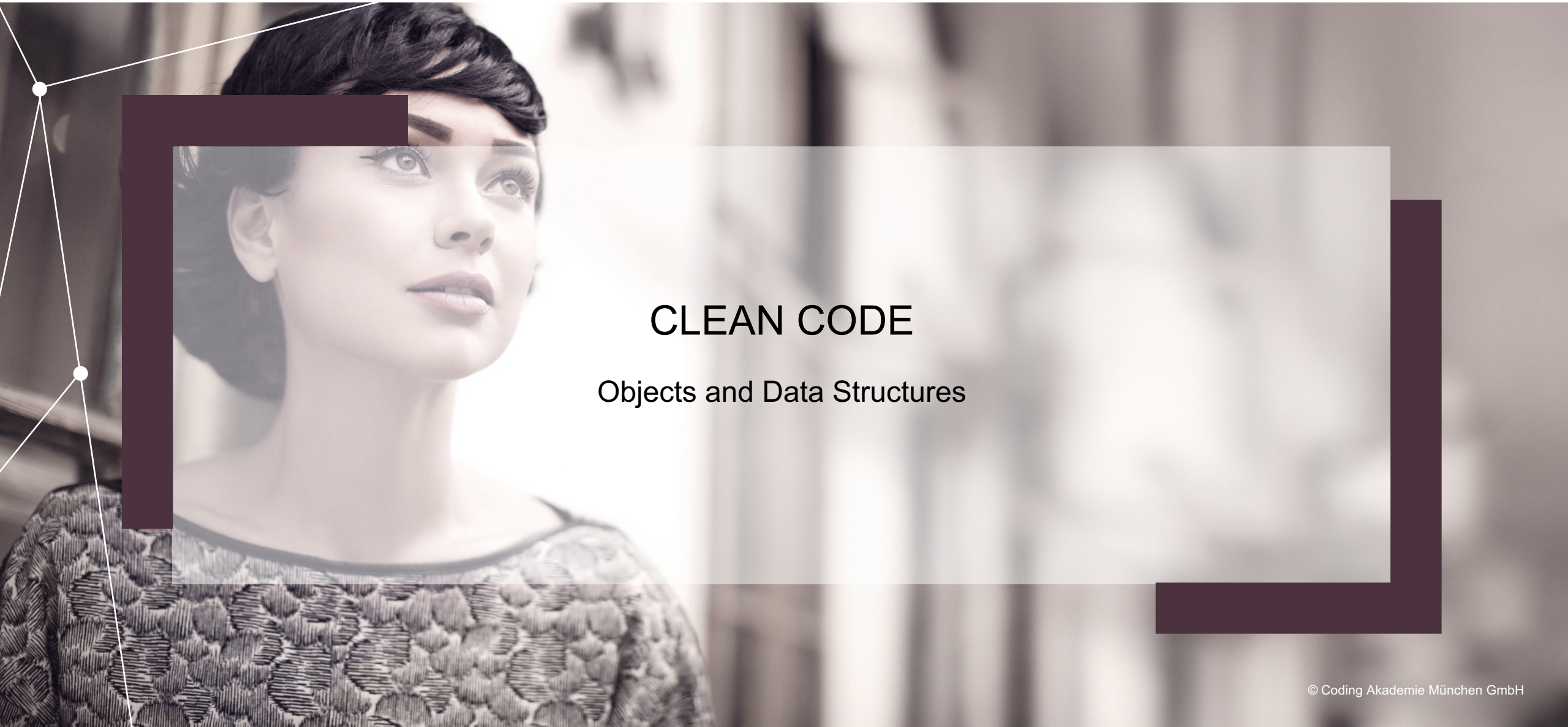
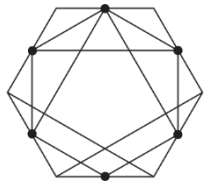


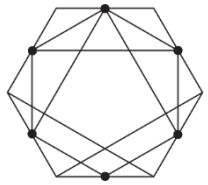
CLEAN CODE

Objects and Data Structures

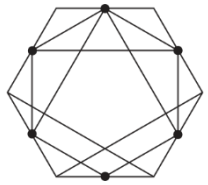


OBJECTS VS. DATA STRUCTURES

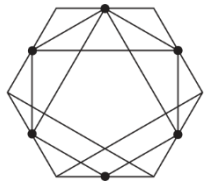
Example: Point in abstract form (object) or data structure (concrete)



```
struct ConcretePoint {
    double x;
    double y;
};
```



```
class AbstractPoint {
public:
    static AbstractPoint from_cartesian(double x, double y);
    static AbstractPoint from_polar(double r, double theta);

    double get_x() const;
    double get_y() const;

};
```

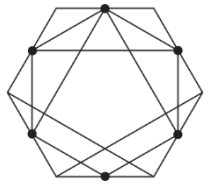


THE EXPRESSION PROBLEM

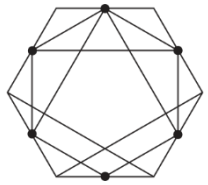
Is it possible to define a type so that we can add

- new subtypes
- new operations

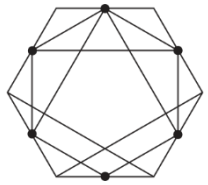
without changing the existing implementation?



```
struct ConcretePoint {
    double x;
    double y;
};

double point_r(ConcretePoint point);
double point_theta(ConcretePoint point);
ConcretePoint add(ConcretePoint lhs, ConcretePoint rhs);
void print(ConcretePoint p);
```



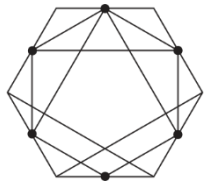
```
class AbstractPoint {
public:
    static AbstractPoint from_cartesian(double x, double y);
    static AbstractPoint from_polar(double r, double theta);

    double get_x() const;
    double get_y() const;
    double get_r() const;
    double get_theta() const;

    AbstractPoint operator+(AbstractPoint rhs);
};

std::ostream& operator<<(std::ostream& s, const AbstractPoint& point);
```



THE EXPRESSION PROBLEM

- For data types we can add new operations but not new subtypes
- For objects we can add new subtypes but not new objects

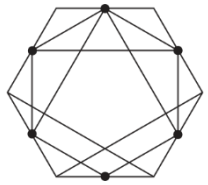
(The missing feature for objects are open methods.)



MINI WORKSHOP: POINTS

Implement classes AbstractPoint and ConcretePoint with attributes/functions for

- X and y coordinates
- Addition of points
- Writing points to a stream (or printing them on the screen)



MINI WORKSHOP: SHAPES

Continue the previous implementation by adding classes that can represent

Squares (described by bottom left corner and length of the sides)
Rectangles (described by bottom left corner, width and height)
Circles (described by center and radius)

both as concrete data type and as classes.

Inherit from a common superclass for the class-based implementation (and possibly the data type).

Add the following functions to both implementations: (1) compute the area, (2) move the shape.

Add triangles and polygons with arbitrary numbers of corners to both implementations and implement the functionality. (You can stub out the implementations for the new data types.)