# Imperative Programming Part 3
# Problem Sheet 1

Peter Jeavons[*]

Trinity Term 2019

1. Assume that you are writing an application that needs to work with the following types of shapes:

   - rectangle (which is defined by its width and height), and
   - square (which is defined by the length of its side).
   - ellipse (which is defined by its semi-major and semi-minor axes),
   - circle (which is defined by its radius),

   Design a set of Scala classes that represent these shapes. Your design should support the following operations:

   - For each shape, it should be possible to retrieve and change its size.
   - Given a set of shapes, it should be possible to identify all squares.
   - Given a set of shapes, it should be possible to identify all circles.

   Note that you should consider carefully whether the obvious design that simply introduces a separate class corresponding to each of the four shapes is in fact the best design choice. Consider alternative designs to this and explain why these are better or worse.

2. The following class is intended to model a rectangle shape, defined by its width and height, which are both mutable.

```scala
class Rectangle(var width: Int, var height: Int) {
  var area = width * height
  ...
}
```

   Comment on any design errors you see in this class, and explain how you would fix them.

3. The class `Slab` defined below should have an invariant that

   `_area = _dimension.width * _dimension.height`.

   Show how the definition given below allows this invariant to be invalidated, and suggest how to improve the design of the class `Slab` so that this invariant is always maintained.

```scala
class Rectangle(var width: Int, var height: Int)

class Slab(private val _dimension: Rectangle) {
  private val _area = _dimension.width * _dimension.height

  def dimension = _dimension
  ...
}
```

[*]Based on earlier material by Mike Spivey, Joe Pitt-Francis and Milos Nikolic.

4. Consider the following classes:

```
class Triangle
class OpaqueTriangle extends Triangle
class Renderer {
  def accept(a: Triangle) = println("Accepted for rendering.")
}
class RayTracingRenderer extends Renderer {
  def accept(a: OpaqueTriangle) = println("Accepted for ray-trace rendering.")
}
```

What will the following code print? Explain your answer.

```
val a: OpaqueTriangle = new OpaqueTriangle
val r1: Renderer = new RayTracingRenderer
r1.accept(a)
val r2: RayTracingRenderer = new RayTracingRenderer
r2.accept(a)
```

Now modify the definition of `RayTracingRenderer` in a way that changes the output produced by `r1.accept(a)`.

5. Consider the following `Ellipse` class with mutable semi-axes `a` and `b`

```
class Ellipse(private var _a: Int, private var _b: Int) {
    def a = _a
    def a_=(a: Int) = {_a = a}

    def b = _b
    def b_=(b: Int) = {_b = b}
}
```

Define a subclass `LoggedEllipse` of this `Ellipse` class that keeps track of how many times the area of the ellipse has been increased from its previous value during the lifetime of the ellipse, and provides a method `getIncreases` to access this number.

Now add a method to the `Ellipse` class that exchanges the two axes of the ellipse.

To illustrate the fragile base class problem, show how this method could be implemented without the need to override it in `LoggedEllipse`, and then show another implementation that does need to be overridden to maintain correctness.

6. The editor case study has a class `Text` for storing and modifying a string of characters, and a subclass `PlaneText` that keeps track of the division of the text into lines.

   (a) Instead of making `PlaneText` a subclass of `Text`, give an outline of how the editor could be reorganised so that `PlaneText` has a `Text` as an instance variable.

   (b) It is proposed to extend the editor for use on a network, and the proposed extension will include a method with heading

   ```
   def transmit(text: Text)
   ```

   With the existing design, instances of both `Text` and `PlaneText` can be passed to this method, because of the subtype relation between them. Show how to maintain this polymorphism in the new design by introducing an appropriate trait.

   (c) Does this proposed change to the relationship between `Text` and `PlaneText` make the fragile base class problem better or worse? What other problems does it cause or cure?

7. [**Programming**] An inexperienced programmer has implemented an equals method for the `Rectangle` class in Question 2 as follows

```
class Rectangle(var width:Int, var height:Int) {
...
def ==(other: Rectangle): Boolean =
   this.width == other.width && this.height == other.height
}
```

They claim that this passes all their test cases:

```
// Tests for equality method

r1 = new Rectangle(20,30)
r2 = new Rectangle(20,30)
r3 = new Rectangle(50,60)

println(r1 == r2, r2 == r3) // Correctly returns "true, false"
```

Devise a suitable additional set of tests to show that this implementation of equality does not lead to desirable behaviour.

Improve the definition of the equals method so that it has all the required properties (see Chapter 30 of *Programming in Scala*). In particular, ensure it defines an equivalence relation, and ensure it allows `Rectangles` to be used with the scala collections library. To ensure this, you may want to make other changes to the `Rectangle` class - if so, explain why these are needed.

Test your new `Rectangle` class carefully. Try your tests on someone else's implementation to see if you can break it.