Inheritance Principles
- Interface Inheritance
- Implementation Inheritance
- Liskov Substitution Principle and Shapes

C++ Object Heap Layout
- Basics
- Single-Inheritance
- Virtual Methods

C++ Multiple Parents Heap Layout
- Multiple-Inheritance
- Common Parents

Discussion & Learning Outcomes
“Wouldn’t it be nice to inherit from several parents?”

**Interface vs. Implementation inheritance**

The classic motivation for inheritance is implementation inheritance
- *Code reuse*
- Child specializes parents, replacing particular methods with custom ones
- Parent acts as library of common behaviours
- Implemented in languages like C++ or Lisp

Code sharing in interface inheritance inverts this relation
- **Behaviour contract**
- Child provides methods, with signatures predetermined by the parent
- Parent acts as generic code frame with room for customization
- Implemented in languages like Java or C#

**Interface Inheritance**

**Implementation inheritance**

[Diagram showing relationships between classes and methods, with arrows indicating inheritance and method calls]
**The Liskov Substitution Principle**

Functions that use pointers or references to base classes must be able to use objects of derived classes without knowing it.

```java
class Rectangle {
    void.setWidth (int w){ this.w=w; }
    void.setHeight(int h){ this.h=h; }
    void getWidth () { return w; }
    void getHeight() { return h; }
}

class Square extends Rectangle {
    void.setWidth (int w){ this.w=h=w; }
    void.setHeight(int h){ this.h=h; w=h; }
}

Rectangle r = new Square(2);
r.setWidth(3);
r.setHeight(4);
assert r.getHeight()==12;
```