SELECTED

JAVA EE COMPONENTS

ORM, DATA MAPPERS, BUSINESS, PRESENTATION

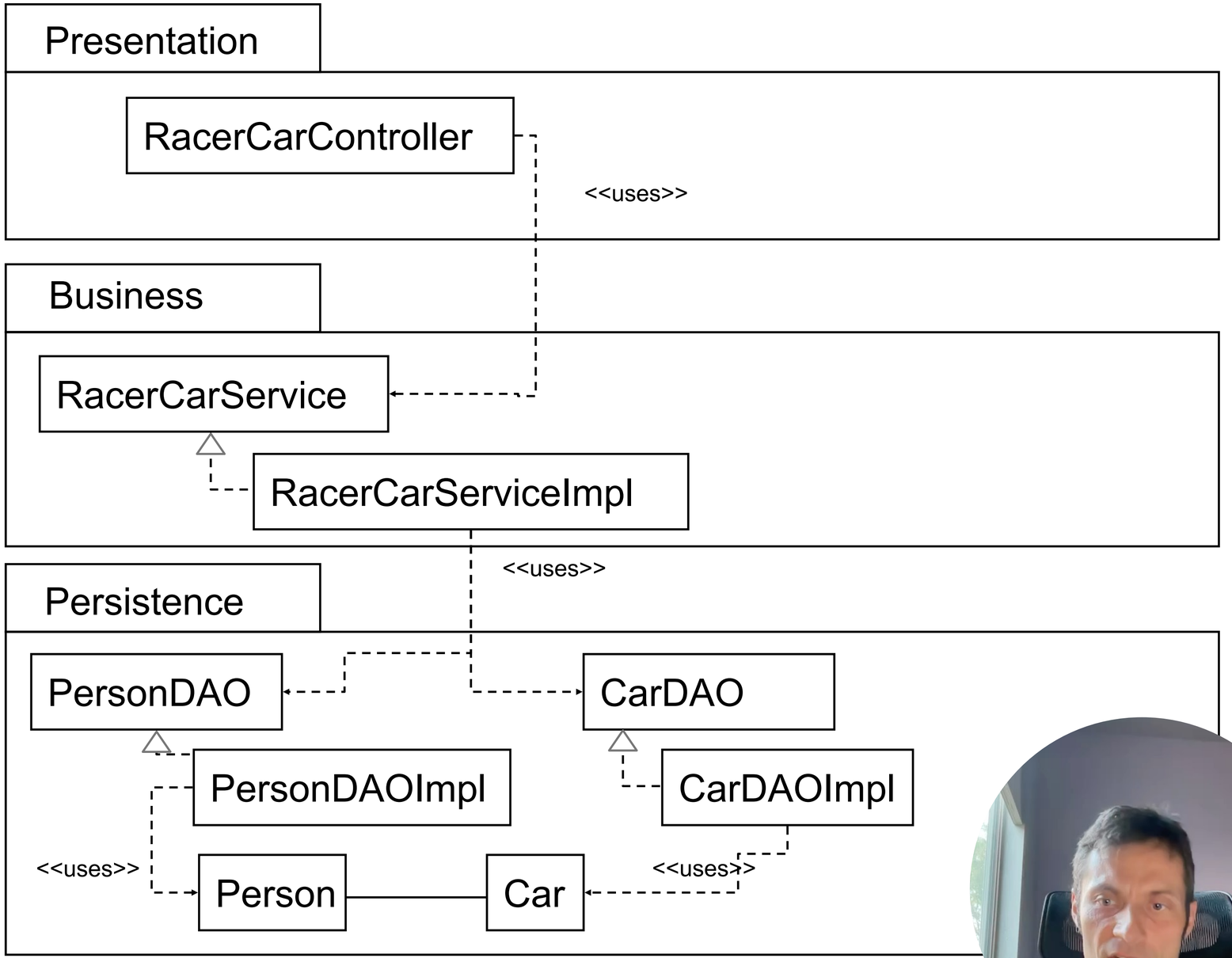
OUTLINE

1. Dependency injection
2. ORM
3. Data mappers
4. Business Layer

JAVA EE

COMPONENTS

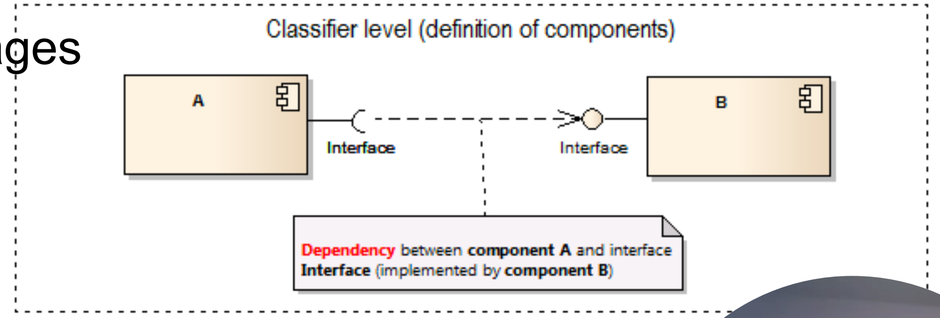
What is it they provide?

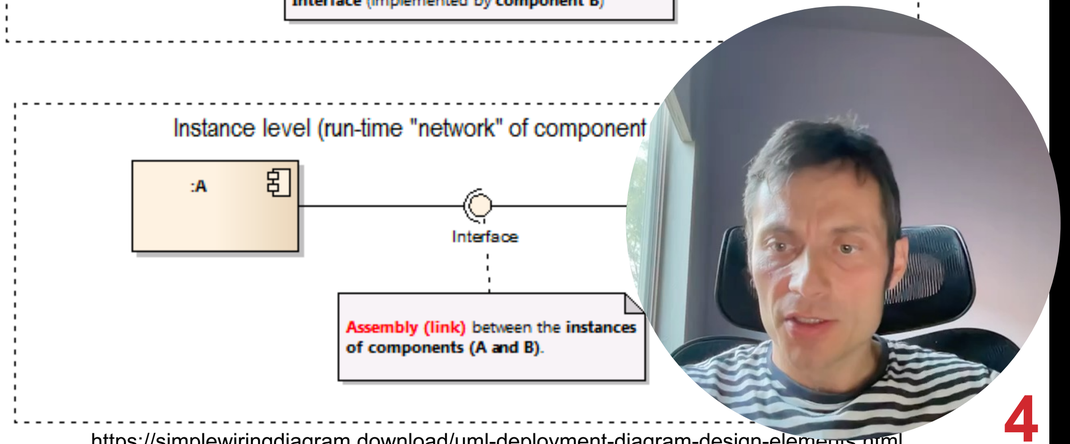


JAVA EE COMPONENTS

Functional components

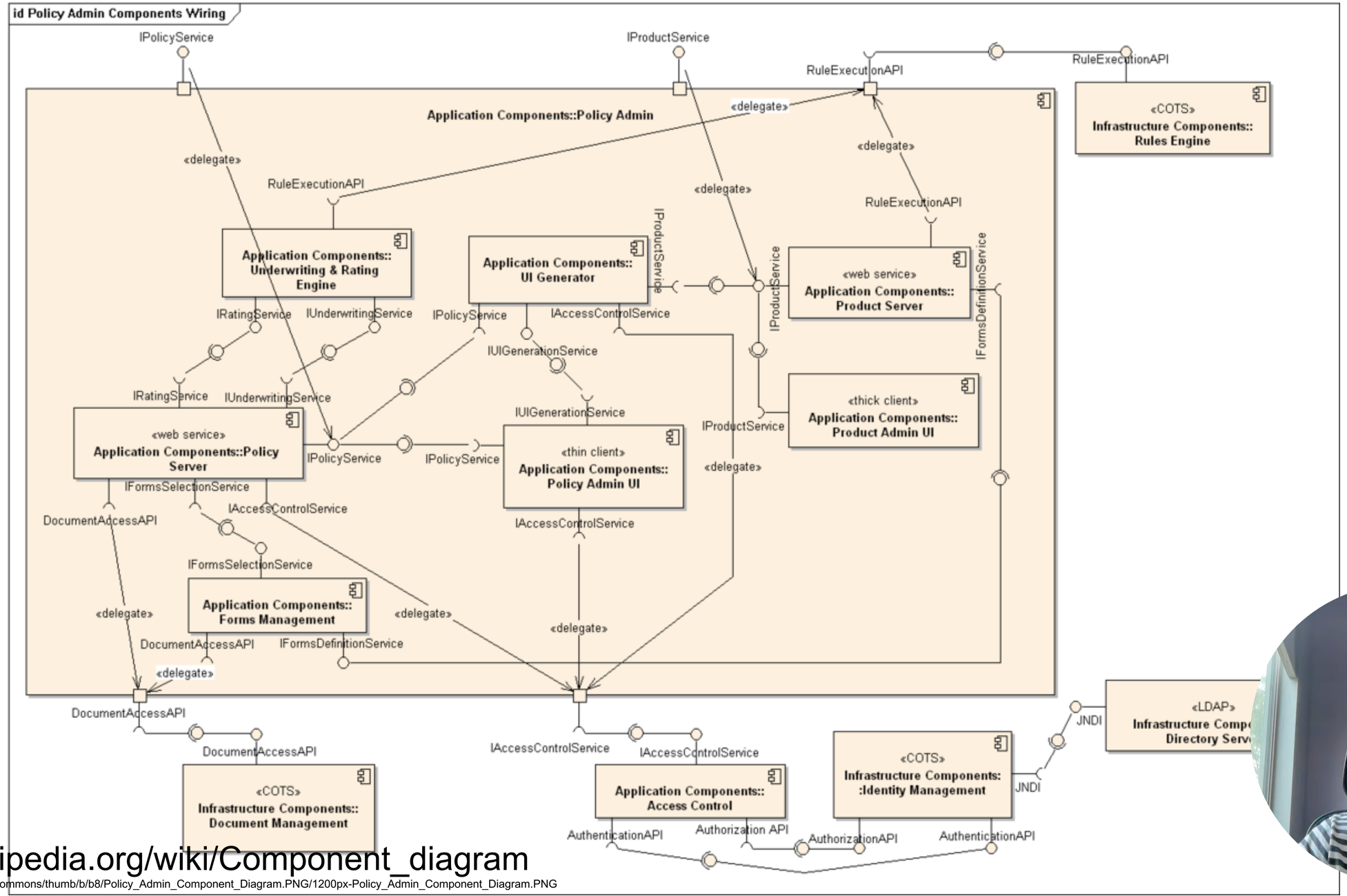
- Enterprise beans = Enterprise JavaBeans (EJB)
  - **Session beans** – transient conversation with client. Once the client is served by the session bean and its data are gone
  - **Message driven beans** – session bean features and message listener – receive messages asynchronously. Interacts with Java Message Service (JMS)
    - Multiple services can interact through messages
- Web page
- Servlet/JSF/JSP/Bean
- Applet
- Entity





https://simplewiringdiagram.download/uml-deployment-diagram-design-elements.html

UFF.. COUPLING!!!!



https://en.wikipedia.org/wiki/Component_diagram

https://upload.wikimedia.org/wikipedia/commons/thumb/b/b8/Policy_Admin_Component_Diagram.PNG/1200px-Policy_Admin_Component_Diagram.PNG

JAVA EE

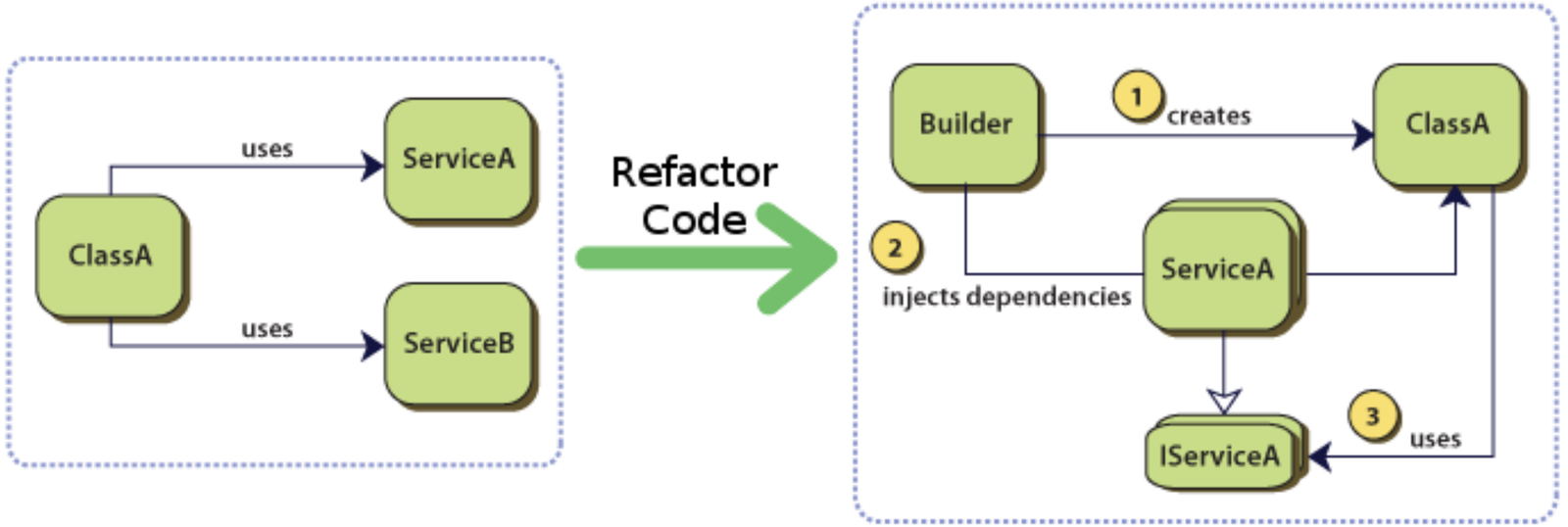
COMPONENTS

Many components needs to be connected / assembled

Introducing high coupling

Contexts and Dependency Injection (CDI)

- Contextual services in Java EE container
- Integration of components with **loose coupling and typesafety**
- Dependency injection



https://medium.com/@elliotchance/a-new-simpler-way-to-do-dependency-injection-in-go-9e191bef50d5

EXAMPLE

DEPENDENCY INJECTION

Naïve application that finds Movies

Use Case : List all movies

```
public interface MovieFinder {
    List findAll(); // e.g. HQL
}
```

Repository

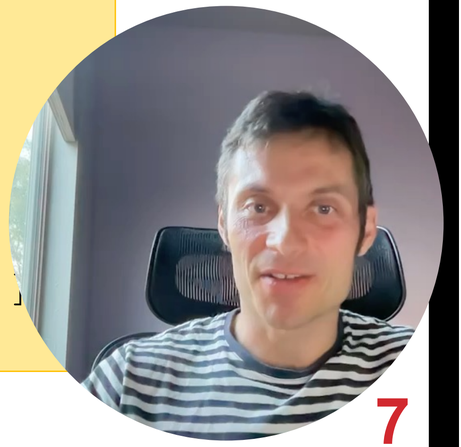
```
class MovieLister...
  private MovieFinder finder;
  public MovieLister() {
    finder = new ColonDelimitedMovieFinder("movies1.txt");
  }
```

Initiated Lister

Concrete subtype of MovieFinder

Service/EJB

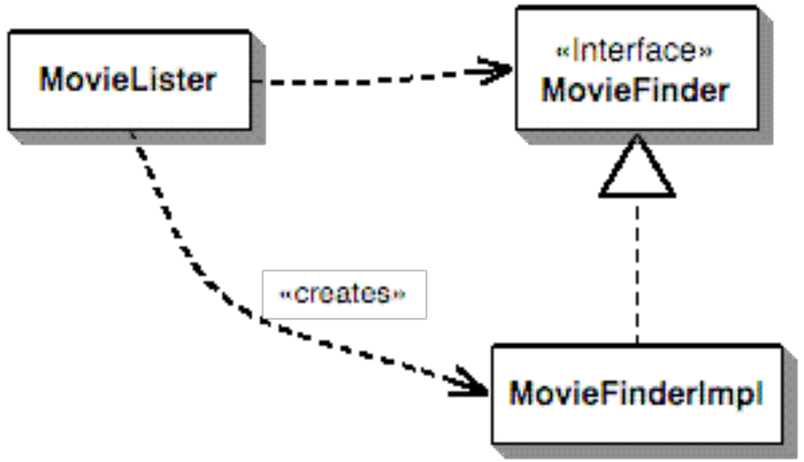
```
class MovieLister...
    public Movie[] moviesDirectedBy(String arg) {
        List allMovies = finder.findAll();
        for (Iterator it = allMovies.iterator(); it.hasNext();) {
            Movie movie = (Movie) it.next();
            if (!movie.getDirector().equals(arg)) it.remove();
        }
        return (Movie[]) allMovies.toArray(new Movie[allMovies.size()]);
    }
```



EXAMPLE

DEPENDENCY INJECTION

Example



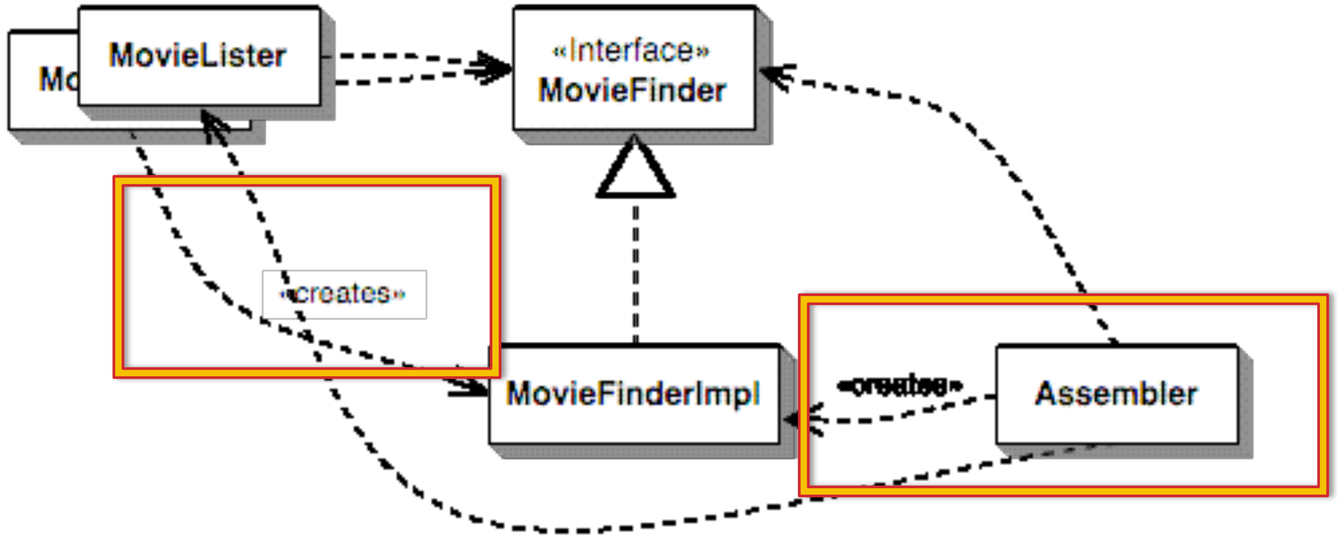

```
class MovieLister...
    public Movie[] moviesDirectedBy(String arg) {
        List allMovies = finder.findAll();
        for (Iterator it = allMovies.iterator(); it.hasNext();) {
            Movie movie = (Movie) it.next();
            if (!movie.getDirector().equals(arg)) it.remove();
        }
        return (Movie[]) allMovies.toArray(new Movie[allMovies.size()]);
    }
```

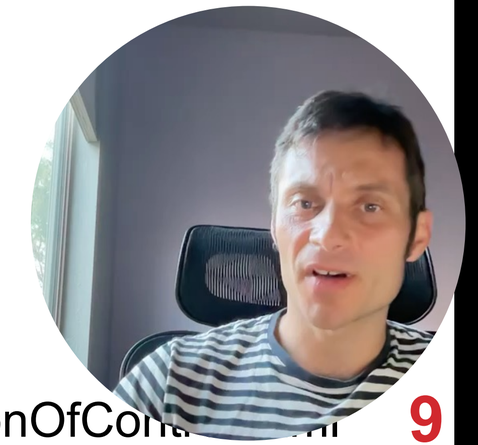
https://martinfowler.com/bliki/InversionOfCont

EXAMPLE

DEPENDENCY INJECTION

Example





https://martinfowler.com/bliki/InversionOfCont

EXAMPLE SPRING SETTER DI

```
class MovieLister {
  private MovieFinder finder;
  public void setFinder(MovieFinder finder) {
    this.finder = finder;
  }
```

Setter method to inject a service

```
class ColonMovieFinder extends MovieFinder {
   public void setFilename(String filename) {
        this.filename = filename;
   }
```

Setter for source files

```
 <beans>
        <bean id="MovieLister" class="spring.MovieLister">
            <property name="finder">
                <ref local="MovieFinder"/>
            </property>
        </bean>
        <bean id="MovieFinder" class="spring.ColonMovieFinder">
            <property name="filename">
                <value>movies1.txt</value>
            </property>
        </bean>
    </beans>
```

XML config

EXAMPLE SPRING SETTER DI

Usage:

```
public void testWithSpring() throws Exception {
  ApplicationContext ctx
            = new FileSystemXmlApplicationContext("spring.xml");
  MovieLister lister
            = (MovieLister) ctx.getBean("MovieLister");
  Movie[] movies
            = lister.moviesDirectedBy("Sergio Leone");

  assertEquals("Once Upon a Time in the West", movies[0].getTitle());
}
```

EXAMPLE

DEPENDENCY INJECTION

DeltaSpike Data Module

https://deltaspike.apache.org/documentation/data.html

```
@Repository
public interface MovieFinder extends EntityRepository<Movie, Long>,
                                     IRepositoryExtension<Movie> {
    List<Movie> findAll(); // e.g. HQL
    @Query(named="MoviesAll") List<Movie> findAllAgain();
}
```

Named query

```
@Stateless
class MovieLister {

   @Inject
   private MovieFinder finder;

   public Movie[] moviesDirectedBy(String arg) {
        List allMovies = finder.findAll();
        for (Iterator it = allMovies.iterator(); it.hasNext()
            Movie movie = (Movie) it.next();
            if (!movie.getDirector().equals(arg)) it.remove();
        }
        return (Movie[]) allMovies.toArray(new Movie[allMovies.size()
    }
```

EJB service

JAVA EE COMPONENTS –

CONTEXT & DEPENDENCY INJECTION

Example

```
import javax.inject.Inject;
import javax….RequestScoped;

@RequestScoped
public class Printer {
  @Inject
  Greeting greeting;
...
```

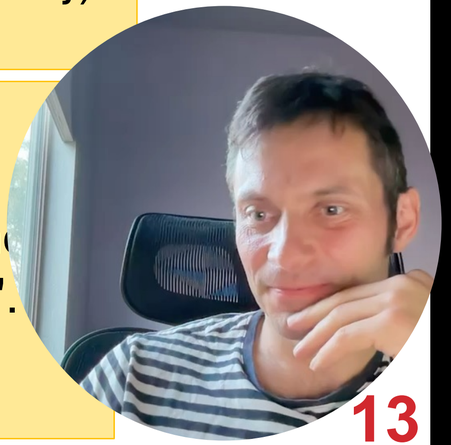
```
import javax.inject.Inject;
import javax….RequestScoped;

@RequestScoped
public class Printer {
   @Inject @Informal
   Greeting greeting;
...
```

Use qualifiers to provide various implementations of a particular bean type.
A qualifier is an annotation that you apply to a bean.

```
@Qualifier @Retention(RUNTIME) @Target({TYPE, METHOD, FIELD, PARAMETER})
public @interface Informal {}
```

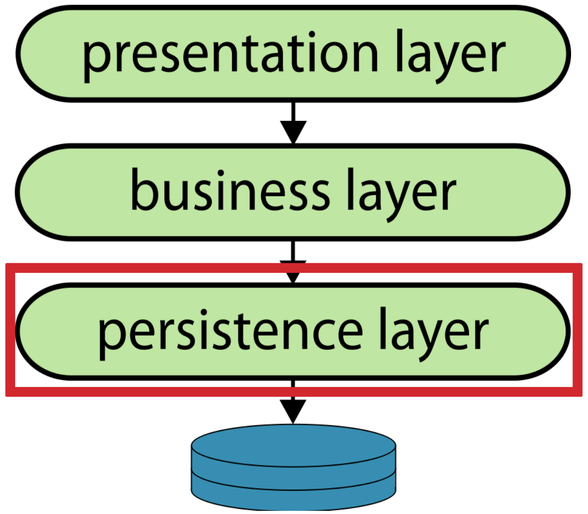
```
@Informal
public class InformalGreeting extends Greeting {
  public String greet(String name) {
        return "Hi, " + name + "!";
  }
}
```

```
@Default
public class Greeting {
 public String greet(String nam
        return "Hello, " + name + ".
 }
}
```

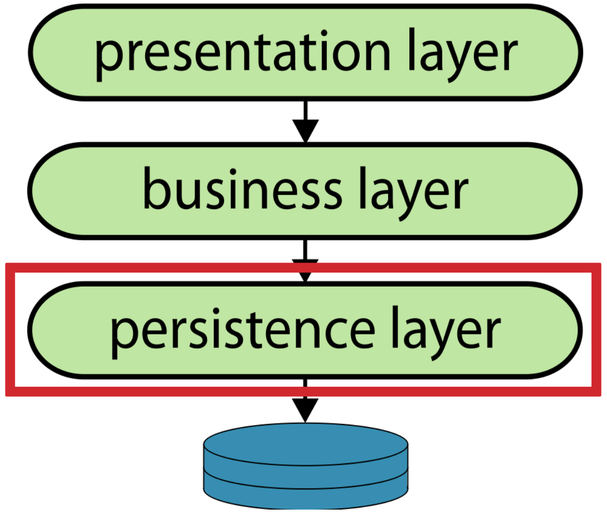


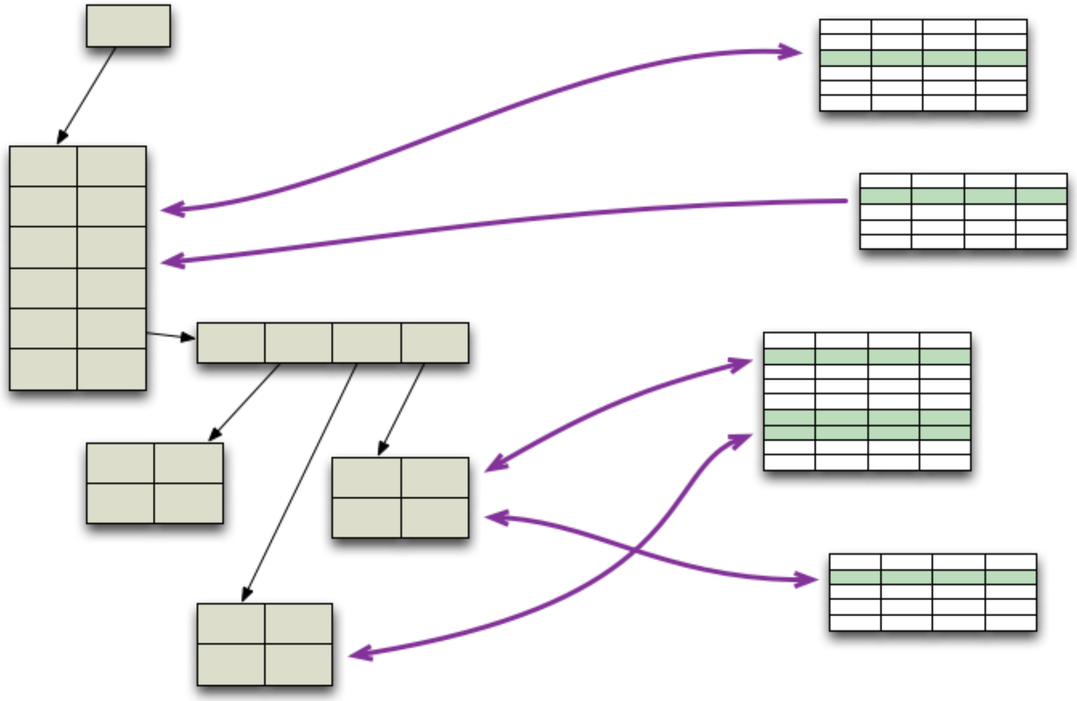
PERSISTENCE LAYER OF EAA

ORM



PERSISTENCE LAYER





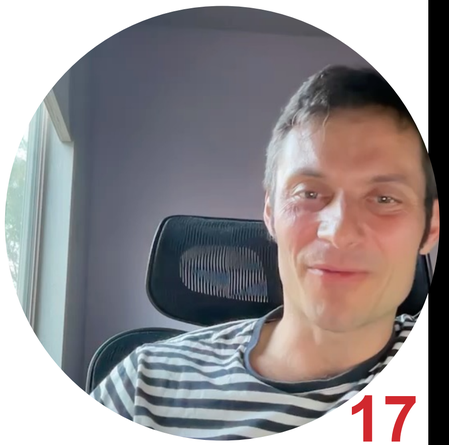
https://martinfowler.com/bliki/images/ormHate/sketch

OBJECT-RELATIONAL MAPPING ORM

- **Quotes**
  - Ted Neward, late 2004
    - "Object-Relational mapping is the Vietnam of our industry”
    - I've seen developers struggle for years with the huge mismatch between relational database models and traditional object models. And all the solutions they come up with seem to make the problem worse.

WHAT IS ORM?

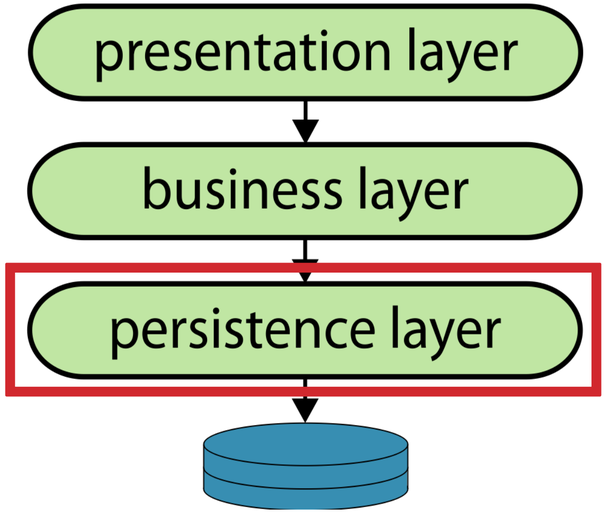
- **Persistence**
  - Helping to achieve persistence – app data outlive the process
- **Relational database**
  - Persistence to RDBMS in Object-oriented app
- **Object-relational mismatch**
  - Paradigm mismatch – object model and relational model incompatible
- **Granularity**
  - More objects than tables in DB (Person>Address)
- **Subtypes**
  - Inheritance in DB?
- **Identity notion**
  - Primary key vs. a == b and a.equals(b) options
- **Associations**
  - Unidirectional/Bidirectional in OOP vs Foreign Key in RDBMS
- **Data navigation**
  - Fundamental difference in OOP and DRBMS
  - Object network vs. JOIN ? Efficiency number of SQL queries (lazy load)



OBJECT-RELATIONAL MAPPING ORM

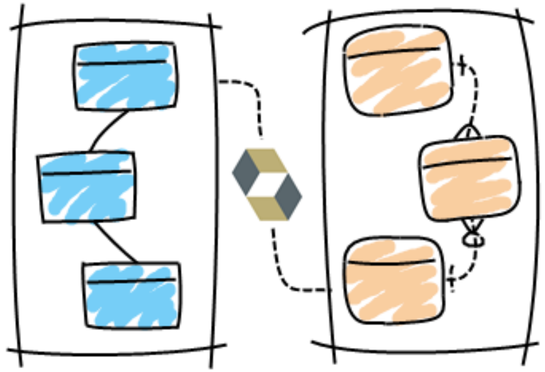
- **Databases are relational**
  - Relations and SQL
  - No inheritance
  - Good performance and centralization
  - ACID, fast to save and retrieve data = GOOD for Persistence
- **Object works**
  - Great for problem decomposition
  - Slow to be used for data persistence
  - Designed to deal with problems not to persist data
  - All nice features and instruments of Object-oriented design
    - Polymorphism, Generics..

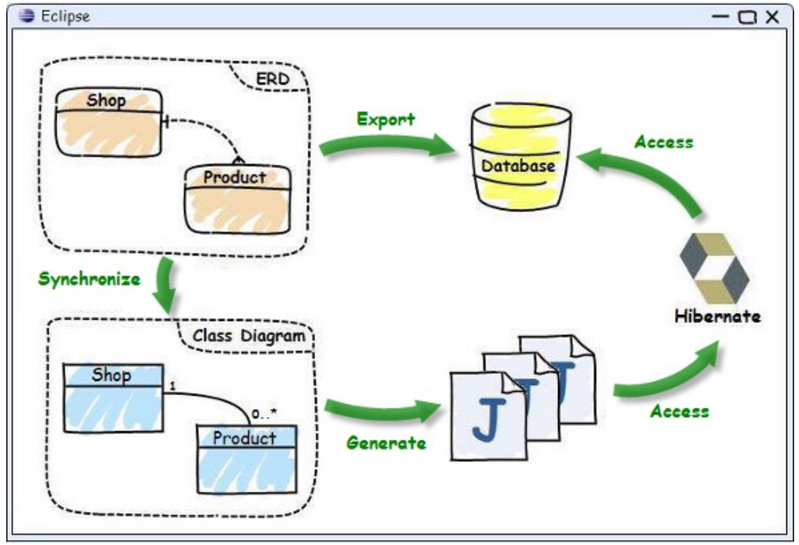
PERSISTENCE LAYER

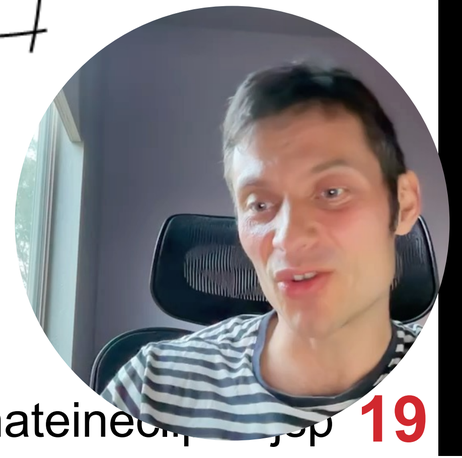


Object-relational mapping (ORM)

- **Databases are relational RDBMS**
- **Objects are not**
  - Technical incompatibility
- **Solution – mapping to both directions**







https://www.visual-paradigm.com/tutorials/hibernateineclipse.jsp

OBJECT-RELATIONAL MAPPING (ORM)

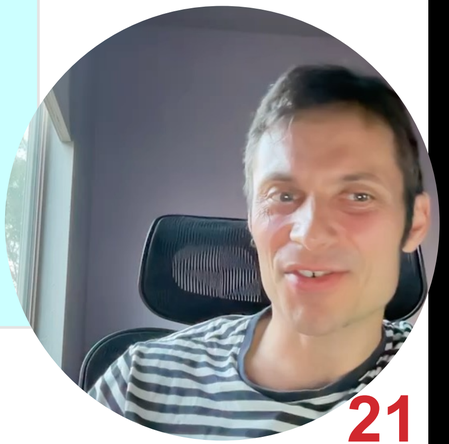
- **Create**

```
public void create() {
      Event e = new Event( "First event!",new Date())
      Session session = sessionFactory.openSession();
      session.beginTransaction();
      session.save(e);
      session.save(new Event( "Second event",new Date()));
      session.getTransaction().commit();
      session.close();
}
```

OBJECT-RELATIONAL MAPPING (ORM)

- **Read**

```
public void read() {
  Session session = sessionFactory.openSession();
  session.beginTransaction();
  List result = session.createQuery("from Event").list();
  for (Event event : (List<Event>) result) {
   out.print("Event "+event.getDate()+"):"+event.getTitle());
  }
  session.getTransaction().commit();
  session.close();
 }
```



OBJECT-RELATIONAL MAPPING (ORM)

- **Entity + annotation**

Note the JavaBeans rules @Component

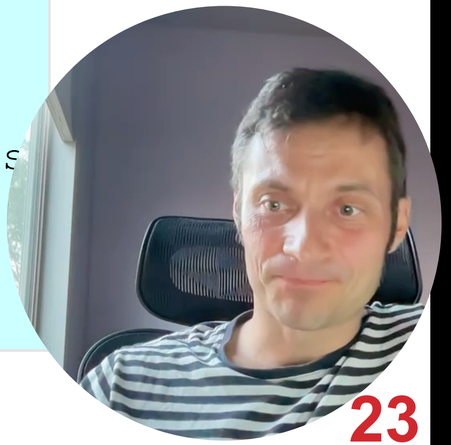
```
@Entity
@Table( name = "EVENTS" )
public class Event {
    ...
 @Id
 @GeneratedValue("increment")
 @GenericGenerator(
    name="increment",
    strategy = "increment")
 public Long getId() {
    return id;
 }
```

```
public String getTitle() {
    return title;
}

@Temporal(TemporalType.TIMESTAMP)
@Column(name = "EVENT_DATE")
public Date getDate() {
    return date;
}
}
```

OBJECT-RELATIONAL MAPPING (ORM)

- **Read/Write**

```
...
Cat fritz = (Cat) sess.load(Cat.class, generatedId);
..
Cat cat = (Cat) sess.get(Cat.class, id);
if (cat==null) {
    cat = new Cat();
    sess.save(cat, id);
}
..
sess.save(cat);
sess.flush(); //force the SQL INSERT
sess.refresh(cat); //re-read the state (after the trigger executes

...
```



OBJECT-RELATIONAL MAPPING (ORM)

Note the JavaBeans rules @Component

- **Inheritance**

```
@Entity
@Inheritance(strategy=
    InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(
                name="planetype”,
 discriminatorType=
    DiscriminatorType.STRING
)
@DiscriminatorValue("Plane")
public class Plane { ... }

@Entity
@DiscriminatorValue("A320")
public class A320 extends Plane {
... }
```

Associations

```
@Entity
public class FlyingObject

  @ManyToOne
  public PropulsionType
                getPropulsion() {
        return altitude;
  }
  @OneToOne(cascade =
                CascadeType.ALL)
  public Navigation getNavi()
        return navi;
  }
```

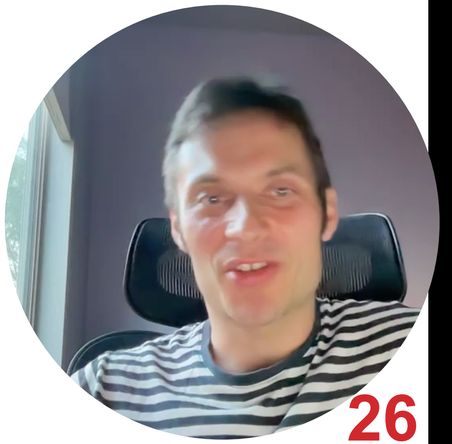
ORM FEATURES

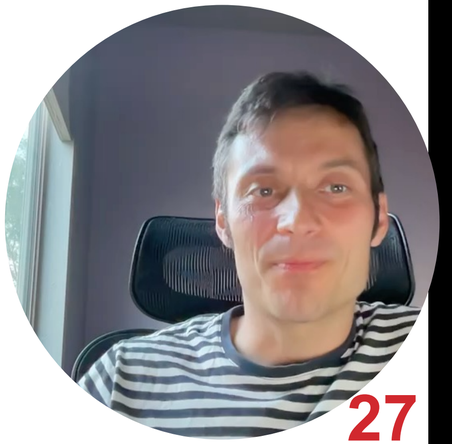
- **Object property mapping**
- **Association mapping (One-one, one-many, many-many)**
  - Lazy loading
- **Inheritance mapping**
  - Single/Table per class/Table per concrete class
- **Generated Keys**
- **Cascades**
- **Locking** – Bob and John modify Person with ID = 1
- **Hibernate Query Language HQL**
- **Mapping to many databases**
- **Cache**
- **Criteria API**

OBJECT-RELATIONAL MAPPING (ORM)

- **Issues**
  - Needs custom equals and hashCode
  - Too much SQL
    - You can make custom/native query
    - N+1 problem with lazy load

- **More? follow links**
  - http://docs.jboss.org/hibernate/orm/4.2/quickstart/en-US/html/
  - http://docs.jboss.org/hibernate/orm/4.2/devguide/en-US/html/





N+1 PROBLEM

Let's say you have a collection of Car objects (database rows), and each Car has a collection of Wheel objects (also rows). In other words, Car → Wheel is a 1-to-many relationship.

Now, let's say you need to iterate through all the cars, and for each one, print out a list of the wheels. The naive O/R implementation would do the following:

SELECT * FROM Cars;

And then **for each Car:**

SELECT * FROM Wheel WHERE CarId = ?

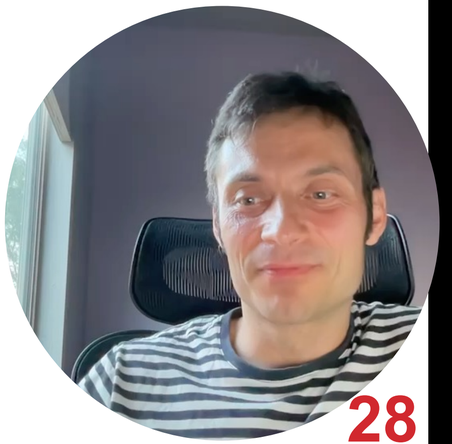
In other words, you have one select for the Cars, and then N additional selects, where N is the total number of cars.

Alternatively, one could get all wheels and perform the lookups in memory:

SELECT * FROM Wheel

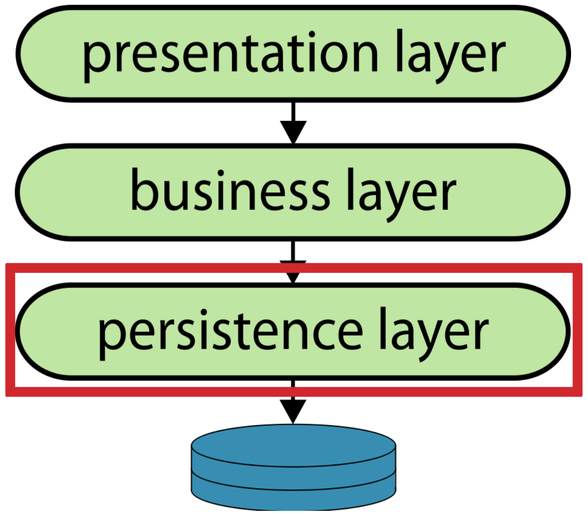
This reduces the number of round-trips to the database from N+1 to 2. Most ORM tools give you several ways to prevent N+1 selects.

Reference: *Java Persistence with Hibernate*, chapter 13.



PERSISTENCE LAYER OF EAA

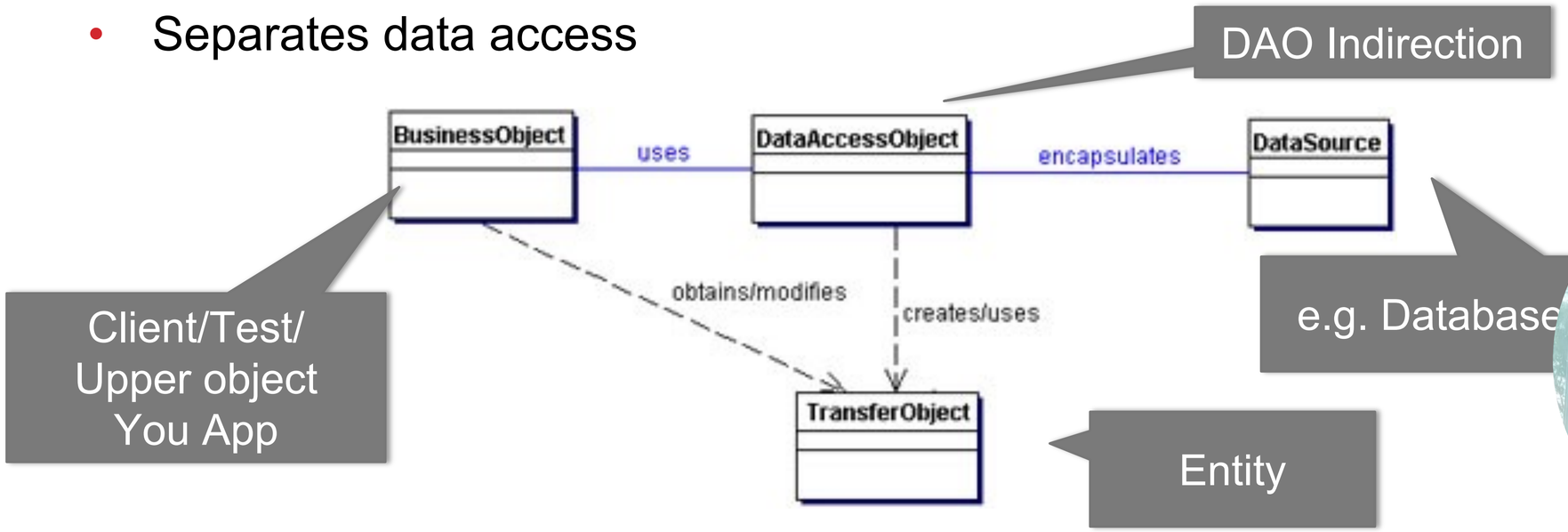
MAPPERS

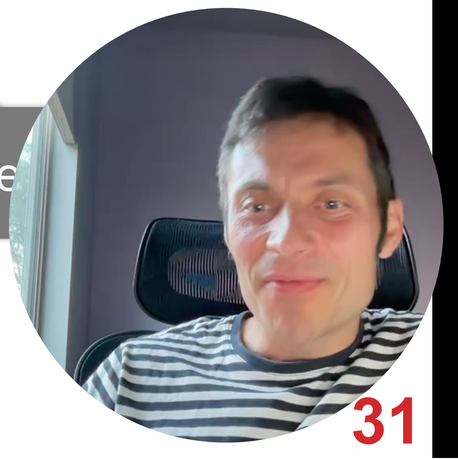


https://www.oracle.com/java/technologies/dataaccessobject.html

DATA ACCESS OBJECT (DAO)
DESIGN PATTERN

- Problem:
  - How to access persistent objects
  - Where to persist data
  - Wrap access to Data Source
  - Indirection
- Object that provides interface to persistence mechanism
- Separates data access

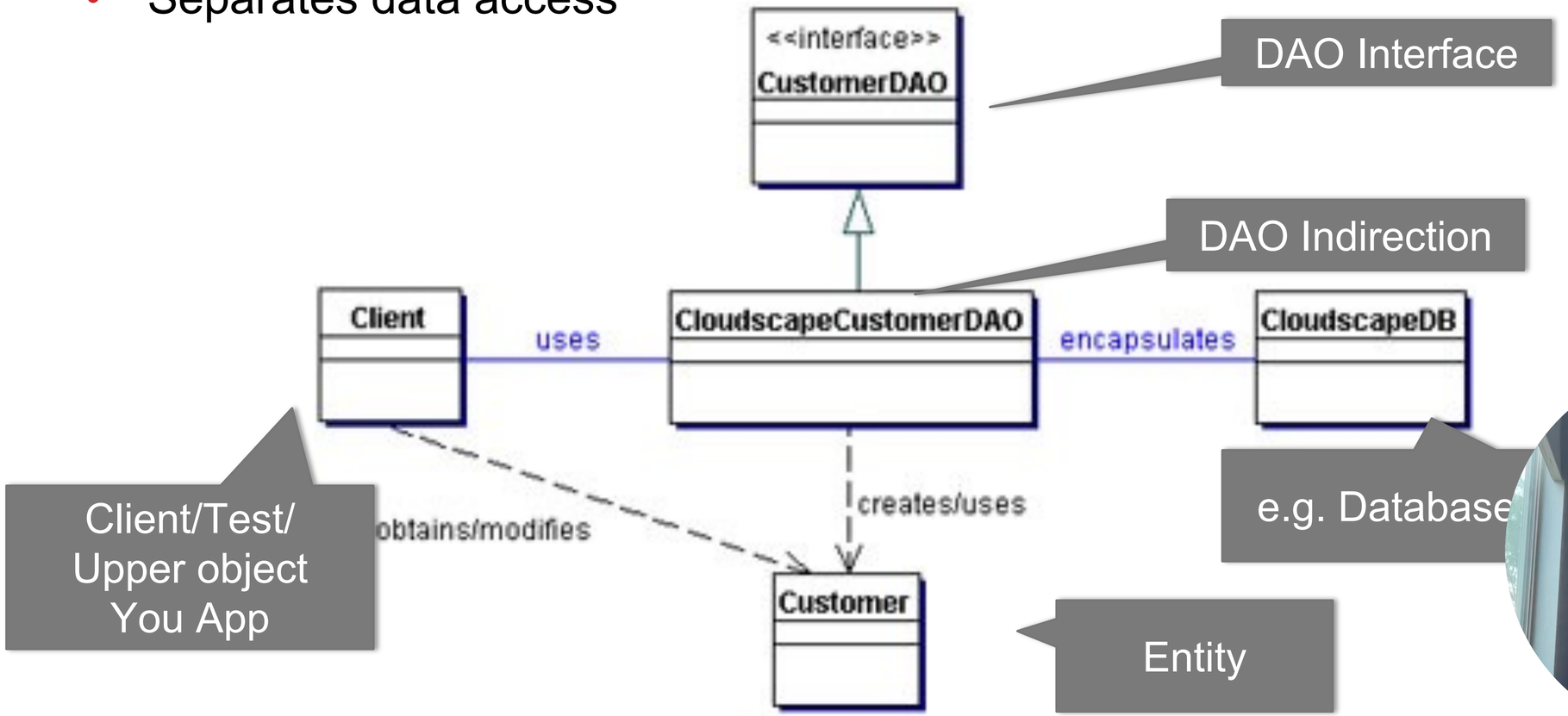


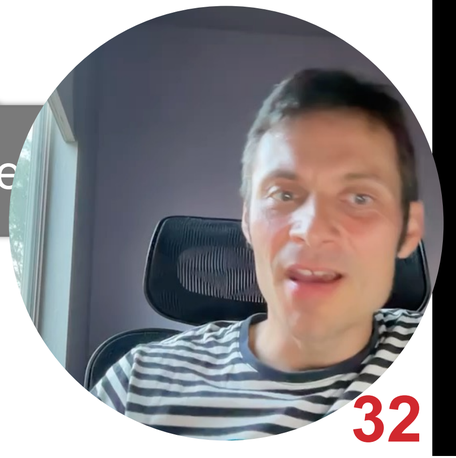


https://www.oracle.com/java/technologies/dataaccessobject.html

DATA ACCESS OBJECT (DAO)
DESIGN PATTERN

- **Design pattern**
  - Object that provides interface to persistence mechanism
  - Separates data access

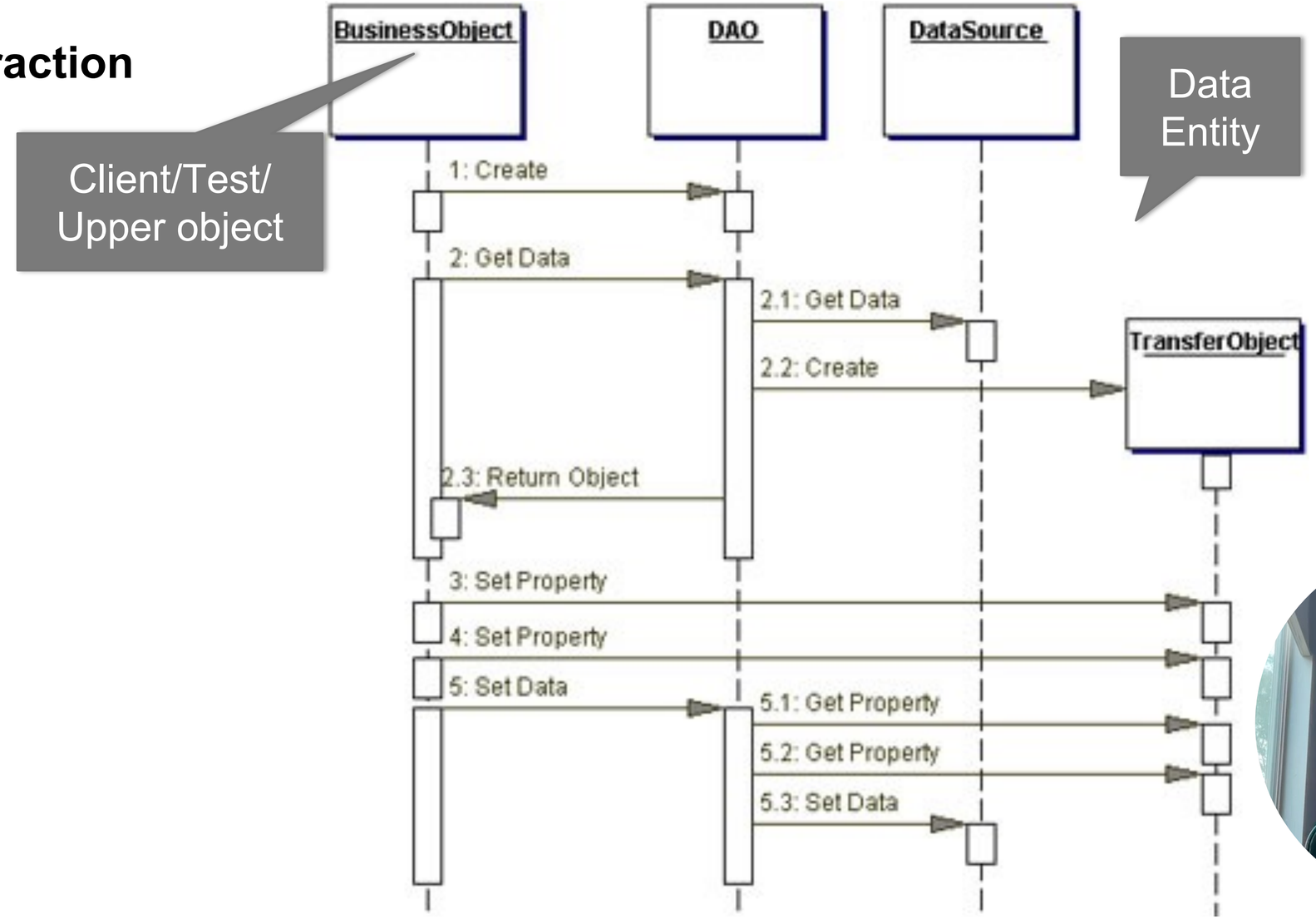


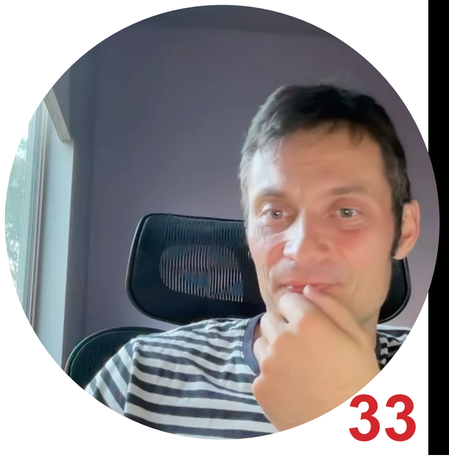


https://www.oracle.com/java/technologies/dataaccessobject.html

DATA ACCESS OBJECT (DAO)
DESIGN PATTERN

- **Interaction**

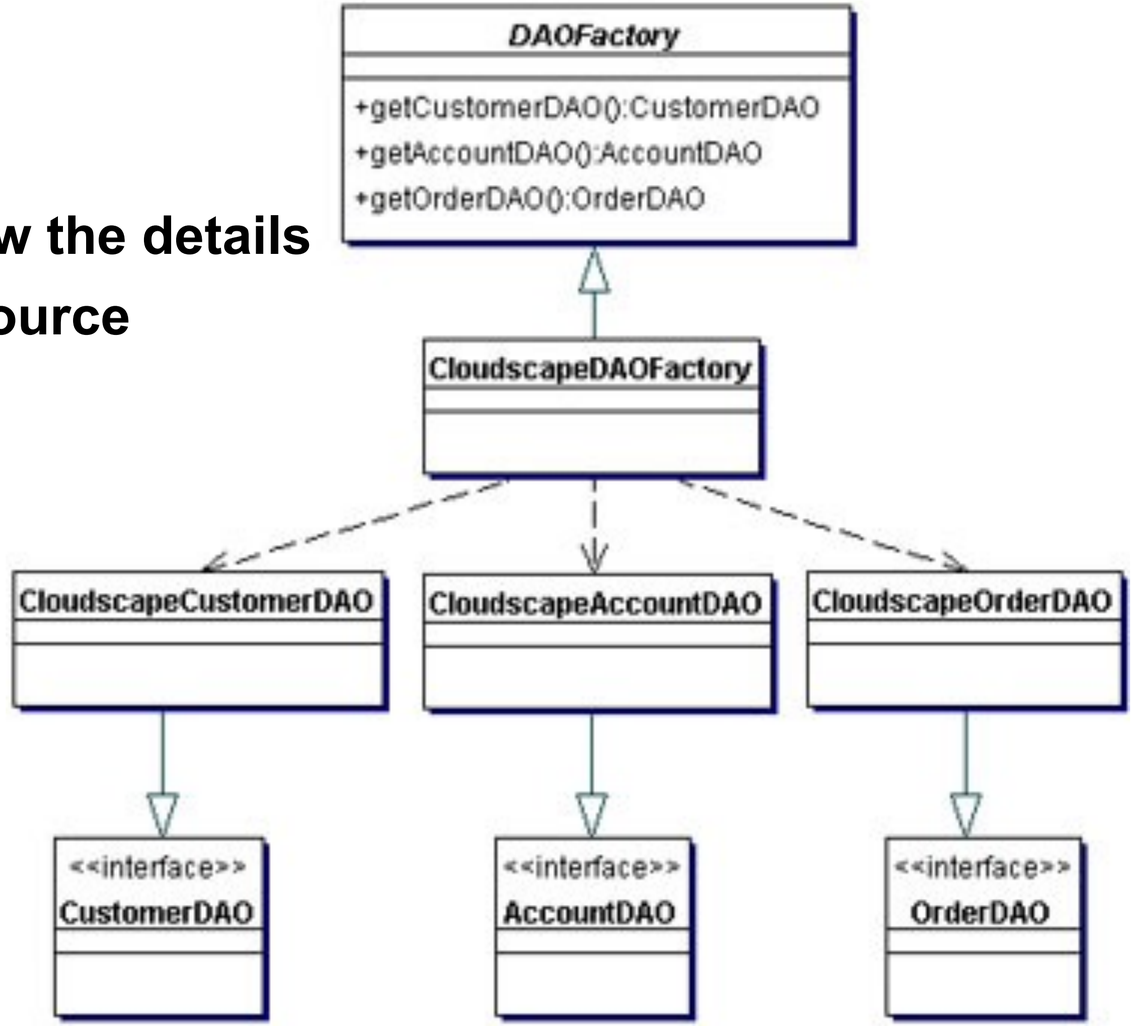




https://www.oracle.com/java/technologies/dataaccessobject.html

DATA ACCESS OBJECT (DAO)
DESIGN PATTERN

- Usual situation
- Different Objects
- Client does not know the details
- Transparent Data Source

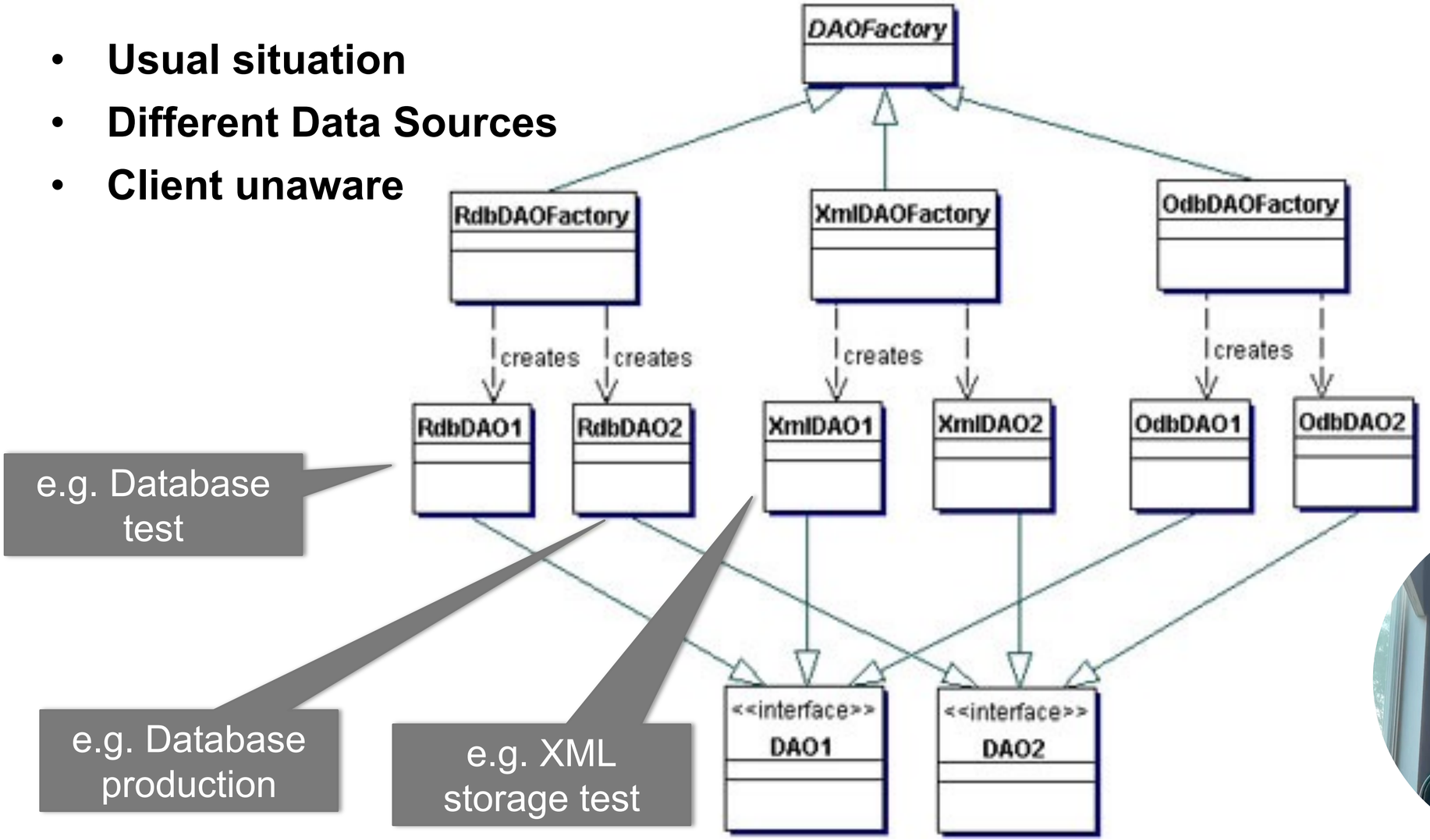


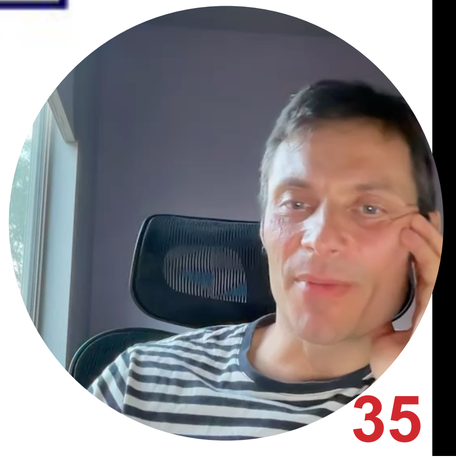
https://www.oracle.com/java/technologies/dataaccessobject.html

DATA ACCESS OBJECT (DAO)

DESIGN PATTERN

- **Usual situation**
- **Different Data Sources**
- **Client unaware**





DATA ACCESS OBJECT (DAO)

DESIGN PATTERN INTERFACE

```
// Interface that all CustomerDAOs must support
public interface CustomerDAO {
  public int insertCustomer(...);
  public boolean deleteCustomer(...);
  public Customer findCustomer(...);
  public boolean updateCustomer(...);
  public RowSet selectCustomersRS(...);
  public Collection selectCustomersTO(...);
  ...
}
```

DATA ACCESS OBJECT (DAO)

DESIGN PATTERN

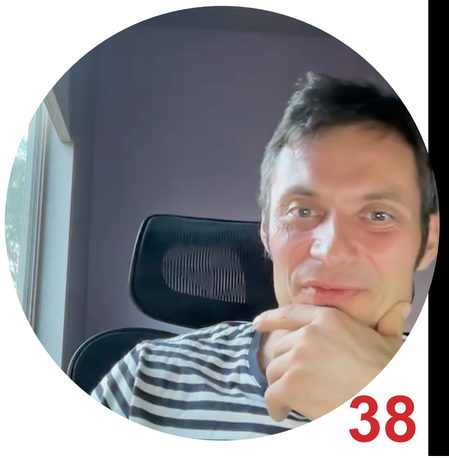
IMPLEMENTATION

```
public class CloudscapeCustomerDAO
        implements CustomerDAO {

  public int insertCustomer(...) {
    // Implement insert customer here.
    // Return created customer number
    // or a -1 on error
  }
  public boolean deleteCustomer(...) {
    // Implement delete customer here
    // True -success, false -failure
  }

  public Customer findCustomer(...) {
    // Find a customer using supplied
    // argument values -search criteria
    // Return Transfer Object if found,
    // return null on error / not found
  }

public boolean updateCustomer(...) {
    // update record here using data
    // from the customerData Object
    // True - success, false – failure
}
…
```

DATA ACCESS OBJECT (DAO)

DESIGN PATTERN ENTITY

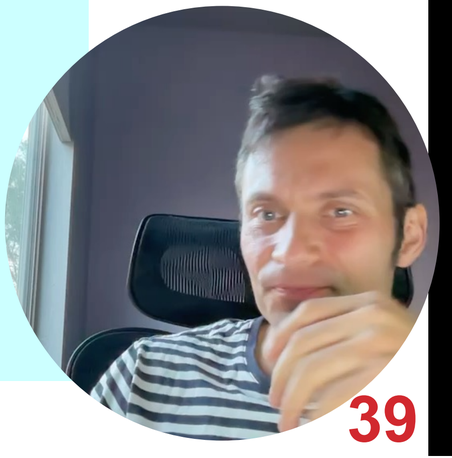
```
@Entity
public class Customer implements java.io.Serializable {
  // member variables
  int customerNumber;
  String name;
  String streetAddress;
  String city;
  ...

  // getter and setter methods...
  ...
}
```



DATA ACCESS OBJECT (DAO)
DESIGN PATTERN

```
// create the required DAO Factory
DAOFactory cloudscapeFactory = ..
// Create a DAO
CustomerDAO custDAO =
     daoFactory.getCustomerDAO();

// create a new customer
int custNo = custDAO.insertCustomer(...);

// Find a customer object.
Customer cust = custDAO.findCustomer(...);

// modify the values in the Transfer Object.
cust.setAddress(...);
cust.setEmail(...);

// update the customer object using the DAO
custDAO.updateCustomer(cust);
```

```
// delete a customer object
custDAO.deleteCustomer(cust);

// select all customers in the same city
Customer criteria=new Customer();
criteria.setCity("New York");
Collection customersList =
    custDAO.selectCustomersTO(criteria);
// returns customersList
….
```



DATA ACCESS OBJECT (DAO)
DESIGN PATTERN

- **Defines CRUD**
  - Create
  - Read
  - Update
  - Delete

- **See more at**
  - https://www.oracle.com/java/technologies/dataaccessobject.html

DATA REPOSITORY
DESIGN PATTERN

- **Problem**
  - CRUD
  - Look up a particular object.
  - We know the ID, name and need the object.
  - Find all orders from a customer.

  - Concentration of query construction code.
  - Large number of domain classes or heavy querying.

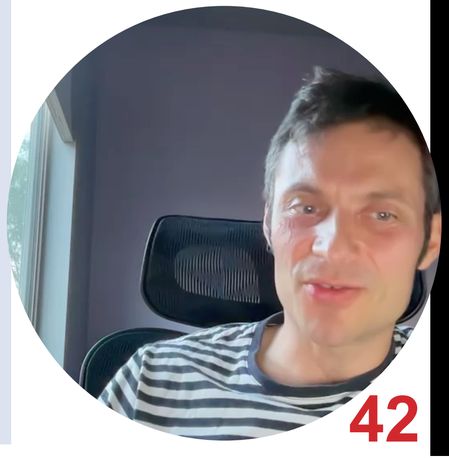
  - ***The goal of Data Repository is to significantly reduce the amount of boilerplate code required to implement data access layers for various persistence stores.***

DATA REPOSITORY
DESIGN PATTERN BY SPRING

Concept

- Central Interface <Generics> <Entity,Key>
- CrudRepository
  - CRUD functionality for the entity class that is being managed.

```
public interface CrudRepository<T, ID extends Serializable>
                                        extends Repository<T, ID> {

        <S extends T> S save(S entity);
        T findOne(ID primaryKey);
        Iterable<T> findAll();
        Long count();
        void delete(T entity);
        boolean exists(ID primaryKey);
        // … more functionality omitted.
```



DATA REPOSITORY

DESIGN PATTERN BY SPRING

Query methods over declared interface

```
public interface PersonRepository extends Repository<User, Long> { … }
```

Declare query methods

```
List<Person> findByLastname(String lastname);
```

Usage

```
public class SomeClient {
  @Autowired
  private PersonRepository repository;

  public void doSomething() {
    List<Person> persons = repository.findByLastname("Matthews");
  }
}
```

DATA REPOSITORY

DESIGN PATTERN BY SPRING

Queries

```
public interface PersonRepository extends Repository<User, Long> {

  List<Person> findByEmailAddressAndLastname(EmailAddress emailAddress, String lastname);

  // Enables the distinct flag for the query
  List<Person> findDistinctPeopleByLastnameOrFirstname(String lastname, String firstname);
  List<Person> findPeopleDistinctByLastnameOrFirstname(String lastname, String firstname);

  // Enabling ignoring case for an individual property
  List<Person> findByLastnameIgnoreCase(String lastname);
  // Enabling ignoring case for all suitable properties
  List<Person> findByLastnameAndFirstnameAllIgnoreCase(String lastname, String firstna

  // Enabling static ORDER BY for a query
  List<Person> findByLastnameOrderByFirstnameAsc(String lastname);
  List<Person> findByLastnameOrderByFirstnameDesc(String lastname);
```

DATA REPOSITORY

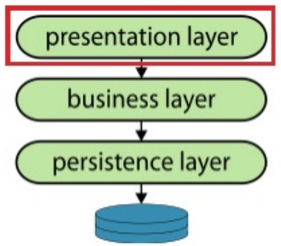
DESIGN PATTERN BY SPRING

Using Pageable and Sort in query methods

```
Page<User> findByLastname(String lastname, Pageable pageable);

List<User> findByLastname(String lastname, Sort sort);

List<User> findByLastname(String lastname, Pageable pageable);
```

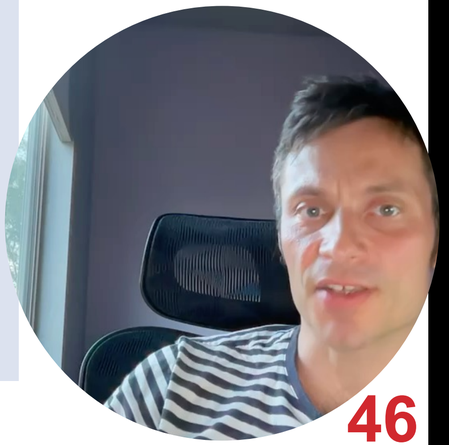
DATA REPOSITORY

DESIGN PATTERN BY SPRING



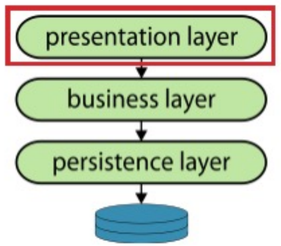
Sample usage

```
@Controller
@RequestMapping("/users")
public class UserController {


  @Autowired UserRepository repository;


  @RequestMapping
  public String showUsers(Model model, Pageable pageable) {


    model.addAttribute("users", repository.findAll(pageable));
    return "users";
  }
}
```



DATA REPOSITORY

DESIGN PATTERN BY SPRING



Sample usage

```
@Controller
@RequestMapping("/users")
public class UserController {
  private final UserRepository userRepository;
  @Autowired
  public UserController(UserRepository userRepository) {
        this.userRepository = userRepository;
  }
  @RequestMapping("/{id}")
  public String showUserForm(@PathVariable("id") Long id, Model model) {
        User user = userRepository.findOne(id);
        model.addAttribute("user", user);
        return "user";
  }
}
```

DATA REPOSITORY
DESIGN PATTERN BY DELTASPIKE

Sample usage POM.XML

```
<dependency>
    <groupId>org.apache.deltaspike.modules</groupId>
    <artifactId>deltaspike-data-module-api</artifactId>
    <version>${deltaspike.version}</version>
    <scope>compile</scope>
</dependency>


<dependency>
     <groupId>org.apache.deltaspike.modules</groupId>
     <artifactId>deltaspike-data-module-impl</artifactId>
     <version>${deltaspike.version}</version>
     <scope>runtime</scope>
</dependency>
```

DATA REPOSITORY
DESIGN PATTERN BY DELTASPIKE

Sample usage

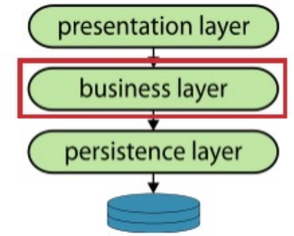
```
public interface EntityRepository<E, PK extends Serializable> {
  E save(E entity);
  void remove(E entity);
  void refresh(E entity);
  void flush();
  E findBy(PK primaryKey);
  List<E> findAll();
  List<E> findBy(E example, SingularAttribute<E, ?>... attributes);
  List<E> findByLike(E example, SingularAttribute<E, ?>... attributes);
  Long count();
  Long count(E example, SingularAttribute<E, ?>... attributes);
  Long countLike(E example, SingularAttribute<E, ?>... attributes); }
```

DATA REPOSITORY
DESIGN PATTERN BY DELTASPIKE

Sample usage

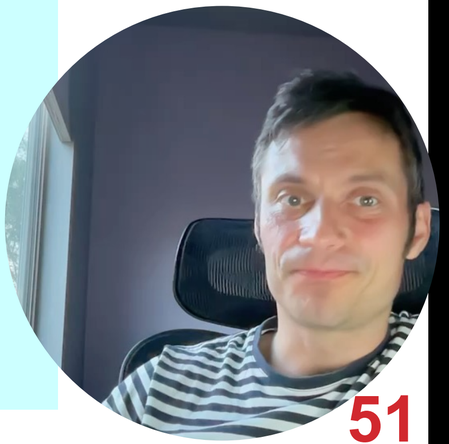
```
@Repository
 public interface PersonRepository extends EntityRepository<Person, Long> {


    List<Person> findByAgeBetweenAndGender(int minAge, int maxAge, Gender g);


    @Query("select p from Person p where p.ssn = ?1")
    Person findBySSN(String ssn);


    @Query(named=Person.BY_FULL_NAME)
    Person findByFullName(String firstName, String lastName);


    Person findBySsn(String ssn);
}
```

DATA REPOSITORY
DESIGN PATTERN BY DELTASPIKE



Sample usage : Its an injectable component

```
@Stateless
public class MyService {


    @Inject
    private PersonRepositorypersonRepository;


    List<Person> result = personRepository.findAllByAge(18, 65)
        .orderAsc("p.lastName", false).orderDesc("p.age”,false)
        .lockMode(LockModeType.WRITE)
        .hint("org.hibernate.timeout", Integer.valueOf(10))
        .getResultList();


   // Query API style
   QueryResult<Person> paged = personRepository.findByAge(age)
           .maxResults(10)  .firstResult(50);
```



NAMED QUERY HINT

1. Go to `Member.java`

```
@Entity
@XmlRootElement
@Table(uniqueConstraints = @UniqueConstraint(columnNames = "email"))
@NamedQueries({
    @NamedQuery(name = ”Member.findAll", query = "SELECT m FROM Member m"),
})
public class Member implements Serializable { ..
```

2. Go to `MemberRepository.java` replace `findAllOrderedByName`

```
public List<Member> findAllOrderedByName() {
    return em.createNamedQuery("Member.findAll", Member.class).getResultList(
}
```

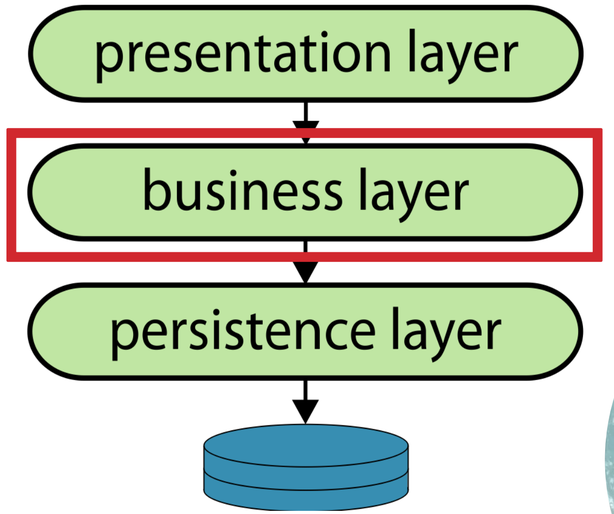
DATA REPOSITORY
DESIGN PATTERN

Details:

- **Spring**
  - http://docs.spring.io/spring-data/data-commons/docs/1.6.1.RELEASE/reference/html/repositories.html

- **DeltaSpike**
  - https://deltaspike.apache.org/documentation/data.html

BUSINESS LAYER OF EAA



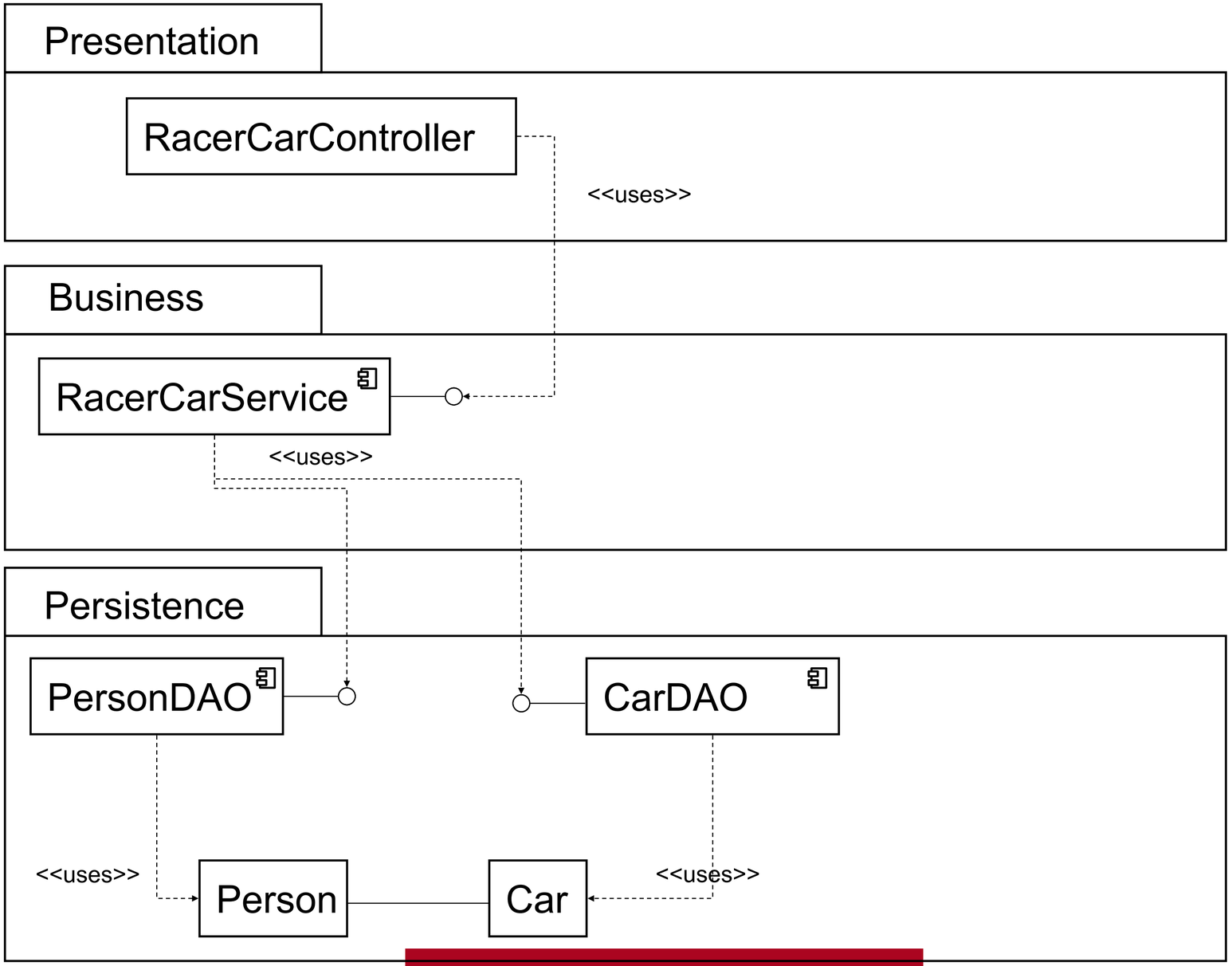
BUSINESS LAYER OF EAA

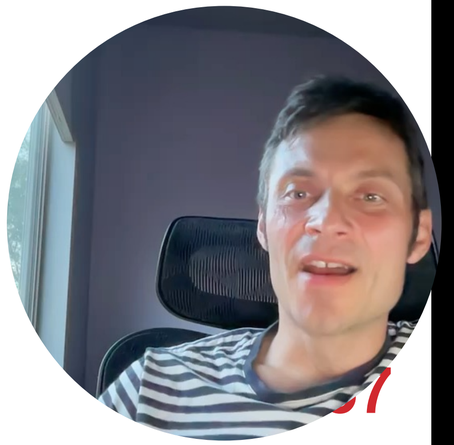
This layer embeds business logic or domain logic. This logic is part of the application that encodes the real-world business rules that determine how data can be created, displayed, stored, and changed.

Defines an application's boundary with a layer of services that establishes a set of available operations and coordinates the application's response in each operation. Enforces constraints, security, validation, work-flow, transaction, concurrency, internal event handling, etc.

Link to details

BUSINESS LAYER OF EAA





EJB EXAMPLE

```
@Stateless
public class DeliveryService {

   @PersistenceContext
    EntityManager em;

    public void deliver(Order order){
       System.out.println("Delivered: " + order);
       order.setDelivered(true);
    }
    public Order status(long orderId) {
       Order found = this.em.find(Order.class, orderId);
       if(found == null) { found = new Order(); }
       return found;
    }
}
```

EJB EXAMPLE

```
@Stateless
public class DeliveryService {

   @PersistenceContext
    EntityManager em;

    public void deliver(Order order) throws BusinessException {
       if (!order.isPaid()) {
            throw new BusinessException (“unpaid yet”);
       }
       System.out.println("Delivered: " + order);
       order.setDelivered(true);
    }
    public Order status(long orderId) {
       Order found = this.em.find(Order.class, orderId);
       if(found == null) { found = new Order(); }
       return found;
    }
}
```

Business logic, rules, policies!

Later expressed in OCL

BUSINESS LAYER OF EAA