Inheritance In Java

By Brandon Cox,
Tyler Nuanes,
and Austin Uphus

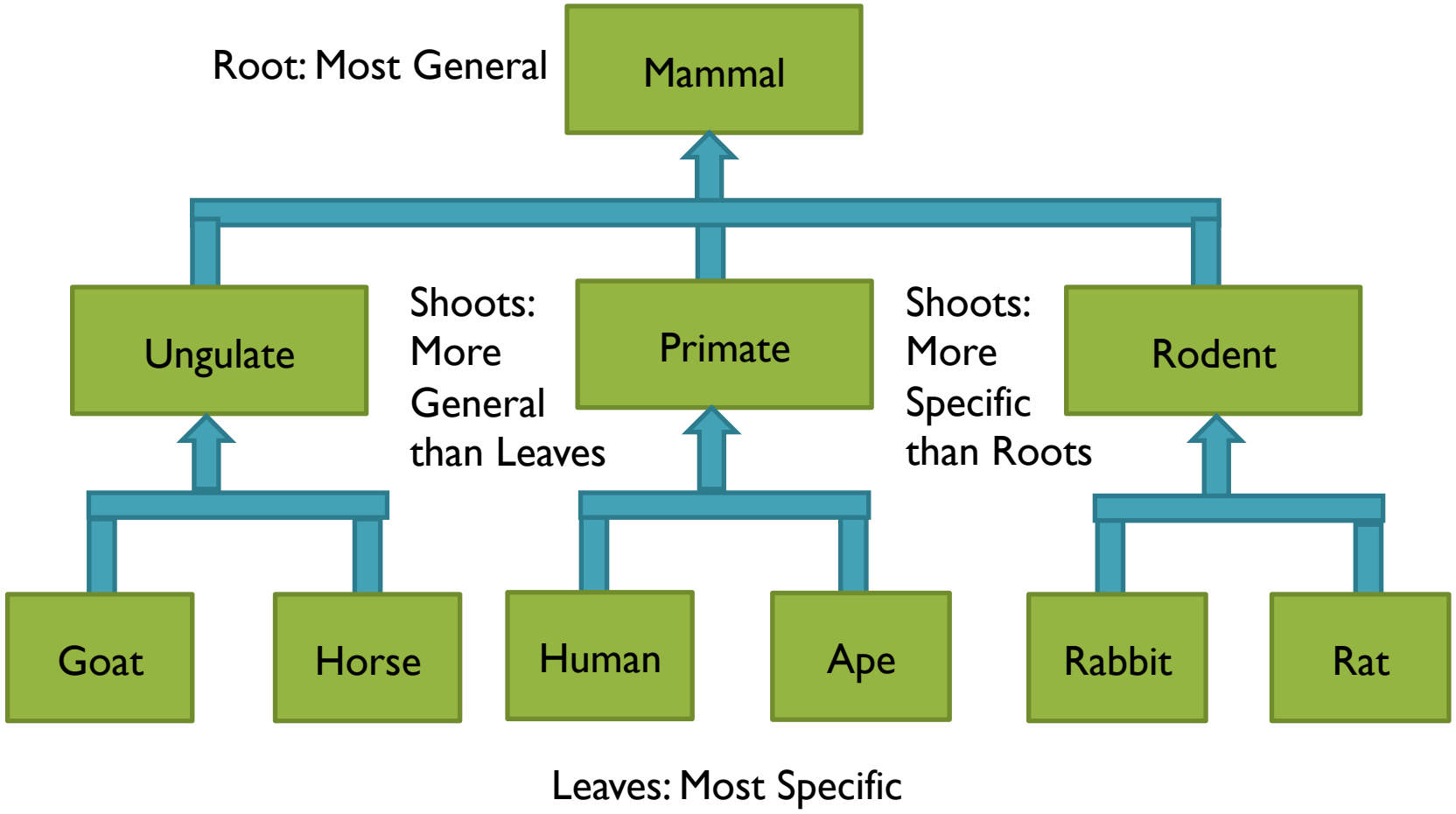
Dr. Defoe
CSSE221-2
September14, 2011

Inheritance Hierarchies

- Expresses the relationship between more general and more specific classes

- Use the *is-a* relationship
  - A primate *is-a* mammal
  - A human *is-a* primate

Class Diagrams: The Inverted Tree



Superclass vs. Subclass

- Describe relative relationships
  - A class can be *both* a superclass and a subclass
- Superclass: more general
- Subclass: more specific; inherits methods and instance variables from the superclass, but not constructors

Subclasses

- public class X extends Y {…}
  - X-subclass
  - Y-superclass

- Include all the methods from the superclass, but you should override or add to them for specialized purposes

Constructors and Methods

- Say you need to construct a superclass inside your construction for a subclass
  - To construct in the superclass:

    super(parameter);
    - Note: Must be the first statement in the subclass constructor

- Say you have a doThis(type parameter) method in your subclass and superclass
  - To call the superclass:

    super.doThis(parameter);

Conclusion: Why Use Inheritance?

- Saves time: allows code and structural reuse
- Prevents errors: using previously tested code leads to greater reliability
- Efficient: avoids redundant code in related classes

Abstract Classes In Java

By Brandon Cox,
Tyler Nuanes,
and Austin Uphus

Dr. Defoe
CSSE221-2
September14, 2011

Abstract Classes

- Cannot be instantiated, meaning you cannot create an object for the class
  - Classes which can be instantiated are called concrete classes
- Abstract methods contain no code
  - They exist for organizational purposes
- public abstract class X {…}
  - Note: you can declare methods abstract in the same way

Example

```
public abstract class Shape {
    //Description of method
    public abstract double getArea();
    //Description of method
    public abstract double getVolume();
}
```

- Note: when inheriting abstract methods, you must override them
- However, Javadoc comments will remain intact, saving coders time

Abstract Variables

- You can have a variable with the *type* of an abstract class, even though the object to which it *refers* cannot be abstract
  - AbstractClass class = new AbstractClass()
    - WRONG!
  - AbstractClass class = new ConcreteClass()
    - RIGHT!

Abstract Classes vs. Interfaces

- Interfaces cannot have instance variables, concrete methods, or constructors

- Abstract classes can have instance variables, concrete methods, and constructors.

Conclusion: Why Use Abstract Classes?

- Prevent errors: force programmers to override methods
  - Why? There are no good default methods
- Saves time: many different classes use the same methods but implement them differently

Citations

- http://stason.org/TULARC/software/object-oriented-programming/1-12-Why-Use-Inheritance-Object-Oriented-Technology.html
- *Big Java 4th Edition* by Cay Horstmann