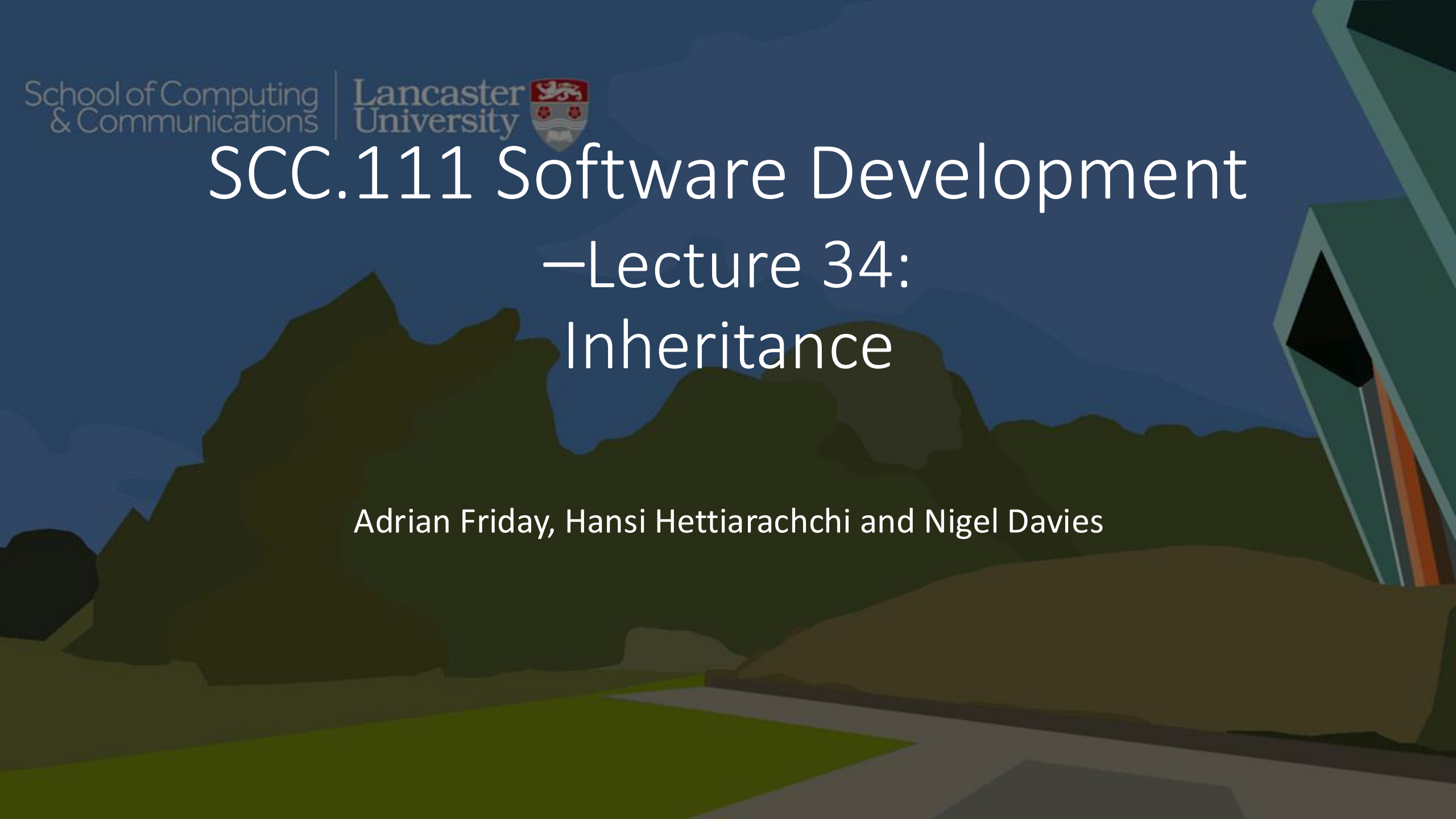


SCC.111 Software Development –Lecture 34: Inheritance

Adrian Friday, Hansi Hettiarachchi and Nigel Davies

Introduction

Lancaster University

- In the last lectures, we looked at:
  - How **version control** can promote collaboration
  - Practical guidance on using **Git** version control
  - Examples of a typical Git Workflows
  - CI/CD: Continuous Integration/Continuous Deployment

- Today we're going to move on to a new concept in Object Oriented programming
  - **Inheritance**

Motivation

Lancaster University

We have seen how classes promote modularity and reuse.

- But what if that class doesn't quite do what we want? Or we want it to do more?
- Wouldn't it be cool if we could **specialize** a class that has already been written, but without changing that code, or duplicating it?

- In OO languages, **inheritance** provides a mechanism to promote reuse and extensibility.

- Code written by one programmer and delivered as a class can then be **extended** by another to suit the needs of the task at hand.

- The original programmer need not know anything about the application space in order to support it. Nor need he release his source code to allow his/her classes to be specialized...

Inheritance: Definition

Lancaster University

One class can inherit from, be derived from, or extend another to create new class.

(note: These are different words for the same thing, used by different languages!)

- We know that classes are simply a grouping of attributes and methods…
- When a class is extended, those attributes and methods implicitly become part of the new class.
- Those attributes and methods can be accessed/invoked on the new class even though they are not explicitly defined in that class.
- The new class can add further capability by creating new attributes and methods, without polluting the original class.

Inheritance: Terminology

Lancaster University

One class can inherit from, be derived from, or extend another to create new class.

- The class being extended is known as the **base class**, **super class** or the **parent** class.
- The new class is known as a **subclass** of the other.
- A class can have many subclasses.

- Some languages permit multiple inheritance languages, where a class can have many super classes too. **C++ and Python are examples.**

- Most OO languages are single inheritance languages, where a class can have only one super class. **Java, C#, Javascript are examples.**

Inheritance: In Java

Lancaster University

When we define a class, we can optionally define its super class too.

- We use the **extends** keyword when defining the class to define its super class.

```
public class Lecturer extends Person
{
    ...
}
```

- All methods and instance variables defined in the super class implicitly become part of the new subclass...
- Any **accessible** methods and attributes can be directly used by methods in the new subclass

Inheritance: In C++

Lancaster University

When we define a class, we can optionally define its super class too.

- We use a **:** when defining the class to define its super class.

```
class Lecturer : public Person
{
    ...
}
```

- All methods and instance variables defined in the super class implicitly become part of the new subclass...
- If we add the public keyword, before the super class name, all the public methods and attributes become public in our class too. Otherwise, they are private to our class...

Inheritance is transitive

Lancaster University

Subclasses can themselves be super classes of other classes!

- If a class is a super class of all its subclasses… and their subclasses…
- e.g. in the example below class Lecturer is a subclass of Person and Mammal
- The **Lecturer** class therefore inherits attributes and methods from both the **Person** and **Mammal** classes.

```
public class Mammal
{
    ...
}
```

```
public class Person extends Mammal
{
    ...
}
```

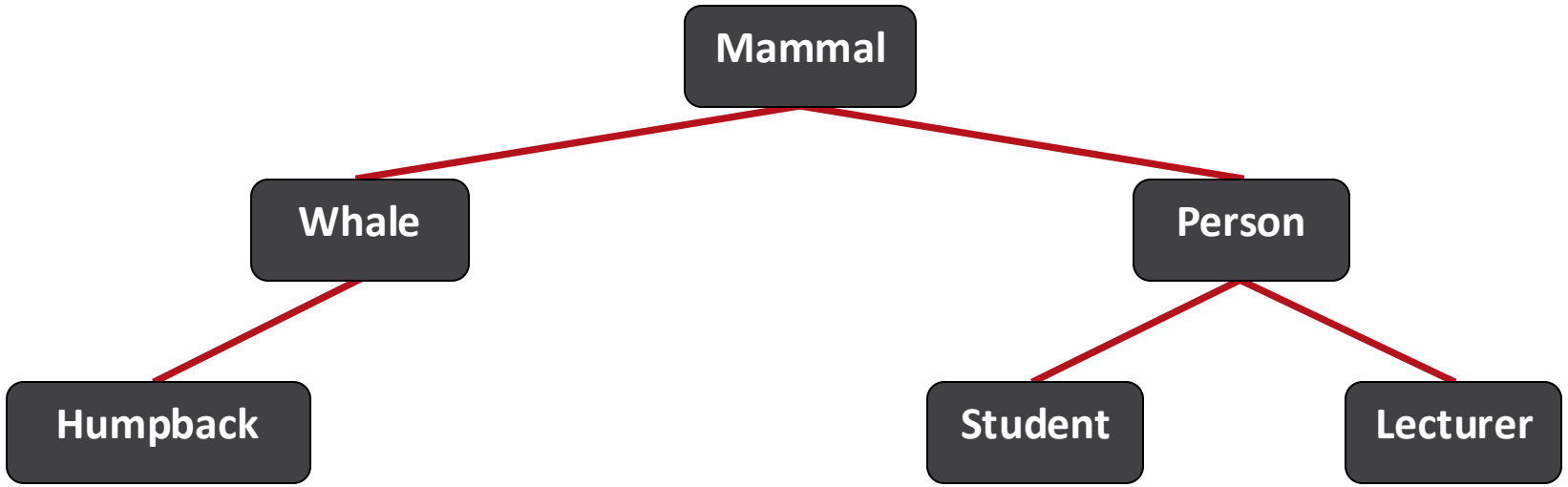
```
public class Lecturer extends Person
{
    ...
}
```

Inheritance hierarchies

Lancaster University

As inheritance is transitive and we can create many subclasses...

- We therefore have a tree structure, known as an **inheritance hierarchy**.
- All the inheritance relationships can be mapped out a bit like a family tree…
- **The attributes and methods your class has depends on its position in that tree.**

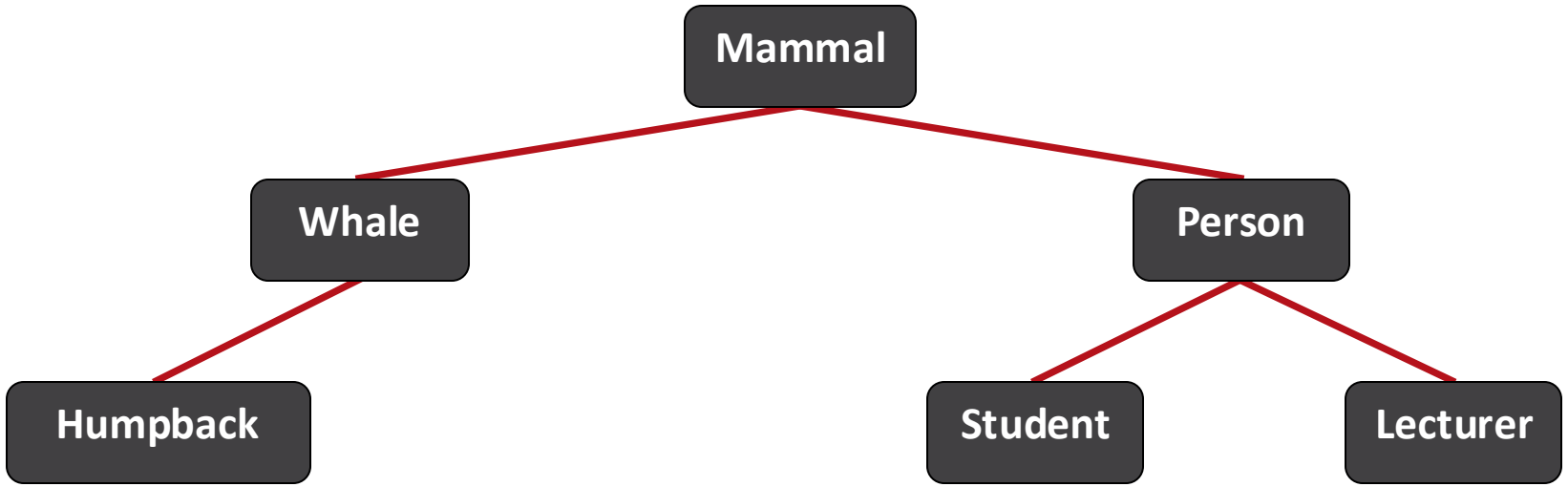


Inheritance hierarchies…

Lancaster University

Note that the level of abstraction changes as we move up and down the hierarchy

- Move up the tree we become more **generalist** and **abstract**.
- Move down the tree we become more **specialist** and **specific**.
- **This is equally true for any OO programs we write.**

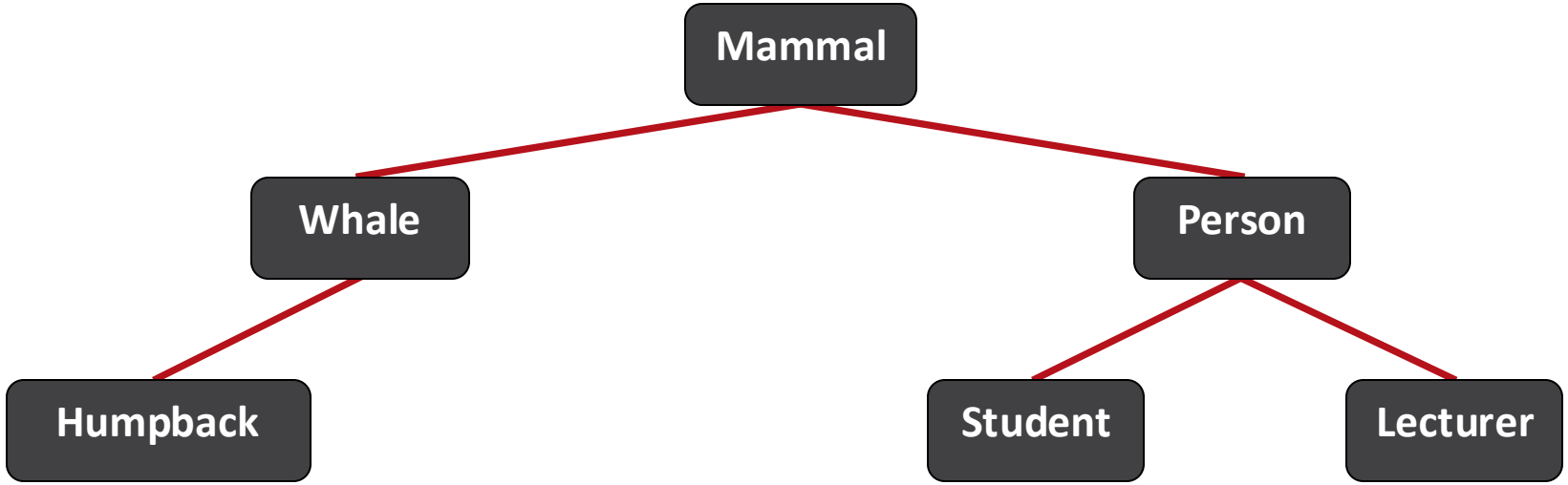


Inheritance hierarchies….

Lancaster University

Inheritance is the primary mechanism to provide extensibility in OO languages

- You can use it to specialise an existing class to suit your needs.
- Deciding where to implement a method or place an attribute is therefore a very important decision!
- To promote reuse, typically implement at as high a level as makes sense (mostly!)



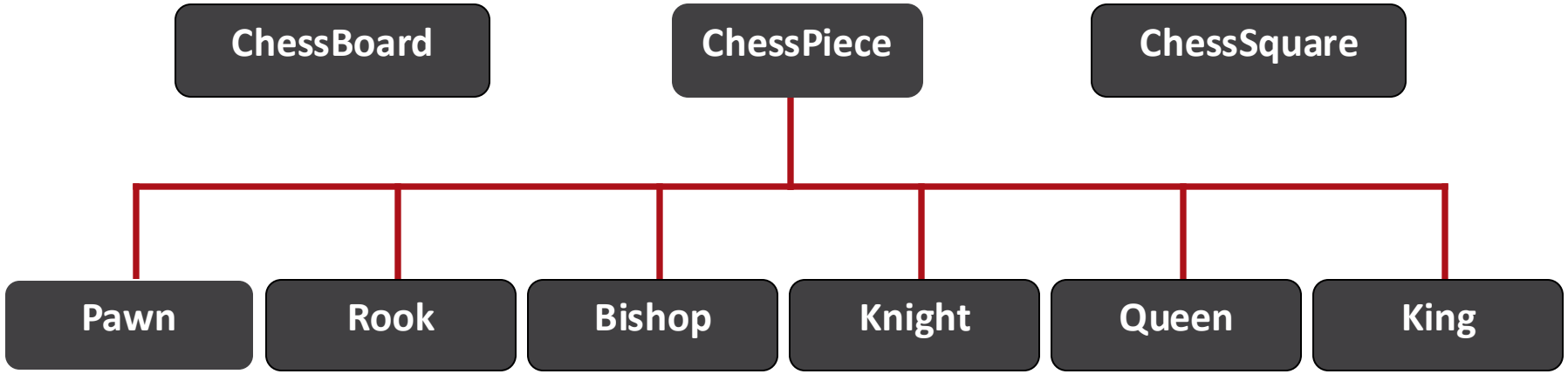
Inheritance case study

Lancaster University

Wizard Chess: A concrete example

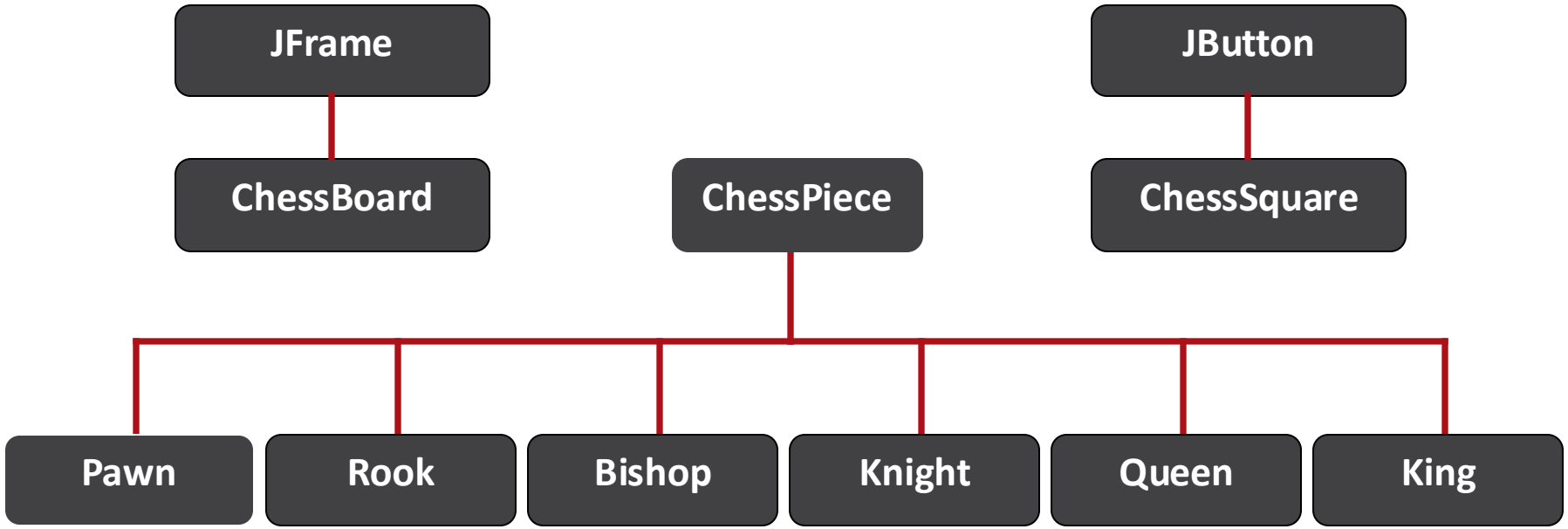
- Visually model the correct behaviour of chess pieces in Swing
- Prevent illegal moves on the board.
- **What classes can we identify?**
- **What names are sensible?**
- **Is there an inheritance relationship between any of these classes?**

Wizard Chess: Inheritance Hierarchy



Are ChessBoard and ChessSquare written from scratch or also specializations?

Wizard Chess: Inheritance Hierarchy…



Are ChessBoard and ChessSquare written from scratch or also specializations?

Wizard Chess: Methods

Lancaster University

Which class do you think these methods should defined in and why?

Attributes:

- The chess square a piece is residing on. — **ChessPiece**
- A two dimensional array of 64 chess squares. — **ChessBoard**
- Image representing a piece. — **ChessPiece**
- The [x,y] location of a chess square — **ChessSquare**

Methods:

- ActionListener() — **ChessBoard**
- moveTo() — **ChessPiece**
- canMoveTo() — **All ChessPiece subclasses!**

Summary

Lancaster University

Today we learned about:

- Principles of inheritance
- How it can help to allow us to specialize existing code to our needs

- Next time we'll learn how to specialize the behaviour of our subclasses.
- Next week's lab will include a little inheritance in action using Swing!