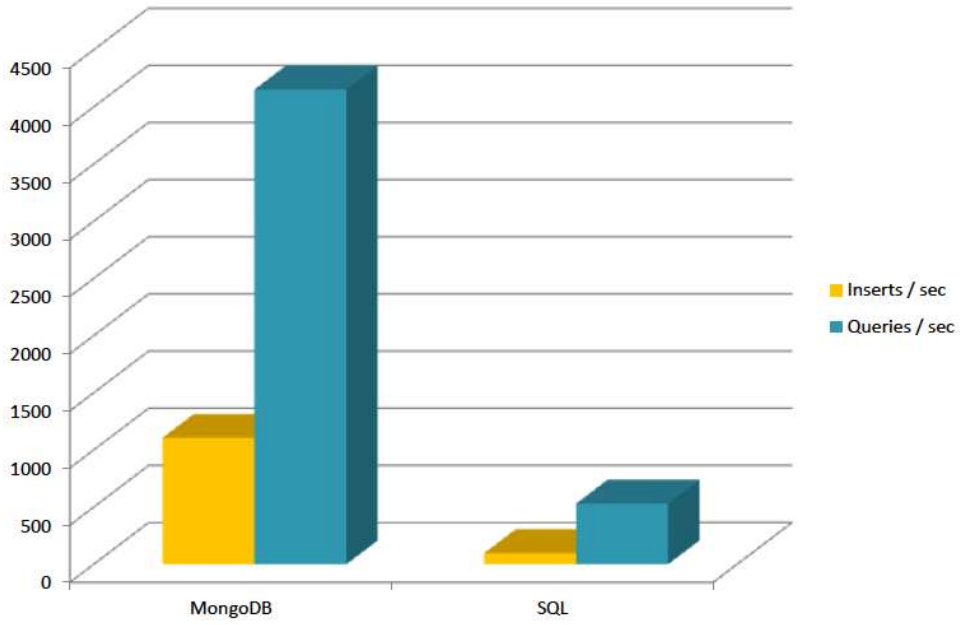
Lesson 5

MONGODB
DTO
REST CLIENT

MONGODB

MongoDB

- Document database
- Fast
- Can handle large datasets
  - Scalable



Document data model (JSON)

Relational - Tables

Customer ID	First Name	Last Name	City
0	John	Doe	New York
1	Mark	Smith	San Francisco
2	Jay	Black	Newark
3	Meagan	White	London
4	Edward	Daniels	Boston

Account Number	Branch ID	Account Type	Customer ID
10	100	Checking	0
11	101	Savings	0
12	101	IRA	0
13	200	Checking	1
14	200	Savings	1
15	201	IRA	2

Document - Collections

```
{   customer_id : 1,
    first_name : "Mark",
    last_name : "Smith",
    city : "San Francisco",
    accounts : [   {
        account_number : 13,
        branch_ID : 200,
        account_type : "Checking"
    },
    {   account_number : 14,
        branch_ID : 200,
        account_type : ”IRA”,
        beneficiaries: […]
    } ]
}
```

Documents are flexible

```
{
        category: bat,
        model: B1403E,
        name: Air Elite,
        brand: “Rip-IT”,
        price: 399.99

        diameter: “2 5/8”,
        barrel: R2 Alloy,
        handle: R2
}
```

```
{
        category: glove,
        model: PRO112PT,
        name: Air Elite,
        brand: “Rawlings”,
        price: “229.99”

        size: 11.25,
        position: outfield,
        pattern: “Pro taper”,
        material: leather,
        color: black
}
```

Insert a document

collection

document

```
db.artists.insert({ artistname: "Jorn Lande" })
```

This inserts a document with `{ artistname: "Jorn Lande" }` as its contents.

Primary key

- _id field as the primary key
- If you don't add a field name with the _id in the field name, then MongoDB will automatically create it

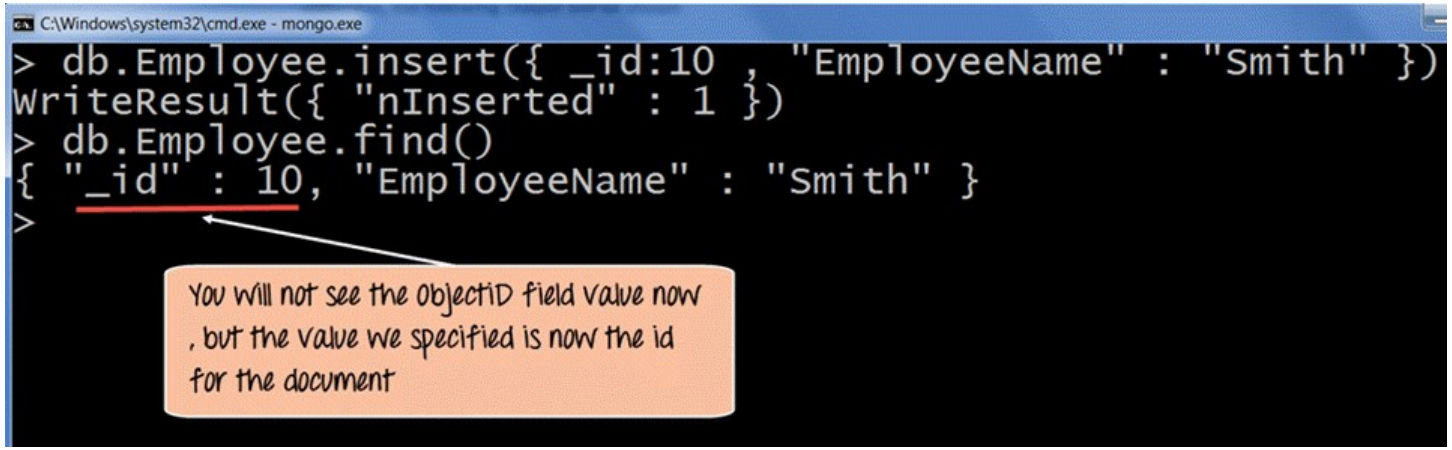
```
> db.Employee.find().forEach(printjson);
{
        "_id" : ObjectId("563479cc8a8a4246bd27d784"),
        "Employeeid" : 1,
        "EmployeeName" : "Smith"
}
{
        "_id" : ObjectId("563479d48a8a4246bd27d785"),
        "Employeeid" : 2,
        "EmployeeName" : "Mohan"
}
{
        "_id" : ObjectId("563479df8a8a4246bd27d786"),
        "Employeeid" : 3,
        "EmployeeName" : "Joe"
}
>
```

Every row has a unique object id

Set _id explicit

_id

```
db.Employee.insert({_id:10, "EmployeeName" : "Smith"})
```



Insert embedded documents

```
db.artists.insert({
    artistname : "Deep Purple",
    albums : [
                {
                    album : "Machine Head",
                    year : 1972,
                    genre : "Rock"
                },
                {
                    album : "Stormbringer",
                    year : 1974,
                    genre : "Rock"
                }
            ]
})
```

Result:

```
WriteResult({ "nInserted" : 1 })
```

Find all document

```
> db.artists.find()
{ "_id" : ObjectId("5780fbf948ef8c6b3ffb0149"), "artistname" : "The
Tea Party" }
{ "_id" : ObjectId("5781c9ac48ef8c6b3ffb014a"), "artistname" : "Jorn
Lande" }
```

Note that MongoDB has created an `_id` field for the documents. If you don't specify one, MongoDB will create one for you. However, you can provide this field when doing the insert if you prefer to have control over the value of the `_id` field.

```
db.artists.insert({ _id: 1, artistname: "AC/DC" })
```

Result:

```
> db.artists.find()
{ "_id" : ObjectId("5780fbf948ef8c6b3ffb0149"), "artistname" : "The
Tea Party" }
{ "_id" : ObjectId("5781c9ac48ef8c6b3ffb014a"), "artistname" : "Jorn
Lande" }
{ "_id" : 1, "artistname" : "AC/DC" }
```

Find with filter criteria

If we're only interested in Deep Purple from the `artists` collection:

```
db.artists.find({ artistname : "Deep Purple" })
```

Result:

```
{ "_id" : ObjectId("5781f85d48ef8c6b3ffb0150"), "artistname" : "Deep
Purple", "albums" : [ { "album" : "Machine Head", "year" : 1972,
"genre" : "Rock" }, { "album" : "Stormbringer", "year" : 1974,
"genre" : "Rock" } ] }
```

Find() method

- Find all users

```
db.users.find()
```

- Find all users with status A

```
db.users.find( { status: "A" } )
```

- Find all users where status is not A

```
db.users.find( { status: { $ne: "A" } } )
```

- Find all users where status = A and age = 50

```
db.users.find( { status: "A", age: 50 } )
```

Find() method

- Find all users where status = A or age = 50

```
db.users.find( { $or: [ { status: "A" } , { age: 50 } ] } )
```

- Find all users with age > 25

```
db.users.find( { age: { $gt: 25 } } )
```

- Find all users with age < 25

```
db.users.find( { age: { $lt: 25 } } )
```

- Find all users with age > 25 and age <= 50

```
db.users.find( { age: { $gt: 25, $lte: 50 } } )
```

MONGODB WITH SPRING BOOT

MongoDB dependency

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-mongodb</artifactId>
</dependency>
```

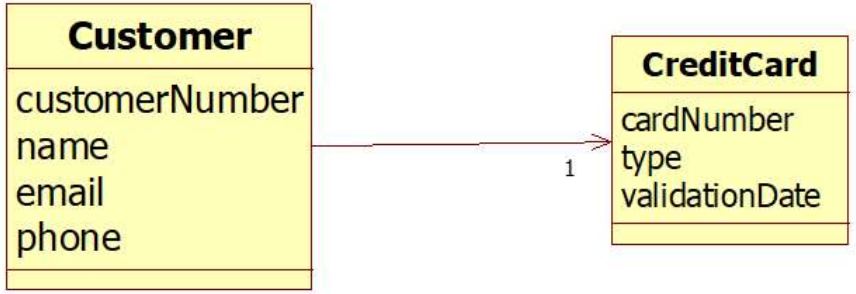
Customer

```
@Document
public class Customer {
  @Id
  private int customerNumber;
  private String name;
  private String email;
  private String phone;
  private CreditCard creditCard;

...
```

@Document

@Id

```
public class CreditCard {
   private String cardNumber;
   private String type;
   private String validationDate;

...
```



Repository

interface

document

Type of the id

```
@Repository
public interface CustomerRepository extends MongoRepository<Customer, Integer> {
   Customer findByPhone(String phone);

   @Query("{'email' : ?0}")
   Customer findCustomerWithPhone(String email);

   @Query("{'creditCard.type' : ?0}")
   List<Customer>  findCustomerWithCreditCardType(String cctype);
}
```

Supported keywords

Keyword	Sample	JPQL snippet
And	findByLastnameAndFirstname	… where x.lastname = ?1 and x.firstname = ?2
Or	findByLastnameOrFirstname	… where x.lastname = ?1 or x.firstname = ?2
Between	findByStartDateBetween	… where x.startDate between 1? and ?2
LessThan	findByAgeLessThan	… where x.age < ?1
GreaterThan	findByAgeGreaterThan	… where x.age > ?1
After	findByStartDateAfter	… where x.startDate > ?1
Before	findByStartDateBefore	… where x.startDate < ?1
IsNull	findByAgeIsNull	… where x.age is null
IsNotNull,NotNull	findByAge(Is)NotNull	… where x.age not null
Like	findByFirstnameLike	… where x.firstname like ?1
NotLike	findByFirstnameNotLike	… where x.firstname not like ?1
StartingWith	findByFirstnameStartingWith	… where x.firstname like ?1 (parameter bound with appended %)
EndingWith	findByFirstnameEndingWith	… where x.firstname like ?1 (parameter bound with prepended %)
Containing	findByFirstnameContaining	… where x.firstname like ?1 (parameter bound wrapped in %)
OrderBy	findByAgeOrderByLastnameDesc	… where x.age = ?1 order by x.lastname desc
Not	findByLastnameNot	… where x.lastname <> ?1
In	findByAgeIn(Collection<Age> ages)	… where x.age in ?1
NotIn	findByAgeNotIn(Collection<Age> age)	… where x.age not in ?1
True	findByActiveTrue()	… where x.active = true
False	findByActiveFalse()	… where x.active = false

application.properties

```
spring.data.mongodb.host=localhost
spring.data.mongodb.port=27017
spring.data.mongodb.database=testdb

logging.level.root=ERROR
logging.level.org.springframework=ERROR
```

Mongo configuration

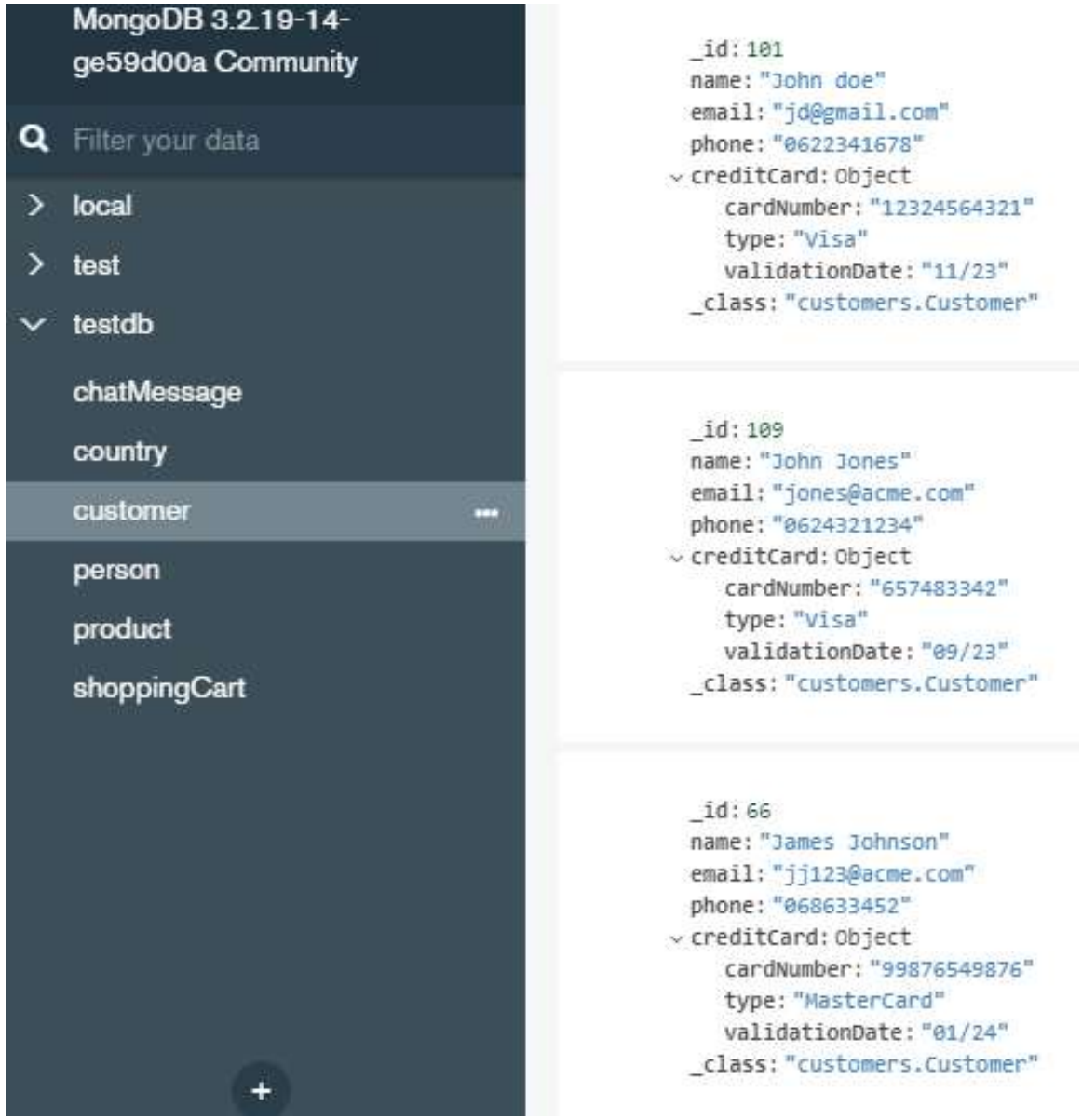
Application (1/2)

```
public void run(String... args) throws Exception {
 // create customer
Customer customer = new Customer(101,"John doe", "johnd@acme.com", "0622341678");
CreditCard creditCard = new CreditCard("12324564321", "Visa", "11/23");
customer.setCreditCard(creditCard);
customerRepository.save(customer);

customer = new Customer(109,"John Jones", "jones@acme.com", "0624321234");
creditCard = new CreditCard("657483342", "Visa", "09/23");
customer.setCreditCard(creditCard);
customerRepository.save(customer);

customer = new Customer(66,"James Johnson", "jj123@acme.com", "068633452");
creditCard = new CreditCard("99876549876", "MasterCard", "01/24");
customer.setCreditCard(creditCard);
customerRepository.save(customer);

//get customers
System.out.println(customerRepository.findById(66).get());
System.out.println(customerRepository.findById(101).get());
```

Application (2/2)

```
System.out.println("-----------All customers ----------------");
System.out.println(customerRepository.findAll());

//update customer
customer = customerRepository.findById(101).get();
customer.setEmail("jd@gmail.com");
customerRepository.save(customer);

System.out.println("-----------find by phone ----------------");
System.out.println(customerRepository.findByPhone("0622341678"));

System.out.println("-----------find by email ----------------");
System.out.println(customerRepository.findCustomerWithPhone("jj123@acme.com"));

System.out.println("-----------find customers with a certain type of creditcard ----------------");
List<Customer> customers = customerRepository.findCustomerWithCreditCardType("Visa");
for (Customer cust : customers){
  System.out.println(cust);
}

}
```

MongoDB

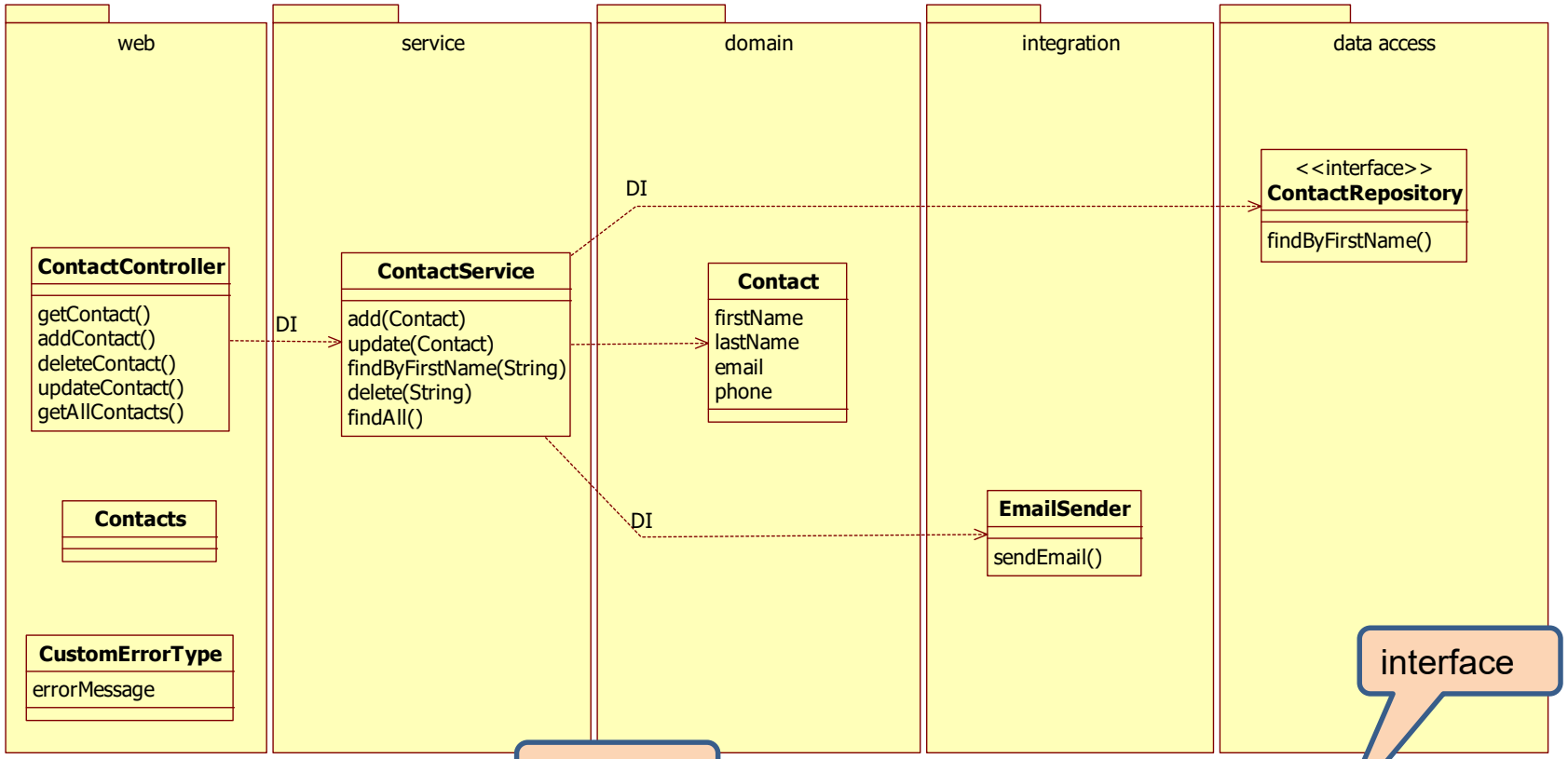

MongoDB 3.2.19-14-ge59d00a Community
Filter your data
local
test
testdb
chatMessage
country
customer
person
product
shoppingCart
_id: 101
name: "John doe"
email: "jd@gmail.com"
phone: "0622341678"
creditCard: Object
cardNumber: "12324564321"
type: "Visa"
validationDate: "11/23"
_class: "customers.Customer"
_id: 109
name: "John Jones"
email: "jones@acme.com"
phone: "0624321234"
creditCard: Object
cardNumber: "657483342"
type: "Visa"
validationDate: "09/23"
_class: "customers.Customer"
_id: 66
name: "James Johnson"
email: "jj123@acme.com"
phone: "068633452"
creditCard: Object
cardNumber: "99876549876"
type: "MasterCard"
validationDate: "01/24"
_class: "customers.Customer"

Main point

- MongoDB is a fast document database that has no schema. *The unified field is the home of all the laws of Nature.*

REST + MONGODB REPOSITORY

Contact and ContactRepository



Document

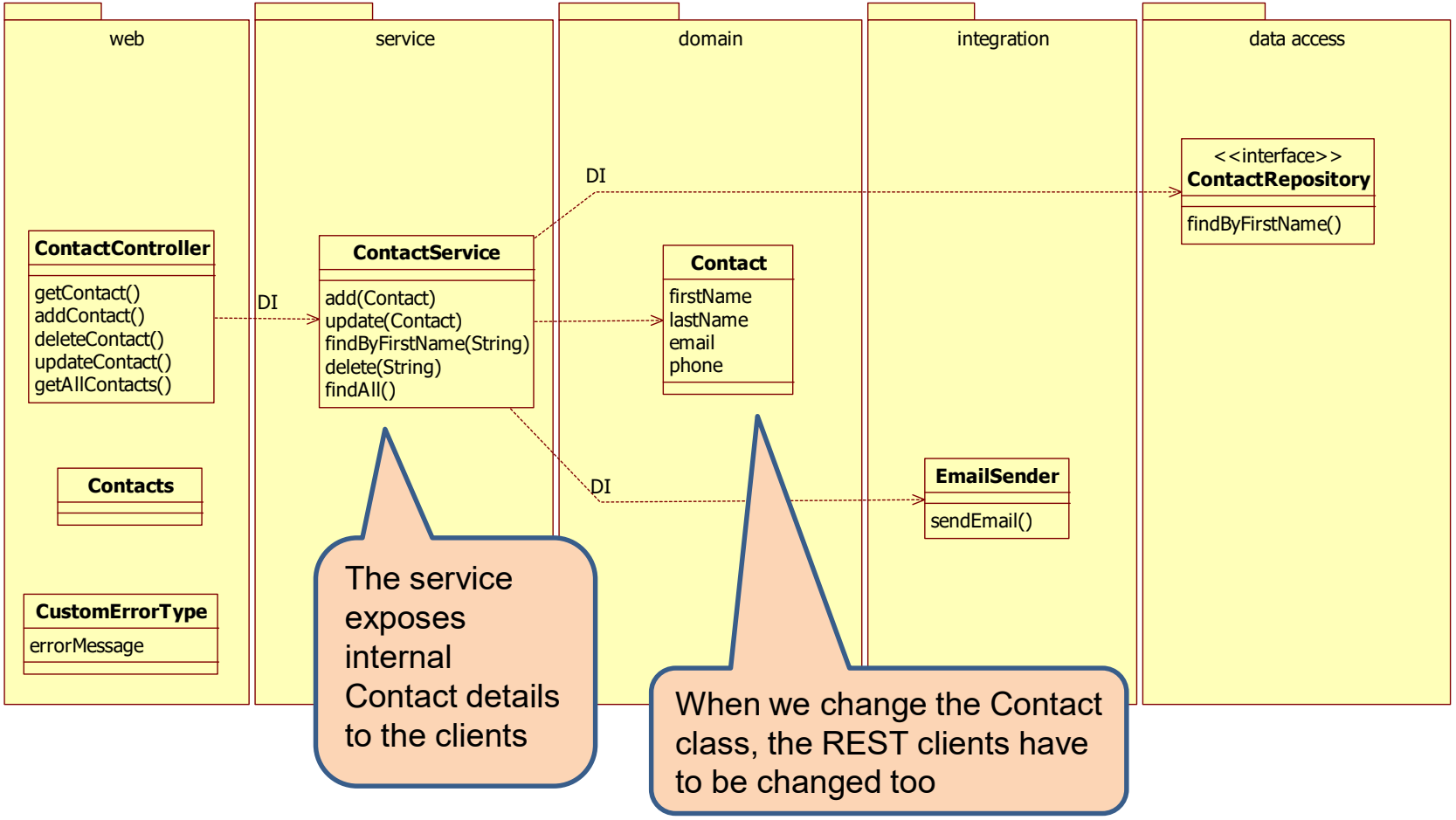
```
@Document
public class Contact {
   @Id
   private String firstName;
   private String lastName;
   private String email;
   private String phone;
…
```

interface

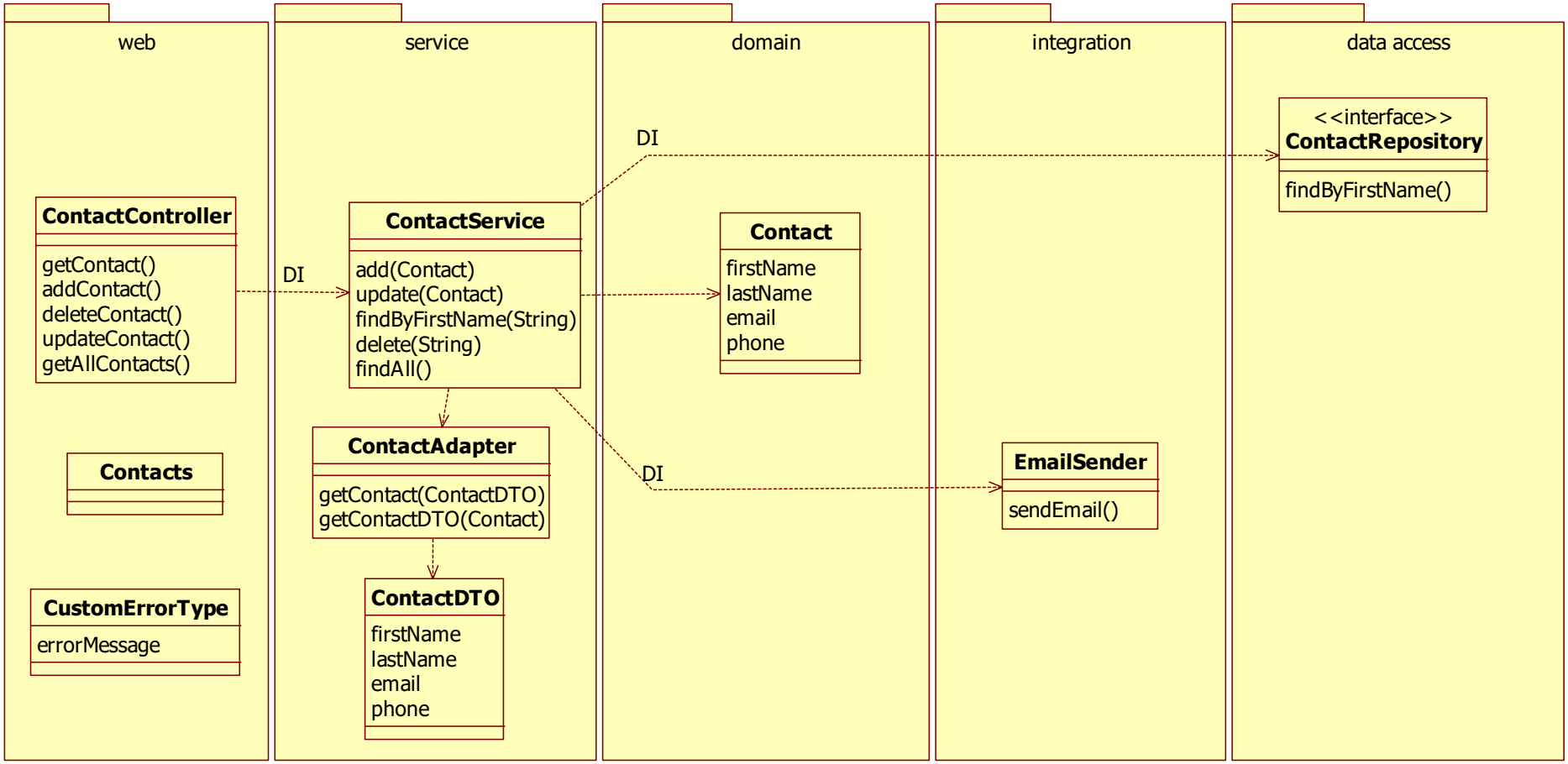
```
public interface ContactRepository extends MongoRepository<Contact, String> {

   Contact findByFirstName(String firstName);
}
```

DATA TRANSFER OBJECT (DTO)

Internal details on the interface



Solution: DTO objects



ContactDTO and ContactAdapter

```java
@Document
public class Contact {
    @Id
    private String firstName;
    private String lastName;
    private String email;
    private String phone;
...
```

```java
public class ContactDTO {
    private String firstName;
    private String lastName;
    private String email;
    private String phone;
...
```

```java
public class ContactAdapter {

    public static Contact getContact(ContactDTO contactDto){
        Contact contact = new Contact();
        if (contactDto != null) {
            contact = new Contact(contactDto.getFirstName(),
                    contactDto.getLastName(),
                    contactDto.getEmail(),
                    contactDto.getPhone());
        }
        return contact;
    }

    public static ContactDTO getContactDTO(Contact contact){
        ContactDTO contactDto = new ContactDTO();
        if (contact != null) {
            contactDto = new ContactDTO(contact.getFirstName(),
                    contact.getLastName(),
                    contact.getEmail(),
                    contact.getPhone());
        }
        return contactDto;
    }
}
```

Convert ContactDTO to Contact

Convert Contact to ContactDTO

ContactService (1/2)

```
@Service
public class ContactService {
   @Autowired
   ContactRepository contactRepository;
   @Autowired
   EmailSender emailSender;

   public void add(ContactDTO contactDto){
      Contact contact = ContactAdapter.getContact(contactDto);
      contactRepository.save(contact);
      emailSender.sendEmail(contact.getEmail(), "Welcome");
   }

   public void update(ContactDTO contactDto){
      Contact contact = ContactAdapter.getContact(contactDto);
      contactRepository.save(contact);
   }

   public ContactDTO findByFirstName(String firstName){
      Contact contact = contactRepository.findByFirstName(firstName);
      return ContactAdapter.getContactDTO(contact);
   }
```

Receive ContactDTO

Receive ContactDTO

Return ContactDTO

ContactService (2/2)

```
    public void delete(String firstName){
        Contact contact = contactRepository.findByFirstName(firstName);
        emailSender.sendEmail(contact.getEmail(), "Good By");
        contactRepository.delete(contact);
    }

    public Collection<ContactDTO> findAll(){
        Collection<Contact> contacts = contactRepository.findAll();
        Collection<ContactDTO> contactDTOs = new ArrayList<ContactDTO>();
        for (Contact contact : contacts){
            contactDTOs.add(ContactAdapter.getContactDTO(contact));
        }
        return contactDTOs;
    }
}
```

Return a list of ContactDTO

ContactController (1/2)

```
@Controller
public class ContactController {
   @Autowired
   private ContactService contactService;


   @GetMapping("/contacts/{firstName}")
   public ResponseEntity<?> getContact(@PathVariable String firstName) {
      ContactDTO contactDto = contactService.findByFirstName(firstName);
      if (contactDto.getFirstName() == null) {
         return new ResponseEntity<CustomErrorType>(new CustomErrorType("Contact with firstname= "
               + firstName + " is not available"), HttpStatus.NOT_FOUND);
      }
      return new ResponseEntity<ContactDTO>(contactDto, HttpStatus.OK);
   }

   @PostMapping("/contacts")
   public ResponseEntity<?> addContact(@RequestBody ContactDTO contactDto) {
      contactService.add(contactDto);
      return new ResponseEntity<ContactDTO>(contactDto, HttpStatus.OK);
   }
```

Return ContactDTO

Receive ContactDTO

Return ContactDTO

ContactController (2/2)

```
@DeleteMapping("/contacts/{firstName}")
  public ResponseEntity<?> deleteContact(@PathVariable String firstName) {
    ContactDTO contact = contactService.findByFirstName(firstName);
    if (!contact.getFirstName().equals(firstName)) {
      return new ResponseEntity<CustomErrorType>(new CustomErrorType("Contact with firstname= " + firstName + " is
not available"),HttpStatus.NOT_FOUND);
    }
    contactService.delete(firstName);
    return new ResponseEntity<>(HttpStatus.NO_CONTENT);
  }

  @PutMapping("/contacts/{firstName}")
  public ResponseEntity<?> updateContact(@PathVariable String firstName, @RequestBody ContactDTO contactDto) {
    contactService.update(contactDto);
    return new ResponseEntity<ContactDTO>(contactDto, HttpStatus.OK);
  }

  @GetMapping("/contacts")
  public ResponseEntity<?> getAllContacts() {
    Contacts allcontacs = new Contacts(contactService.findAll());
    return new ResponseEntity<Contacts>(allcontacs, HttpStatus.OK);
  }
}
```

Receive ContactDTO

Return ContactDTO

Return wrapped list of ContactDTO's

Main point

- A back-end application should never expose its internal classes. *It is the birthright of every human being to be able to contact and live on the level of pure consciousness*

REST CLIENT

Web dependency

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
  <exclusions>
    <exclusion>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-tomcat</artifactId>
    </exclusion>
  </exclusions>
</dependency>
```

Do not start Tomcat

ContactController (2/2)

```
@SpringBootApplication
public class Application implements CommandLineRunner {
  @Autowired
  private RestOperations restTemplate;
  private String serverUrl = "http://localhost:8080/contacts";

  public static void main(String[] args) {
    SpringApplication.run(Application.class, args);
  }

  @Override
  public void run(String... args) throws Exception {
    ...
  }

  @Bean
  RestTemplate restTemplate(){
    return new RestTemplate();
  }
}
```

Inject the REST template

Run method

Create a REST template

ContactController (2/2)

```
@Override
public void run(String... args) throws Exception {
  // add a contact
  restTemplate.postForLocation(serverUrl, new ContactDTO("John","Doe", "jdoe@acme.com", "6739127563"));
  // add a contact
  restTemplate.postForLocation(serverUrl, new ContactDTO("Bob","Jones", "bobby@hotmail.com", "3214532563"));
   // get contact
  ContactDTO contact= restTemplate.getForObject(serverUrl+"/Bob", ContactDTO.class );
  System.out.println("--------------------Contact info from Bob----------------------");
  System.out.println(contact);
  // get all contacts
  Contacts contacts= restTemplate.getForObject(serverUrl, Contacts.class );
  System.out.println("--------------------All contacts----------------------");
  System.out.println(contacts);
  //delete a contact
  restTemplate.delete(serverUrl+"/Bob");
  // update contact
  contact.setEmail("bjones@gmail.com");
  restTemplate.put(serverUrl+"/Bob", contact);
 }

}
```

Use the RestTemplate

post

get

delete

put

Main point

- In a Spring Boot application, you can call a remote REST server using a RestTemplate. *The TM technique makes it easy and effortless to access pure consciousness.*

Connecting the parts of knowledge with the wholeness of knowledge

1. A repository in a Spring Boot application is a simple interface. Spring will create the implementation class.
2. The coupling between client and server are on the level of DTO classes.

---

3. **Transcendental consciousness** is the source of all contexts.
4. **Wholeness moving within itself:** In Unity Consciousness, one realizes that everything in creation are just expressions of the field of Pure Intelligence.