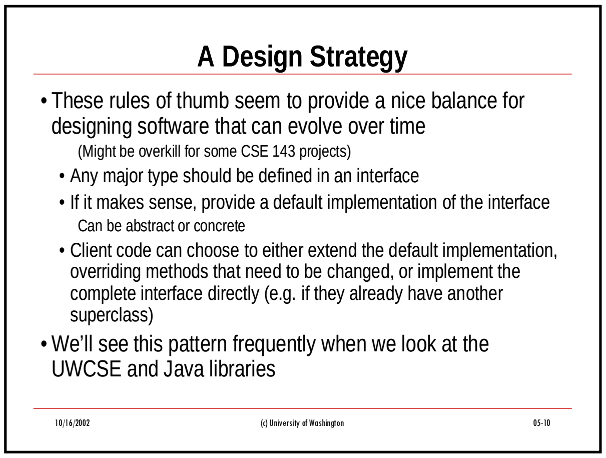
10/16/2002 (c) University of Washington 05-8



Framework Example

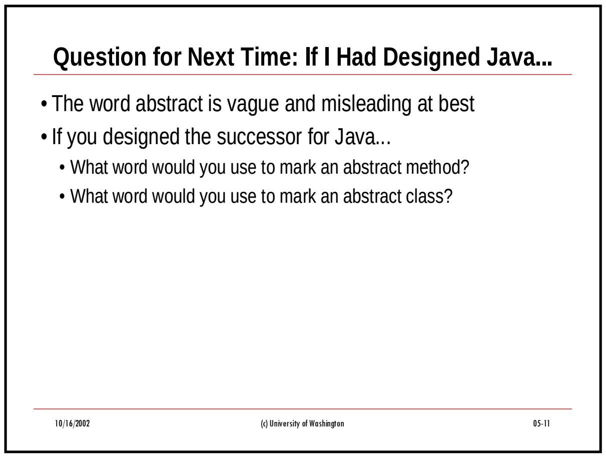
- Example: a framework for Dungeon games

```
abstract class MovingThing implements Actor { … }
    // keeps track of location, perhaps a list of Shapes as appearance
abstract class Character extends MovingThing { … }
    // keeps track of score, provides default implementations of motion,
    // being captured, etc.
    // clients implement their own concrete subclasses of Character,
    // providing their own visual appearance and customizing behavior as desired
abstract class Monster extends MovingThing { … }
    // adds chasing & capturing default behavior
    // clients implement their own concrete subclasses of Monster,
    // providing their own visual appearance and customizing behavior as desired
```



A Design Strategy

- These rules of thumb seem to provide a nice balance for designing software that can evolve over time
  (Might be overkill for some CSE 143 projects)
  - Any major type should be defined in an interface
  - If it makes sense, provide a default implementation of the interface
    Can be abstract or concrete
  - Client code can choose to either extend the default implementation, overriding methods that need to be changed, or implement the complete interface directly (e.g. if they already have another superclass)
- We'll see this pattern frequently when we look at the UWCSE and Java libraries



Question for Next Time: If I Had Designed Java...

- The word abstract is vague and misleading at best
- If you designed the successor for Java...
  - What word would you use to mark an abstract method?
  - What word would you use to mark an abstract class?