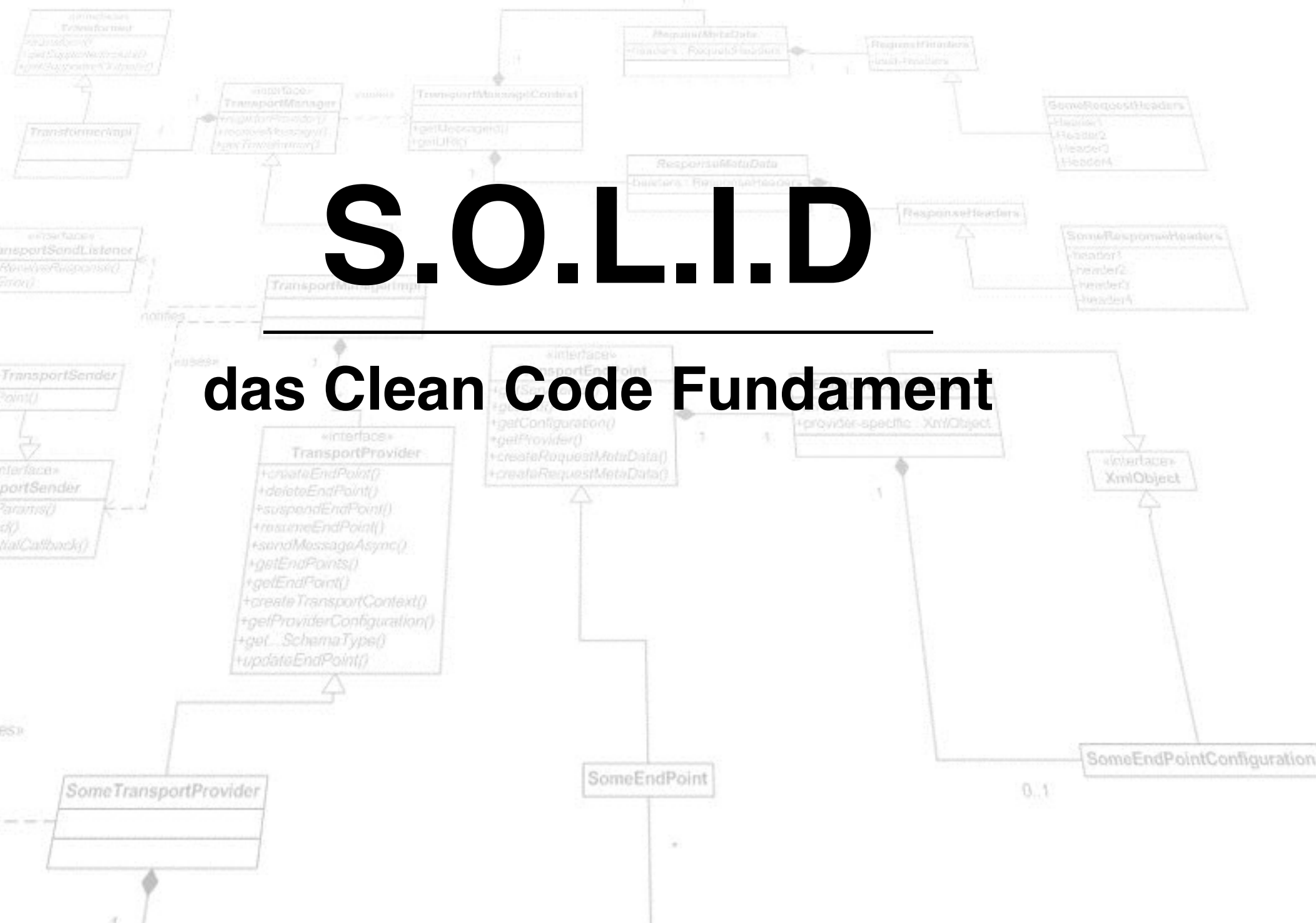


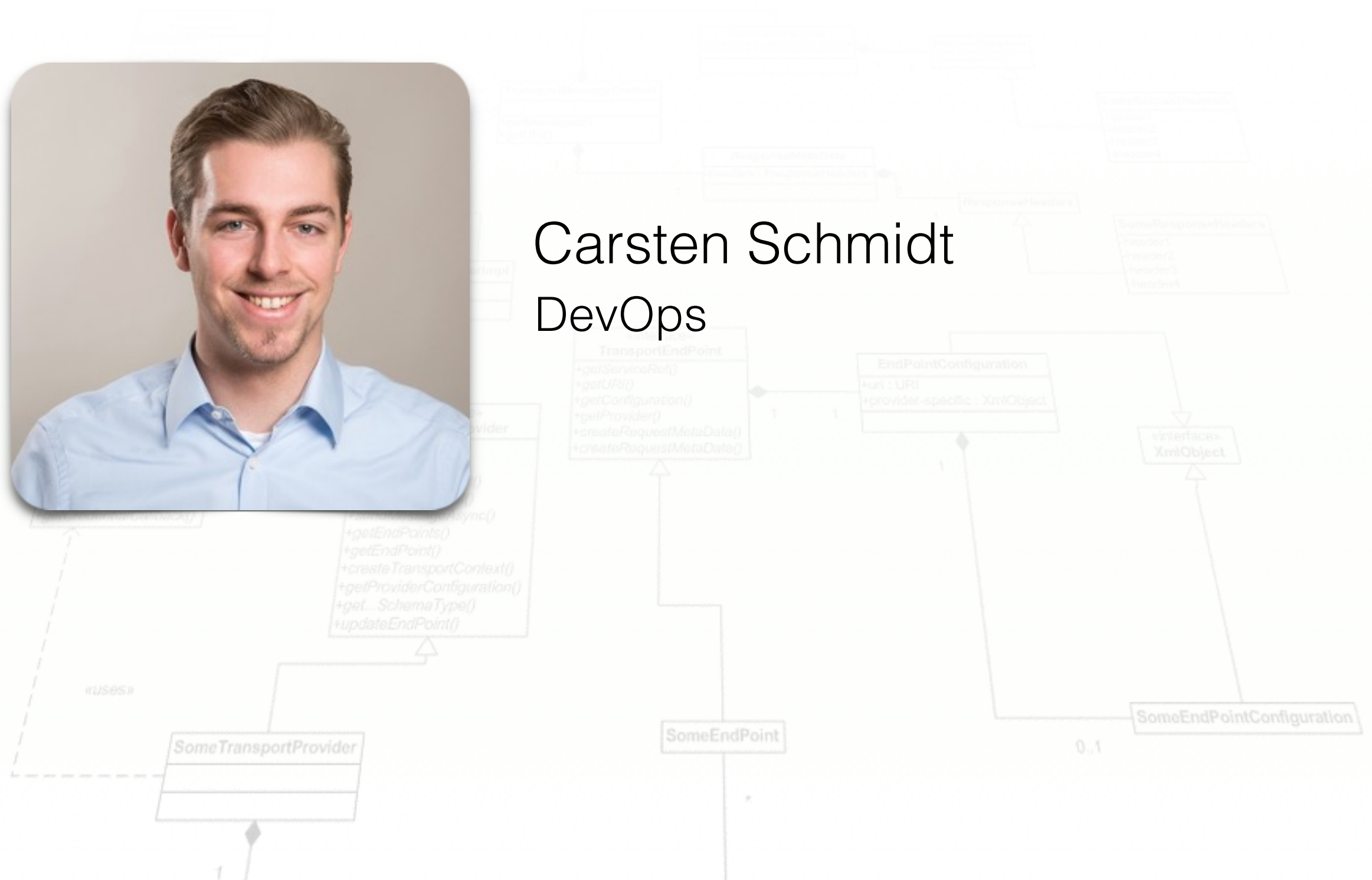
das Clean Code Fundament





Carsten Schmidt

DevOps



S.O.L.I.D

- Single Responsibility Principle
- Open-Closed Principle
- Liskov Substitution Principle
- Interface Segregation Principle
- Dependency Inversion Principle



Robert C. Martin

Uncle Bob

S.O.L.I.D

3. Grad Single Responsibility Principle

5. Grad Open-Closed Principle

4. Grad Liskov Substitution Principle

4. Grad Interface Segregation Principle

4. Grad Dependency Inversion Principle



Robert C. Martin

Uncle Bob

S.O.L.I.D

Vorteile

- Modularisierung
- Wiederverwendbarkeit
- Erweiterbarkeit
- Einfach zu lesen
- Wartbar
- Einfach zu verändern
- Saubere Abstraktionen
- Einfach zu verstehen

S.O.L.I.D

Nachteile

- Mehr Ebenen
- Ist vielleicht langsamer
- Mehr Quelltext
- Schreibaufwand
- "Über-Abstraktionen"
- Fachwissen benötigt

S.O.L.I.D

Zielsetzung

Qualitätssteigerung

Aufwandssenkung

S.O.L.I.D

Zielsetzung

Qualitätssteigerung



Weniger Fehler

Aufwandssenkung



Fehlerbehebung

S.O.L.I.D

Zielsetzung

Qualitätssteigerung



Weniger Fehler



Höhere Flexibilität

Aufwandssenkung



Fehlerbehebung



Softwareänderungen

S.T.U.P.I.D

- Singleton
- Tight Coupling
- Untestability
- Premature Optimization
- Indescriptive Naming
- Duplication

Singleton

Singleton

```
public final class Database {  
  
    private static Database instance;  
  
    private Database() {  
        /* load connection params */  
    }  
  
    public static Database instance() {  
        if(null == instance) {  
            instance = new Database();  
        }  
  
        return instance;  
    }  
}
```

Singleton

```
public final class Database {  
  
    private static Database instance;  
  
    private Database() {  
        /* load connection params */  
    }  
  
    public static Database instance() {  
        if(null == instance) {  
            instance = new Database();  
        }  
  
        return instance;  
    }  
}
```

Singleton

```
public final class Database {  
    private static Database instance;
```

```
public class ProductManager {  
    public String getProductName(long productId) {  
        Database db = Database.instance();  
        return (String)db.from('products').select('pname');  
    }  
}
```

Singleton

```
public final class Database {  
    private static Database instance;
```

```
public class ProductManager {  
    public String getProductname(long productId) {  
        Database db = Database.instance();  
        return (String)db.from('products').select('pname');  
    }  
}
```

```
        return instance;  
    }  
}
```


Singleton

```
public final class Database {  
    private static Database instance;
```

```
public class ProductManager {  
    public String getProductName(long productId) {  
        Database db = Database.instance();  
        return (String)db.from('products').select('pname');  
    }  
}
```

Tight Coupling

Tight Coupling



Tight Coupling

```
public class House {  
  
    private Window window;  
    private Door door;  
  
    public House() {  
        window = new Window();  
        door = new Door();  
    }  
}
```

Tight Coupling

```
public class House {  
  
    private Window window;  
    private Door door;  
  
    public House() {  
        window = new Window();  
        door = new Door();  
    }  
}
```

Kopplung!



Tight Coupling

```
public class House {  
  
    private Window window;  
    private Door door;  
  
    public House(Window window, Door door) {  
        this.window = window;  
        this.door = door;  
    }  
}
```

Tight Coupling

```
public class House {  
  
    private Window window;  
    private Door door;  
  
    public House(Window window, Door door) {  
        this.window = window;  
        this.door = door;  
    }  
}
```

Dynamisch!



Untestability

Premature Optimization



Indescriptive Naming

```
if(i == j) {  
    itKrv = itKrv * 0.2;  
}  
  
String tanga = "rtval-${pc}";
```

Duplication

Duplication

Duplication

Duplication

Duplication

Duplication

Duplication

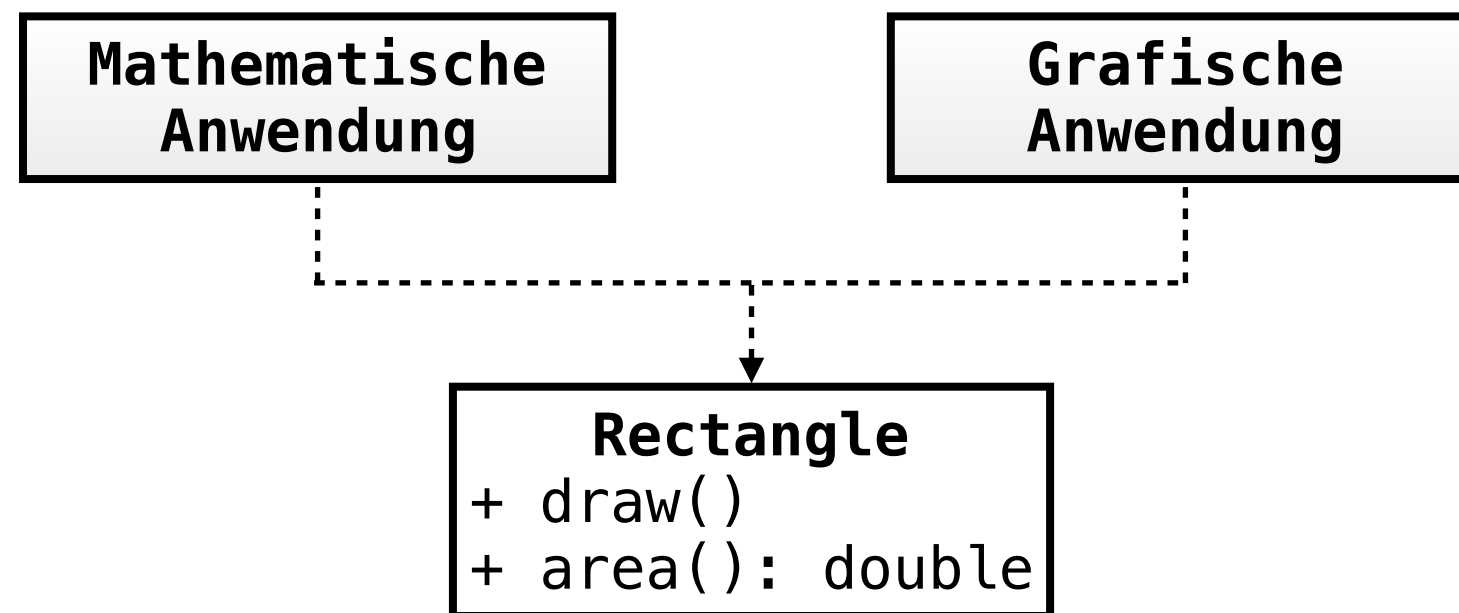
Duplication

Single Responsibility Principle

Es sollte nie mehr als einen Grund geben, eine Klasse zu ändern

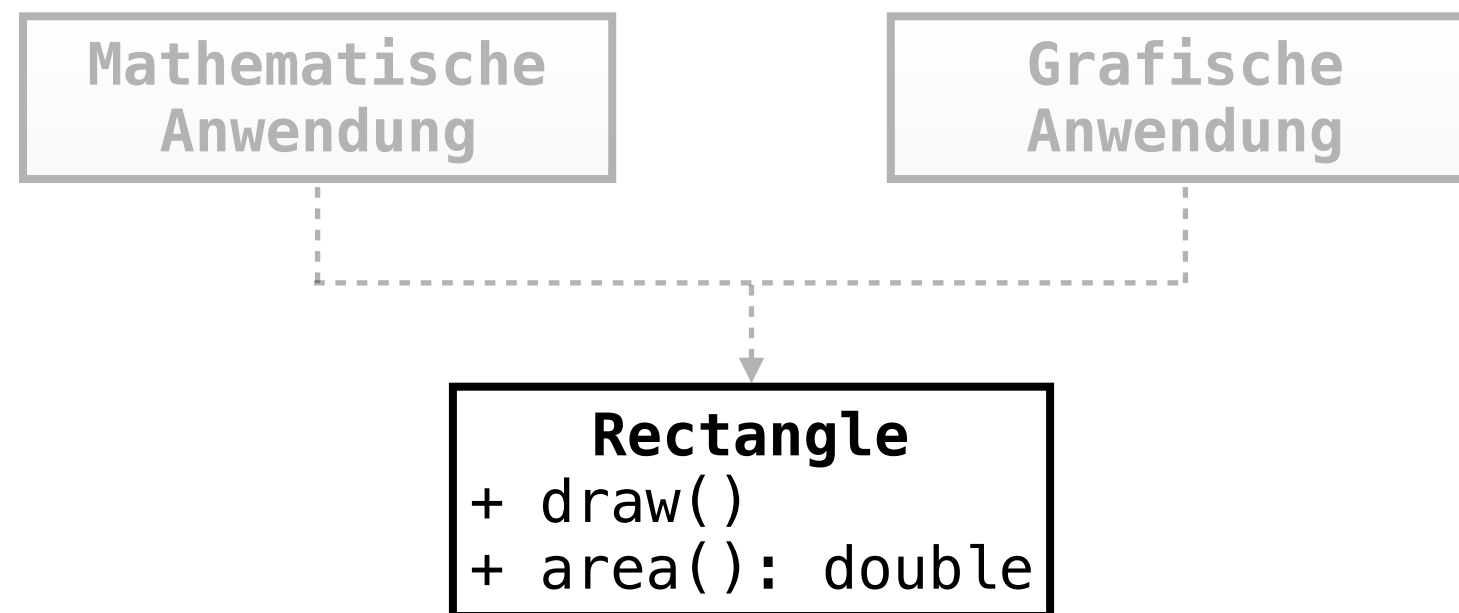
Single Responsibility Principle

Es sollte nie mehr als einen Grund geben, eine Klasse zu ändern



Single Responsibility Principle

Es sollte nie mehr als einen Grund geben, eine Klasse zu ändern

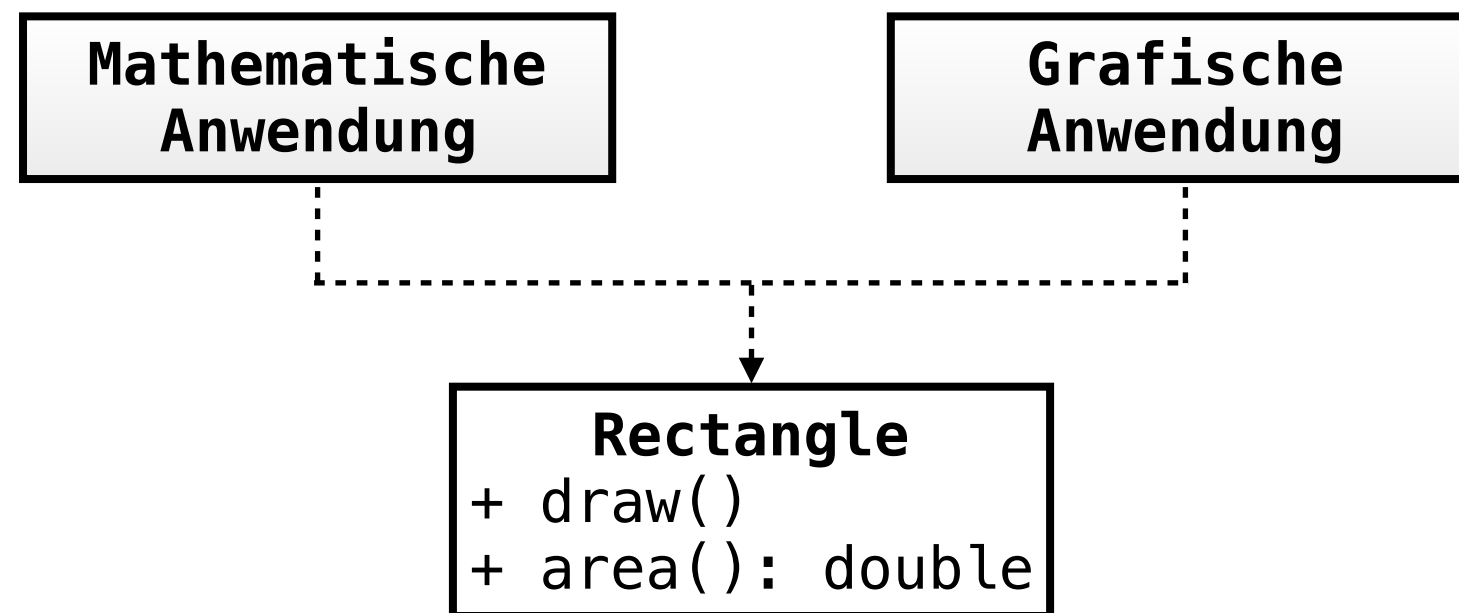


Aufgaben

- Mathematisches Modell
- Zeichenlogik

Single Responsibility Principle

Es sollte nie mehr als einen Grund geben, eine Klasse zu ändern

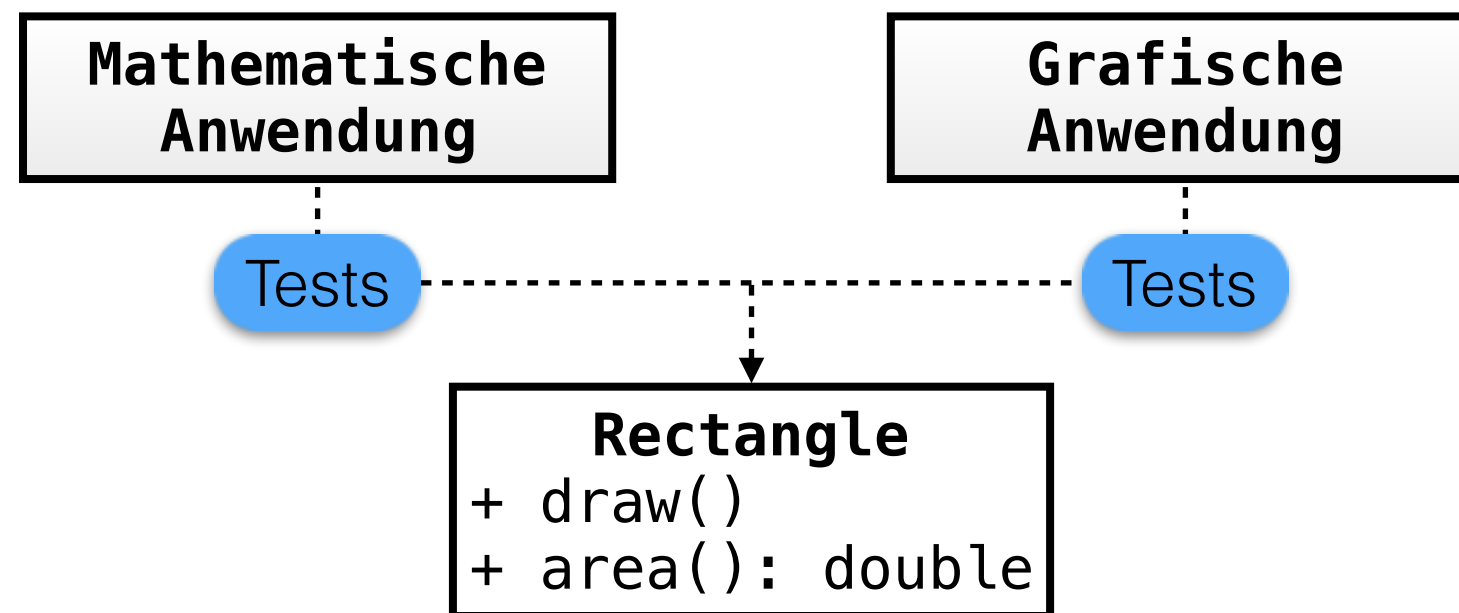


Aufgaben

- Mathematisches Modell
- Zeichenlogik

Single Responsibility Principle

Es sollte nie mehr als einen Grund geben, eine Klasse zu ändern

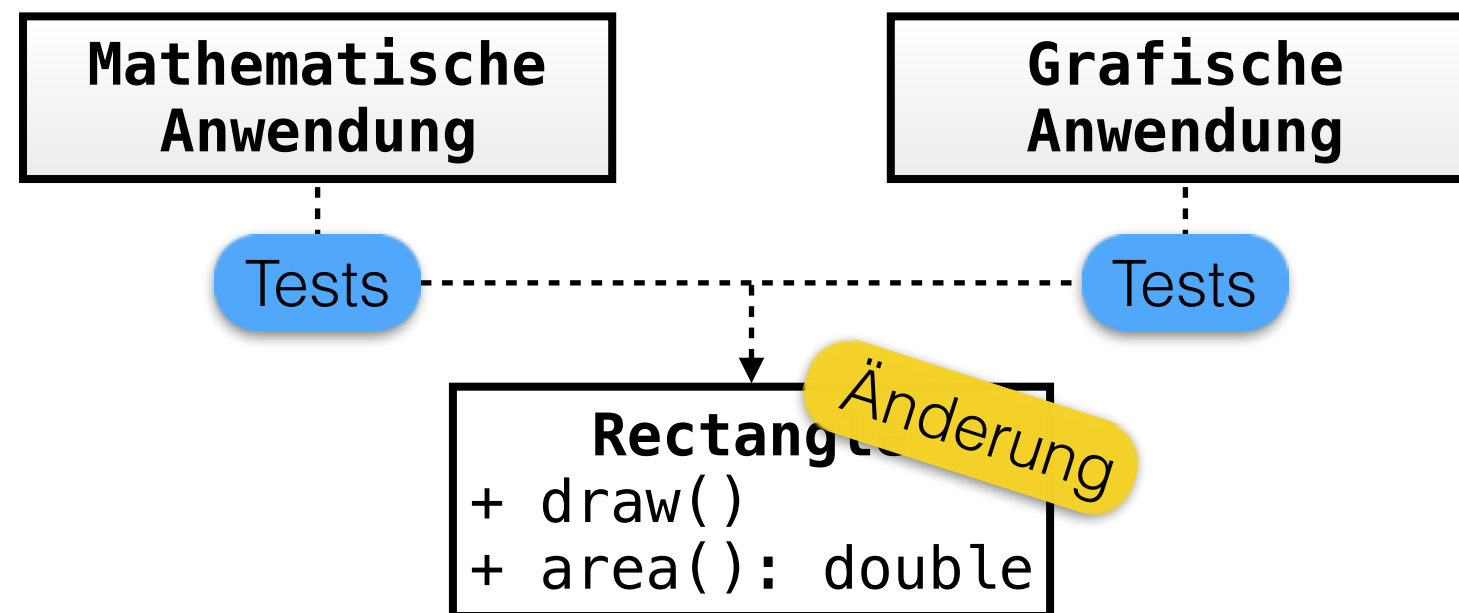


Aufgaben

- Mathematisches Modell
- Zeichenlogik

Single Responsibility Principle

Es sollte nie mehr als einen Grund geben, eine Klasse zu ändern



Aufgaben

- Mathematisches Modell
- Zeichenlogik

Single Responsibility Principle

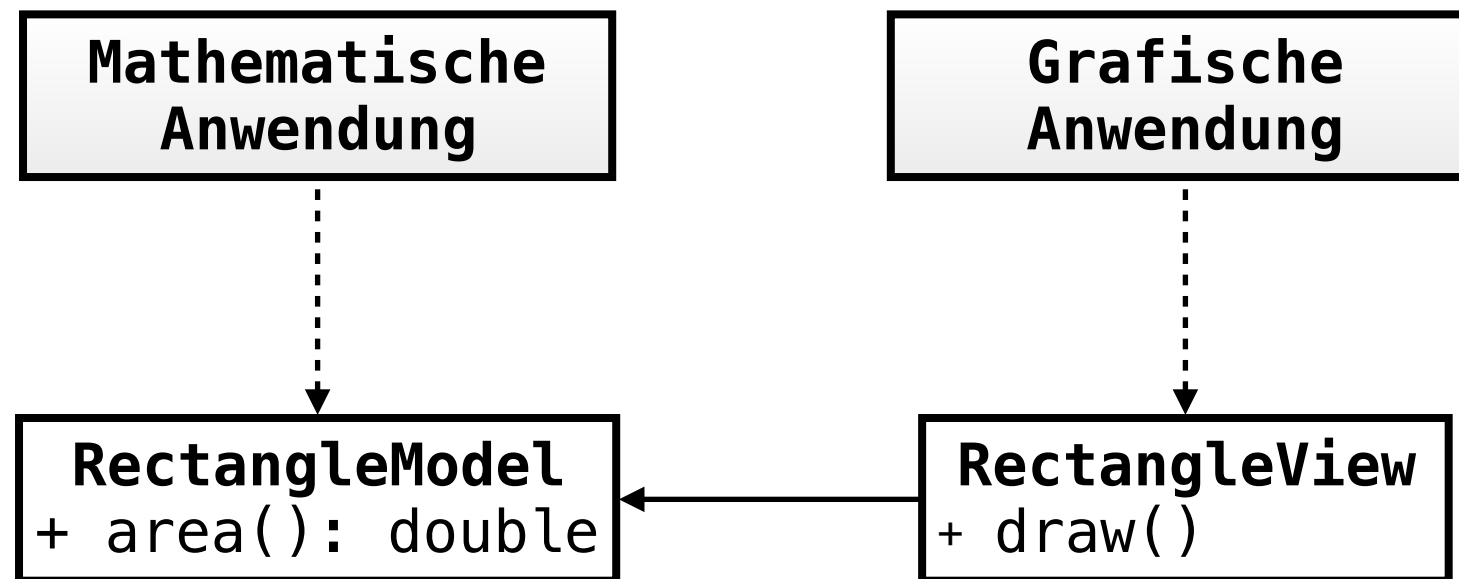
Es sollte nie mehr als einen Grund geben, eine Klasse zu ändern

**Mathematische
Anwendung**

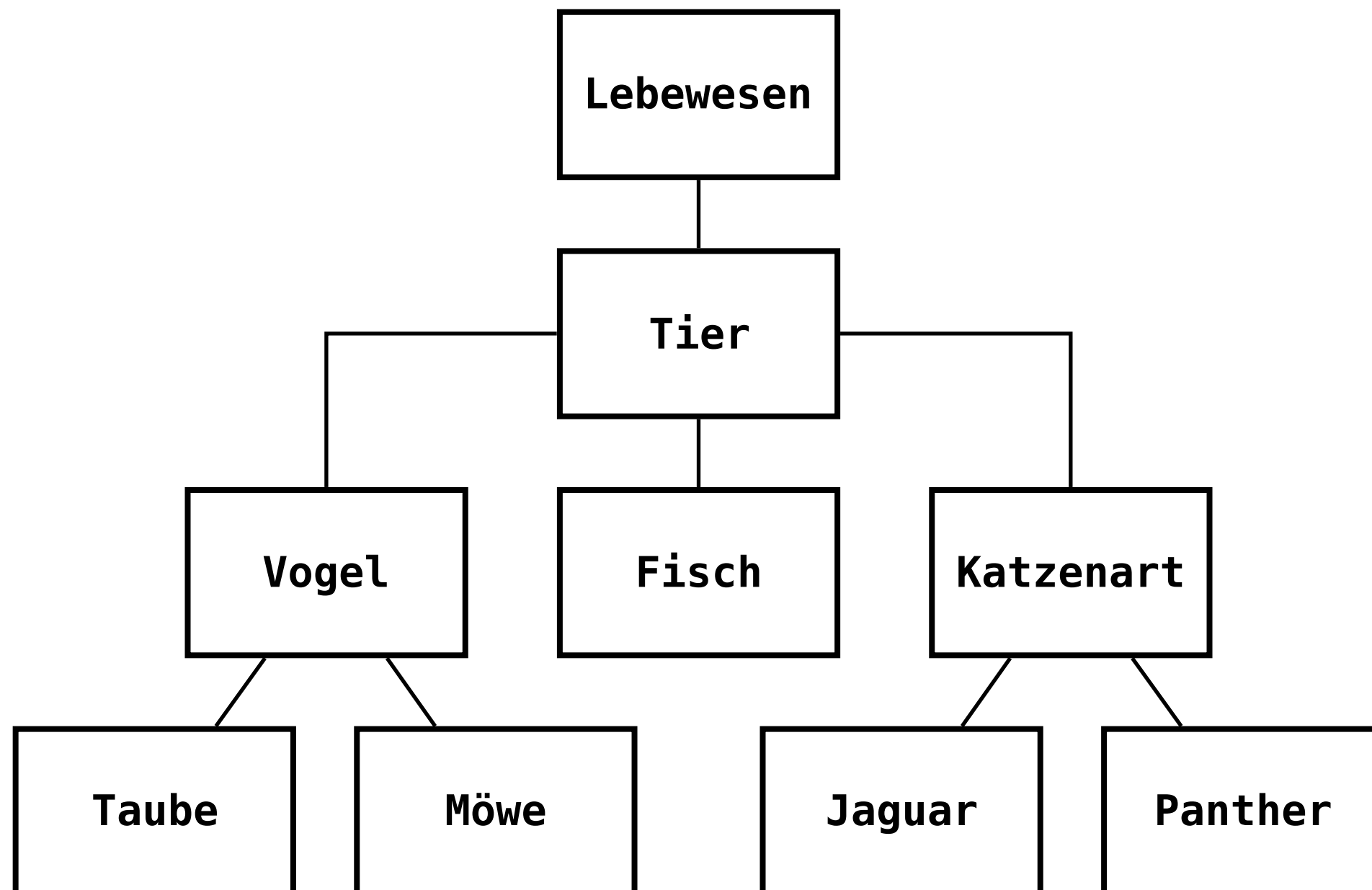
**Grafische
Anwendung**

Single Responsibility Principle

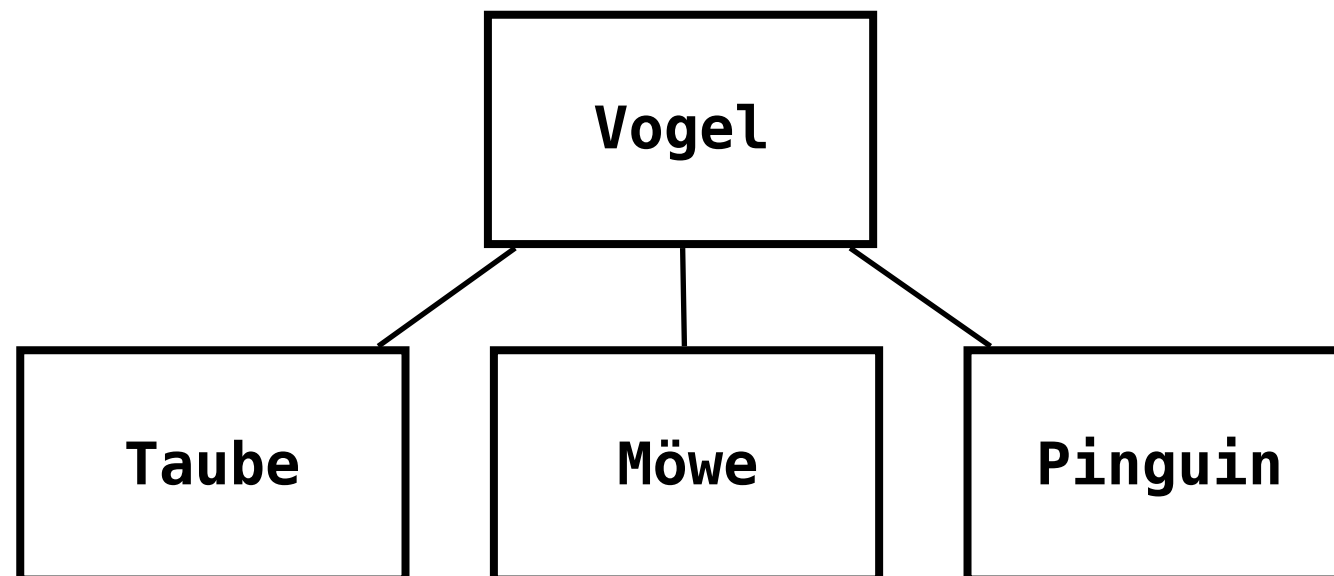
Es sollte nie mehr als einen Grund geben, eine Klasse zu ändern



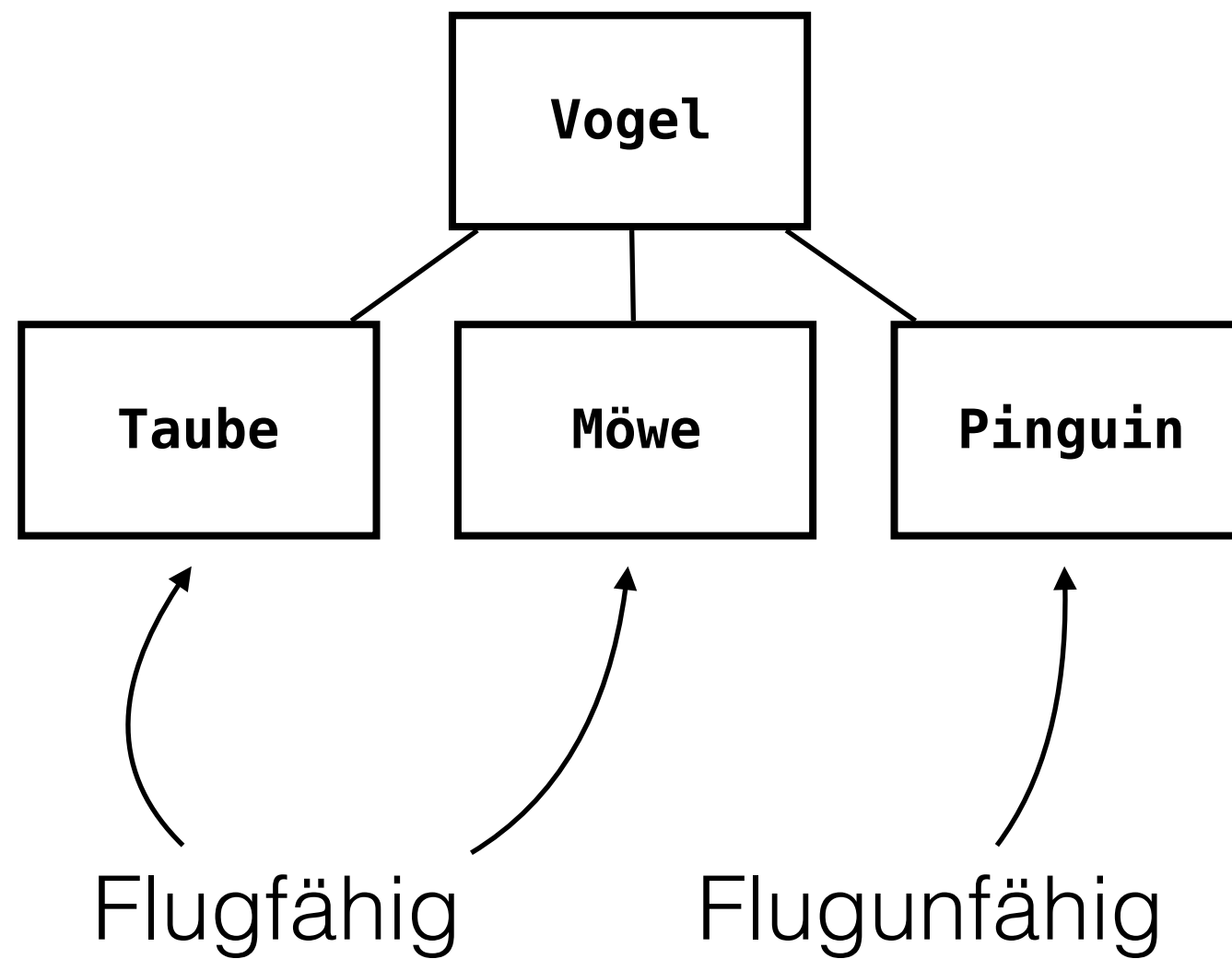
Objekt = Physisches Modell



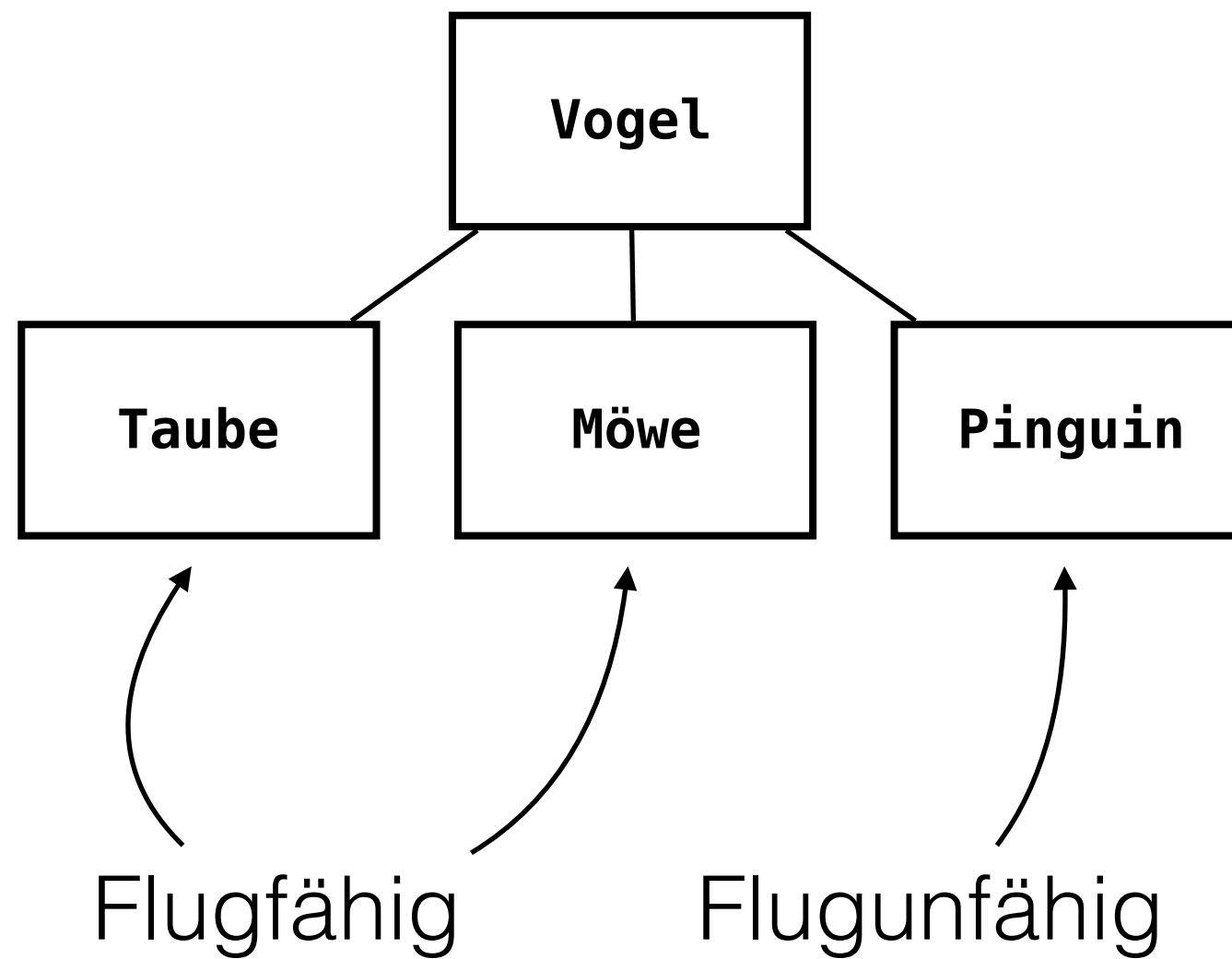
Objekt = Physisches Modell



Objekt = Physisches Modell



Objekt \neq Physisches Modell



Objekt \neq Physisches Modell

```
interface Modem
{
    void dial(String number);
    void hangup();
    void send(char c);
    char receive();
}
```

Objekt \neq Physisches Modell

```
interface Modem
{
    void dial(String number);
    void hangup();
    void send(char c);
    char receive();
}
```

Aufgaben

- Verbindungsfunktionen
- Übertragungsfunktionen

Objekt \neq Physisches Modell

<<Connection>>

+ dial(String number)
+ hangup()

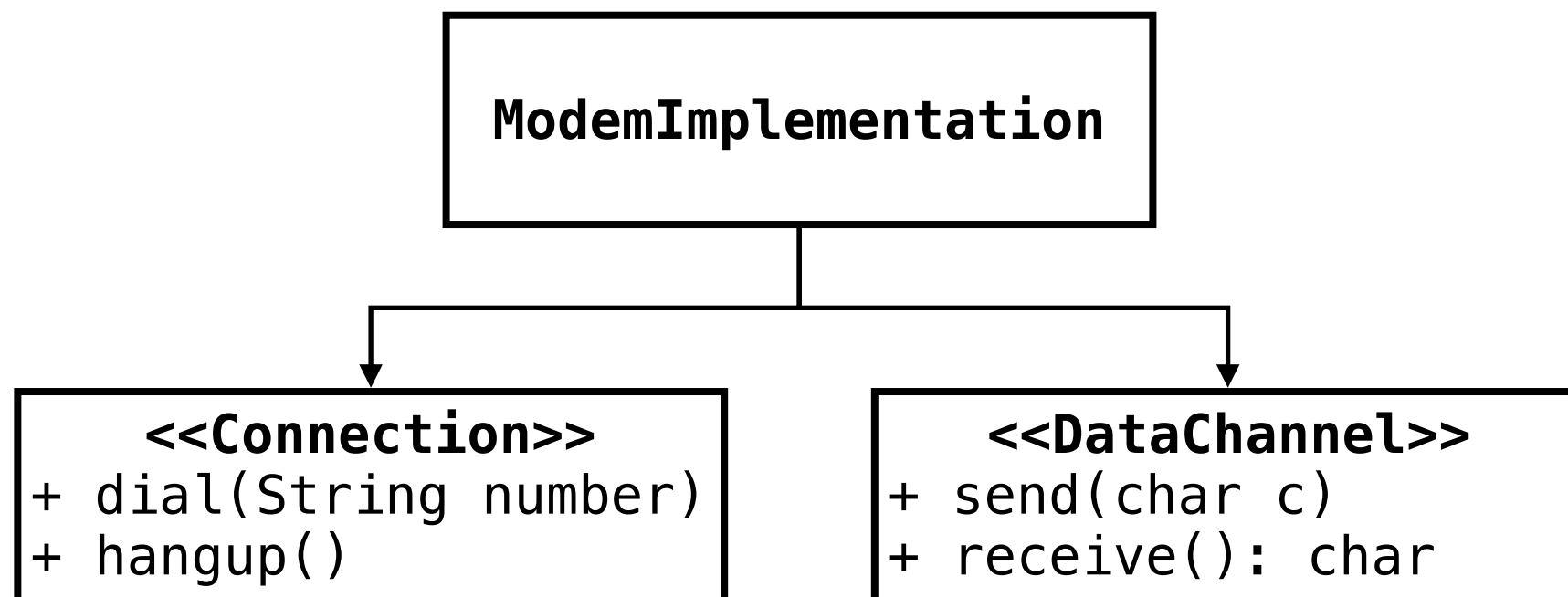
<<DataChannel>>

+ send(char c)
+ receive(): char

Aufgaben

- Verbindungsfunktionen
- Übertragungsfunktionen

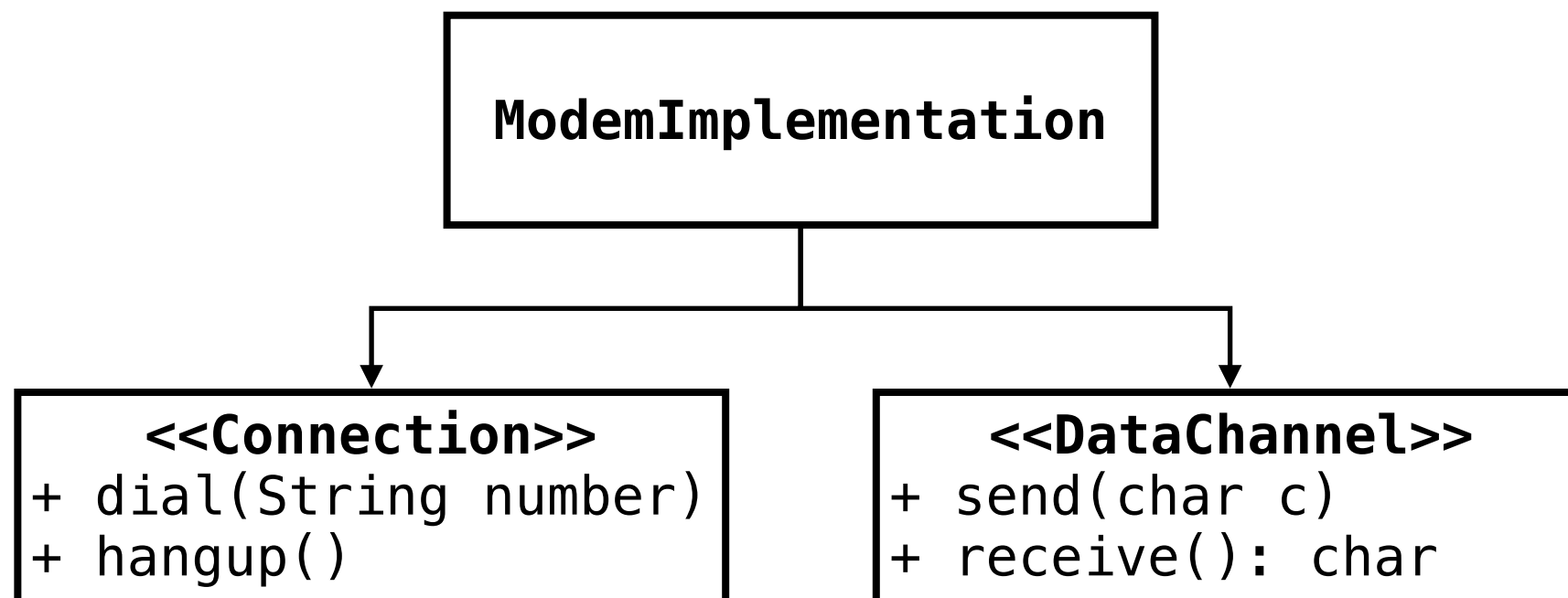
Objekt \neq Physisches Modell



Aufgaben

- Verbindungsfunktionen
- Übertragungsfunktionen

Objekt \neq Physisches Modell



Aufgaben

- Verbindungsfunktionen
- Übertragungsfunktionen

Objekt = funktionale Einheit

```
public class UserService {  
    public void register(String email, String password) {  
        if (!email.contains("@")) {  
            throw new ValidationException("Email is not an email!");  
        }  
  
        User user = new User(email, password);  
        database.save(user);  
  
        MailMessage message = new MailMessage("Welcome", "Welcome to the site!");  
        String from = "no-reply@site.com";  
        smtpClient.send(from, email, message);  
    }  
}
```

```
public class UserService {  
    public void register(String email, String password) {  
        if (!email.contains("@")) {  
            throw new ValidationException("Email is not an email!");  
        }  
  
        User user = new User(email, password);  
        database.save(user);  
  
        MailMessage message = new MailMessage("Welcome", "Welcome to the site!");  
        String from = "no-reply@site.com";  
        smtpClient.send(from, email, message);  
    }  
}
```

```
public class UserService {  
  
    public void register(String email, String password) {  
  
        if (!isValidEmail(email)) {  
            throw new ValidationException("Email is not an email!");  
        }  
  
        User user = new User(email, password);  
        database.save(user);  
  
        MailMessage message = new MailMessage("Welcome", "Welcome to the site!");  
        String from = "no-reply@site.com";  
        smtpClient.send(from, email, message);  
    }  
  
    private bool isValidEmail(String email) {  
        return email.contains("@");  
    }  
  
}
```



```
public class UserService {  
  
    public void register(String email, String password) {  
        if (!isValidEmail(email)) {  
            throw new ValidationException("Email is not an email!");  
        }  
  
        User user = new User(email, password);  
        database.save(user);  
  
        MailMessage message = new MailMessage("Welcome", "Welcome to the site!");  
        String from = "no-reply@site.com";  
        smtpClient.send(from, email, message);  
    }  
  
    private bool isValidEmail(String email) {  
        return email.contains("@");  
    }  
  
}
```

```
public class UserService {  
  
    public void register(String email, String password) {  
  
        if (!isValidEmail(email)) {  
            throw new ValidationException("Email is not an email!");  
        }  
  
        User user = new User(email, password);  
        database.save(user);  
  
        MailMessage message = new MailMessage("Welcome", "Welcome to the site!");  
        String from = "no-reply@site.com";  
        smtpClient.send(from, email, message);  
    }  
  
    private bool isValidEmail(String email) {  
        return email.contains("@");  
    }  
  
}
```

```
public class UserService {  
  
    public void register(String email, String password) {  
  
        if (!isValidEmail(email)) {  
            throw new ValidationException("Email is not an email!");  
        }  
  
        User user = new User(email, password);  
        database.save(user);  
  
        MailMessage message = new MailMessage("Welcome", "Welcome to the site!");  
        String from = "no-reply@site.com";  
        smtpClient.send(from, email, message);  
    }  
  
    private bool isValidEmail(String email) {  
        return email.contains("@");  
    }  
  
}
```

```
public class UserService {  
  
    public void register(String email, String password) {  
        validateEmail(email);  
  
        User user = new User(email, password);  
        database.save(user);  
  
        MailMessage message = new MailMessage("Welcome", "Welcome to the site!");  
        String from = "no-reply@site.com";  
        smtpClient.send(from, email, message);  
    }  
  
    private void validateEmail(String email) {  
        if (!isValidEmail(email)) {  
            throw new ValidationException("Email is not an email!");  
        }  
    }  
  
    private bool isValidEmail(String email) {  
        return email.contains("@");  
    }  
  
}
```

```
public class UserService {  
  
    public void register(String email, String password) {  
        validateEmail(email);  
  
        User user = new User(email, password);  
        database.save(user);  
  
        MailMessage message = new MailMessage("Welcome", "Welcome to the site!");  
        String from = "no-reply@site.com";  
        smtpClient.send(from, email, message);  
    }  
  
    private void validateEmail(String email) {  
        if (!isValidEmail(email)) {  
            throw new ValidationException("Email is not an email!");  
        }  
    }  
  
    private bool isValidEmail(String email) {  
        return email.contains("@");  
    }  
  
}
```

```
public class UserService {  
  
    public void register(String email, String password) {  
        validateEmail(email);  
  
        User user = new User(email, password);  
        database.save(user);  
  
        MailMessage message = new MailMessage("Welcome", "Welcome to the site!");  
        String from = "no-reply@site.com";  
        smtpClient.send(from, email, message);  
    }  
  
    private void validateEmail(String email) {  
        if (!isValidEmail(email)) {  
            throw new ValidationException("Email is not an email!");  
        }  
    }  
  
    private bool isValidEmail(String email) {  
        return email.contains("@");  
    }  
  
}
```

```
public class UserService {  
  
    public void register(String email, String password) {  
        validateEmail(email);  
  
        User user = new User(email, password);  
        database.save(user);  
  
        MailMessage message = new MailMessage("Welcome", "Welcome to the site!");  
        String from = "no-reply@site.com";  
        smtpClient.send(from, email, message);  
    }  
  
    private void validateEmail(String email) {  
        if (!isValidEmail(email)) {  
            throw new ValidationException("Email is not an email!");  
        }  
    }  
  
    private bool isValidEmail(String email) {  
        return email.contains("@");  
    }  
  
}
```

```
public class UserService {  
  
    public void register(String email, String password) {  
        validateEmail(email);  
  
        User user = new User(email, password);  
        database.save(user);  
  
        MailMessage message = new MailMessage("Welcome", "Welcome to the site!");  
        sendEmail(message, email);  
    }  
  
    private void validateEmail(String email) {  
        if (!isValidEmail(email)) {  
            throw new ValidationException("Email is not an email!");  
        }  
    }  
  
    private bool isValidEmail(String email) {  
        return email.contains("@");  
    }  
  
    private void sendEmail(MailMessage message, String email) {  
        smtpClient.send("no-reply@site.com", email, message);  
    }  
  
}
```



```
public class UserService {  
  
    public void register(String email, String password) {  
        validateEmail(email);  
  
        User user = new User(email, password);  
        database.save(user);  
  
        MailMessage message = new MailMessage("Welcome", "Welcome to the site!");  
        sendEmail(message, email);  
    }  
  
    private void validateEmail(String email) {  
        if (!isValidEmail(email)) {  
            throw new ValidationException("Email is not an email!");  
        }  
    }  
  
    private bool isValidEmail(String email) {  
        return email.contains("@");  
    }  
  
    private void sendEmail(MailMessage message, String email) {  
        smtpClient.send("no-reply@site.com", email, message);  
    }  
  
}
```

```
public class UserService {  
  
    public void register(String email, String password) {  
        validateEmail(email);  
  
        User user = new User(email, password);  
        database.save(user);  
  
        MailMessage message = new MailMessage("Welcome", "Welcome to the site!");  
        sendEmail(message, email);  
    }  
  
    private void validateEmail(String email) {  
        if (!isValidEmail(email)) {  
            throw new ValidationException("Email is not an email!");  
        }  
    }  
  
    private bool isValidEmail(String email) {  
        return email.contains("@");  
    }  
  
    private void sendEmail(MailMessage message, String email) {  
        smtpClient.send("no-reply@site.com", email, message);  
    }  
  
}
```

```
public class UserService {  
  
    public void register(String email, String password) {  
        validateEmail(email);  
  
        User user = new User(email, password);  
        database.save(user);  
  
        MailMessage message = new MailMessage("Welcome", "Welcome to the site!");  
        sendEmail(message, email);  
    }  
  
    private void validateEmail(String email) {  
        if (!isValidEmail(email)) {  
            throw new ValidationException("Email is not an email!");  
        }  
    }  
  
    private bool isValidEmail(String email) {  
        return email.contains("@");  
    }  
  
    private void sendEmail(MailMessage message, String email) {  
        smtpClient.send("no-reply@site.com", email, message);  
    }  
  
}
```

```
public class UserService {  
  
    public void register(String email, String password) {  
        validateEmail(email);  
  
        User user = new User(email, password);  
        database.save(user);  
  
        MailMessage message = new MailMessage("Welcome", "Welcome to the site!");  
        sendEmail(message, email);  
    }  
  
    private void validateEmail(String email) {  
        if (!isValidEmail(email)) {  
            throw new ValidationException("Email is not an email!");  
        }  
    }  
  
    private bool isValidEmail(String email) {  
        return email.contains("@");  
    }  
  
    private void sendEmail(MailMessage message, String email) {  
        smtpClient.send("no-reply@site.com", email, message);  
    }  
  
}
```

```
public class UserService {
```

```
    public void register(String email, String password) {  
        validateEmail(email);
```

```
        public class EmailService {
```

```
            public void validateEmail(String email) {
```

```
                if (!isValidEmail(email)) {
```

```
                    throw new ValidationException("Email is not an email!");
```

```
                }
```

```
            }
```

```
            public bool isValidEmail(String email) {
```

```
                return email.contains("@");
```

```
            }
```

```
            public void sendEmail(MailMessage message, String email) {
```

```
                smtpClient.send("no-reply@site.com", email, message);
```

```
            }
```

```
        }
```

```
    private void sendEmail(MailMessage message, String email) {
```

```
        smtpClient.send("no-reply@site.com", email, message);
```

```
    }
```

```
}
```

```
public class UserService {

    public void register(String email, String password) {
        validateEmail(email);

        User user = new User(email, password);
        database.save(user);

        MailMessage message = new MailMessage("Welcome", "Welcome to the site!");
        sendEmail(message, email);
    }

    private void validateEmail(String email) {
        if (!isValidEmail(email)) {
            throw new ValidationException("Email is not an email!");
        }
    }

    private bool isValidEmail(String email) {
        return email.contains("@");
    }

    private void sendEmail(MailMessage message, String email) {
        smtpClient.send("no-reply@site.com", email, message);
    }
}
```

```
public class UserService {  
  
    public void register(String email, String password) {  
        validateEmail(email);  
  
        User user = new User(email, password);  
        database.save(user);  
  
        MailMessage message = new MailMessage("Welcome", "Welcome to the site!");  
        sendEmail(message, email);  
    }  
  
    private void validateEmail(String email) {  
        if (!isValidEmail(email)) {  
            throw new ValidationException("Email is not an email!");  
        }  
    }  
  
    private bool isValidEmail(String email) {  
        return email.contains("@");  
    }  
  
    private void sendEmail(MailMessage message, String email) {  
        smtpClient.send("no-reply@site.com", email, message);  
    }  
  
}
```

```
public class UserService {  
  
    private EmailService emailService;  
  
    public UserService(EmailService emailService) {  
        this.emailService = emailService;  
    }  
  
    public void register(String email, String password) {  
        emailService.validateEmail(email);  
  
        User user = new User(email, password);  
        database.save(user);  
  
        MailMessage message = new MailMessage("Welcome", "Welcome to the site!");  
        emailService.sendEmail(message, email);  
    }  
}
```



```
public class UserService {  
  
    private EmailService emailService;  
  
    public UserService(EmailService emailService) {  
        this.emailService = emailService;  
    }  
  
    public void register(String email, String password) {  
        emailService.validateEmail(email);  
  
        User user = new User(email, password);  
        database.save(user);  
  
        MailMessage message = new MailMessage("Welcome", "Welcome to the site!");  
        emailService.sendEmail(message, email);  
    }  
}
```

```
public class UserService {  
  
    private EmailService emailService;  
  
    public UserService(EmailService emailService) {  
        this.emailService = emailService;  
    }  
  
    public void register(String email, String password) {  
        emailService.validateEmail(email);  
  
        User user = new User(email, password);  
        database.save(user);  
  
        MailMessage message = new MailMessage("Welcome", "Welcome to the site!");  
        emailService.sendEmail(message, email);  
    }  
}
```

```
public class UserService {  
  
    private EmailService emailService;  
  
    public UserService(EmailService emailService) {  
        this.emailService = emailService;  
    }  
  
    public void register(String email, String password) {  
        emailService.validateEmail(email);  
  
        User user = new User(email, password);  
        database.save(user);  
  
        MailMessage message = new MailMessage("Welcome", "Welcome to the site!");  
        emailService.sendEmail(message, email);  
    }  
}
```

```
public class UserService {  
  
    private EmailService emailService;  
  
    public UserService(EmailService emailService) {  
        this.emailService = emailService;  
    }  
  
    public void register(String email, String password) {  
        emailService.validateEmail(email);  
  
        createUser(email, password);  
  
        MailMessage message = new MailMessage("Welcome", "Welcome to the site!");  
        emailService.sendEmail(message, email);  
    }  
  
    private User createUser(String email, String password) {  
        User user = new User(email, password);  
        database.save(user);  
        return user;  
    }  
}
```

```
public class UserService {  
  
    private EmailService emailService;  
  
    public UserService(EmailService emailService) {  
        this.emailService = emailService;  
    }  
  
    public void register(String email, String password) {  
        emailService.validateEmail(email);  
  
        createUser(email, password);  
  
        MailMessage message = new MailMessage("Welcome", "Welcome to the site!");  
        emailService.sendEmail(message, email);  
    }  
  
    private User createUser(String email, String password) {  
        User user = new User(email, password);  
        database.save(user);  
        return user;  
    }  
}
```

```
public class UserService {  
  
    private EmailService emailService;  
  
    public UserService(EmailService emailService) {  
        this.emailService = emailService;  
    }  
  
    public void register(String email, String password) {  
        emailService.validateEmail(email);  
  
        createUser(email, password);  
  
        MailMessage message = new MailMessage("Welcome", "Welcome to the site!");  
        emailService.sendEmail(message, email);  
    }  
  
    private User createUser(String email, String password) {  
        User user = new User(email, password);  
        database.save(user);  
        return user;  
    }  
}
```

```
public class UserService {  
  
    private EmailService emailService;  
  
    public UserService(EmailService emailService) {  
        this.emailService = emailService;  
    }  
  
    public void register(String email, String password) {  
        emailService.validateEmail(email);  
  
        createUser(email, password);  
  
        MailMessage message = new MailMessage("Welcome", "Welcome to the site!");  
        emailService.sendEmail(message, email);  
    }  
  
    private User createUser(String email, String password) {  
        User user = new User(email, password);  
        database.save(user);  
        return user;  
    }  
}
```

```
public class UserService {  
  
    private EmailService emailService;  
  
    public UserService(EmailService emailService) {  
        this.emailService = emailService;  
    }  
  
    public void register(String email, String password) {  
        emailService.validateEmail(email);  
  
        User user = createUser(email, password);  
  
        welcomeUser(user);  
    }  
  
    private User createUser(String email, String password) {  
        User user = new User(email, password);  
        database.save(user);  
        return user;  
    }  
  
    private void welcomeUser(User user) {  
        MailMessage message = new MailMessage("Welcome", "Welcome to the site!");  
        emailService.sendEmail(message, user.email);  
    }  
  
}
```



```
public class UserService {  
  
    private EmailService emailService;  
  
    public UserService(EmailService emailService) {  
        this.emailService = emailService;  
    }  
  
    public void register(String email, String password) {  
        emailService.validateEmail(email);  
  
        User user = createUser(email, password);  
  
        welcomeUser(user);  
    }  
  
    private User createUser(String email, String password) {  
        User user = new User(email, password);  
        database.save(user);  
        return user;  
    }  
  
    private void welcomeUser(User user) {  
        MailMessage message = new MailMessage("Welcome", "Welcome to the site!");  
        emailService.sendEmail(message, user.email);  
    }  
  
}
```

```
public class UserService {  
  
    private EmailService emailService;  
  
    public UserService(EmailService emailService) {  
        this.emailService = emailService;  
    }  
  
    public void register(String email, String password) {  
        emailService.validateEmail(email);  
  
        User user = createUser(email, password);  
  
        welcomeUser(user);  
    }  
  
    private User createUser(String email, String password) {  
        User user = new User(email, password);  
        database.save(user);  
        return user;  
    }  
  
    private void welcomeUser(User user) {  
        MailMessage message = new MailMessage("Welcome", "Welcome to the site!");  
        emailService.sendEmail(message, user.email);  
    }  
  
}
```

```
public class UserService {  
  
    private UserValidator userValidator;  
    private WelcomingService welcomingService;  
    private UserStorage userStorage;  
  
    public UserService(UserValidator validator, WelcomingService service,  
                      UserStorage storage)  
    {  
        userValidator = validator;  
        welcomingService = service;  
        userStorage = storage;  
    }  
  
    public void register(String email, String password) {  
        User user = createUser(email, password);  
  
        userValidator.validate(user)  
        userStorage.persist(user);  
        welcomingService.welcome(user);  
    }  
  
    private User createUser(String email, String password) {  
        return new User(email, password);  
    }  
  
}
```



Dokumentiere Code

Wird zu oft *und* gebraucht?



Assoziiere Methoden mit der Schnittstelle

ValidateEmail hat nichts mit einem UserService zu tun

Open-Closed Principle

Offen für Erweiterung, aber geschlossen gegenüber Modifikationen



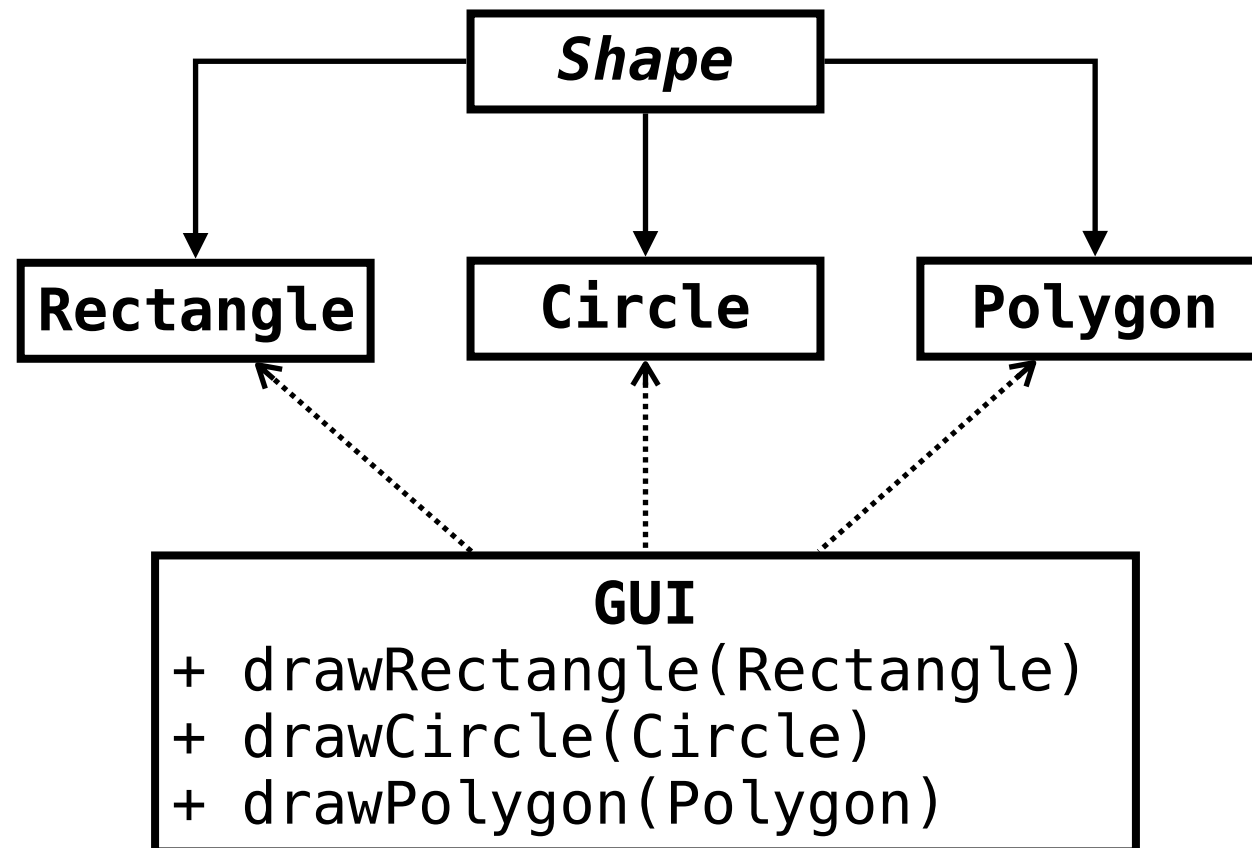
Erweiterung

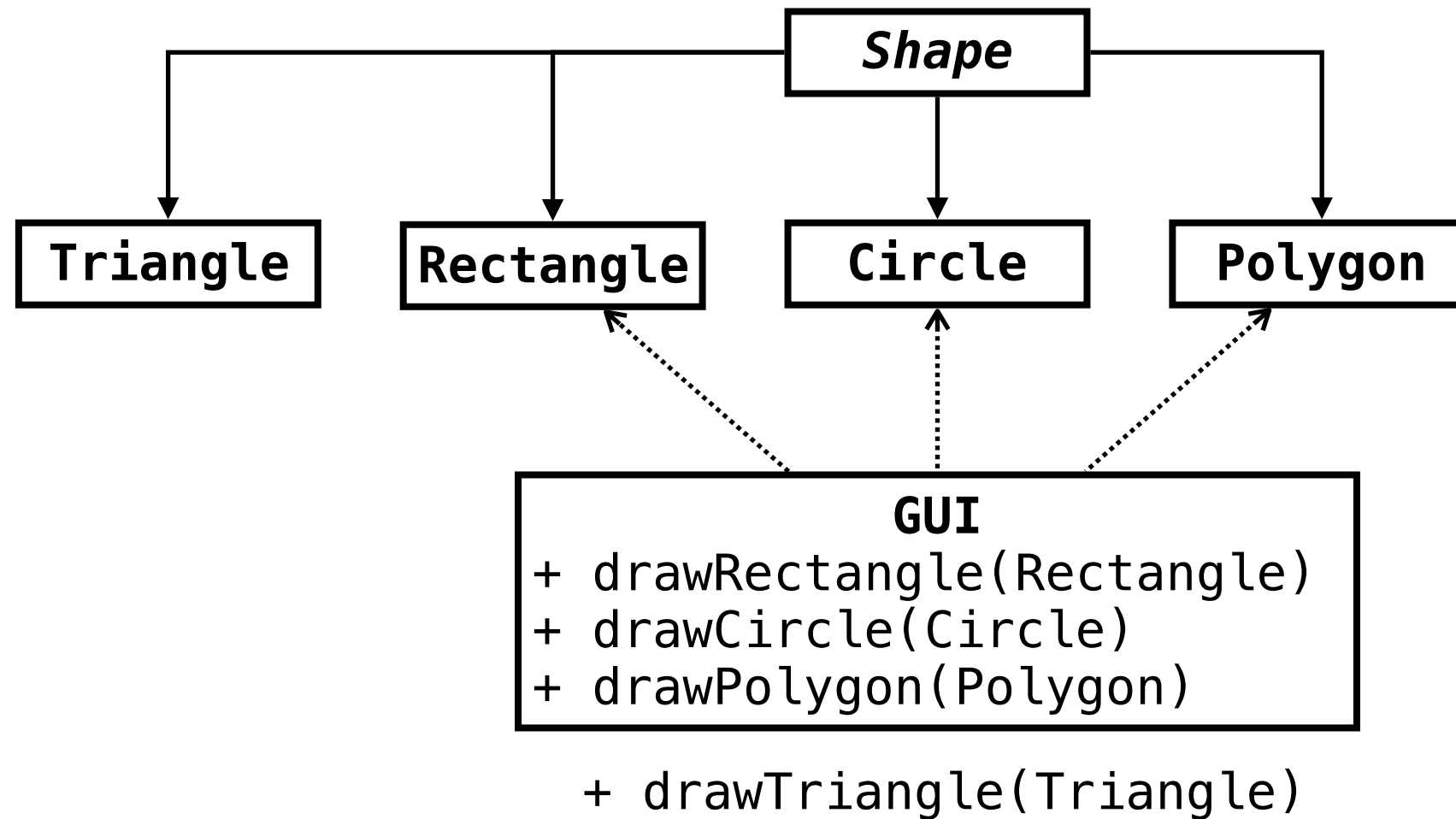
Das Verhalten des Moduls kann durch andere Module erweitert oder verändert werden

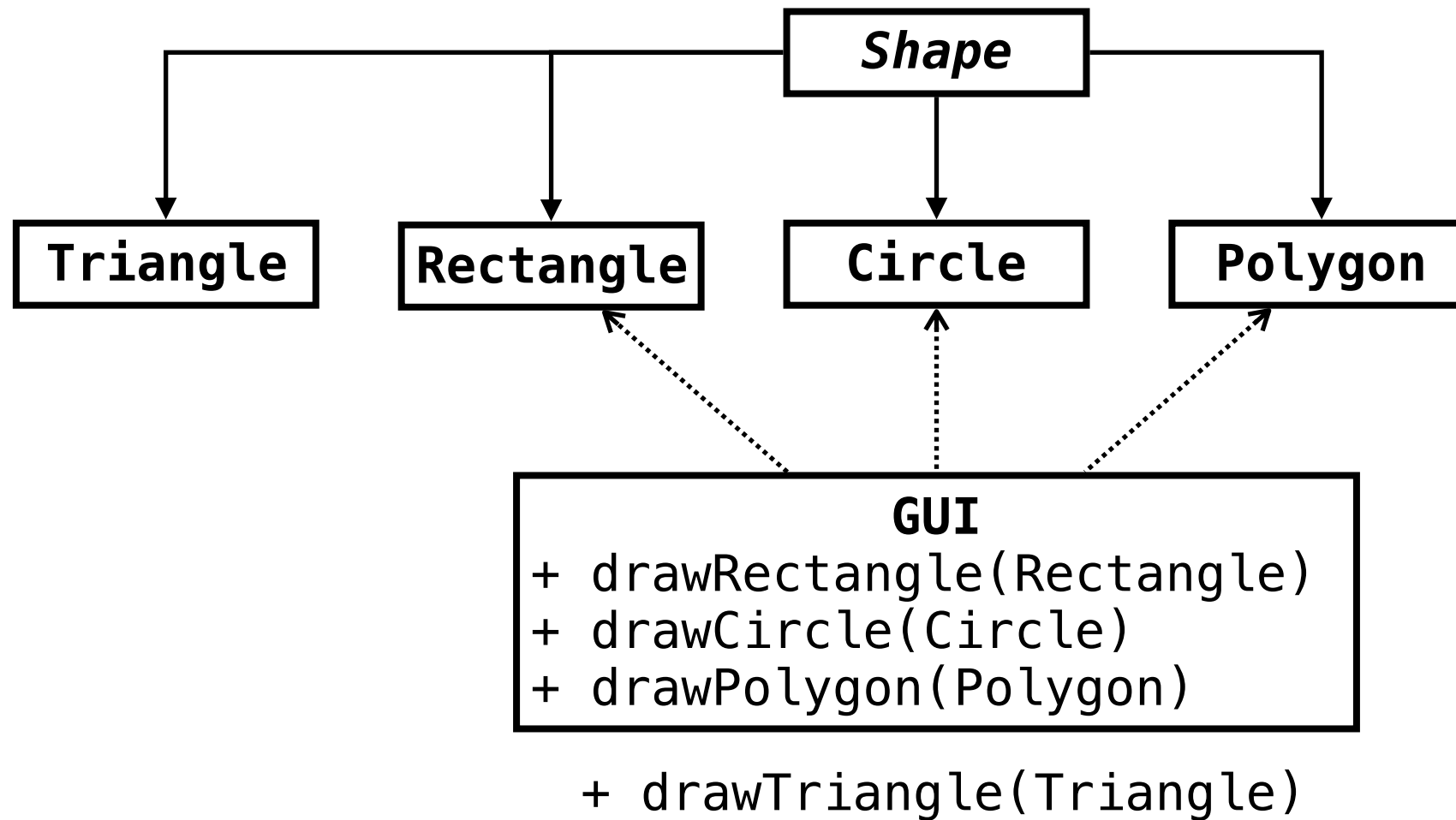


Modifikation

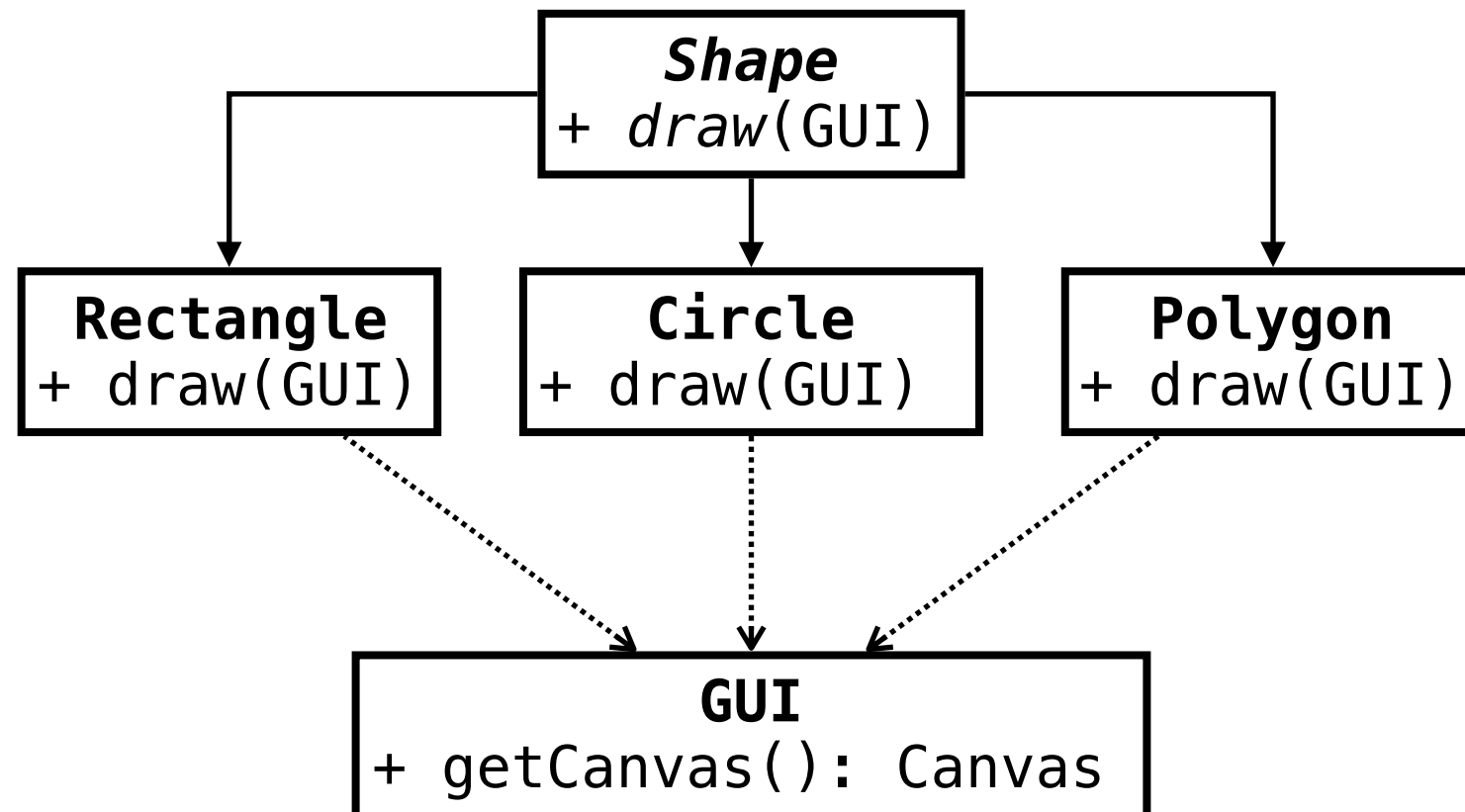
Der Quelltext des Moduls ist geschlossen und wird in keinem Fall angepasst oder erweitert

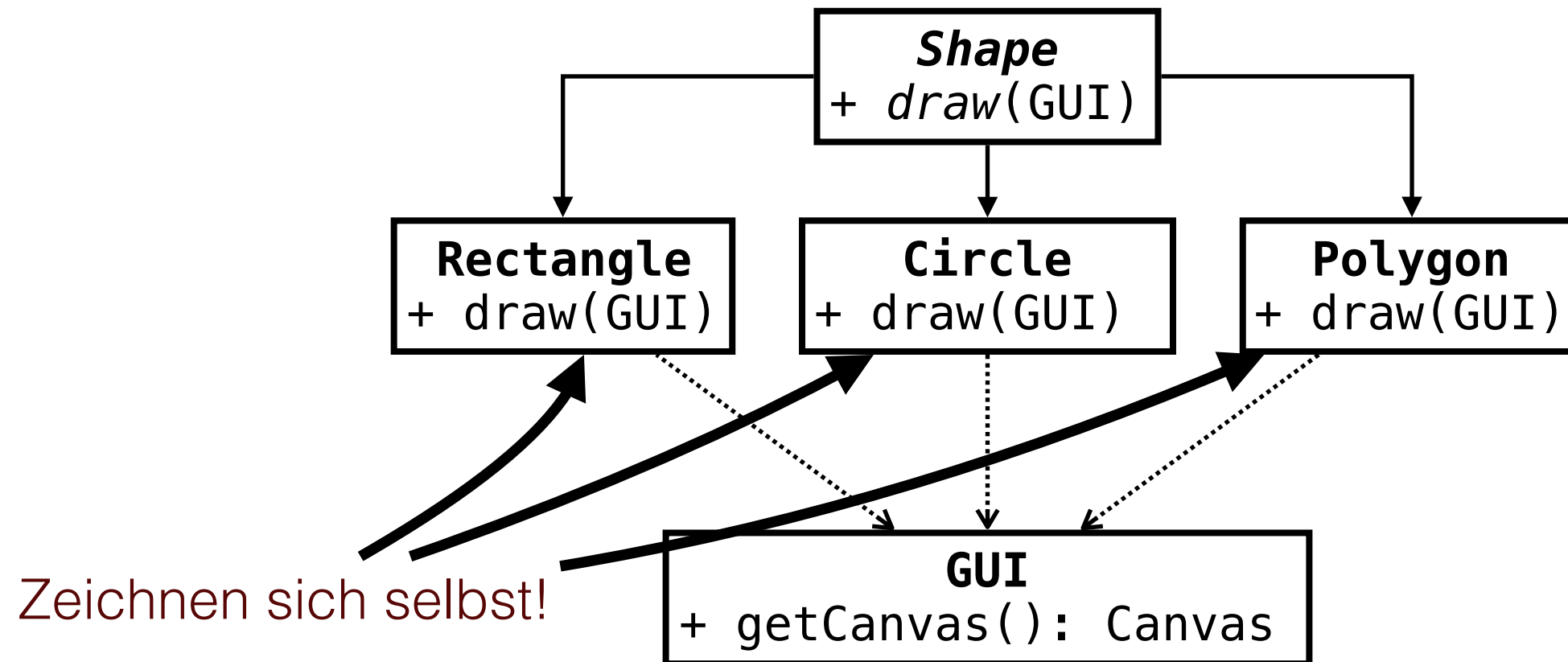












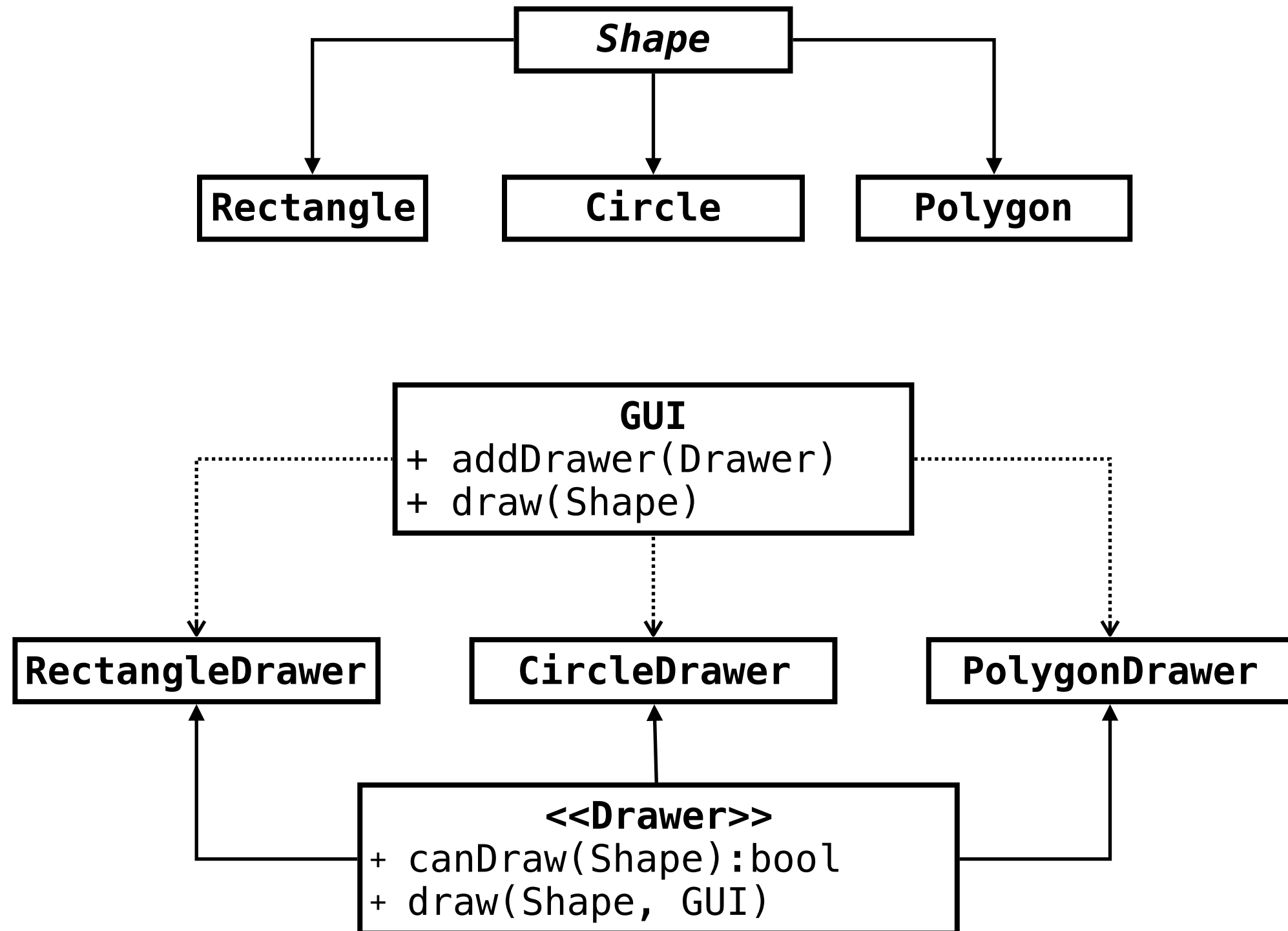
SRP

OCP

LSP

ISP

DIP



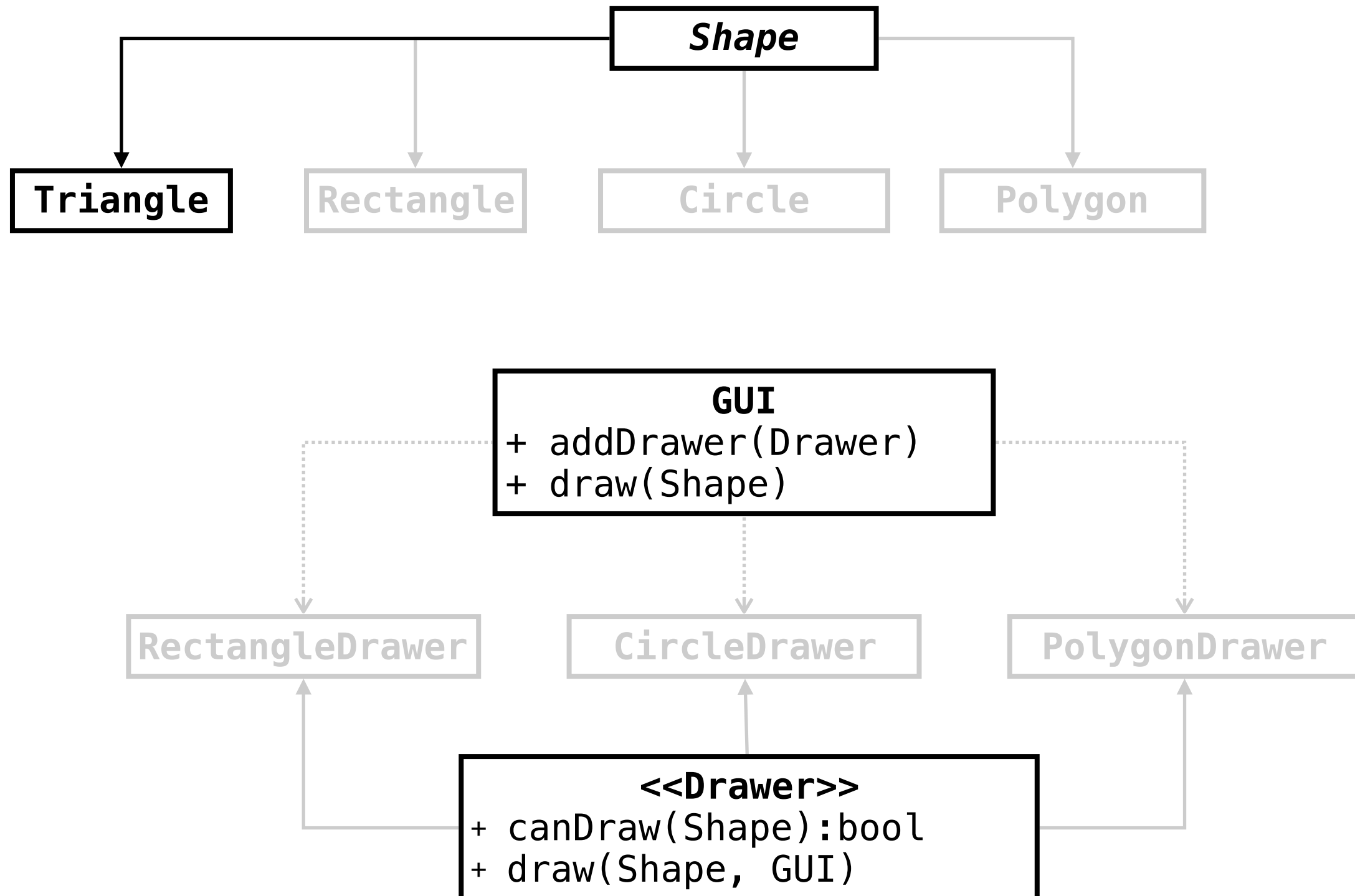
SRP

OCP

LSP

ISP

DIP



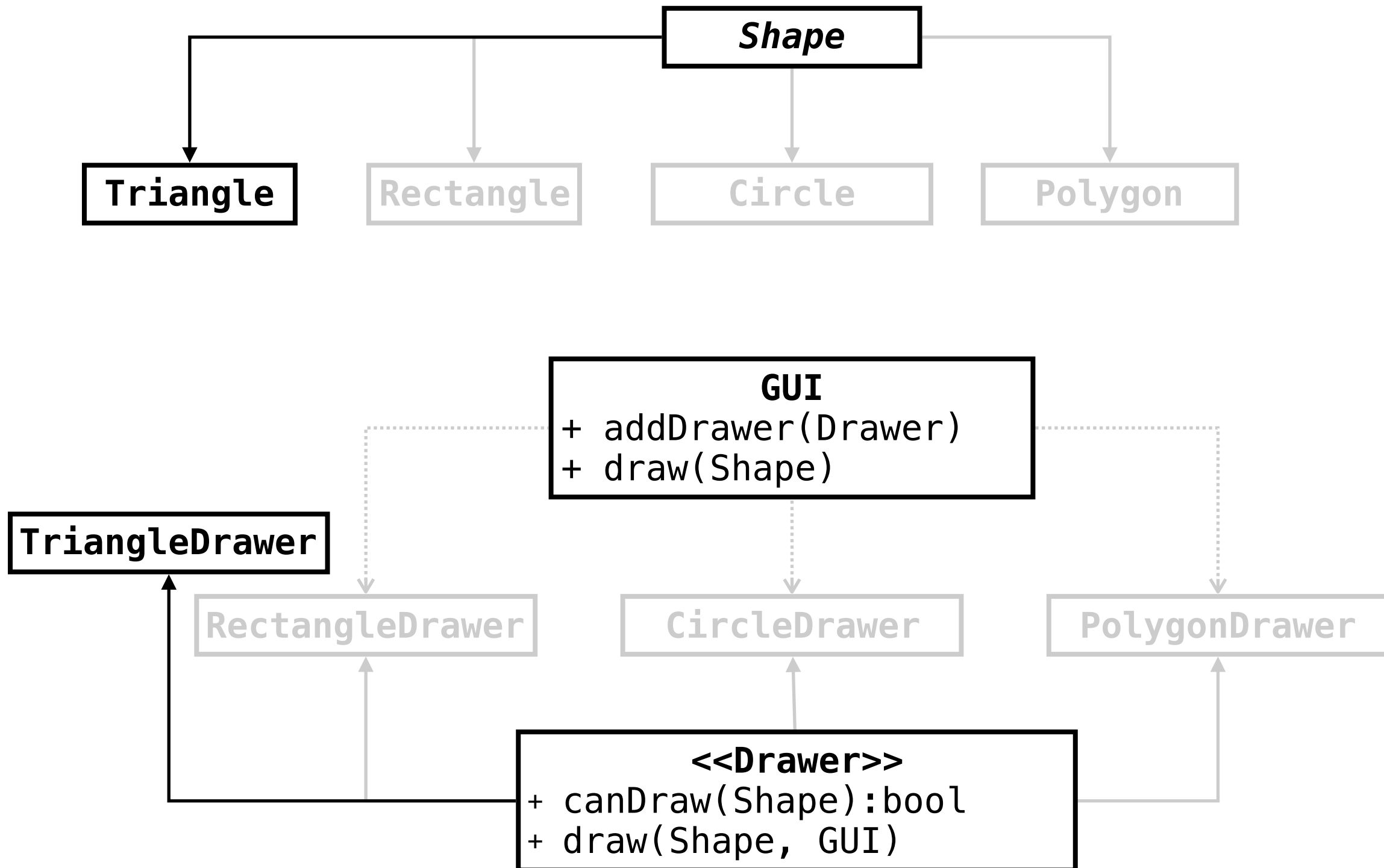
SRP

OCP

LSP

ISP

DIP



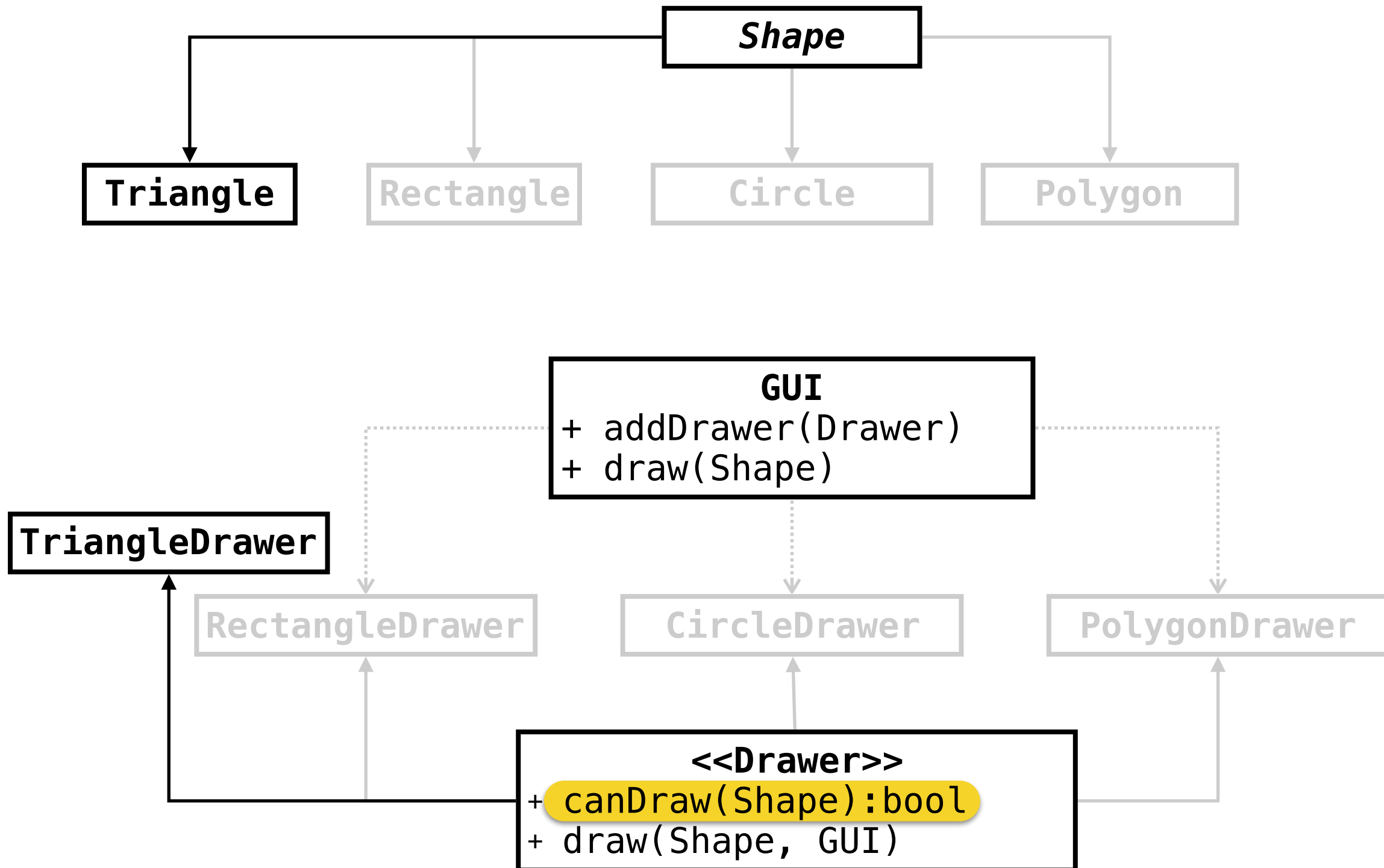
SRP

OCP

LSP

ISP

DIP



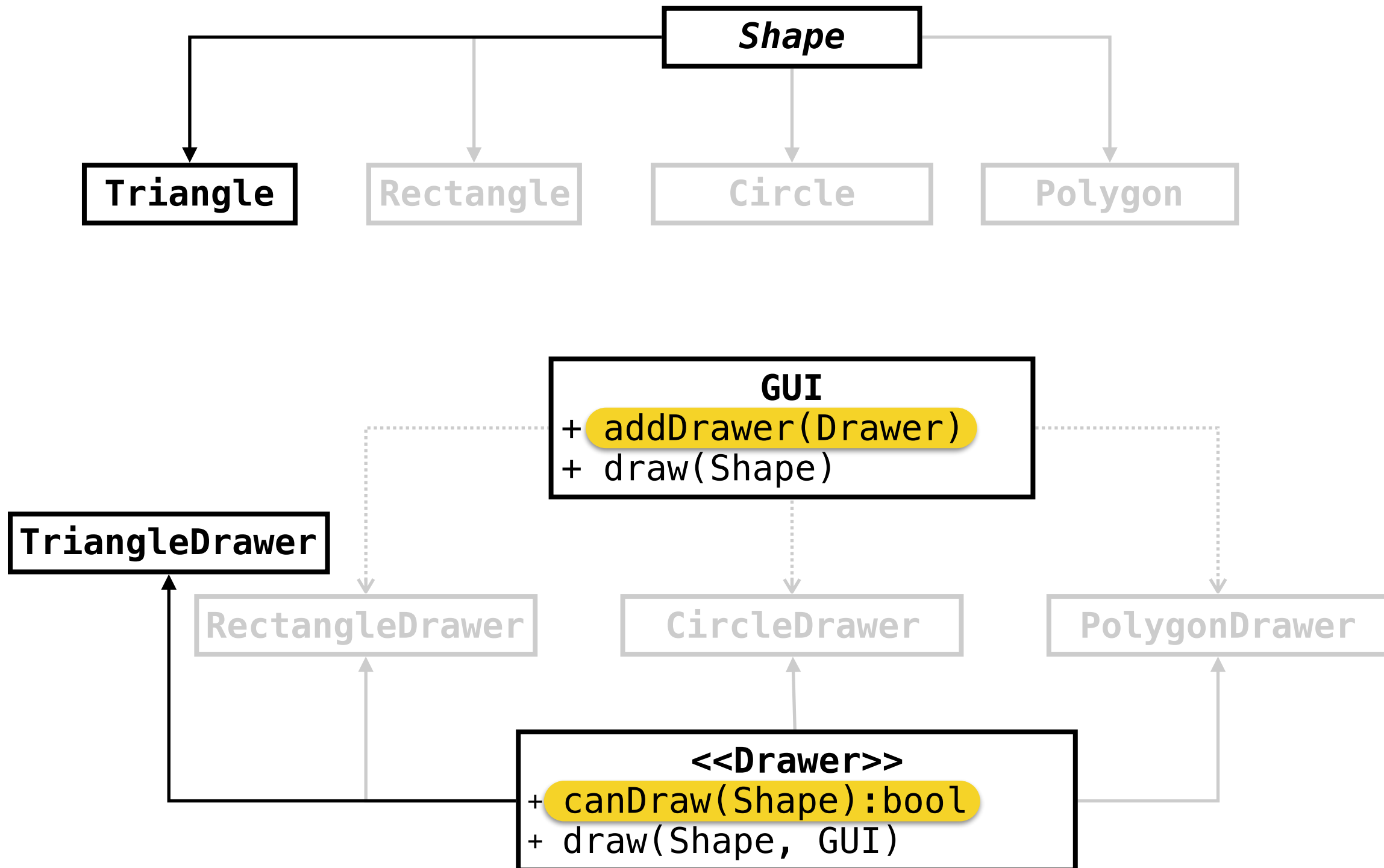
SRP

OCP

LSP

ISP

DIP



SRP

OCP

LSP

ISP

DIP



Konkrete Sequenz

- Berechnung durchführen
- Ausgabe

Konkrete Sequenz

- Berechnung durchführen
- Ausgabe

Abstrakte Logik

- Operator bereitstellen
- Berechnung

Konkrete Sequenz

- Berechnung durchführen
- Ausgabe

Abstrakte Logik

- Operator bereitstellen
- Berechnung

$$3 + 7 = 10$$

```
abstract class Calculation {  
    final public void doCalc(int arg1, int arg2) {  
        int result    = getResult(arg1, arg2);  
        char operator = getOperator();  
  
        printCalc(arg1, operator, arg2, result);  
    }  
  
    private void printCalc(int arg1, char operator, int arg2, int result) {  
        String output = String.format("%d %s %d = %d");  
        System.out.println(output);  
    }  
  
    protected abstract char getOperator();  
  
    protected abstract int getResult(int arg1, int arg2);  
}
```

```
abstract class Calculation {  
    final public void doCalc(int arg1, int arg2) {  
        int result    = getResult(arg1, arg2);  
        char operator = getOperator();  
  
        printCalc(arg1, operator, arg2, result);  
    }  
  
    private void printCalc(int arg1, char operator, int arg2, int result) {  
        String output = String.format("%d %s %d = %d");  
        System.out.println(output);  
    }  
  
    protected abstract char getOperator();  
  
    protected abstract int getResult(int arg1, int arg2);  
}
```

```
abstract class Calculation {  
    final public void doCalc(int arg1, int arg2) {  
        int result = getResult(arg1, arg2);  
        char operator = getOperator();  
  
        printCalc(arg1, operator, arg2, result);  
    }  
  
    private void printCalc(int arg1, char operator, int arg2, int result) {  
        String output = String.format("%d %s %d = %d");  
        System.out.println(output);  
    }  
  
    protected abstract char getOperator();  
  
    protected abstract int getResult(int arg1, int arg2);  
}
```

```
class Multiplication extends Calculation {  
    @Override  
    protected char getOperator() {  
        return 'x';  
    }  
  
    @Override  
    protected int getResult(int arg1, int arg2) {  
        return arg1 * arg2;  
    }  
}
```

– Quelltext –

```
Calculation calc = new Multiplication();  
calc.doCalc(6, 6);
```

– Ausgabe –

```
6 x 6 = 36
```


Liskov Substitution Principle

Ein abgeleiteter Typ verhält sich wie in der Basis vereinbart

Liskov Substitution Principle

Ein abgeleiteter Typ verhält sich wie in der Basis vereinbart

Liskov Substitution Principle

Ein abgeleiteter Typ verhält sich wie in der Basis vereinbart

„Eine stärkere Forderung [als Kovarianz und Kontravarianz] wird benötigt, die das Verhalten von Untertypen einschränkt: Eigenschaften, die anhand der Spezifikation des vermeintlichen Typs eines Objektes bewiesen werden können, sollten auch dann gelten, wenn das Objekt einem Untertyp dieses Typs angehört:

Sei $q(x)$ eine beweisbare Eigenschaft von Objekten x des Typs T .
Dann soll $q(y)$ für Objekte y des Typs S wahr sein, wobei S ein Untertyp von T ist.“

Barbara Liskov et al., 1993

„Sei $q(x)$ eine beweisbare Eigenschaft von Objekten x des Typs T . Dann soll $q(y)$ für Objekte y des Typs S wahr sein, wobei S ein Untertyp von T ist.“

Geometrische Form: T

Besitzt eine Fläche

Ellipse: $S \Rightarrow$ Geometrische Form

Besitzt eine Fläche

Besitzt einen Umfang

Kreis: $S \Rightarrow$ Ellipse

Besitzt eine Fläche

Besitzt einen Umfang

„Sei $q(x)$ eine beweisbare Eigenschaft von Objekten x des Typs T . Dann soll $q(y)$ für Objekte y des Typs S wahr sein, wobei S ein Untertyp von T ist.“

Geometrische Form: T

Besitzt eine Fläche

Ellipse: $S \Rightarrow$ Geometrische Form

Besitzt eine Fläche

Besitzt einen Umfang

Kann zweidimensional skaliert werden

Kreis: $S \Rightarrow$ Ellipse

Besitzt eine Fläche

Besitzt einen Umfang

„Sei $q(x)$ eine beweisbare Eigenschaft von Objekten x des Typs T . Dann soll $q(y)$ für Objekte y des Typs S wahr sein, wobei S ein Untertyp von T ist.“

Geometrische Form: T

Besitzt eine Fläche

Ellipse: $S \Rightarrow$ Geometrische Form

Besitzt eine Fläche

Besitzt einen Umfang

Kann zweidimensional skaliert werden

Kreis: $S \Rightarrow$ Ellipse

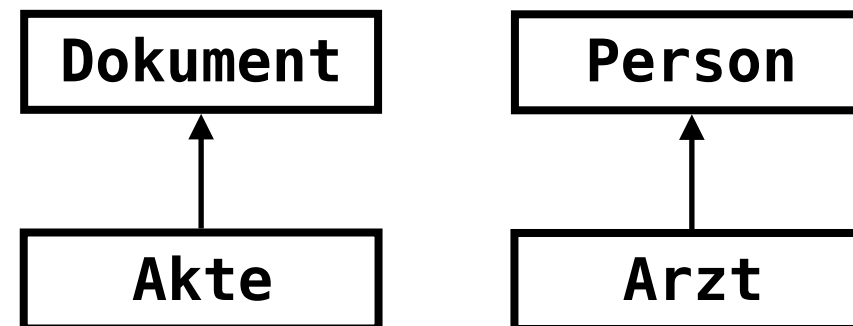
Besitzt eine Fläche

Besitzt einen Umfang

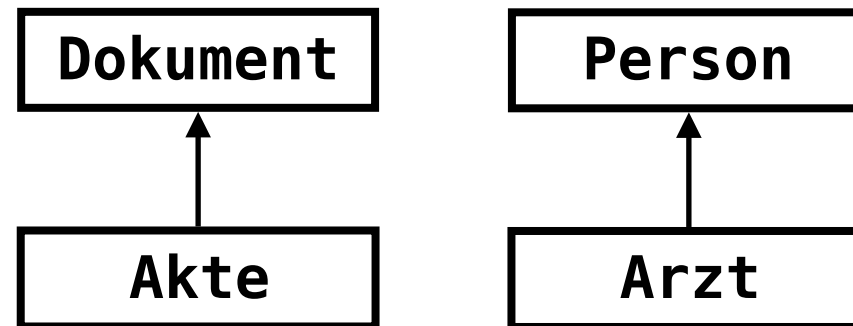


Kann eindimensional skaliert werden

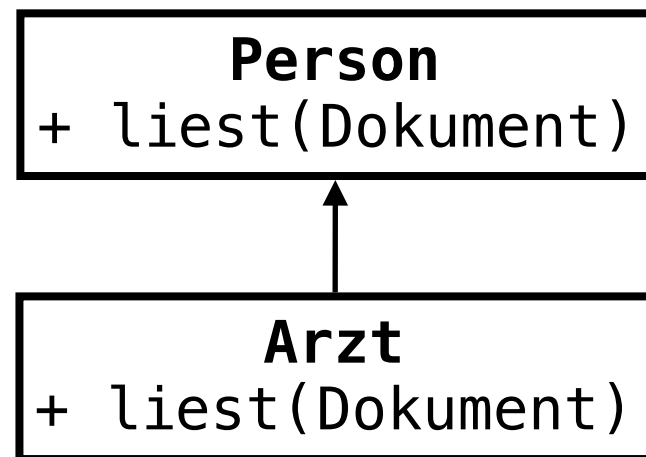
„Sei $q(x)$ eine beweisbare Eigenschaft von Objekten x des Typs T . Dann soll $q(y)$ für Objekte y des Typs S wahr sein, wobei S ein Untertyp von T ist.“



„Sei $q(x)$ eine beweisbare Eigenschaft von Objekten x des Typs T . Dann soll $q(y)$ für Objekte y des Typs S wahr sein, wobei S ein Untertyp von T ist.“

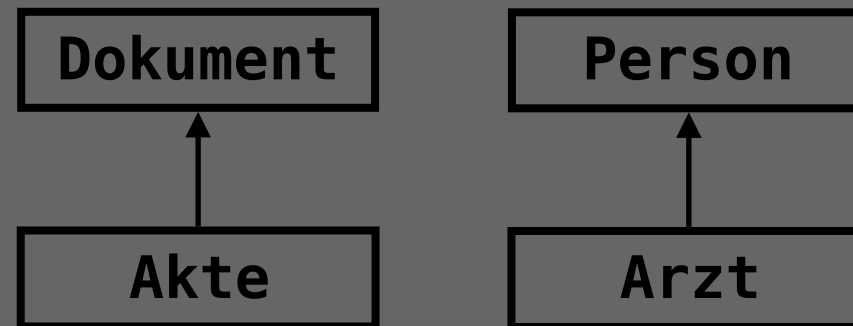


Invarianz

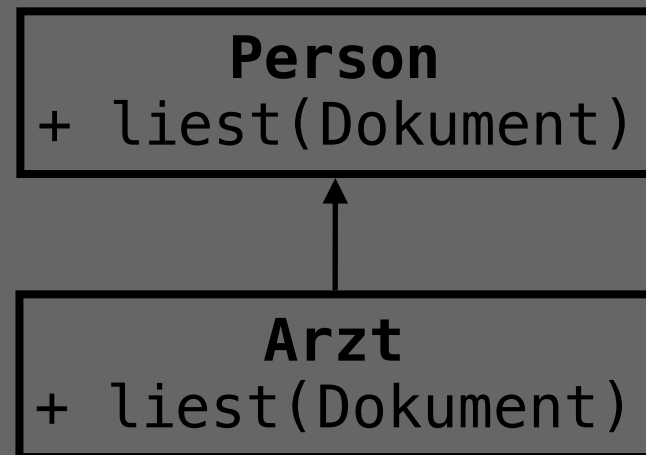


Person kann Dokument lesen
Arzt kann Dokument lesen

„Sei $q(x)$ eine beweisbare Eigenschaft von Objekten x des Typs T . Dann soll $q(y)$ für Objekte y des Typs S wahr sein, wobei S ein Untertyp von T ist.“

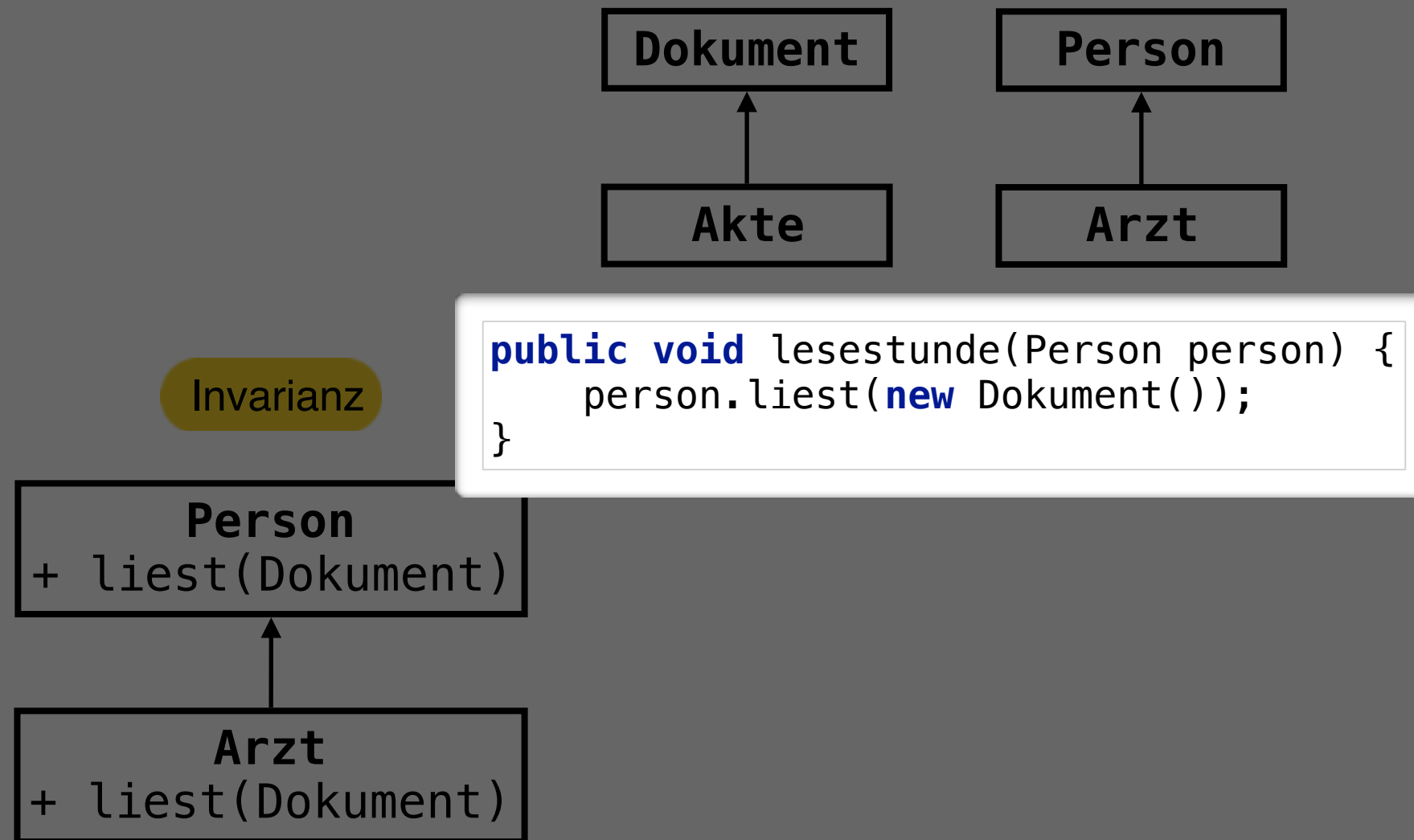


Invarianz



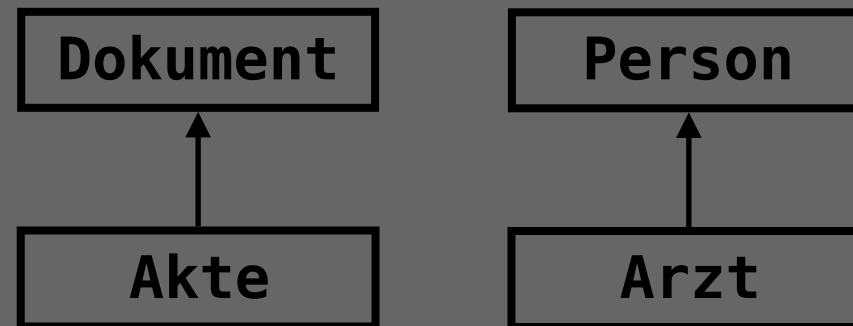
Person kann Dokument lesen
Arzt kann Dokument lesen

„Sei $q(x)$ eine beweisbare Eigenschaft von Objekten x des Typs T . Dann soll $q(y)$ für Objekte y des Typs S wahr sein, wobei S ein Untertyp von T ist.“

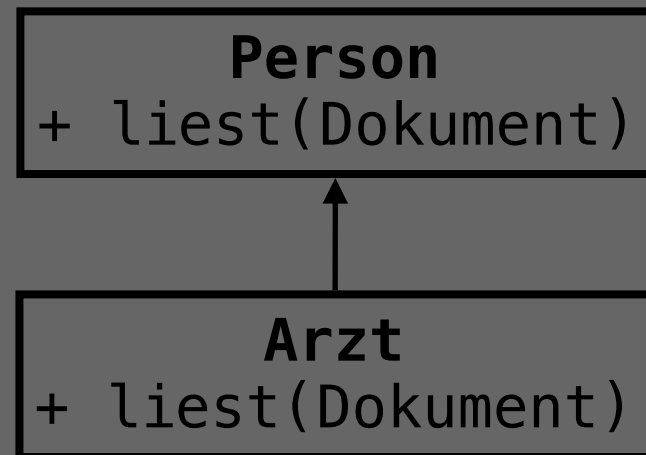


Person kann Dokument lesen
Arzt kann Dokument lesen

„Sei $q(x)$ eine beweisbare Eigenschaft von Objekten x des Typs T . Dann soll $q(y)$ für Objekte y des Typs S wahr sein, wobei S ein Untertyp von T ist.“

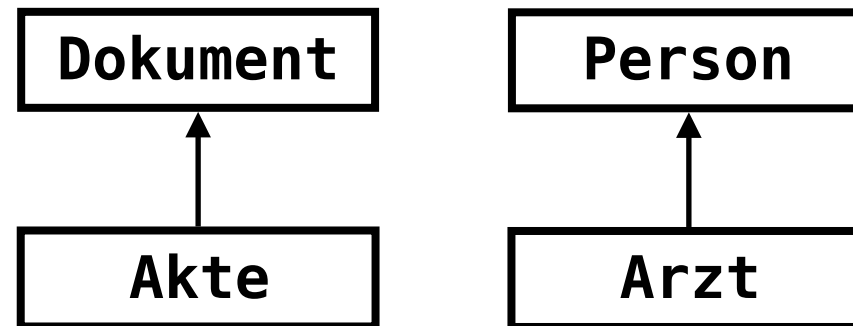


Invarianz

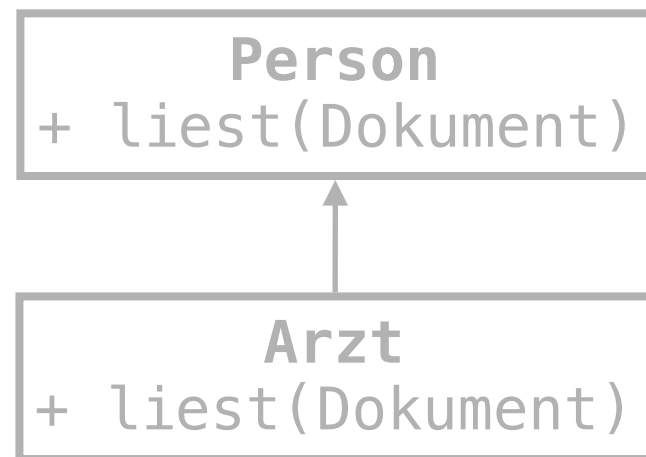


Person kann Dokument lesen
Arzt kann Dokument lesen

„Sei $q(x)$ eine beweisbare Eigenschaft von Objekten x des Typs T . Dann soll $q(y)$ für Objekte y des Typs S wahr sein, wobei S ein Untertyp von T ist.“

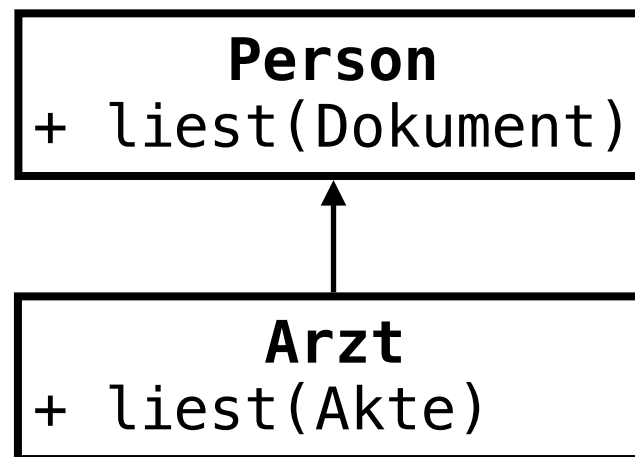


Invarianz



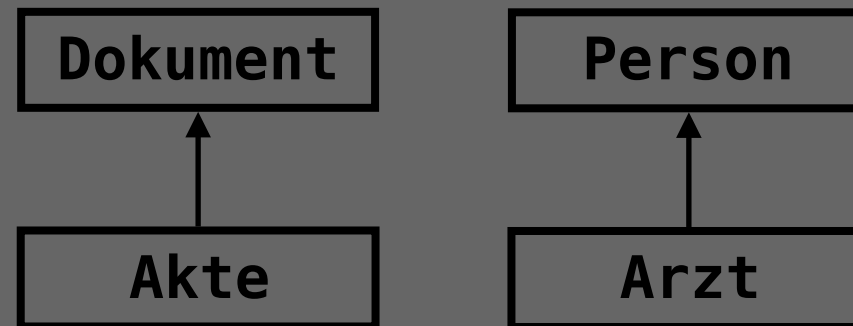
Person kann Dokument lesen
Arzt kann Dokument lesen

Kovarianz

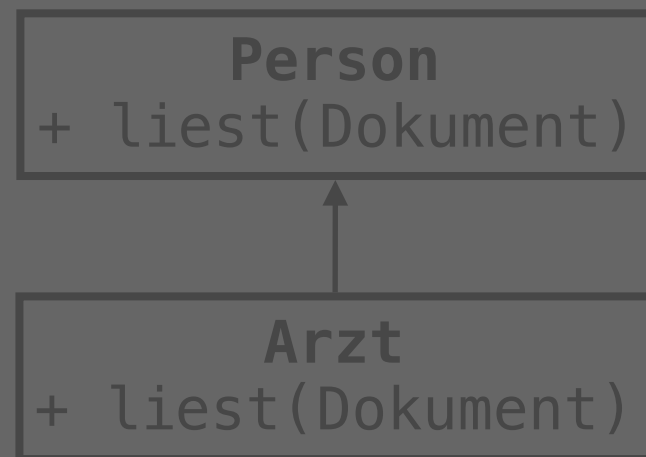


Person kann Dokument lesen
Arzt kann Akte lesen

„Sei $q(x)$ eine beweisbare Eigenschaft von Objekten x des Typs T . Dann soll $q(y)$ für Objekte y des Typs S wahr sein, wobei S ein Untertyp von T ist.“

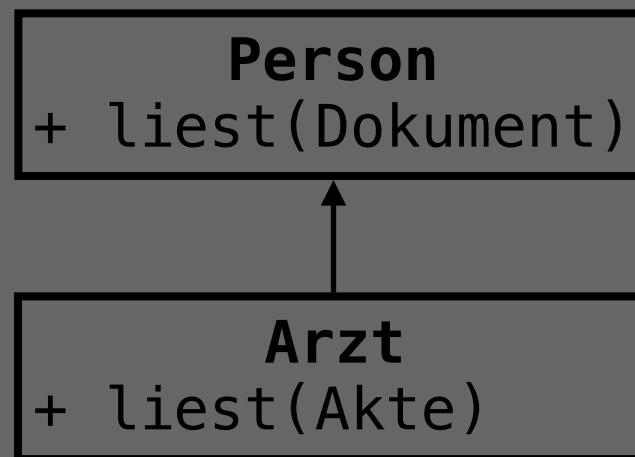


Invarianz



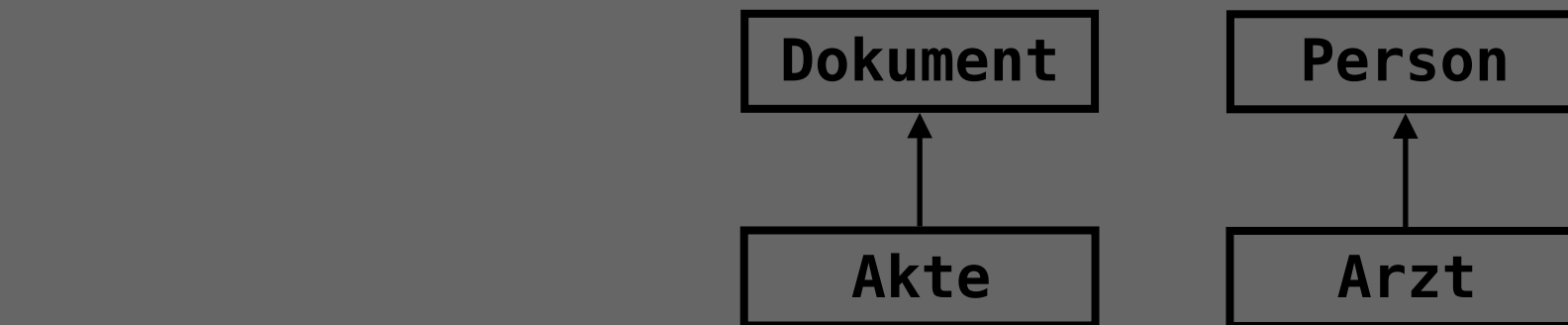
Person kann Dokument lesen
Arzt kann Dokument lesen

Kovarianz



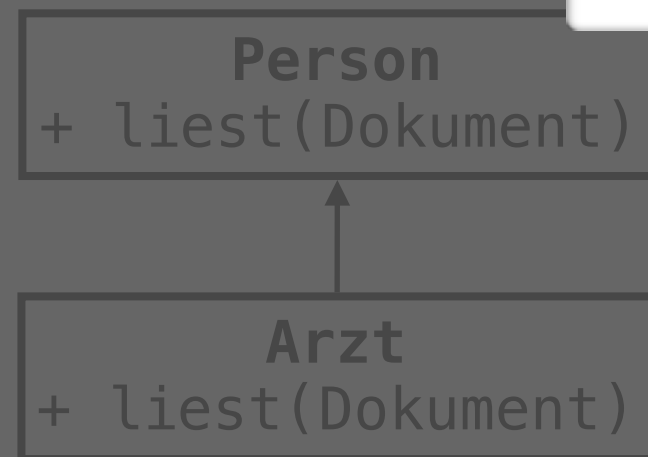
Person kann Dokument lesen
Arzt kann Akte lesen

„Sei $q(x)$ eine beweisbare Eigenschaft von Objekten x des Typs T . Dann soll $q(y)$ für Objekte y des Typs S wahr sein, wobei S ein Untertyp von T ist.“

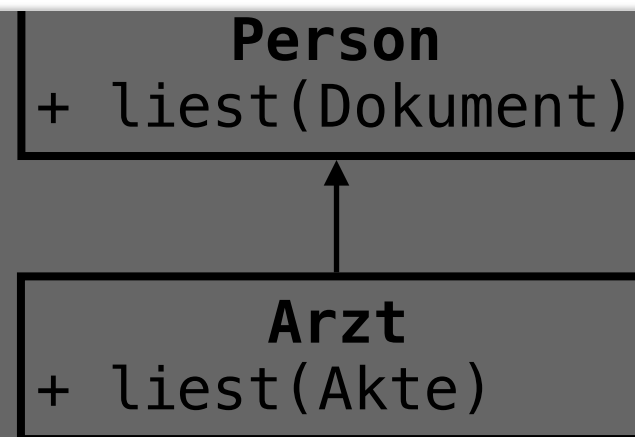


Invarianz

```
public void lesestunde(Person person) {
    person.liest(new Dokument());
}
```

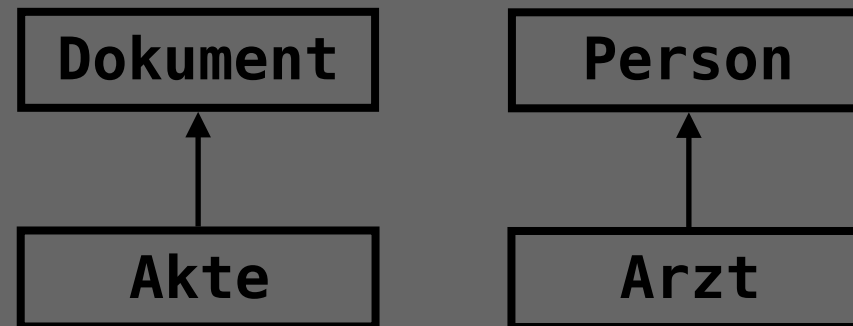


Person kann Dokument lesen
Arzt kann Dokument lesen

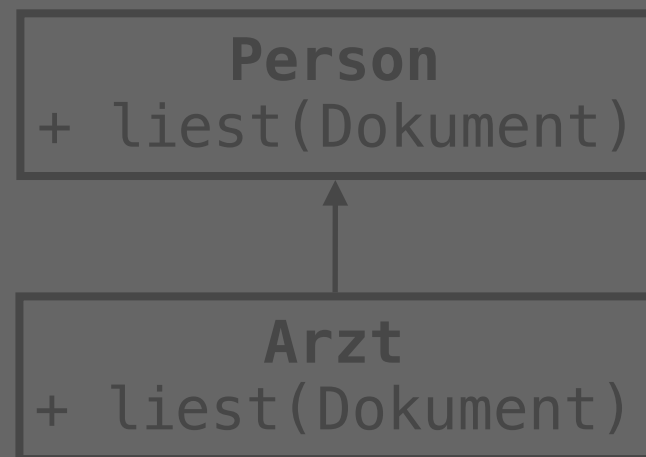


Person kann Dokument lesen
Arzt kann Akte lesen

„Sei $q(x)$ eine beweisbare Eigenschaft von Objekten x des Typs T . Dann soll $q(y)$ für Objekte y des Typs S wahr sein, wobei S ein Untertyp von T ist.“

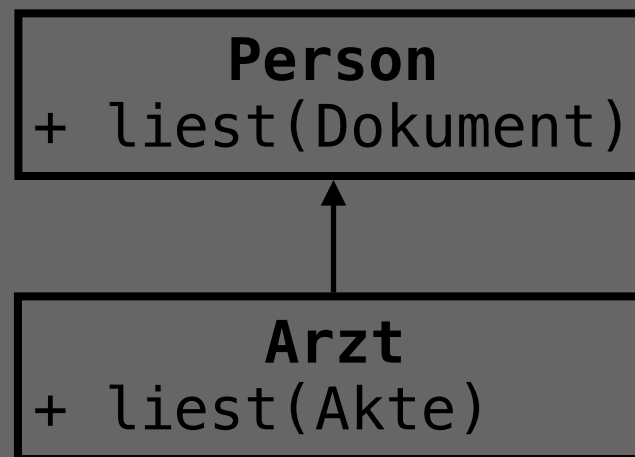


Invarianz



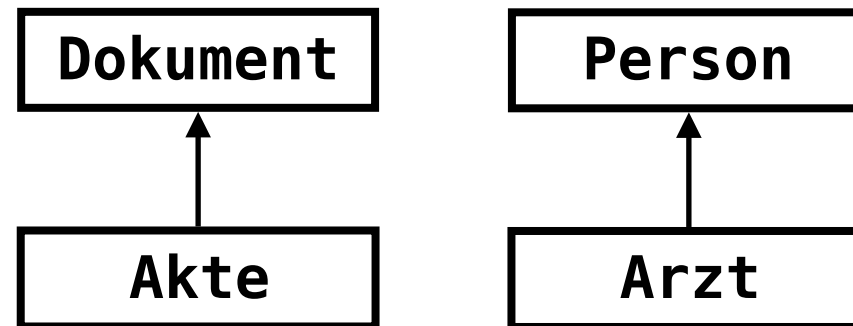
Person kann Dokument lesen
Arzt kann Dokument lesen

Kovarianz

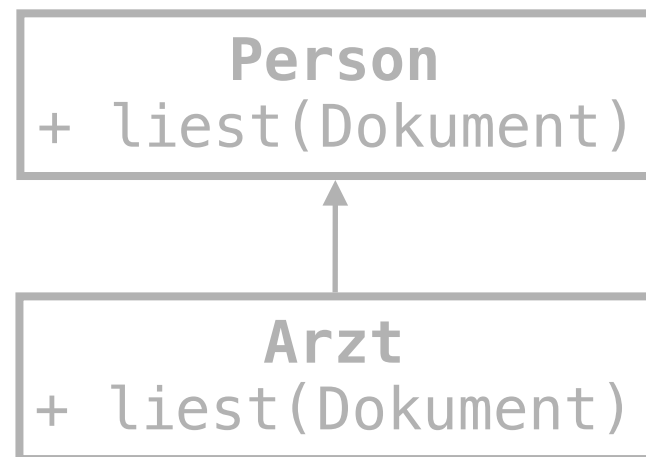


Person kann Dokument lesen
Arzt kann Akte lesen

„Sei $q(x)$ eine beweisbare Eigenschaft von Objekten x des Typs T . Dann soll $q(y)$ für Objekte y des Typs S wahr sein, wobei S ein Untertyp von T ist.“

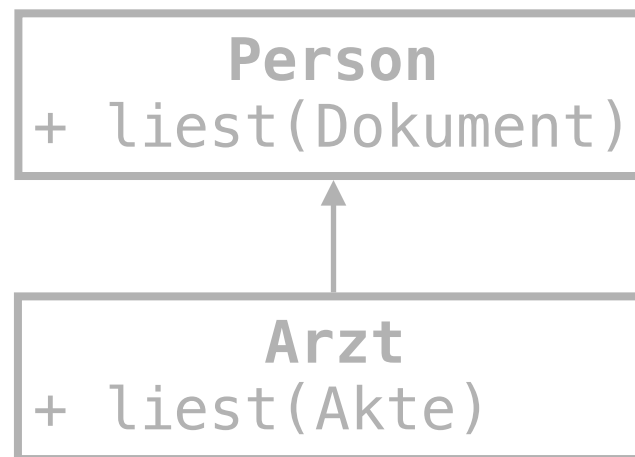


Invarianz



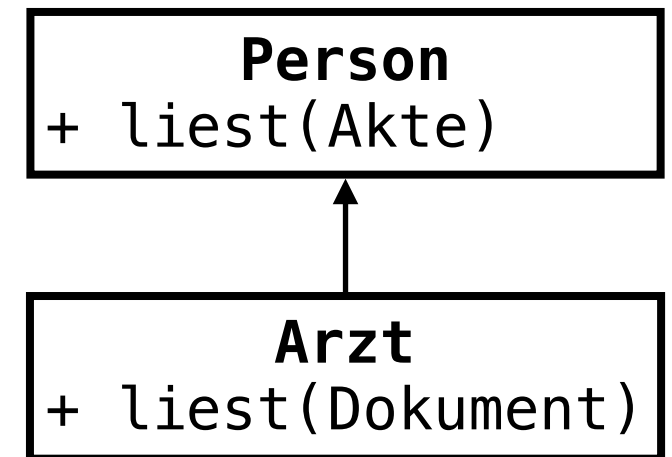
Person kann Dokument lesen
Arzt kann Dokument lesen

Kovarianz



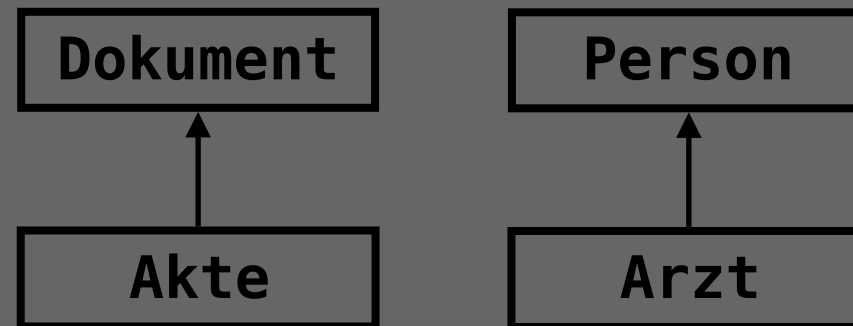
Person kann Dokument lesen
Arzt kann Akte lesen

Kontravarianz

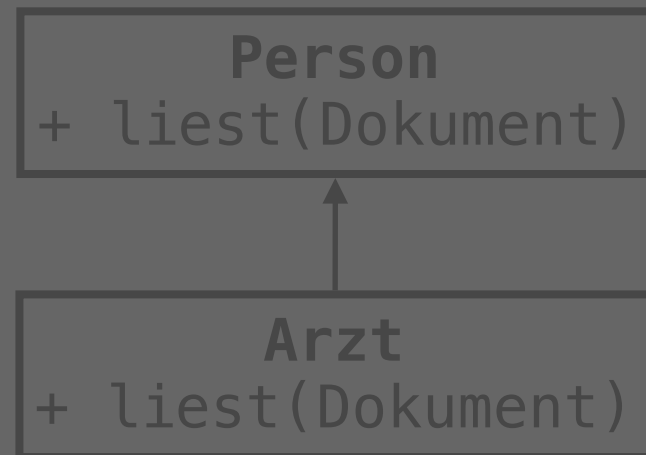


Person kann Akte lesen
Arzt kann Dokument lesen

„Sei $q(x)$ eine beweisbare Eigenschaft von Objekten x des Typs T . Dann soll $q(y)$ für Objekte y des Typs S wahr sein, wobei S ein Untertyp von T ist.“

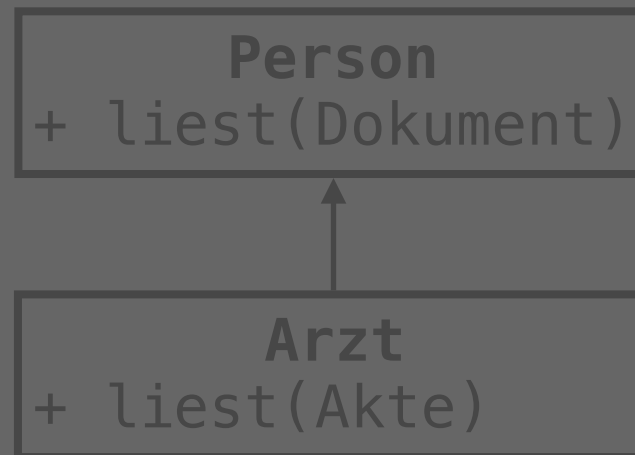


Invarianz



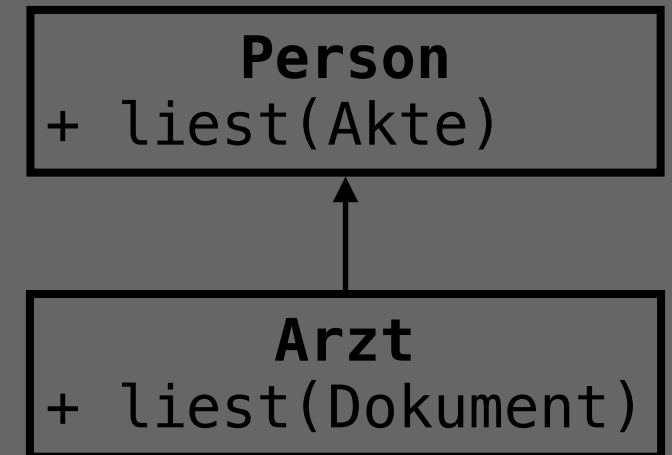
Person kann Dokument lesen
Arzt kann Dokument lesen

Kovarianz



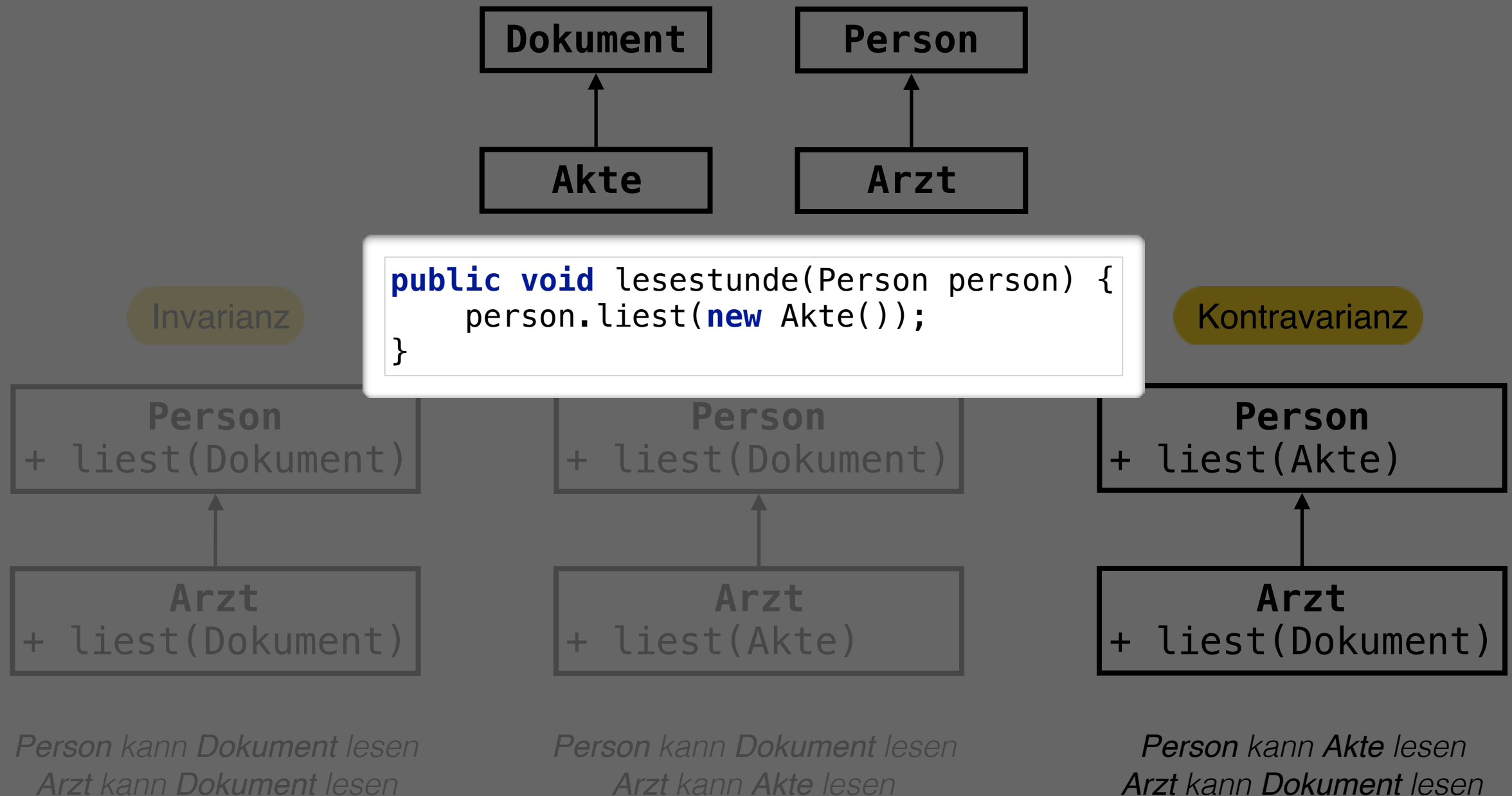
Person kann Dokument lesen
Arzt kann Akte lesen

Kontravarianz

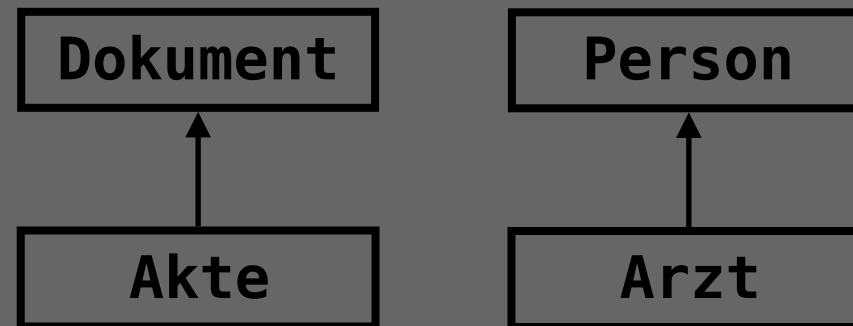


Person kann Akte lesen
Arzt kann Dokument lesen

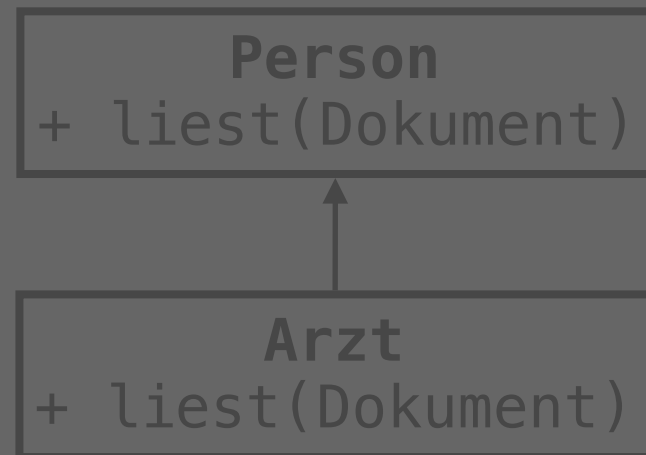
„Sei $q(x)$ eine beweisbare Eigenschaft von Objekten x des Typs T . Dann soll $q(y)$ für Objekte y des Typs S wahr sein, wobei S ein Untertyp von T ist.“



„Sei $q(x)$ eine beweisbare Eigenschaft von Objekten x des Typs T . Dann soll $q(y)$ für Objekte y des Typs S wahr sein, wobei S ein Untertyp von T ist.“

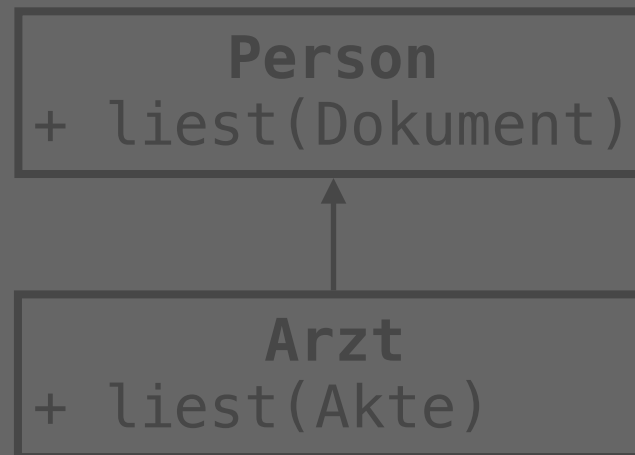


Invarianz



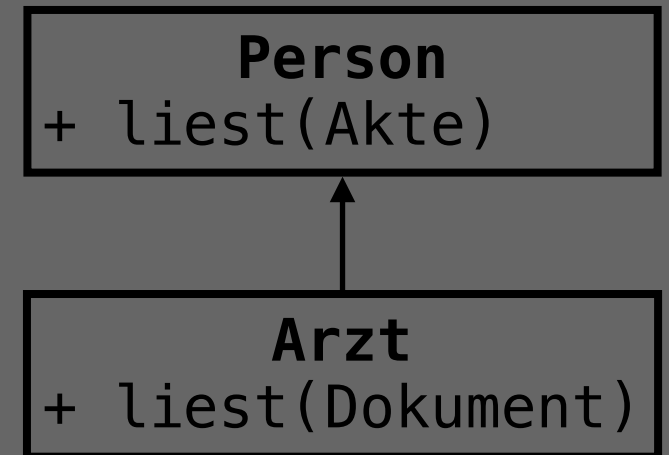
Person kann Dokument lesen
Arzt kann Dokument lesen

Kovarianz



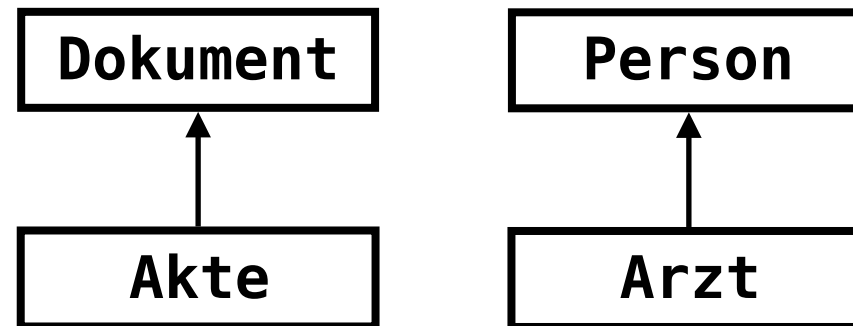
Person kann Dokument lesen
Arzt kann Akte lesen

Kontravarianz

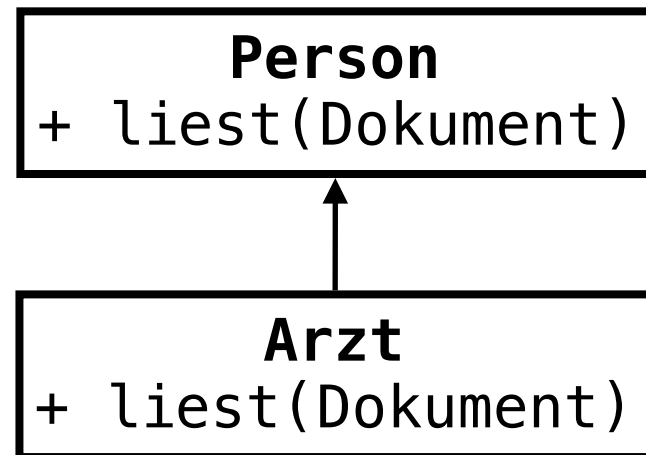


Person kann Akte lesen
Arzt kann Dokument lesen

„Sei $q(x)$ eine beweisbare Eigenschaft von Objekten x des Typs T . Dann soll $q(y)$ für Objekte y des Typs S wahr sein, wobei S ein Untertyp von T ist.“

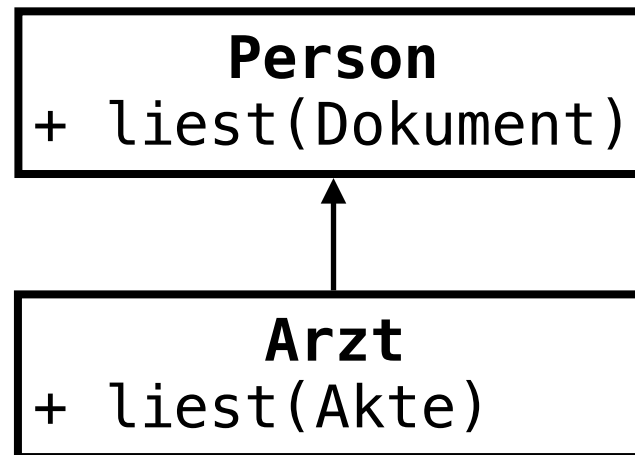


Invarianz



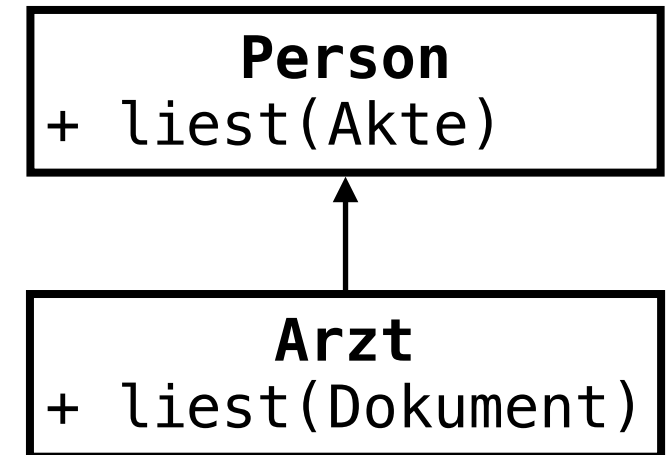
Person kann Dokument lesen
Arzt kann Dokument lesen

Kovarianz



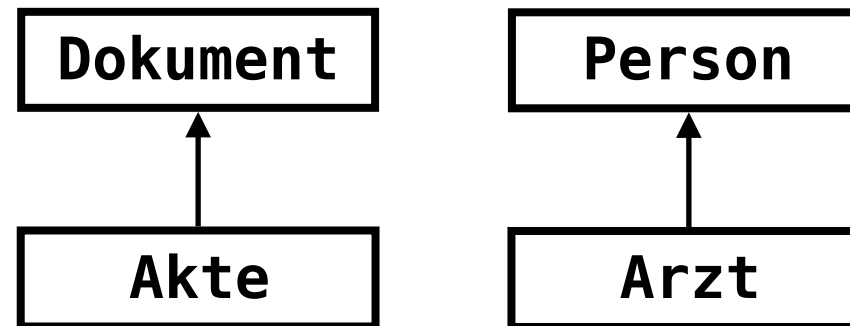
Person kann Dokument lesen
Arzt kann Akte lesen

Kontravarianz

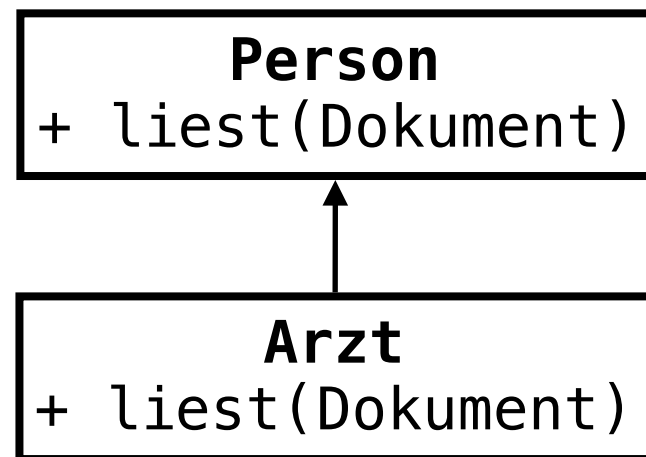


Person kann Akte lesen
Arzt kann Dokument lesen

„Sei $q(x)$ eine beweisbare Eigenschaft von Objekten x des Typs T . Dann soll $q(y)$ für Objekte y des Typs S wahr sein, wobei S ein Untertyp von T ist.“



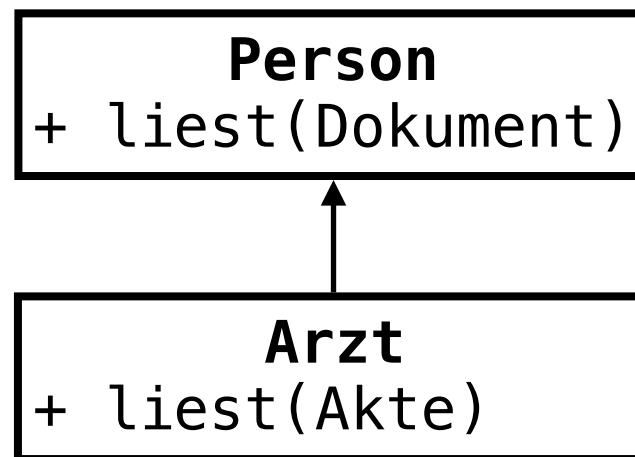
Invarianz



Person kann Dokument lesen
Arzt kann Dokument lesen

*(Person)Arzt kann
 Dokument lesen*

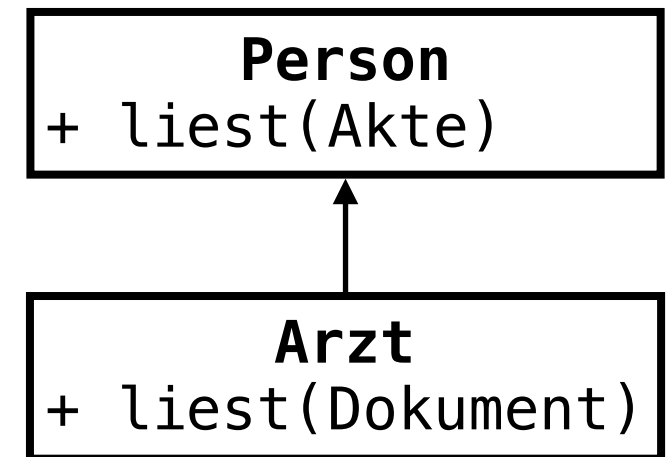
Kovarianz



Person kann Dokument lesen
Arzt kann Akte lesen

 *(Person)Arzt kann nicht
 lesen*

Kontravarianz



Person kann Akte lesen
Arzt kann Dokument lesen

*(Person)Arzt kann auch
 Akte lesen*

„Sei $q(x)$ eine beweisbare Eigenschaft von Objekten x des Typs T . Dann soll $q(y)$ für Objekte y des Typs S wahr sein, wobei S ein Untertyp von T ist.“

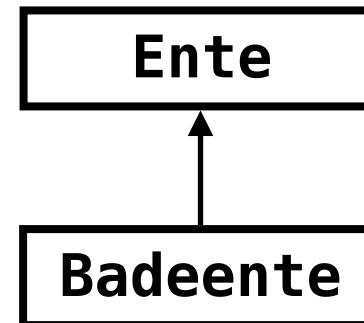
Vorbedingung

*Die übergebenen Parameter **abstahieren** oder gleichen*

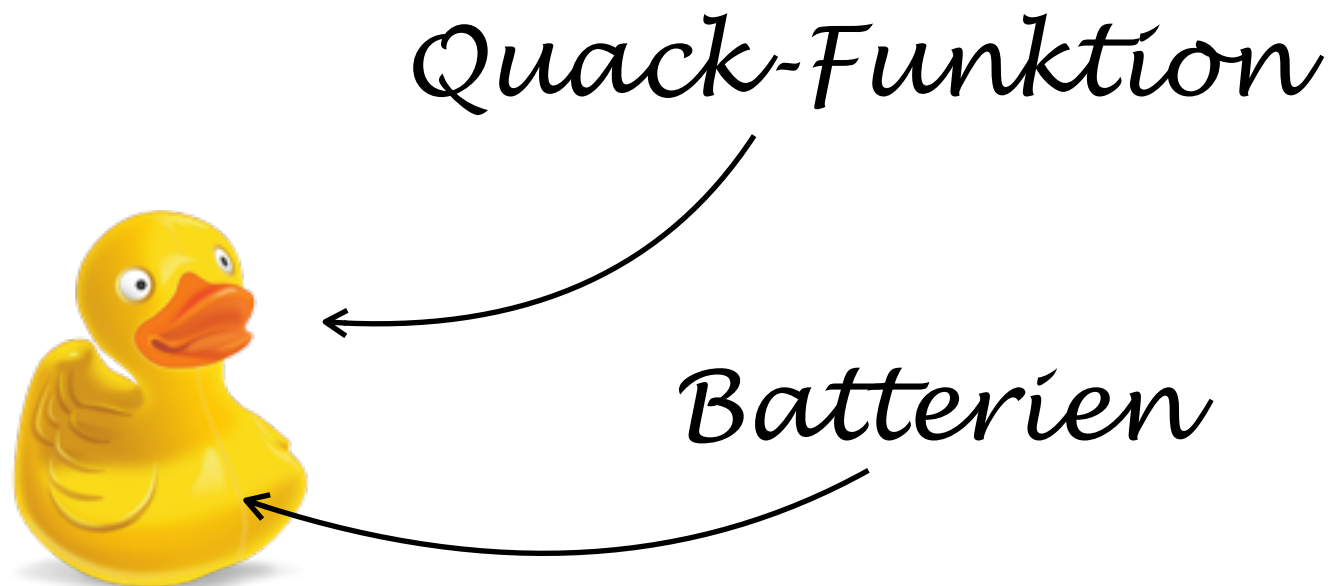
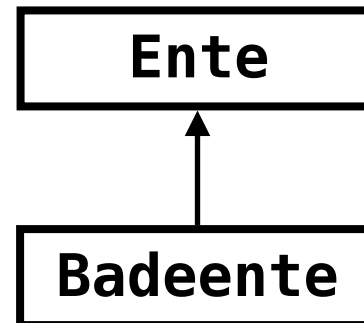
Nachbedingung

*Der Rückgabewert **spezialisiert** oder gleicht*

„Sei $q(x)$ eine beweisbare Eigenschaft von Objekten x des Typs T . Dann soll $q(y)$ für Objekte y des Typs S wahr sein, wobei S ein Untertyp von T ist.“



„Sei $q(x)$ eine beweisbare Eigenschaft von Objekten x des Typs T . Dann soll $q(y)$ für Objekte y des Typs S wahr sein, wobei S ein Untertyp von T ist.“



„Sei $q(x)$ eine beweisbare Eigenschaft von Objekten x des Typs T . Dann soll $q(y)$ für Objekte y des Typs S wahr sein, wobei S ein Untertyp von T ist.“

Ente

```
public class Duck {  
    public void quack() {  
        System.out.println("Quack!");  
    }  
}
```

Badeente

```
public class RubberDuck extends Duck {  
    private Battery battery;  
  
    @Override  
    public void quack() {  
        if(!battery.isCharged()) {  
            throw new NotEnoughPowerException();  
        }  
  
        System.out.println("Quack!");  
    }  
}
```



„Sei $q(x)$ eine beweisbare Eigenschaft von Objekten x des Typs T . Dann soll $q(y)$ für Objekte y des Typs S wahr sein, wobei S ein Untertyp von T ist.“

Ente

```
public class Duck {  
    public void quack() {  
        System.out.println("Quack!");  
    }  
}
```

Badeente

```
public class RubberDuck extends Duck {  
    private Battery battery;  
    @Override  
    public void quack() {  
        if(!battery.isCharged()) {  
            throw new NotEnoughPowerException();  
        }  
        System.out.println("Quack!");  
    }  
}
```



Interface Segregation Principle

Mehr Abstraktion mit weniger Details

Interface Segregation Principle

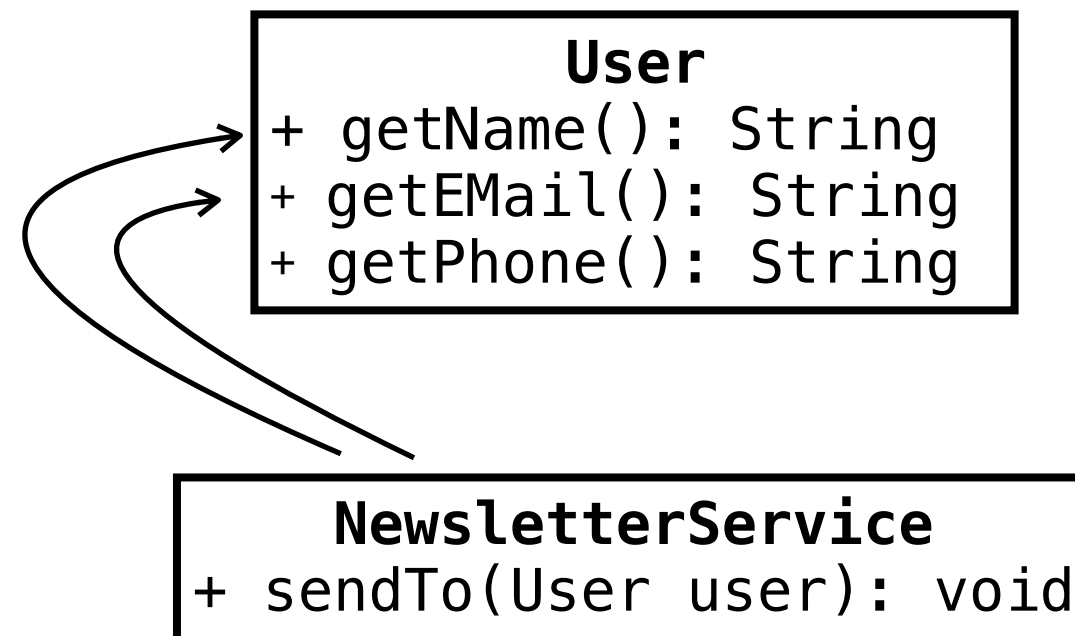
Mehr Abstraktion mit weniger Details

```
User  
+ getName(): String  
+ getEmail(): String  
+ getPhone(): String
```

```
NewsletterService  
+ sendTo(User user): void
```

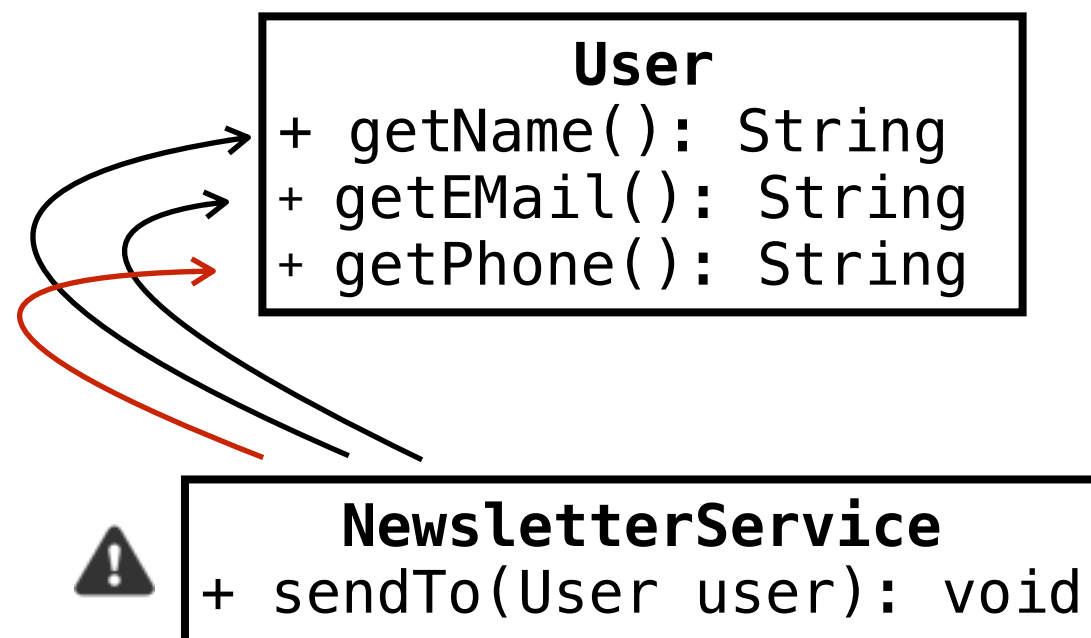
Interface Segregation Principle

Mehr Abstraktion mit weniger Details



Interface Segregation Principle

Mehr Abstraktion mit weniger Details



Interface Segregation Principle

Mehr Abstraktion mit weniger Details

```
User  
+ getName(): String  
+ getEmail(): String  
+ getPhone(): String
```

```
<<NewsletterReceiver>>  
+ getName(): String  
+ getEmail(): String
```

```
NewsletterService  
+ sendTo(User user): void
```

Interface Segregation Principle

Mehr Abstraktion mit weniger Details

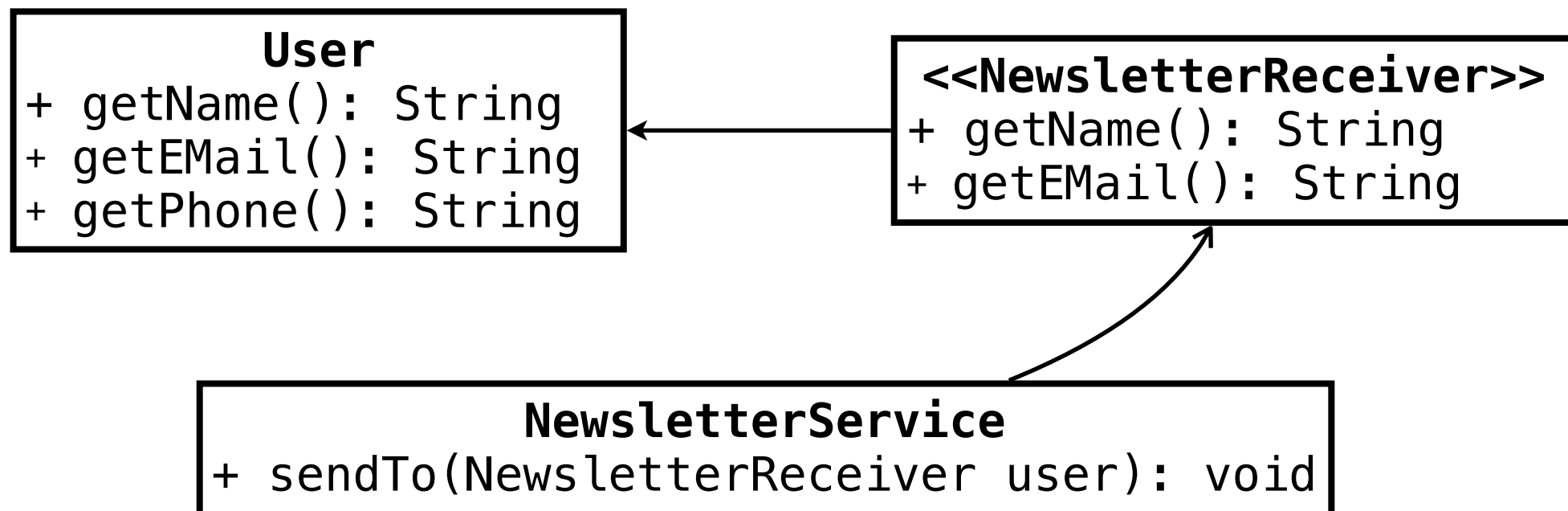
```
User  
+ getName(): String  
+ getEmail(): String  
+ getPhone(): String
```

```
<<NewsletterReceiver>>  
+ getName(): String  
+ getEmail(): String
```

```
NewsletterService  
+ sendTo(NewsletterReceiver user): void
```


Interface Segregation Principle

Mehr Abstraktion mit weniger Details



Interface Segregation Principle

Mehr Abstraktion mit weniger Details

```
public interface WorkerInterface {  
    void work();  
    void sleep();  
}
```

Interface Segregation Principle

Mehr Abstraktion mit weniger Details

```
public interface WorkerInterface {  
  
    void work();  
  
    void sleep();  
  
}
```

```
public class HumanWorker implements WorkerInterface {  
  
    @Override  
    public void work() {  
        // ... arbeiten  
    }  
  
    @Override  
    public void sleep() {  
        // ... schlafen  
    }  
  
}
```

```
public class RobotWorker implements WorkerInterface {  
  
    @Override  
    public void work() {  
        // ... arbeiten  
    }  
  
    @Override  
    public void sleep() {  
    }  
  
}
```

Interface Segregation Principle

Mehr Abstraktion mit weniger Details

```
public interface WorkerInterface {  
  
    void work();  
  
    void sleep();  
  
}
```

```
public class HumanWorker implements WorkerInterface {  
  
    @Override  
    public void work() {  
        // ... arbeiten  
    }  
  
    @Override  
    public void sleep() {  
        // ... schlafen  
    }  
  
}
```

```
public class RobotWorker implements WorkerInterface {  
  
    @Override  
    public void work() {  
        // ... arbeiten  
    }  
  
    @Override  
    public void sleep() {  
    }  
  
}
```

Interface Segregation Principle

Mehr Abstraktion mit weniger Details

```
public interface WorkerInterface {  
  
    void work();  
  
    void sleep();  
  
}
```

```
public class HumanWorker implements WorkerInterface {  
  
    @Override  
    public void work() {  
        // ... arbeiten  
    }  
  
    @Override  
    public void sleep() {  
        // ... schlafen  
    }  
  
}
```

```
public class RobotWorker implements WorkerInterface {  
  
    @Override  
    public void work() {  
        // ... arbeiten  
    }  
  
    @Override  
    public void sleep() {  
    }  
  
}
```

ungenutzte Funktion

Interface Segregation Principle

Mehr Abstraktion mit weniger Details



Plane sorgfältig



Adapter-Pattern

Dependency Inversion Principle

Sei nicht so detailverliebt, nutze Schnittstellen

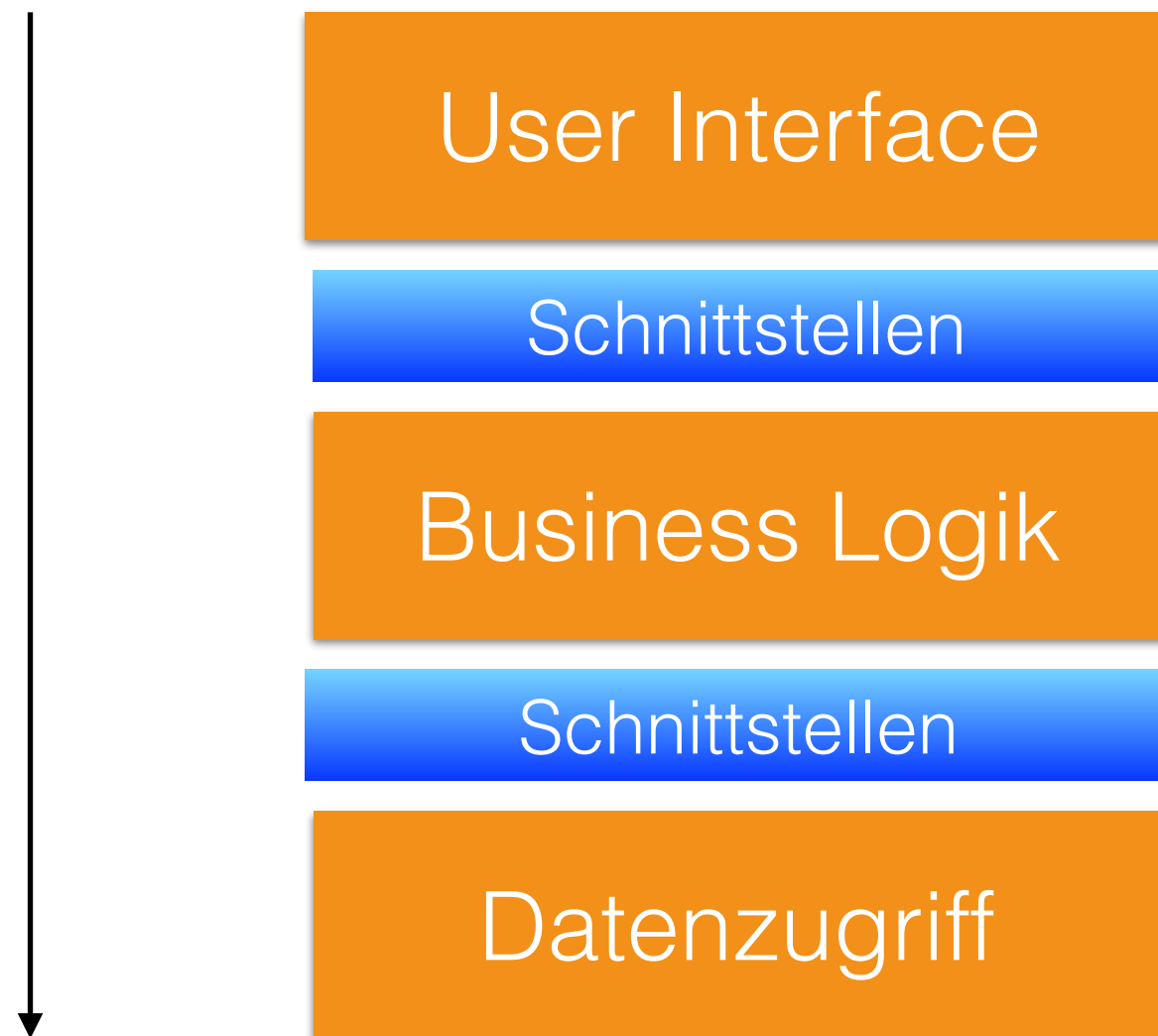
Dependency Inversion Principle

Sei nicht so detailverliebt, nutze Schnittstellen



Dependency Inversion Principle

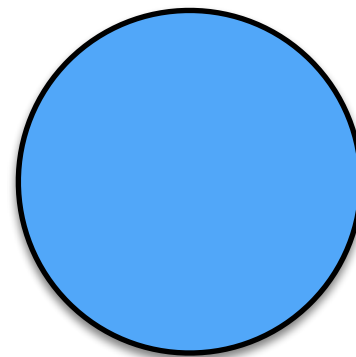
Sei nicht so detailverliebt, nutze Schnittstellen



Dependency Inversion Principle

Sei nicht so detailverliebt, nutze Schnittstellen

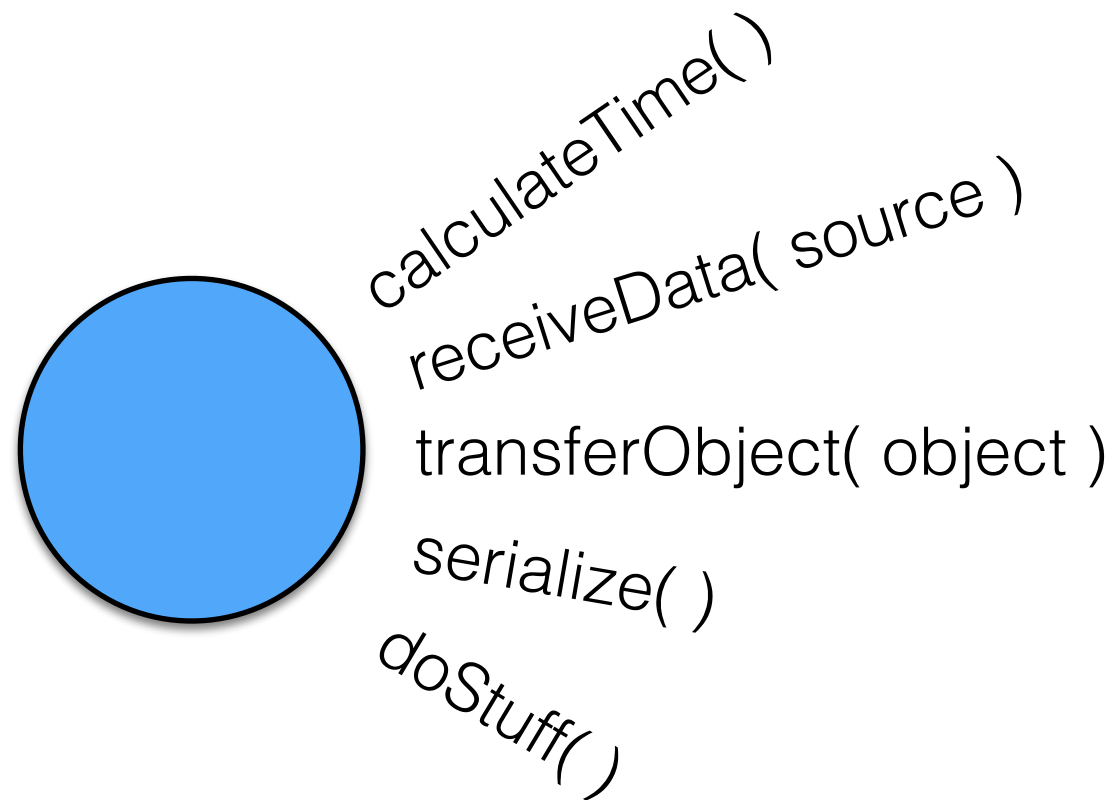
Test



Dependency Inversion Principle

Sei nicht so detailverliebt, nutze Schnittstellen

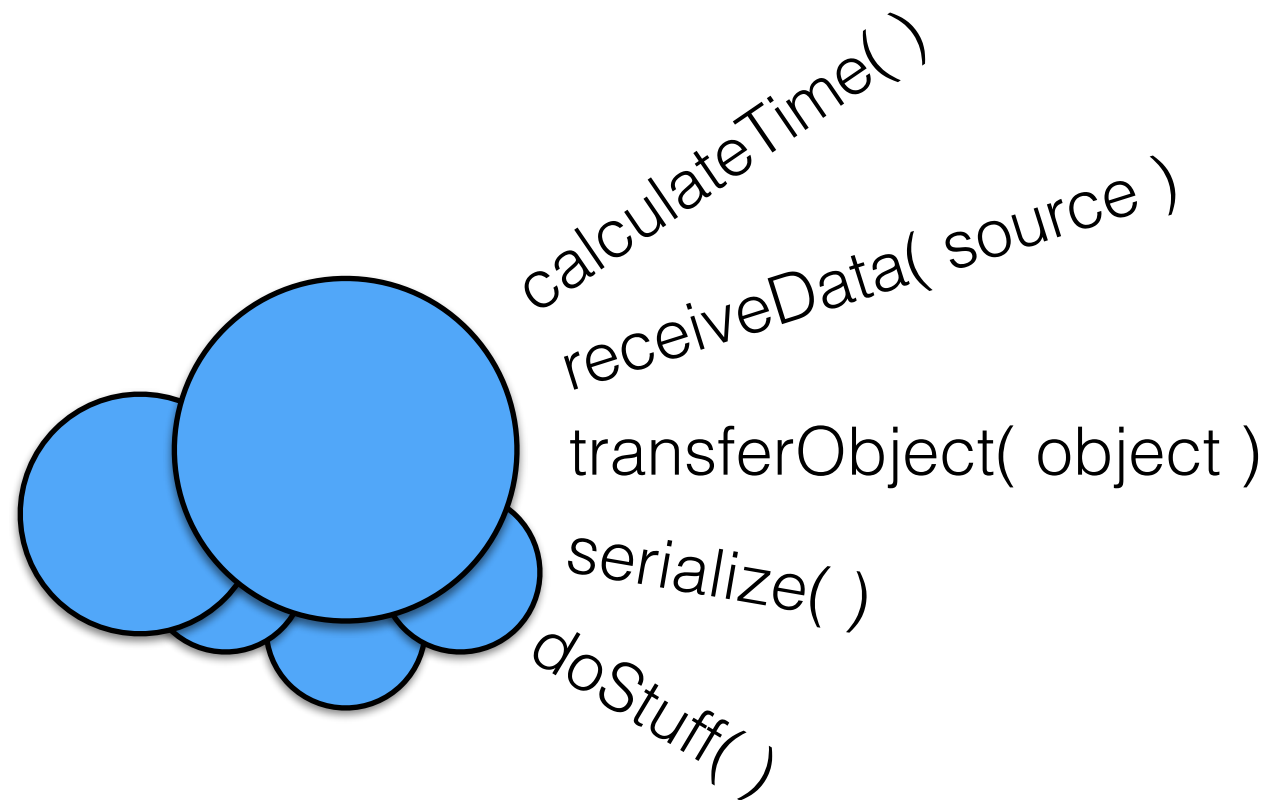
Test



Dependency Inversion Principle

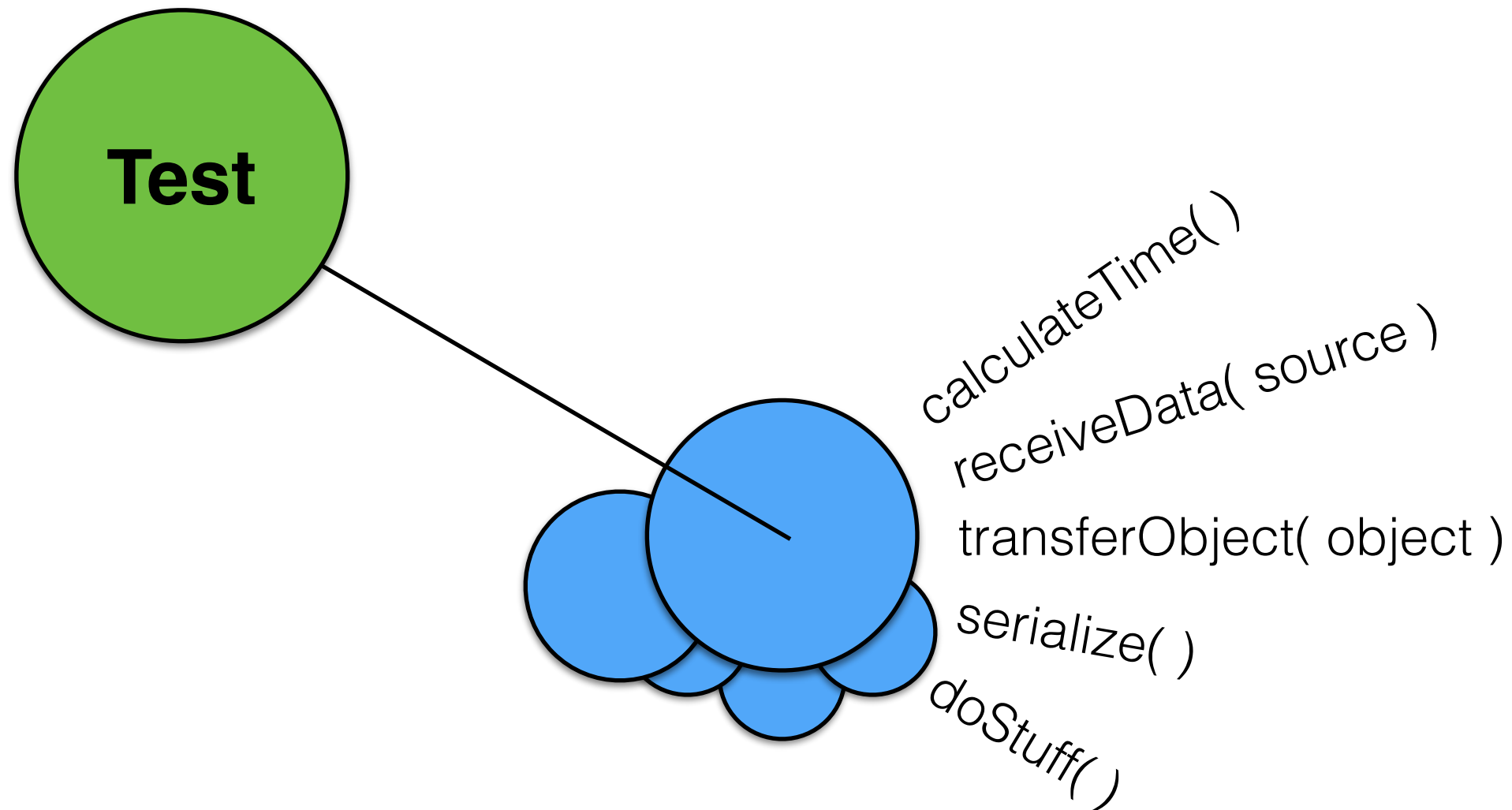
Sei nicht so detailverliebt, nutze Schnittstellen

Test



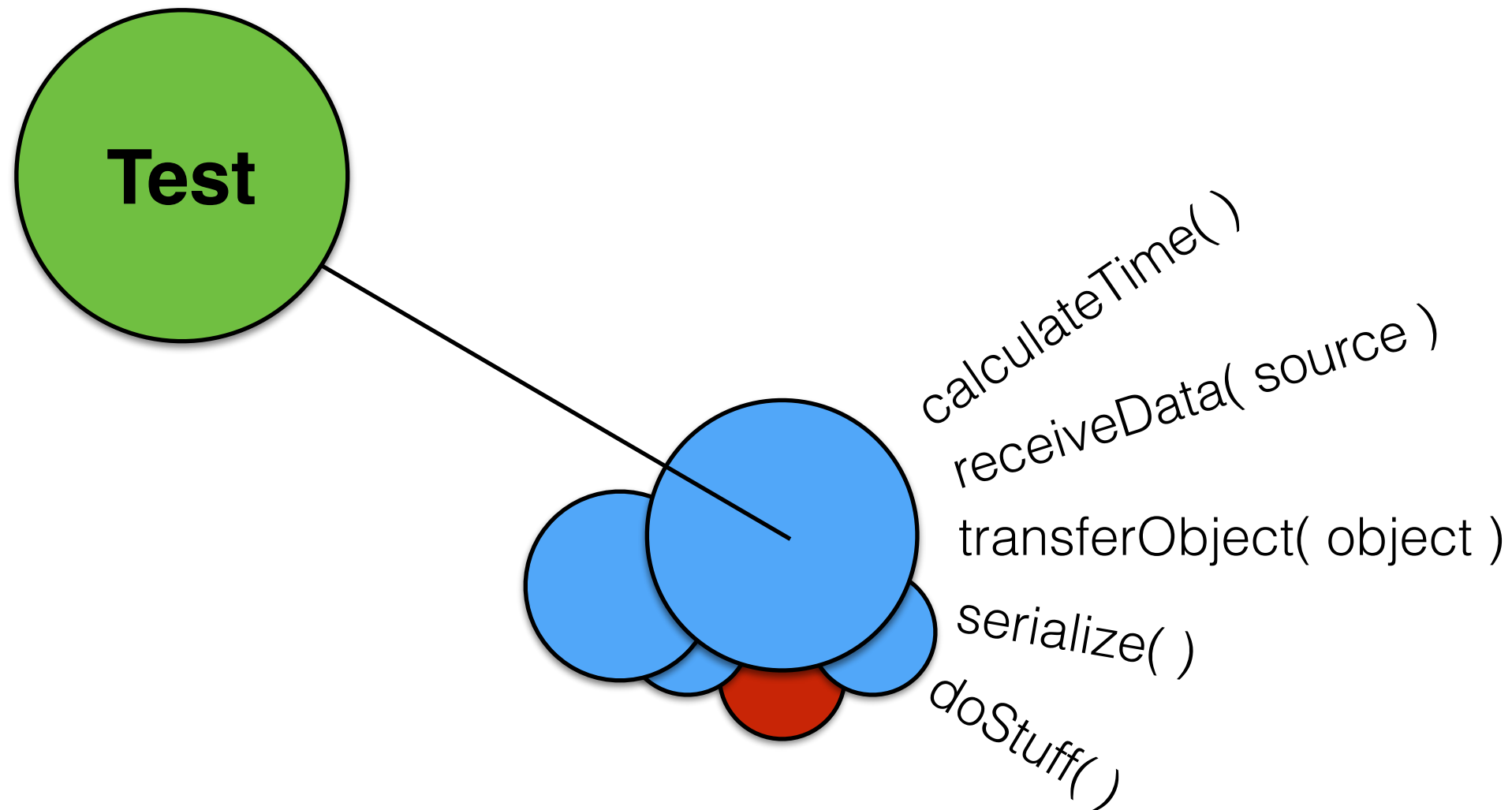
Dependency Inversion Principle

Sei nicht so detailverliebt, nutze Schnittstellen



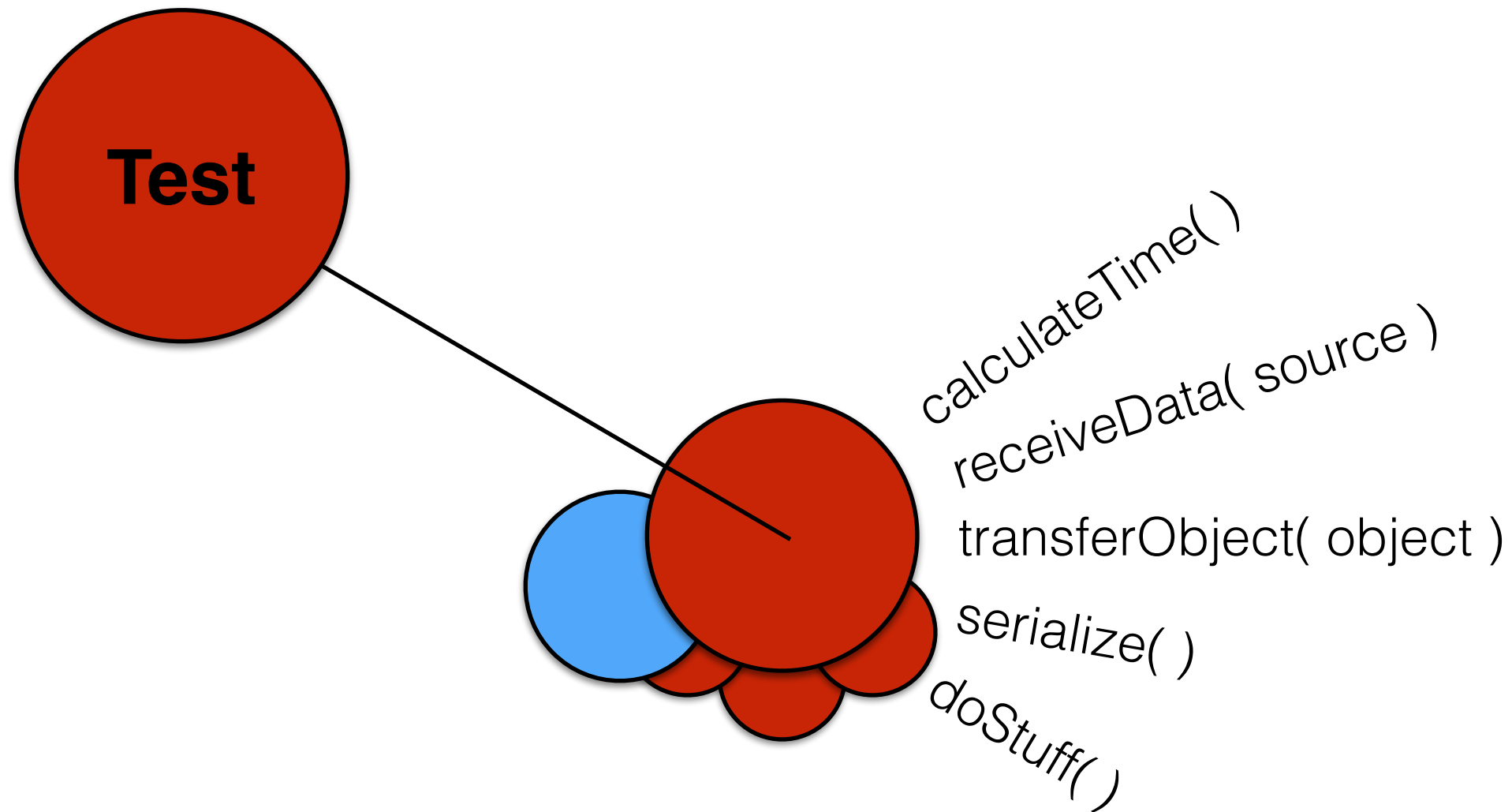
Dependency Inversion Principle

Sei nicht so detailverliebt, nutze Schnittstellen



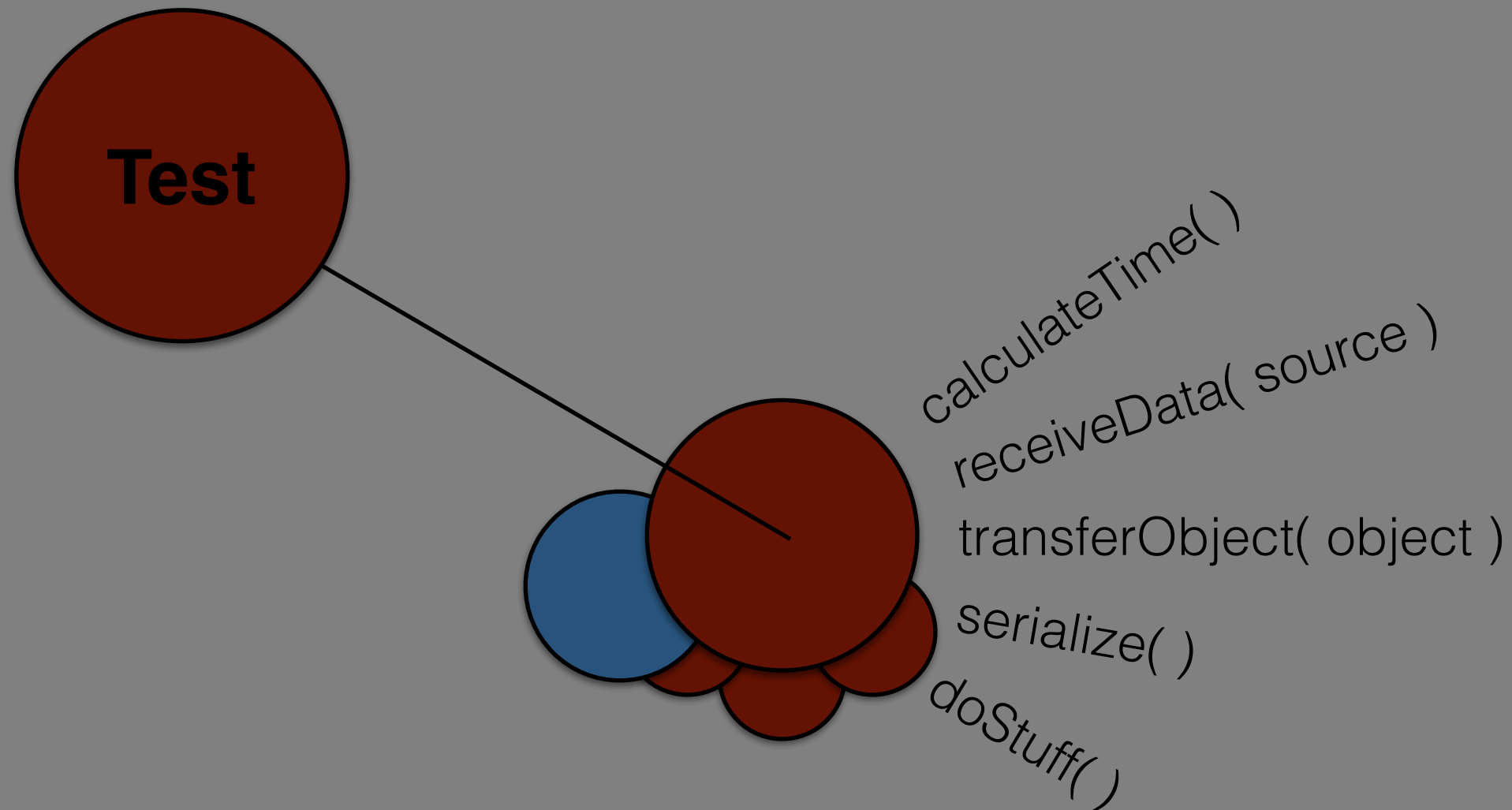
Dependency Inversion Principle

Sei nicht so detailverliebt, nutze Schnittstellen



Dependency Inversion Principle

Sei nicht so detailverliebt, nutze Schnittstellen



Dependency Inversion Principle

Sei nicht so detailverliebt, nutze Schnittstellen

```
public class ConfigurationWizard {  
  
    private WizardView view;  
    private DatabaseConnection database;  
    private EntityConfigurator configurator;  
    private Entity entity;  
  
    public ConfigurationWizard() {  
        database = DatabaseConnection.getInstance();  
        entity = database.getWizardEntityForSession(Session.getInstance());  
        configurator = EntityConfigurator.get(entity);  
        view = configurator.initView(new WizardView(entity));  
        populate();  
        initEvents();  
    }  
  
    // ... Rest der Klasse  
  
}
```

Dependency Inversion Principle

Sei nicht so detailverliebt, nutze Schnittstellen



Dependency Inversion Principle

Sei nicht so detailverliebt, nutze Schnittstellen

```
class Lampe {  
    private boolean leuchtet = false;  
  
    public void anschalten() {  
        leuchtet = true;  
    }  
  
    public void ausschalten() {  
        leuchtet = false;  
    }  
}
```

```
class Schalter {  
    private Lampe lampe;  
    private boolean gedrueckt;  
  
    public Schalter(Lampe lampe) {  
        this.lampe = lampe;  
    }  
  
    public void drueckeSchalter() {  
        gedrueckt = !gedrueckt;  
        if(gedrueckt) {  
            lampe.anschalten();  
        } else {  
            lampe.ausschalten();  
        }  
    }  
}
```

Dependency Inversion Principle

Sei nicht so detailverliebt, nutze Schnittstellen

```
class Lampe {  
    private boolean leuchtet = false;  
  
    public void anschalten() {  
        leuchtet = true;  
    }  
  
    public void ausschalten() {  
        leuchtet = false;  
    }  
}
```

```
class Schalter {  
    private Lampe lampe;  
    private boolean gedrueckt;  
  
    public Schalter(Lampe lampe) {  
        this.lampe = lampe;  
    }  
  
    public void drueckeSchalter() {  
        gedrueckt = !gedrueckt;  
        if(gedrueckt) {  
            lampe.anschalten();  
        } else {  
            lampe.ausschalten();  
        }  
    }  
}
```

Dependency Inversion Principle

Sei nicht so detailverliebt, nutze Schnittstellen

```
class Lampe {  
    private boolean leuchtet = false;  
  
    public void anschalten() {  
        leuchtet = true;  
    }  
  
    public void ausschalten() {  
        leuchtet = false;  
    }  
}
```

```
class Schalter {  
    private Lampe lampe;  
    private boolean gedrueckt;  
  
    public Schalter(Lampe lampe) {  
        this.lampe = lampe;  
    }  
  
    public void drueckeSchalter() {  
        gedrueckt = !gedrueckt;  
        if(gedrueckt) {  
            lampe.anschalten();  
        } else {  
            lampe.ausschalten();  
        }  
    }  
}
```

Dependency Inversion Principle

Sei nicht so detailverliebt, nutze Schnittstellen

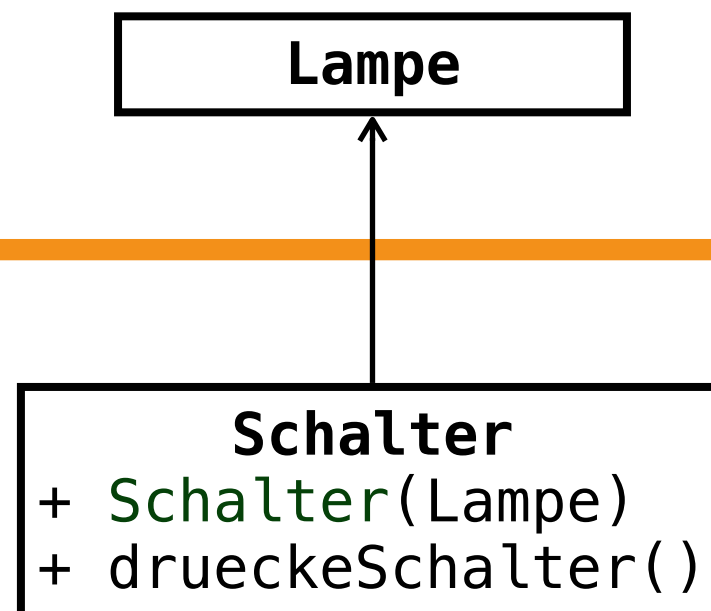
```
class Lampe {  
    private boolean leuchtet = false;  
  
    public void anschalten() {  
        leuchtet = true;  
    }  
  
    public void ausschalten() {  
        leuchtet = false;  
    }  
}
```

Schalter ist abhängig von Lampe

```
class Schalter {  
    private Lampe lampe;  
    private boolean gedrueckt;  
  
    public Schalter(Lampe lampe) {  
        this.lampe = lampe;  
    }  
  
    public void drueckeSchalter() {  
        gedrueckt = !gedrueckt;  
        if(gedrueckt) {  
            lampe.anschalten();  
        } else {  
            lampe.ausschalten();  
        }  
    }  
}
```

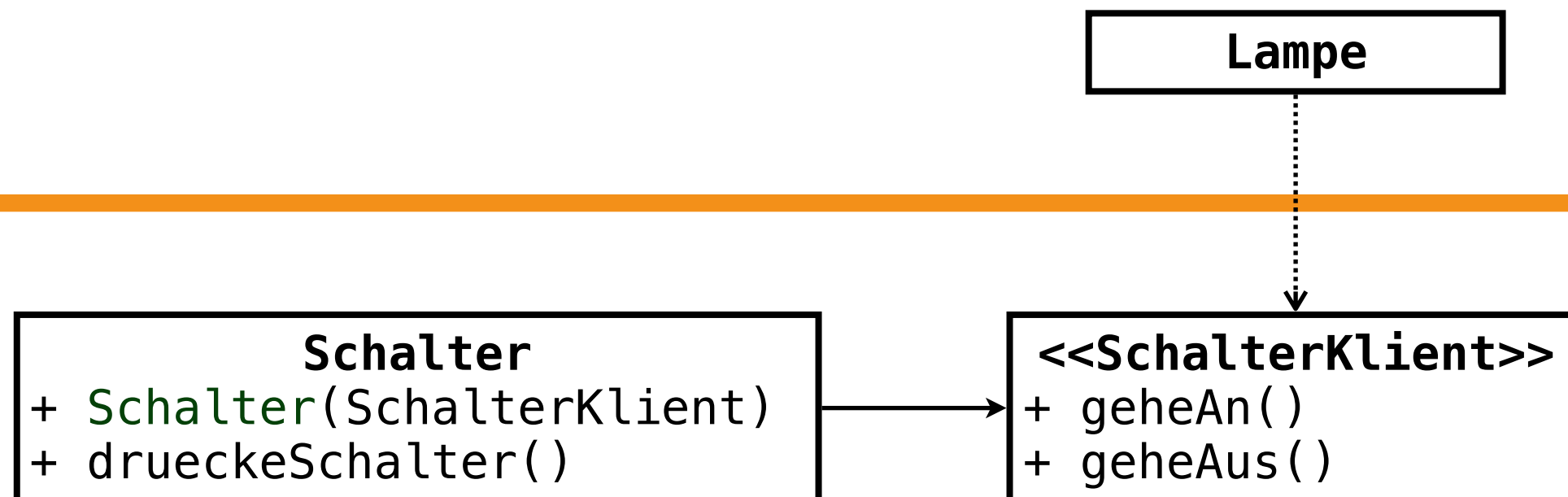
Dependency Inversion Principle

Sei nicht so detailverliebt, nutze Schnittstellen



Dependency Inversion Principle

Sei nicht so detailverliebt, nutze Schnittstellen



Dependency Inversion Principle

Sei nicht so detailverliebt, nutze Schnittstellen

Abhängigkeiten von außen geben

DI Container verwenden



Das **Reinheitsgebot**
des Softwareentwicklers

