Lecture 12 Summary

- Inheritance
  - Superclasses / subclasses
  - Inheritance in Java
  - Overriding methods
  - Abstract classes and methods
  - Final classes and methods
- Multiplicity

By the end of this lecture, you will be able to incorporate *inheritance* and *multiplicity* into your class models.

You will also be able to use inheritance and *override* methods in Java.

Process So far...

- Identify objects and create an *object model*
- Observe commonalities in object model
  - objects with the same attributes/behaviour
- Classify common objects into a *class model*
  - remove repetition (number of objects/relations)

Additional Step

- Identify objects and create an *object model*
- Observe commonalities in object model
  - objects with the same attributes/behaviour
- Classify common objects into a *class model*
  - remove repetition (number of objects/relations)
- **Find commonalities in class model and abstract them using *inheritance*.**
  - **this process is called *generalization***

In procedural languages, what is the benefit of dividing your code up into multiple procedures?

In OO languages, what is the benefit of classifying objects in your object model?

Slides by Mark Hancock
(adapted from notes by Craig Schock) 5

The process of *abstraction* helps to reduce the *complexity* of the problem space.
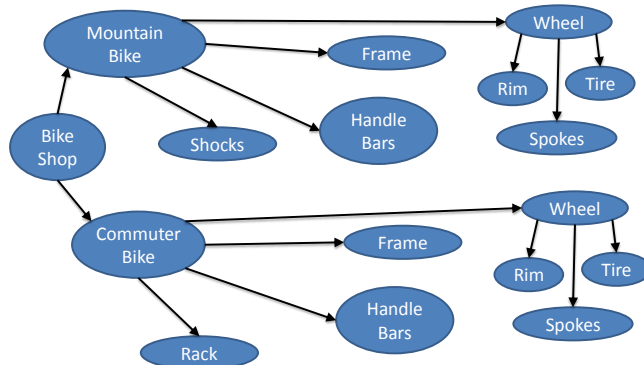
Generalization is another form of abstraction.

Slides by Mark Hancock
(adapted from notes by Craig Schock) 6

Draw the class model...

Slides by Mark Hancock
(adapted from notes by Craig Schock) 7

What are the commonalities in the *object model*?

What are the commonalities in the *class model*?

Slides by Mark Hancock
(adapted from notes by Craig Schock) 8

Inheritance
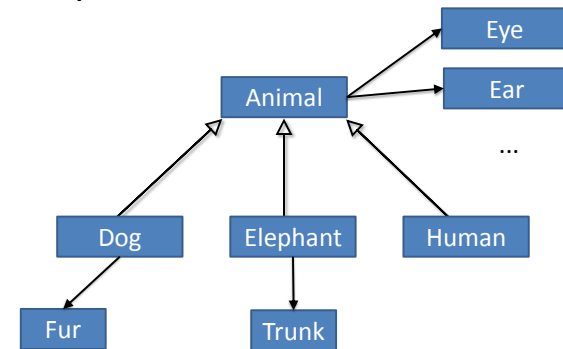
- A new kind of relationship between classes:
  – *is-a* or *is-a-kind-of*
- Used to describe a *group* of classes in an abstract way

Slides by Mark Hancock
(adapted from notes by Craig Schock)

9

Example

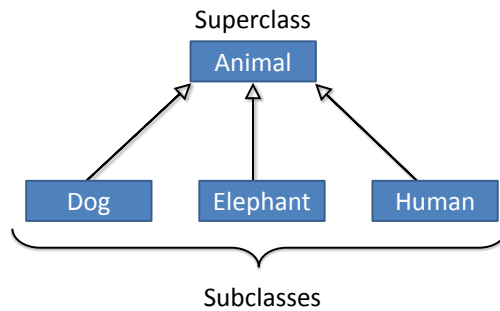Slides by Mark Hancock
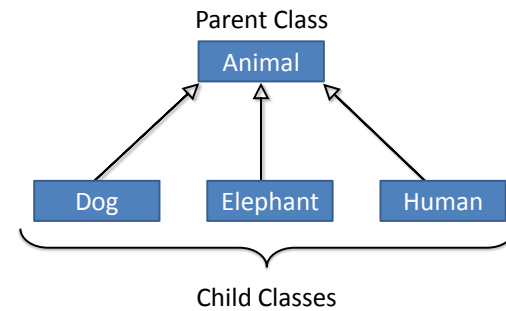(adapted from notes by Craig Schock)

10

Terminology

Superclass

Subclasses

Slides by Mark Hancock
(adapted from notes by Craig Schock)

11

Terminology

Parent Class

Child Classes

Slides by Mark Hancock
(adapted from notes by Craig Schock)

12

Exercise: *generalize* the bike shop class model.

Java Syntax

```
public class <subclass> extends <superclass>
{
    ...
}
```

Example

Animal.java:
```
public class Animal {
    private Eye leftEye;
    private Eye rightEye;

    private Ear leftEar;
    private Ear rightEar;
}
```

Dog.java:
```
public class Dog extends Animal {
    private Fur fur;
}
```

Elephant.java:
```
public class Elephant extends Animal {
    private Trunk trunk;
}
```

Human.java:
```
public class Human extends Animal {
}
```

Which code makes sense?

```
1. public class MainProgram {
2.     public static void main(String[] args) {
3.         Animal a = new Dog();
4.         Elephant e = new Animal();
5.
6.         LinkedList<Animal> animals =
7.                     new LinkedList<Animal>();
8.         animals.add( new Human() );
9.         animals.add( new Elephant() );
10.
11.         LinkedList<Dog> dogs =
12.                     new LinkedList<Dog>();
13.         dog.add( new Dog() );
14.         dog.add( new Human() );
15.     }
16.}
```

What if we want a method called 'speak' that plays the right sound for each animal?

Where does this method belong?

Slides by Mark Hancock
(adapted from notes by Craig Schock)

Consider the following code...

```
public class MainProgram
{
    public static void main(String[] args)
    {
        LinkedList<Animal> animals =
                        new LinkedList<Animal>();
        ... // fill up the list

        for (Animal a : animals)
        {
            a.speak();    What should happen here?
        }
    }
}
```

Slides by Mark Hancock
(adapted from notes by Craig Schock)

Overriding Methods

- A subclass can override a method in the superclass
- Automatically happens by using the same method *signature*
  – same name
  – same parameters
  – same return type

Slides by Mark Hancock
(adapted from notes by Craig Schock)

Example

```
public class Animal {            public class Dog extends Animal {
    ...                              ...

    public void speak() {            public void speak() {
        ...                              playAudioClip("bark.wav");
    }                                }
}                                }
```

What code goes here?

Slides by Mark Hancock
(adapted from notes by Craig Schock)

Abstract Classes & Methods

- An *abstract class* is a class that cannot be *instantiated*
  - No instances can be created
- An *abstract method* is a method that must be *overridden* by any subclass

Example

```
public abstract class Animal {
    ...

    public abstract void speak();
}
```

Reconsider the following code...

```
public class MainProgram
{
    public static void main(String[] args)
    {
        LinkedList<Animal> animals =
                    new LinkedList<Animal>();
        ... // fill up the list

        for (Animal a : animals)
        {
            a.speak();
        }
    }
}
```

Can you think of a situation where you would want to prevent a method from being overridden?

`final`

- You can prevent a method from being overridden by adding the keyword 'final'.

- You can prevent a class from being inherited from by adding the keyword 'final'.

Slides by Mark Hancock (adapted from notes by Craig Schock) 25

Example

```java
public class Security {
    public final boolean matchPassword(String password) {
        ...
    }
}
```

Slides by Mark Hancock (adapted from notes by Craig Schock) 26

Does the following code make sense?

```java
public abstract final ParentClass {
}
```

Slides by Mark Hancock (adapted from notes by Craig Schock) 27

Root Classes

- There is a class called *Object* in Java
- Every class *is an* Object.
- If you do not specify a superclass through the *extends* keyword, Java automatically inherits from Object
- Object is the *root class* for Java
- Not all languages have a root class (e.g., C++)

Slides by Mark Hancock (adapted from notes by Craig Schock) 28

Single Inheritance

- Java only allows single inheritance:
  - A class can only have *one* superclass
  - This superclass may itself inheret from another class, and so on, until Object is reached
  - Object is the only class with no superclass

- C++ allows for multiple inheritance
  - A class in C++ can have many superclasses

Inheritance Summary

- Commonalities between classes in the class model can be abstracted using *inheritance*.
- Inheritance introduces the *is-a* relationship to our class models.
- In Java, a class can inherit from a *superclass* using the *extends* keyword.
- An instance of a subclass can be *substituted* for a reference to a superclass

Inheritance Summary

- Methods in a superclass can be *overridden* in a subclass
- An *abstract* class cannot be instantiated (only its non-abstract subclasses can).
- You can prevent a class from being subclassed or a method from being overridden with the *final* keyword.
- Java uses single inheritance and has a root class called *Object*.

Multiplicity

- The has-a relationship can be further decomposed:
  - has one
  - has many
  - belongs to one
  - belongs to many

Multiplicity

Driver	1 → 1	Car
Tree	1 → *	Leaf
Purchase	* → 1	Customer
Document	* → *	Word
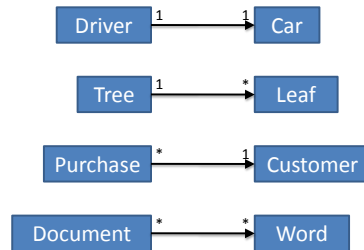
Slides by Mark Hancock
(adapted from notes by Craig Schock)
33

Exercise: add indications in the bike shop class model for multiplicity.

Slides by Mark Hancock
(adapted from notes by Craig Schock)
34

Lecture 12 Summary

- Inheritance
  - Superclasses / subclasses
  - Inheritance in Java
  - Overriding methods
  - Abstract classes and methods
  - Final classes and methods
- Multiplicity

Slides by Mark Hancock
(adapted from notes by Craig Schock)
35

Next Class

- Polymorphism (theory)
- Interfaces (a.k.a. pure abstract classes)

Slides by Mark Hancock
(adapted from notes by Craig Schock)
36