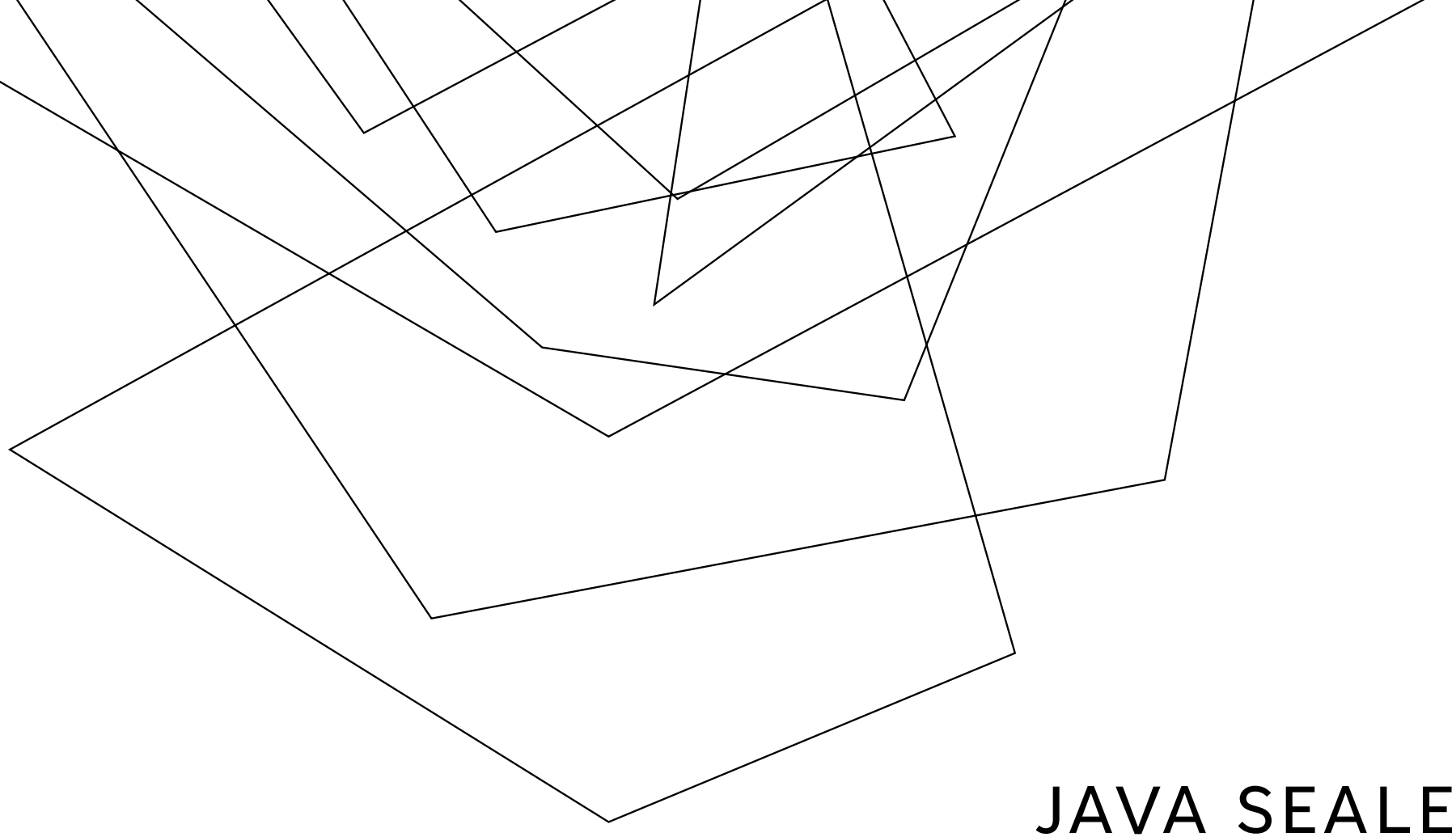


JAVA SEALED CLASSES

Efthimios Alepis

INTRODUCTION

- Sealed classes and interfaces restrict which other classes or interfaces may extend or implement them
- By sealing a class, you can specify which classes are permitted to extend it and prevent any other arbitrary class from doing so

GOALS

- Allow the author of a class or interface to control which code is responsible for implementing it.
- Provide a more declarative way than access modifiers to restrict the use of a superclass.
- Support future directions in pattern matching by providing a foundation for the exhaustive analysis of patterns.

HOW TO USE THEM

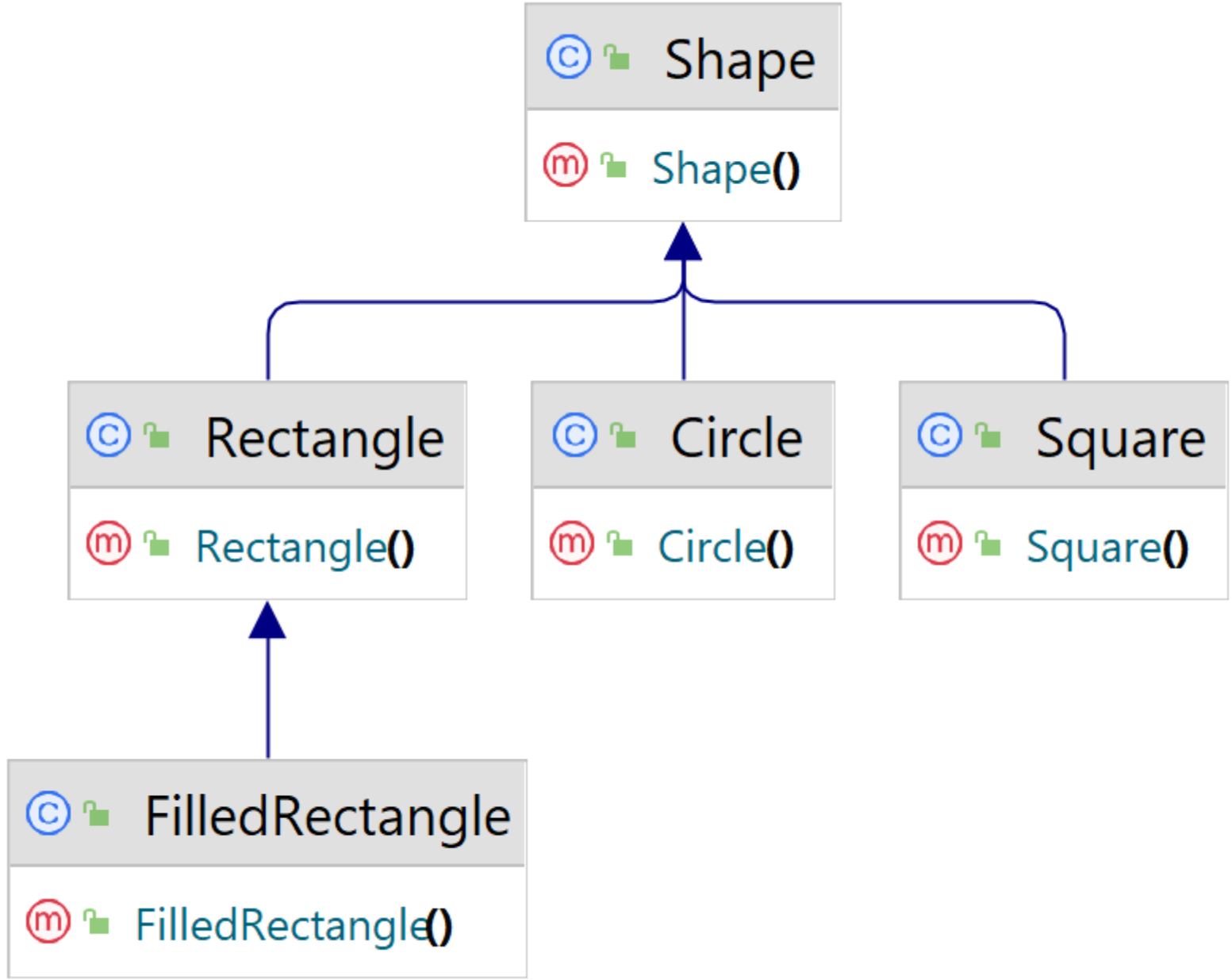
- To seal a class, add the sealed modifier to its declaration. Then, after any extends and implements clauses, add the permits clause. This clause specifies the classes that may extend the sealed class.
- Alternatively, you can define permitted subclasses in the same file as the sealed class. If you do so, then you can omit the permits clause.

CONSTRAINTS ON PERMITTED SUBCLASSES

- Permitted subclasses have the following constraints:
  - They must be accessible by the sealed class at compile time.
  - They must directly extend the sealed class.
  - They must have exactly one of the following modifiers to describe how it continues the sealing initiated by its superclass:
    - final: Cannot be extended further
    - sealed: Can only be extended by its permitted subclasses
    - non-sealed: Can be extended by unknown subclasses; a sealed class cannot prevent its permitted subclasses from doing this
  - They must be in the same module as the sealed class (if the sealed class is in a named module) or in the same package (if the sealed class is in the unnamed module).

DECLARING SEALED INTERFACES

- Like sealed classes, to seal an interface, add the sealed modifier to its declaration.
- Then, after any extends clause, add the permits clause, which specifies the classes that can implement the sealed interface and the interfaces that can extend the sealed interface.


Shape
Shape()
Rectangle
Rectangle()
Circle
Circle()
Square
Square()
FilledRectangle
FilledRectangle()

EXAMPLE

```
public sealed class Shape permits Circle, Square, Rectangle{

}
```

```
public final class Circle extends Shape {
    public float radius;
}
```

```
public non-sealed class Square extends Shape {
    public double side;
}
```

```
public sealed class Rectangle extends Shape permits
FilledRectangle {
    public double length, width;
}
```

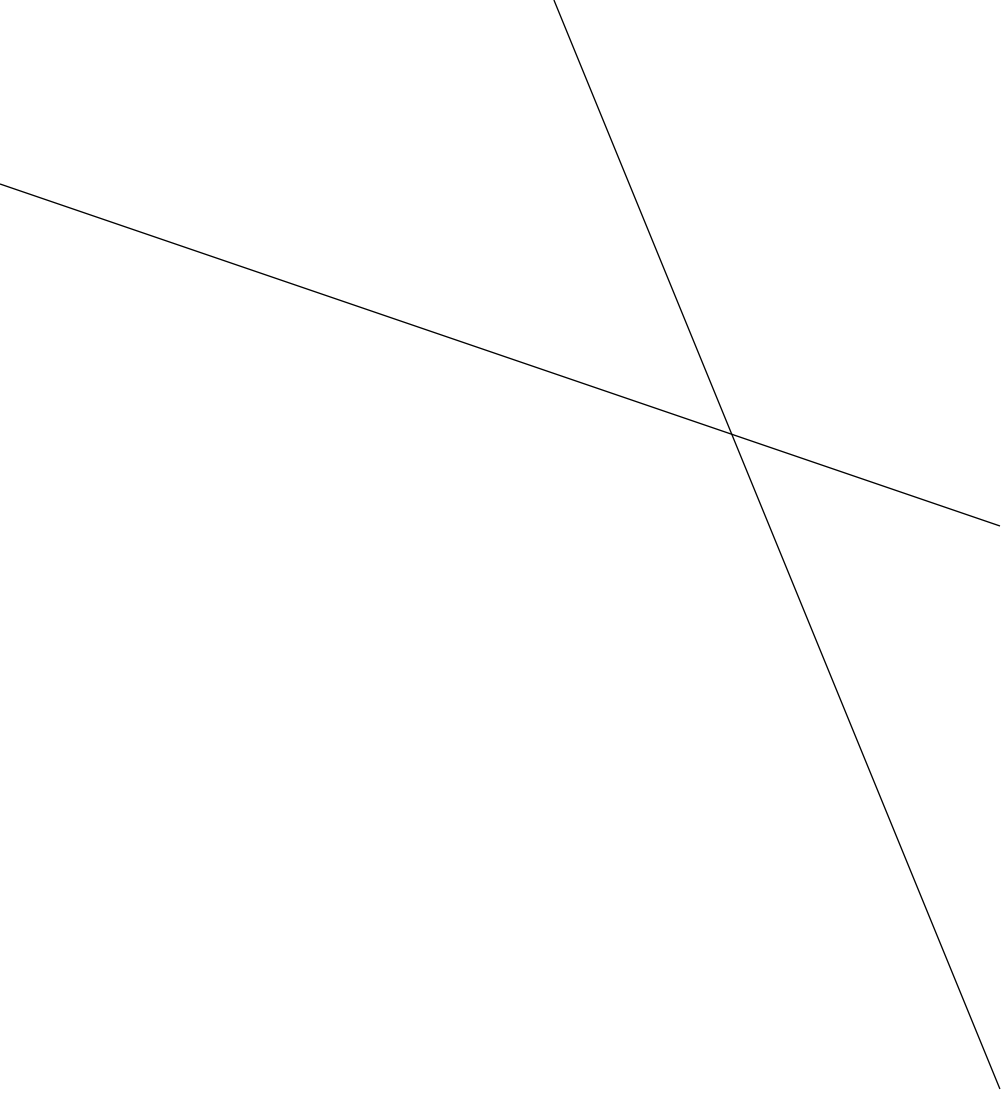
```
public final class FilledRectangle extends Rectangle {
    public int red, green, blue;
}
```

REFLECTION

- Sealed classes are also supported by the reflection API, where two public methods have been added to the java.lang.Class:
    - The isSealed method returns true if the given class or interface is sealed.
    - Method getPermittedSubclasses returns an array of objects representing all the permitted subclasses.

SUMMARY

The permits clause allows a sealed class, such as the Shape class shown earlier, to be accessible-for-invocation by code in any module, but accessible-for-implementation by code in only the same module as the sealed class (or same package if in the unnamed module). This makes the type system more expressive than the access-control system. With access control alone, if Shape is accessible-for-invocation by code in any module (because its package is exported), then Shape is also accessible-for-implementation in any module; and if Shape is not accessible-for-implementation in any other module, then Shape is also not accessible-for-invocation in any other module.



FURTHER READING

https://openjdk.org/jeps/409

https://docs.oracle.com/en/java/javase/20/language/sealed-classes-and-interfaces.html