Software Engineering I (02161)

Week 10

Assoc. Prof. Hubert Baumeister

DTU Compute
Technical University of Denmark

Spring 2018

Recap

- Observer Pattern
- MVC
- Presentation Layer Example
- S.O.L.I.D.
  - Simple Responsibility Principle (SRP)
  - Open/Closed Principle (OCP)

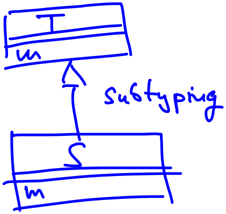
Contents

Liskov Substition Principle (LSP)

Subtype property S is a **subtype** of T:
"Let $\phi(x)$ be a *property* provable about objects x of type T. Then $\phi(y)$ should be true for objects y of type S where S is a subtype of T."

Liskov, B. H.; Wing, J. M. (November 1994). A behavioral notion of subtyping.

```
void f(T t) {
    ..
   t.m(args);
    ...
}
f(new T());
f(new S());
```



Liskov Substition Principle (LSP)

Subtype property S is a **subtype** of T:
”Let $\phi(x)$ be a *property* provable about objects x of type T. Then $\phi(y)$ should be true for objects y of type S where S is a subtype of T.”

Liskov, B. H.; Wing, J. M. (November 1994). A behavioral notion of subtyping.

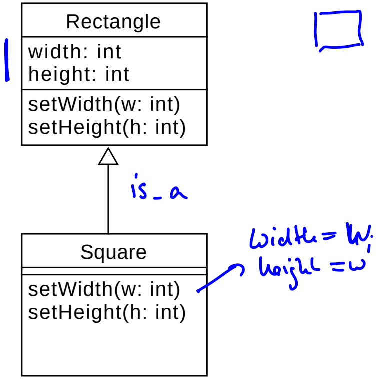
```
void f(T t) {
    ..
   t.m(args);
    ...
}
f(new T());
f(new S());
```

if `f(new T())` gives a result, then `f(new S())` gives a result

- *S* has to understand the same methods as *T*
- Dynamic typed languages (Smalltalk, Ruby, JavaScript, ...): ”duck typing”
  - ”If it walks like a duck and it quacks like a duck, then it must be a duck.”
- In Java by construction: `class S extends T {...}`

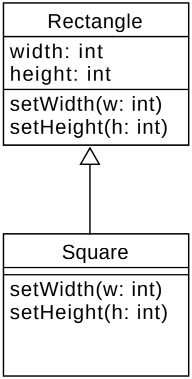
Behaviour conformance

- However: "Gives a result" is not enough
- Objects $y : S$ should behave like objects $x : T$
  - $y : S$ should fullfil all the expectations one has from $x : T$



Behaviour conformance

- However: "Gives a result" is not enough
- Objects $y : S$ should behave like objects $x : T$
    - $y : S$ should fullfil all the expectations one has from $x : T$



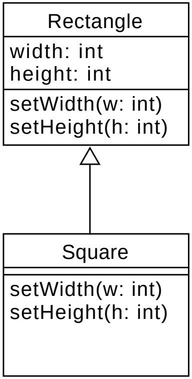
```
public void testArea(Rectangle r) {
  r.setWidth(10); r.setHeight(20);
  assertEquals(200, r.width * r.height);
}
```

What happens

- `testArea(new Rectangle())`? ✓

Behaviour conformance

- However: ”Gives a result” is not enough
- Objects $y : S$ should behave like objects $x : T$
  - $y : S$ should fullfil all the expectations one has from $x : T$

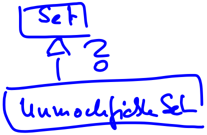


```
public void testArea(Rectangle r) {
  r.setWidth(10); r.setHeight(20);
  assertEquals(200, r.width * r.height);
}
```

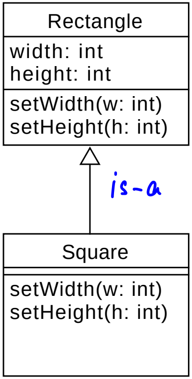
What happens

- `testArea(new Rectangle())`? ✓
- `testArea(new Square())`? f

Behaviour conformance



- However: "Gives a result" is not enough
- Objects $y : S$ should behave like objects $x : T$
  - $y : S$ should fullfil all the expectations one has from $x : T$



```
public void testArea(Rectangle r) {
  r.setWidth(10); r.setHeight(20);
  assertEquals(200, r.width * r.height);
}
```

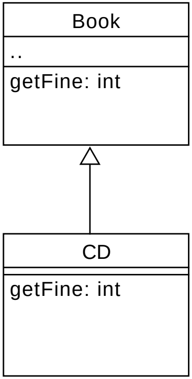
What happens

- `testArea(new Rectangle())`?
- `testArea(new Square())`?

We have found a property $\phi(x)$ that holds for $x : T$ but not for $y : S$

Square should not inherit from Rectangle

GetFine Example



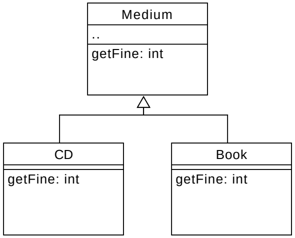
- Book getFine returns 100
- CD getFine returns 200

```
public void testFine(Book b) {
   assertEquals(100,b.getFine()};
}
testFine(new Book(...))? ✓
testFine(new CD(...))? f
```

Problem with LSP: Which properties ϕ? All possible observations or only those that we **expect** from T?

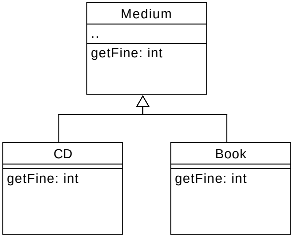
$\rightarrow$ Make expectations explicit: Design by contract

GetFine Example solution



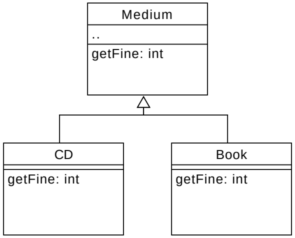
```
public void testFineBook {
   Book b = new Book(...)
   assertEquals(100,b.getFine()};
}
public void testFineCd {
   CD c = new CD(...)
   assertEquals(200,c.getFine());
}
```

GetFine Example solution



```
public void testFineBook {
   Book b = new Book(...)
   assertEquals(100,b.getFine()};
}
public void testFineCd {
   CD c = new CD(...)
   assertEquals(200,c.getFine());
}


public void testFineMedium(Medium m)  {
   assertTrue(m.getFine() >= 0);
}
testFineMedium(new Book(...)) -> Ok
testFineMedium(new Cd(...)) -> Ok
```

GetFine Example solution

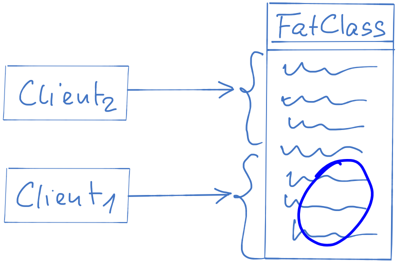


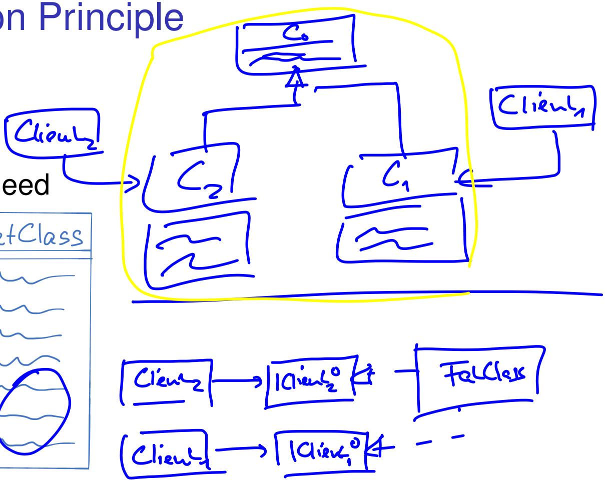
```
public void testFineBook {
   Book b = new Book(...)
   assertEquals(100,b.getFine()};
}
public void testFineCd {
   CD c = new CD(...)
   assertEquals(200,c.getFine());
}


public void testFineMedium(Medium m) {
   assertTrue(m.getFine() >= 0);
}
testFineMedium(new Book(...)) -> Ok
testFineMedium(new Cd(...)) -> Ok
```

Conclusion: When creating a subclass, make sure that it satisfies all *expectations* from the superclass

Interface Segregation Principle

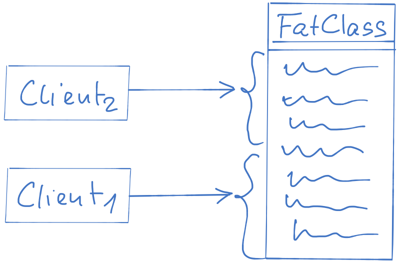
Clients 1/2 depend on functionality they don't need



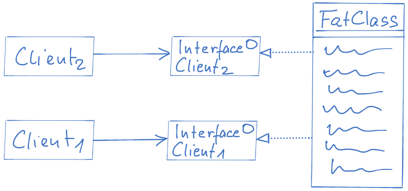


Interface Segregation Principle

Clients 1/2 depend on functionality they don't need

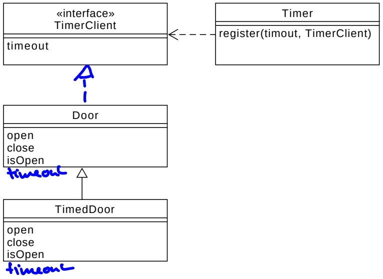


Separate out needed functionality in interfaces

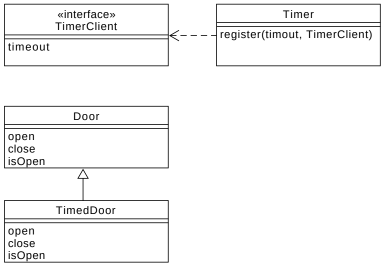


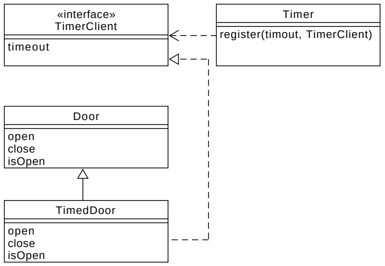
$\rightarrow$ Single Responsibility Principle

Timed Door Example



Timed Door Example



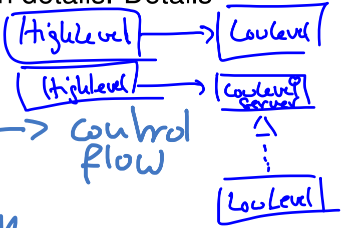


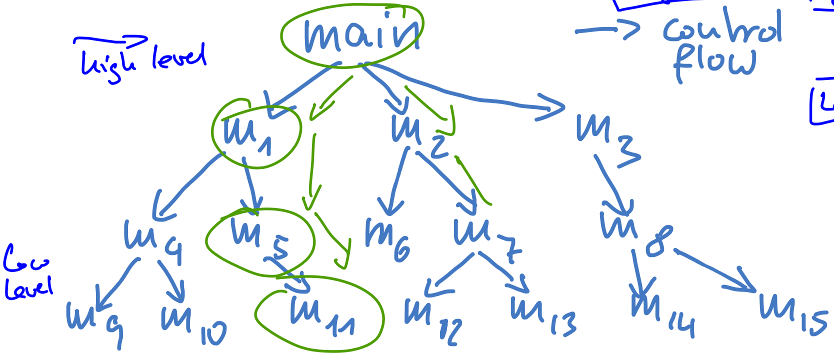
Dependency-Inversion Principle (DIP

A. ”High-level modules should not depend on low-level modules. Both should depend on abstractions.”

B. ”Abstractions should not depend upon details. Details should depend upon abstractions.”

Robert C. Martin (2007) Agile Principles, Patterns, and Practices in C#



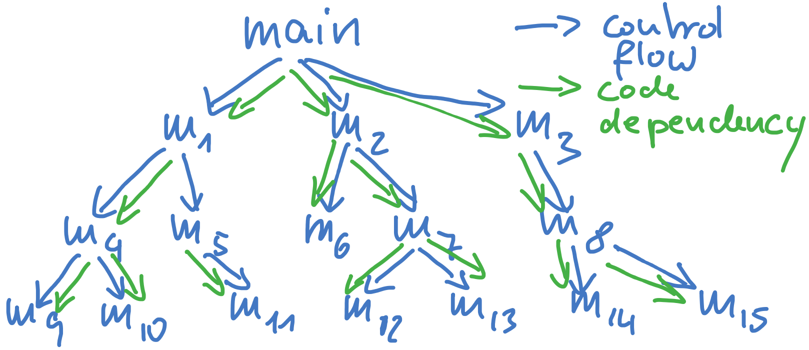


Dependency-Inversion Principle (DIP

A. ”High-level modules should not depend on low-level modules. Both should depend on abstractions.”

B. ”Abstractions should not depend upon details. Details should depend upon abstractions.”

Robert C. Martin (2007) Agile Principles, Patterns, and Practices in C#



Copy

```
public class Copier {  bool tf = true ;
    static void copy() {
        int c;       if (tf) { ch = Tape.read() } else {
        while( (ch = Keyboard.read()}) != -1) {
            Printer.write(ch);
        }
    }
}
```

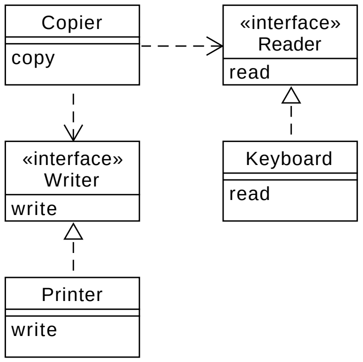
$$\begin{tikzcd}
 & \text{Copy} \arrow[dl, bend right] \arrow[dl] \arrow[dr] \arrow[dr, bend left] & \\
\text{Keyboard.read} & & \text{Printer.write}
\end{tikzcd}$$

Copy

```
public class Copier {
//remember to reset these flags
public static bool ptFlag = false;
public static bool punchFlag = false;

  public static void copy()
  {
     int c;
     while((c=(ptFlag ? PaperTape.read() : Keyboard.read())) != -1) {
        punchFlag ? PaperTape.punch(c) : Printer.write(c);
     }
  }
}
```

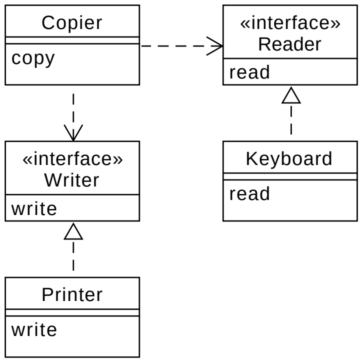
How would you solve that problem?

Solution: Dependency Inversion

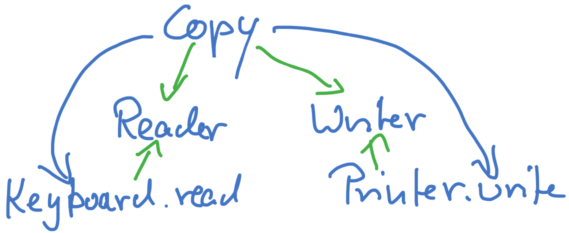


```
public class Copier {
  public static void copy(Reader r,
                          Writer w)
  {
     int c;
     while((c=(r.read())) != -1) {
        w.write(c);
     }
  }
}
```

Solution: Dependency Inversion



```
public class Copier {
  public static void copy(Reader r,
                          Writer w)
  {
     int c;
     while((c=(r.read())) != -1) {
        w.write(c);
     }
  }
}
```



Furnace Example

```
const byte TERMOMETER = 0x86; const byte FURNACE = 0x87;
const byte ENGAGE = 1;
const byte DISENGAGE = 0;

void Regulate(double minTemp, double maxTemp) {
for(;;) {
    while (in(THERMOMETER) > minTemp)
      wait(1);
    out(FURNACE,ENGAGE);
    while (in(THERMOMETER) < maxTemp)
      wait(1);
    out(FURNACE,DISENGAGE);
  }
}
```

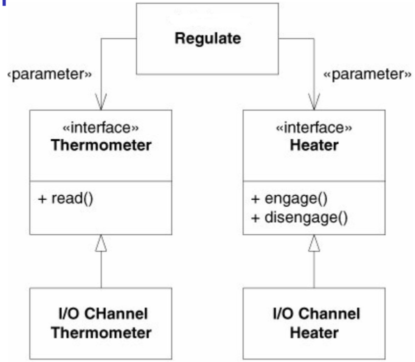
Hardaddresses

Low level

high coupling

high level

Furnace Example Solution



Robert C. Martin (2007) Agile Principles, Patterns, and Practices in C#

```
void Regulate(Thermometer t, Heater h, double minTemp, double maxTemp)
{
for(;;)
  {
    while (t.Read() > minTemp)
      wait(1);
    h.Engage();
    while (t.Read() < maxTemp)
      wait(1);
    h.Disengage();
  }
}
```

Summary

- High level code should not depend on lower level code
- Dependency inversion breaks this dependency
- Heuristics (use with consideration)
  - ”No variable should hold a reference to a concrete class”
  - ”No class should derive from a concrete class”
  - ”No method should override an implemented method of any of its base classes”

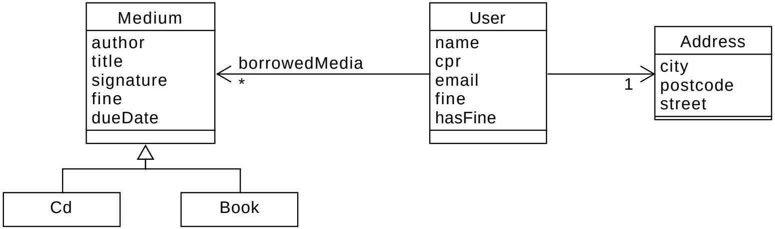
Contents

Persistence

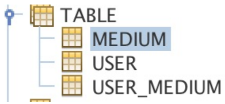
- Important: not required for the exam project
- Example of layered architecture and dependency inversion principle
- Java Persistence API (JPA)
- Sqlite embedded database (relational (SQL) database)

Relational databases

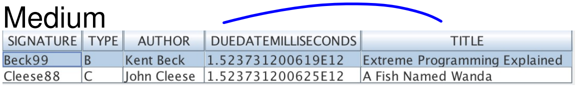
Data stored in tables: classes and certain types of relations



Database model

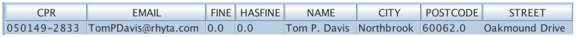


Medium



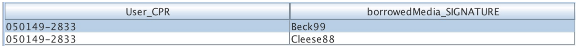
SIGNATURE	TYPE	AUTHOR	DUEDATEMILLISECONDS	TITLE
Beck99	B	Kent Beck	1.523731200619E12	Extreme Programming Explained
Cleese88	C	John Cleese	1.523731200625E12	A Fish Named Wanda

User



CPR	EMAIL	FINE	HASFINE	NAME	CITY	POSTCODE	STREET
050149-2833	TomPDavis@rhyta.com	0.0	0.0	Tom P. Davis	Northbrook	60062.0	Oakmound Drive

Borrowed media association



User_CPR	borrowedMedia_SIGNATURE
050149-2833	Beck99
050149-2833	Cleese88

JPA Annotations

Medium

```
@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "TYPE",
                     discriminatorType = DiscriminatorType.STRING,
                     length = 20)
@DiscriminatorValue("M")
public abstract class Medium {
  private String title;
  private String author;
  @Id private String signature;
  private long dueDateMilliseconds;
```

Book

```
@Entity
@DiscriminatorValue("B")
public class Book extends Medium {
```

Cd

```
@Entity
@DiscriminatorValue("C")
public class Cd extends Medium {
```

JPA Annotations

User

```
@Entity
public class User {
  @Id
  private String cpr;
  private String name;
  private String email;
  @Embedded
  private Address address;
  @OneToMany
  private List<Medium> borrowedMedia = new ArrayList<>();
  private double fine = 0d;
  private boolean hasFine = false;
```

$\rightarrow$ generates a seperate table

Address

```
@Embeddable
public class Address {
  private String street;
  private int postCode;
  private String city;
```

JPA usage: Methods in LibraryApp

Entity manager based on META-INF/persistence.xml

```
Map<String, String> properties = new HashMap<String, String>();
properties.put("javax.persistence.jdbc.url",
               "jdbc:sqlite:lib/db/production.db");
EntityManagerFactory emf =
      Persistence.createEntityManagerFactory("library", properties);
EntityManager em = emf.createEntityManager();
```

Add a medium

high level code

```
public void addMedium(Medium medium) {
  checkAdministratorLoggedIn();   ← business logic
  em.getTransaction().begin();
  em.persist(medium);
  em.getTransaction().commit();
```

database code (business logic)
↳ low level code

Borrow a medium

```
public void borrowMedium(Medium medium, User user) throws Exception {
  user.borrowMedium(medium, dateServer.getDate());   ← business logic
  em.getTransaction().begin();
  em.merge(m);       ← update m
  em.merge(user);    ← update u
  em.getTransaction().commit();
```

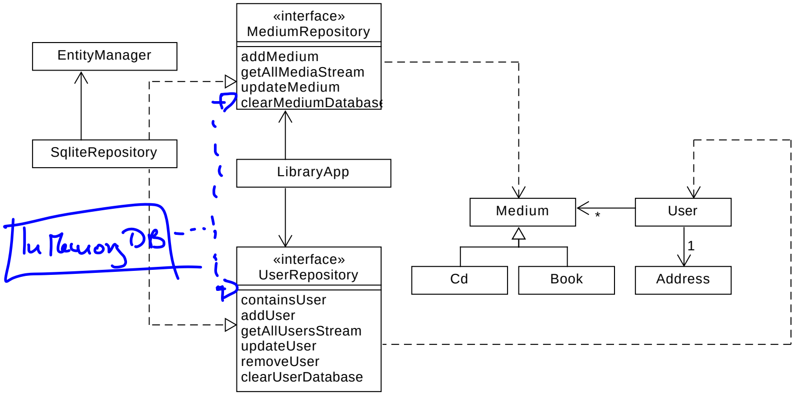
database

Get all media

```
public Stream<Medium> getMediaStream() {
   return em.createQuery("SELECT m FROM Medium m", Medium.class)
         .getResultStream();
}
```

How to improve the design?

Improvment with Dependency Inversion



JPA usage: Methods in LibraryApp

Dependency **injection** via constructor

```
public LibraryApp(MediumRepository mediumRepo,
                  UserRepository userRepo) {
  this.mediumRepository = mediumRepo;
  this.userRepository = userRepo;
}
SqliteRepository repo = new SqliteRepository();
new LibraryApp(repo,repo);
```

Add a medium

```
public void addMedium(Medium medium) {
  checkAdministratorLoggedIn();
  mediumRepository.addMedium(medium);
```

Borrow a medium

```
public void borrowMedium(Medium medium, User user) throws Exception {
  user.borrowMedium(medium, dateServer.getDate());
  userRepository.updateUser(user);
  mediumRepository.updateMedium(medium);
```

Get all media

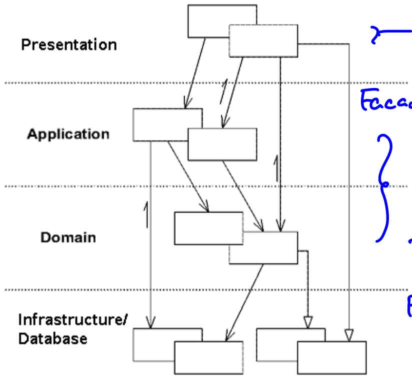
```
public Stream<Medium> getMediaStream() {
   return mediumRepository.getAllMediaStream();
}
```

InMemoryRepository

```
public class InMemoryRepository implements MediumRepository,
                                           UserRepository {
  List<Medium> media = new ArrayList<>();
  List<User> users = new ArrayList<>();

  public void addMedium(Medium medium) {
    media.add(medium);
  }
  public Stream<Medium> getAllMediaStream() {
    return media.stream();
  }
  public void updateMedium(Medium m) { }
  public boolean contains(User user) {
    return users.contains(user);
  }
  public void addUser(User user) {
    users.add(user);
  }
  public Stream<User> getAllUsersStream() {
    return users.stream();
  }
  public void removeUser(User user) {
    users.remove(user);
  }
  public void updateUser(User user) { }
}
```

Contents

Layered Architecture



- Questions 1:
  - Presentation layer depends on application layer?
  - Application layer depends on presentation layer?
  - Low coupling between the layers

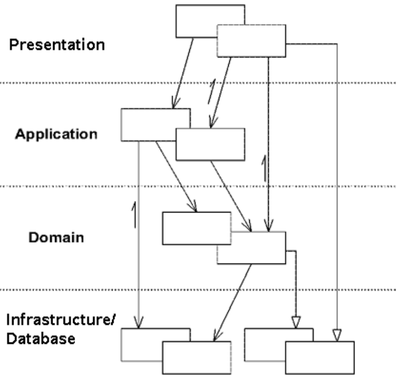
→ present. no business logic

Facade

business logic + no database logic

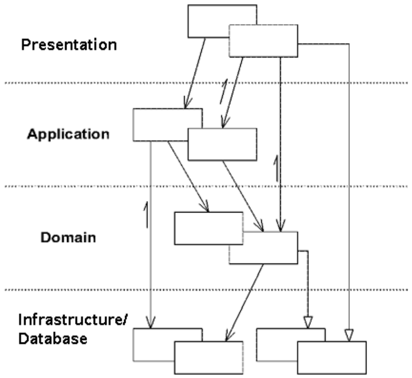
Facades / Interfaces → Dependency inversion

Layered Architecture



- Questions 1:
  - Presentation layer depends on application layer?
  - Application layer depends on presentation layer?
  - Low coupling between the layers
- Question 2: Sql statements in domain layer
  - Domain layer depends on database layer?
  - Database layer depends on domain layer?
  - Low coupling between the layers?

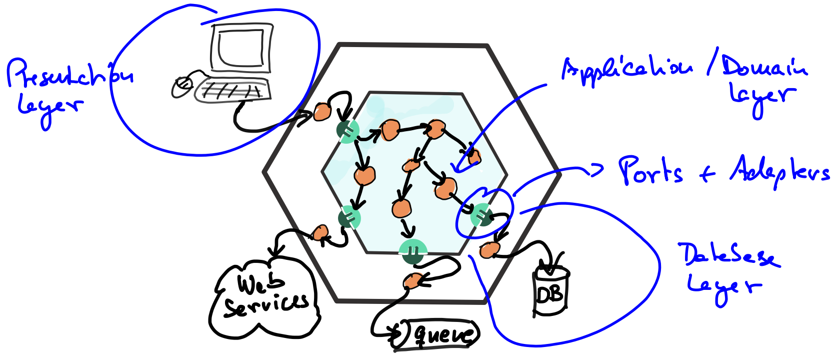
Layered Architecture



- Questions 1:
  - Presentation layer depends on application layer?
  - Application layer depends on presentation layer?
  - Low coupling between the layers
- Question 2: Sql statements in domain layer
  - Domain layer depends on database layer?
  - Database layer depends on domain layer?
  - Low coupling between the layers?

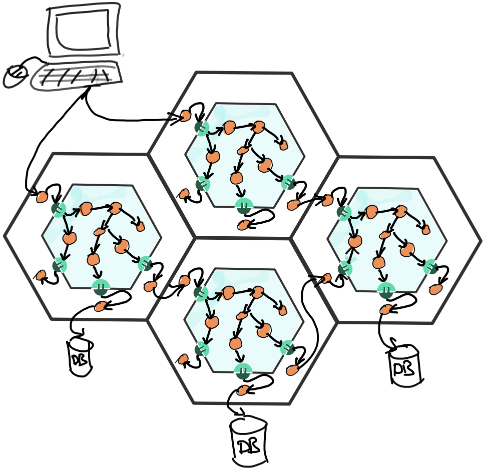
- Keep business logic out of presentation/database layer
- Keep database code out of application/domain layer

Hexagonal Architecture / Ports and Adapters



- Two types of ports
  - Primary: ”input” ports: interfaces or facades
  - Secondary: ”output” ports: usually interfaces (dependency inversion)
- Dependency Injection injects adapters
- Example: LibraryApp

Microservices



- Application divided in communicating, pluggable ”mini applications”
- Run on their own VM or container
- Communicate through Web services or message queues

Next Week

- Design by Contract