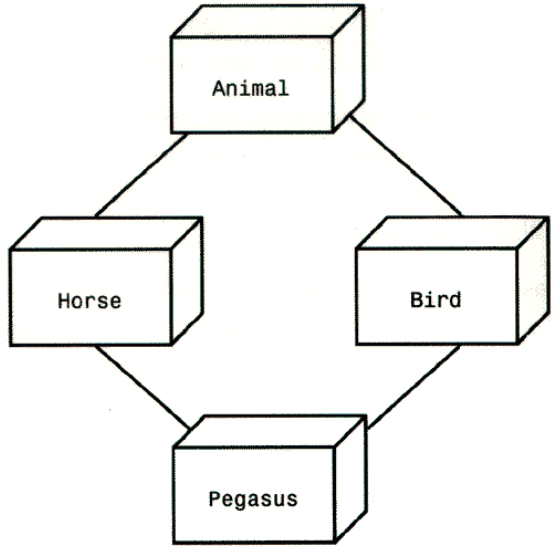
Software and Programming II

Introduction to Inheritance

Department of Computer Science and Information Systems
Birkbeck, University of London

keith@dcs.bbk.ac.uk

October 7, 2014



Topics

- Inheritance Hierarchies
- Implementing Subclasses
- Overriding Methods

Inheritance Hierarchies

In object-oriented programming, inheritance is a relationship between:

- A superclass: a more generalised class
- A subclass: a more specialised class

The subclass *inherits* data (variables) and behaviour (methods) from the superclass

Classes form a hierarchy

- Classes are arranged is a treelike hierarchy
- There is one class at the top, or root, of the hierarchy, named `Object`

  In computer science we draw trees upside-down, with the root at the top
- Every class except Object has one *parent* class, or superclass
- Each class is a subclass of its superclass

What is the class hierarchy for?

- Classes inherit from their superclasses
- A class has not only its own fields and methods, but also:
  - Every field described in any class above it
  - Every method described in any class above it
  - Classes do not inherit constructors, however
  - Please note: this all depends on the *access modifiers*
- Hence, a class may contain much more information than is obvious from the class description
- The *access modifiers* are private, protected, *package*, and public

Example of inheritance

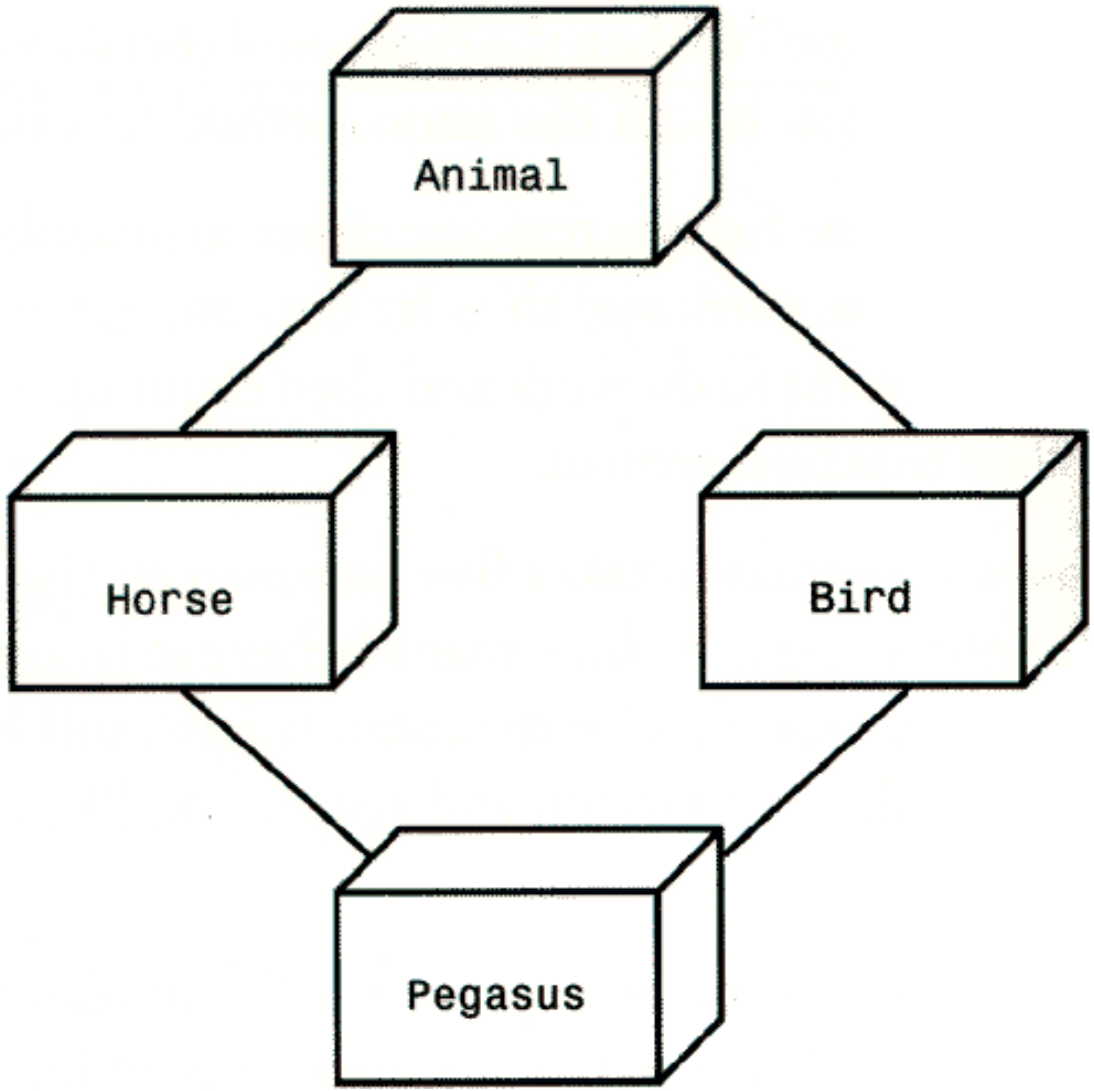
```
package inherit;

public class Employee extends Person {
  private double hourlyWage;

  // cons

  public Employee(String name){
    // call constructor of parent class
    super(name);
  }

  public void pay(double hoursWorked){
    System.out.println("Pay: " + name + " " + hoursWorked * hourlyWage);
  }



}
```

- An `Employee` has a `name`, and age, an `hourlyWage`, and `birthday` and `pay` methods.
- In addition, the class *inherits* some methods from the `Object` class — dependent upon the access modifiers

The modified Person class

```
package inherit;

public class Person {
  // fields with implicit access modifiers
  protected String name;
  protected int age;

  // constructor
  public Person(String name){
    this.name = name;
    age = 0;
  }

  // methods

  public String getName(){
    return name;
  }

  public void birthday(){
    age++;
    System.out.println("Happy birthday!");
  }
}
```

Another example hierarchy



The *Substitution* Principle

Since the subclass `Horse` *is-a* `Animal`

- `Horse` shares common traits with `Animal`
- You can substitute a `Horse` object in an algorithm that expects an `Animal` object

```
Horse horse = new Horse(...);
processAnimal(horse);
```

The *is-a* relationship is represented by an arrow in a class diagram and means that the subclass can behave as an object of the superclass.

Quiz Question Hierarchy

Imagine that there are different types of quiz questions:

1. Fill-in-the-blank
2. Single answer choice
3. Multiple answer choice
4. Numeric answer
5. Free Response

and a question can:

- Display it's text
- Check for a correct answer

Question Class

`Question.java` and `QuestionDemo.java`

General notes

- Use a *Single Class* for variation in *Values*, and
- *Inheritance* for variation in *Behaviour*

Example

if two vehicles only vary by fuel efficiency, use an instance variable for the variation, not inheritance

```
// Car instance variable
double milesPerGallon;
```

- Therefore, if two vehicles behave differently, use inheritance (but don't over-do it!)

Implementing Subclasses

Consider implementing `ChoiceQuestion` to handle:

```
In which country was the inventor of Java born?
1. Australia
2. Canada
3. Denmark
4. United States
```

How does `ChoiceQuestion` differ from `Question`?

- It stores choices (1,2,3 and 4) in addition to the question
- There must be a method for adding multiple choices
- The `display` method will show these choices below the question, numbered appropriately

Inheriting from the Superclass

- Subclasses inherit from the superclass:
  - All public methods that it does not override
  - All instance variables
- The Subclass can
  - Add new instance variables
  - Add new methods
  - Change the implementation of inherited methods

Overriding Superclass Methods

- Can you re-use any methods of the Question class?
  - Inherited methods perform exactly the same
  - If you need to change how a method works:
    - Write a new more specialised method in the subclass
    - Use the same method name as the superclass method you want to replace
    - It must take all of the same parameters
  - This will override the superclass method
- The new method will be invoked with the same method name when it is called on a subclass object
- We use the reserved word `extends` to inherit from the superclass

Overriding Methods

- The `ChoiceQuestion` class needs a `display` method that overrides the `display` method of the (revised) `Question` class
- They are two different method implementations
- The two methods named display are:

`Question display`	Displays the instance variable `text`
`ChoiceQuestion display`	Overrides `Question display` method Displays the instance variable `text` Displays the local list of choices

Calling Superclass Methods

- Consider the `display` method of the `ChoiceQuestion` class
- It needs to display the question **AND** the list of choices
- `text` is a private instance variable of the superclass
- How do you get access to it to print the question?
- Call the `display` method of the superclass `Question` of course!
- How do we do this?
- From a subclass, preface the method name with:

```
super.
```

```
public void display() {
  // Display the question text
  super.display(); // OK
  // Display the answer choices
  . . .
}
```

Calling the Superclass Constructor

- When a subclass is instantiated, it will call the superclass constructor with no arguments
- If you wish to call a more specific constructor, you can invoke it by using replacing the superclass name with the reserved word `super` followed by `()`:

```
public ChoiceQuestion(String questionText){
  super(questionText);
  choices = new ArrayList<String>();
}
```

- The call to `super` must be the first statement in your constructor

Summary I

- A subclass inherits data and behaviour from a superclass
- You can always use a subclass object in place of a superclass object
- A subclass inherits all methods that it does not override
- A subclass can override a superclass method by providing a new implementation

Summary II

- An overriding method can extend or replace the functionality of the superclass method
- Use the reserved word `super` to call a superclass method
- Unless specified otherwise, the subclass constructor calls the superclass constructor with no arguments
- To call a superclass constructor, use the `super` reserved word in the first statement of the subclass constructor
- The constructor of a subclass can pass arguments to a superclass constructor, using the reserved word `super`

Questions