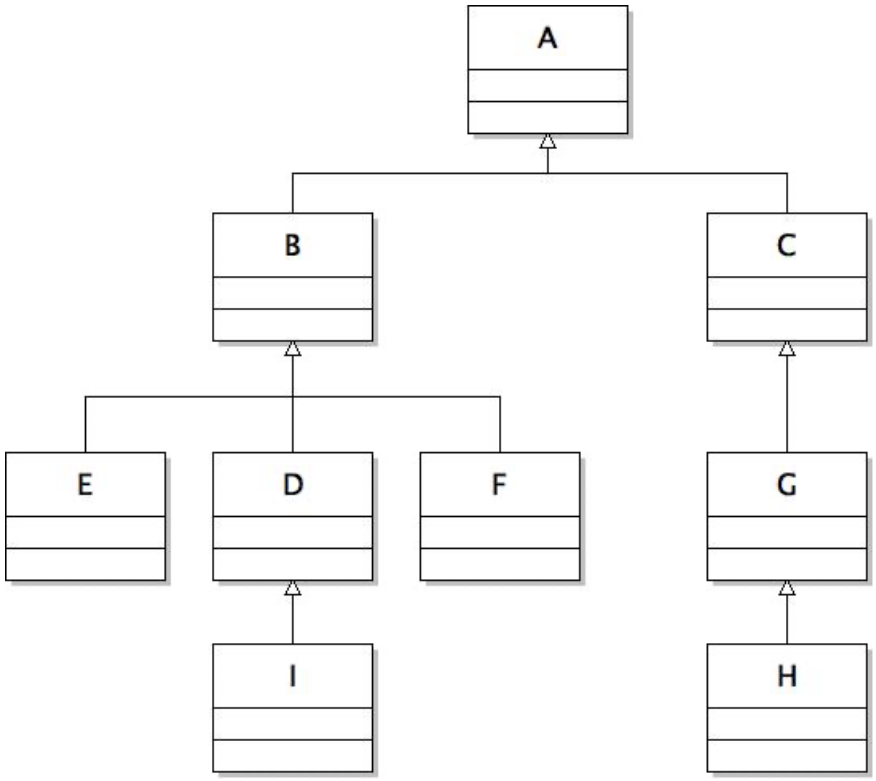
Object Oriented Software Engineering
Tutorial 2

Liskov's Substitution Principle(LSP)

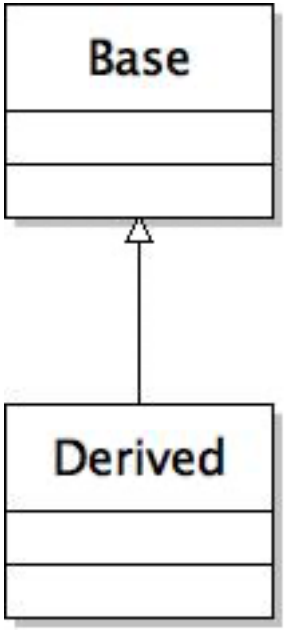
Graham McDonald

Liskov's Substitution Principle(LSP)



- When creating class hierarchies, ensure that the new derived classes just extend without replacing the functionality of old classes.

Liskov's Substitution Principle(LSP)



- Likov's Substitution Principle states that if a program module is using a Base class, then the reference to the Base class can be replaced with a Derived class without affecting the functionality of the program module.

How to Achieve LSP

- **Demand no more:** The subclass should accept any arguments that the superclass would accept.

- **Promise no less:** Any assumption that is valid when the superclass is used must be valid when the subclass is used.

LSP – Design by Contract

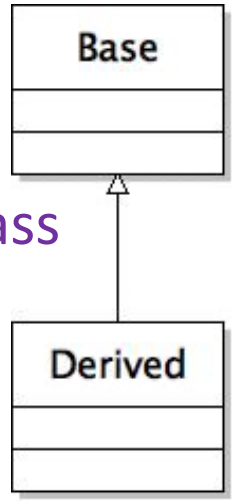
The contract of a method:

- **Precondition:** A property that must be true before calling the method. If the precondition fails, the results are undefined
- **Postcondition:** A property that is guaranteed to be true after calling the method. If the postcondition fails, it shall not return.
- **Invariants:** A property that is guaranteed not to change after executing the the method

LSP – Design by Contract

The LSP in terms of contract

- A derived class is substitutable for its base class if
  1. Its preconditions are not stronger than the base class method
  2. Its postconditions are no weaker than the base class method.
  3. The invariants remain the same



LSP Rules

Rule 1:

- When you override a method in a base class, the **precondition** of the overriding method should be *weaker* than the **precondition** of the overriden method.

LSP Rules

Rule 2:

- When you override a method in a base class, the **postcondition** of the overriding method should be *stronger* than the **postcondition** of the overriden method

Example of LSP Violation

```
public class Rectangle {
    private double width, height;

    public double area() {
        return width * height;
    }

    public void setWidth(double w) {
        width = w;
    }

    public void setHeight(double h) {
        height = h;
    }

    public double getWidth() {
        return width;
    }

    public double getHeight() {
        return height;
    }
}
```

```
public class Square extends Rectangle {
    public void setWidth(double w) {
        super.setWidth(w);
        super.setHeight(w);
    }

    public void setHeight(double h) {
        super.setWidth(h);
        super.setHeight(h);
    }
}
```

Can you see what is wrong with the Square class?

What is wrong with this class?

```
public class Person {
  private String firstName, lastName;

  public boolean equals(Person other) {
    return this.firstName.equals(other.firstName)
        && this.lastName.equals(other.lastName);
  }
}
```

Does the code pass or fail LSP rules?