Lecture 22

There is another kind of relationship that exists in the 'real world' that we would want to model in our software. This is a relationship between classes rather than objects: one class may be a *generalisation* of another.

We show that, in the real world, this relationship leads to the formation of hierarchies of classes called *class hierarchies* or generalisation hierarchies.

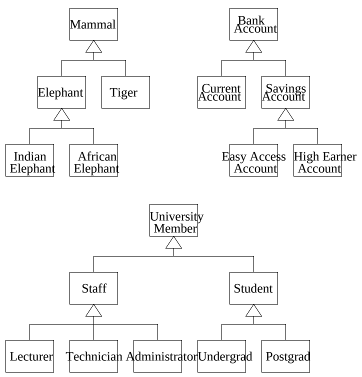
We see that Java offers a mechanism for easy modeling of generalisation: *inheritance*. We discuss the inheritance of instance variables and whether we can 'block' inheritance.

The example is a lecturer's research publications.

Class hierarchies

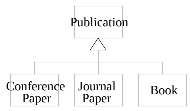
- In the 'real-world', classes often have *subclasses*. E.g. elephants are a subclass of mammals.
- Equivalently, by inverting our perspective, we can say that a class may have a *superclass*. E.g. mammals are a superclass of elephants.
- Sometimes subclasses themselves have subclasses. E.g. Indian elephants are, in turn, a subclass of elephants.
- The result is a *class hierarchy*.
- In a given hierarchy, it is sometimes necessary to distinguish the special case of being a *direct* or *immediate subclass/ superclass*. E.g. Indian elephants and elephants in general are subclasses of mammal; elephant is a direct/ immediate subclass of mammal

Example class hierarchies



Note how this is a relationship between classes, rather than between individual objects.

Our main example



- Publications have a title, author and a year of publication
- Conference papers have a title, author, year of publication, a conference name, and start and end page numbers
- Journal papers have a title, author, year of publication, a journal name and start and end page numbers
- Books have a title, author, year of publication and publisher

Duplicating the variables: usually a bad approach

```
public class Publication
{
   private String title;
   private PersonName author;
   private int yearOfPub;
}

public class ConferencePaper
{
   private String title;
   private PersonName author;
   private int yearOfPub;
   private String confName;
   private int startPage;
   private int endPage;
}
```

— similar duplication in `JournalPaper` and `Book`

Inheritance: a better alternative

- We indicate that one class is a subclass of another using the keyword `extends`
- Then the subclass *inherits* the instance variables of its superclass
- I.e. by inheritance, the subclass incorporates the instance variables and instance methods of the superclass
- It can also *extend* the superclass by declaring further instance variables/methods of its own

Using inheritance

```
public class Publication
{
   private String title;
   private PersonName author;
   private int yearOfPub;
}

public class ConferencePaper
             extends Publication
{
   private String confName;
   private int startPage;
   private int endPage;
}
```

— objects of class `ConferencePaper` still have 6 instance variables

Visibility of inherited variables

- Recall: `public` variables are directly accessible from all classes; `private` variables are directly accessible only within the class in which they are defined
- So, on the previous slide, the methods within `ConferencePaper` cannot directly assign to, retrieve from or compute with `title`, `author` and `yearOfPub`
- The access modifier `protected`: makes a variable/method visible to subclasses (and to any classes in the same package)

Using `protected`

```
public class Publication
{
   protected String title;
   protected PersonName author;
   protected int yearOfPub;
}

public class ConferencePaper
             extends Publication
{
   private String confName;
   private int startPage;
   private int endPage;
}
```

— `ConferencePaper` now has 6 visible instance variables.

(If `ConferencePaper` is to have subclasses of its own, then it may want to make its variables visible by making them `protected`)

Final remarks

- Some people believe that you should avoid `protected`
  - Subclasses can access inherited variables using `public` getters/setters in the interface of the superclass
  - This approach reduces coupling
- What if you don't want to inherit all the instance variables?
  - Too bad: instance variables are always inherited
  - Actually, there is a way of 'hiding' inherited instance variables
  - Rather than do this, you might do better to think about reworking your hierarchy