**Module Design Patterns**
Prof. Dr. Dominik Gruntz
Prof. Dr. Wolfgang Weck

# Worksheet Immutability: Solution

An immutable object must encapsulate all its *observable* attributes in such a way that they are set in the constructor only and only read afterwards.

**Rules:**

1. The methods declared in the class must not change the (visible) state of the object.
2. All non-private fields have to be declared final and have to be of primitive or immutable type.
3. No setter methods.
4. References returned by getter methods have to be of primitive or immutable type or have to be (deeply) cloned
5. Class has to be declared final (not extensible) because immutability cannot be forced onto a subclass, i.e. subclasses could add mutable fields which would violate the Liskov substitution principle.
6. References passed to the object with a constructor have to be of primitive or immutable type or have to be (deeply) cloned
7. Do not inherit from a base class which has non-static mutable fields
8. The `this` reference is not allowed to escape during construction

**Implementation:**

```
public final class ImmutableLine {
      private final Point start, end;
      public Line(Point start, Point end) {
            this.start = (Point) start.clone();
            this.end = (Point) end.clone();
      }
      public Point getStartPoint () {
            return (Point) start.clone();
      }
      public Point getEndPoint () {
            return (Point) end.clone();
      }
      public ImmutableLine withStartPoint(Point start) {
            return this.start.equals(start)) ? this
                  : return new ImmutableLine(start, end);
      }
      public ImmutableLine withEndPoint(Point end) {
            return this.end.equals(end)) ? this
                  : return new ImmutableLine(start, end);
      }
      @Override
      public Object clone() {     // violates the condition that x.clone() != x
            return this;          // which is specified in the specification of
      }                           // Object.clone  (as a SHOULD requirement).
      // As an alternative method clone() could also actually create a copy,
      // or the method could be omitted altogether (as in class java.lang.String).

      @Override
      public String toString() {
            return String.format("[Line: start=%s, end=%s]", start, end);
      }
}
```

**Test:**

```
      Point p1 = new Point(1, 2);
      Point p2 = new Point(3, 4);
      Line l1 = new Line(p1, p2);
      System.out.println(l1); p1.x = 5;
      System.out.println(l1); // should once again print the same!!
```