Lesson 4

REST VALIDATION
BACK-END DESIGN

ERROR HANDLING

Calculator

```
@Controller
public class CalcController {

  @PostMapping("/calc")
  public ResponseEntity<?> calculate(@RequestBody Calculation calculation) {
    double result=0.0;

    switch(calculation.getOperation()){
      case "+" : {result = calculation.getNumber1() + calculation.getNumber2(); break;}
      case "-" : {result = calculation.getNumber1() - calculation.getNumber2(); break;}
      case "*" : {result = calculation.getNumber1() * calculation.getNumber2(); break;}
      case "/" : {result = calculation.getNumber1() / calculation.getNumber2(); break;}
    }
    CalculationResult calculationResult = new CalculationResult(calculation.getNumber1(),
            calculation.getNumber2(),calculation.getOperation(), result);
    return new ResponseEntity<CalculationResult>(calculationResult, HttpStatus.OK);
  }
}
```

```
public class Calculation {

  private int number1;
  private int number2;
  private String operation;
...
```

```
public class CalculationResult {
  private int number1;
  private int number2;
  private String operation;
  private double result;
...
```

Calculator

POST localhost:8080/calc

Params Authorization Headers (9) Body

none form-data x-www-form-urlencoded

```
1 {
2     "number1": "3",
3     "number2": "6",
4     "operation": "+"
5 }
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2     "number1": 3,
3     "number2": 6,
4     "operation": "+",
5     "result": 9.0
6 }
```

Divide by zero

POST localhost:8080/calc

Params Authorization Headers (9) Body Pre-request

none form-data x-www-form-urlencoded raw

```
1 {
2     "number1": "3",
3     "number2": "0",
4     "operation": "/"
5 }
```

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize JSON

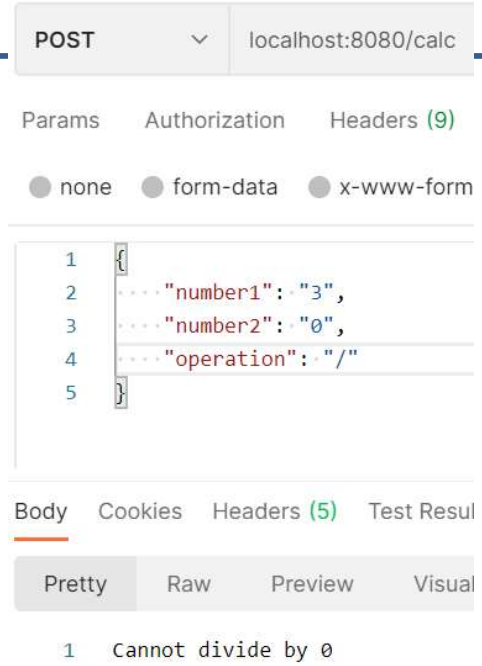
```
1 {
2     "timestamp": "2021-06-14T14:33:00.044+00:00",
3     "status": 500,
4     "error": "Internal Server Error",
5     "message": "",
6     "path": "/calc"
7 }
```

Global exception handler

```
@ControllerAdvice
public class CustomExceptionHandler extends ResponseEntityExceptionHandler {

    @ExceptionHandler(value = { ArithmeticException.class})
    protected ResponseEntity<Object> handleConflict(RuntimeException ex, WebRequest request) {
        String bodyOfResponse = "Cannot divide by 0";
        return handleExceptionInternal(ex, bodyOfResponse, new HttpHeaders(), HttpStatus.CONFLICT, request);
    }
}
```

Array of exceptions



VALIDATION

Supporting XML and JSON

constraints

```
public class Contact {

    @NotNull
    @Size(min=2, max=20)
    private String firstName;

    @NotNull
    @Size(min=2, max=20)
    private String lastName;

    @NotNull
    @Email
    private String email;

    @NotNull
    @Size(min=10, max=10)
    private String phone;
```

@Valid

```
@PostMapping("/contacts")
public ResponseEntity<?> addContact(@RequestBody @Valid Contact contact) {
    contacts.put(contact.getFirstName(), contact);
    return new ResponseEntity<Contact>(contact, HttpStatus.OK);
}

@PutMapping("/contacts/{firstName}")
public ResponseEntity<?> updateContact(@PathVariable String firstName, @RequestBody @Valid Contact contact) {
    contacts.put(firstName, contact);
    return new ResponseEntity<Contact>(contact, HttpStatus.OK);
}
```

Validation result

POST localhost:8080/contacts/

Params Authorization Headers (9) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2     "firstName": "Leo34",
3     "lastName": "Jones",
4     "email": "jdoegmail.com",
5     "phone": "65298765"
6 }
```

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2     "timestamp": "2021-06-14T10:00:25.990+00:00",
3     "status": 400,
4     "error": "Bad Request",
5     "message": "",
6     "path": "/contacts/"
7 }
```

Bad request

Handle validation errors in the method

```
@PostMapping("/contacts")
public ResponseEntity<?> addContact(@RequestBody @Valid Contact contact, Errors errors) {
  if (errors.hasErrors()) {
    Map<String, Object> fieldError = new HashMap<>();
    List<FieldError> fieldErrors= errors.getFieldErrors();
    for (FieldError error : fieldErrors) {
      fieldError.put(error.getField(), error.getDefaultMessage());
    }

    Map<String, Object> map = new HashMap<>();
    map.put("isSuccess", false);
    map.put("data", null);
    map.put("status", HttpStatus.BAD_REQUEST);
    map.put("fieldError", fieldError);
    return new ResponseEntity<Object>(map,HttpStatus.BAD_REQUEST);
  }
  contacts.put(contact.getFirstName(), contact);
  return new ResponseEntity<Contact>(contact, HttpStatus.OK);
}
```

Map with all the errors

You have to do this for all methods with validation

Handle validation errors in the method

POST localhost:8080/contacts/

Params Authorization Headers (9) Body Pre-request Sc

none form-data x-www-form-urlencoded raw bi

```
1 {
2     "firstName": "Leo34",
3     "lastName": "Jones",
4     "email": "jdoegmail.com",
5     "phone": "65298765"
6 }
```

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2     "data": null,
3     "fieldError": {
4         "phone": "size must be between 10 and 10",
5         "email": "must be a well-formed email address"
6     },
7     "isSuccess": false,
8     "status": "BAD_REQUEST"
9 }
```

Handle validation errors in the controller

@ExceptionHandler

```
@ExceptionHandler(MethodArgumentNotValidException.class)
public ResponseEntity<Object> handleValidationExceptions(MethodArgumentNotValidException ex) {
   Map<String, Object> fieldError = new HashMap<>();
   List<FieldError> fieldErrors= ex.getBindingResult().getFieldErrors();
   for (FieldError error : fieldErrors) {
      fieldError.put(error.getField(), error.getDefaultMessage());
   }

   Map<String, Object> map = new HashMap<>();
   map.put("isSuccess", false);
   map.put("data", null);
   map.put("status", HttpStatus.BAD_REQUEST);
   map.put("fieldError", fieldError);
   return new ResponseEntity<Object>(map,HttpStatus.BAD_REQUEST);
}
```

This method is called for all validation errors in this controller

You have to do this for all controllers

Handle validation errors in the controller

POST localhost:8080/contacts/

Params Authorization Headers (9) Body Pre-request Script Tes

none form-data x-www-form-urlencoded raw binary G

```
1  {
2      "firstName": "Leo35",
3      "lastName": "J",
4      "email": "jdoe",
5      "phone": "6529876"
6  }
```

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize JSON

```
1  {
2      "data": null,
3      "fieldError": {
4          "lastName": "size must be between 2 and 20",
5          "phone": "size must be between 10 and 10",
6          "email": "must be a well-formed email address"
7      },
8      "isSuccess": false,
9      "status": "BAD_REQUEST"
10 }
```

Handle validation errors for all controllers

@ControllerAdvice

```
@ControllerAdvice
public class CustomExceptionHandler extends ResponseEntityExceptionHandler {

  protected ResponseEntity<Object> handleMethodArgumentNotValid(MethodArgumentNotValidException ex,
          HttpHeaders headers, HttpStatus status, WebRequest request) {

    Map<String, Object> fieldError = new HashMap<>();
    List<FieldError> fieldErrors= ex.getBindingResult().getFieldErrors();
    for (FieldError error : fieldErrors) {
      fieldError.put(error.getField(), error.getDefaultMessage());
    }

    Map<String, Object> map = new HashMap<>();
    map.put("isSuccess", false);
    map.put("data", null);
    map.put("status", HttpStatus.BAD_REQUEST);
    map.put("fieldError", fieldError);
    return new ResponseEntity<Object>(map,HttpStatus.BAD_REQUEST);
  }
}
```

This method is called for all validation errors in all controllers

Handle validation errors for all controllers

POST localhost:8080/contacts/

Params Authorization Headers (9) Body Pre-request Script Tes

none form-data x-www-form-urlencoded raw binary G

```
1  {
2      "firstName": "Leo35",
3      "lastName": "J",
4      "email": "jdoe",
5      "phone": "6529876"
6  }
```

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize JSON

```
1  {
2      "data": null,
3      "fieldError": {
4          "lastName": "size must be between 2 and 20",
5          "phone": "size must be between 10 and 10",
6          "email": "must be a well-formed email address"
7      },
8      "isSuccess": false,
9      "status": "BAD_REQUEST"
10 }
```

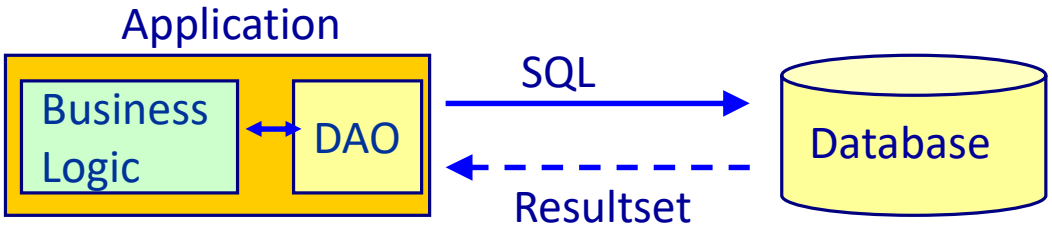
Main point

- The @ControllerAdvice class handles validation errors for all Rest controllers. *A quality of Cosmic Consciousness is the ability to know what is right in every situation and to handle every situation with maximum effectiveness.*

BACK-END DESIGN

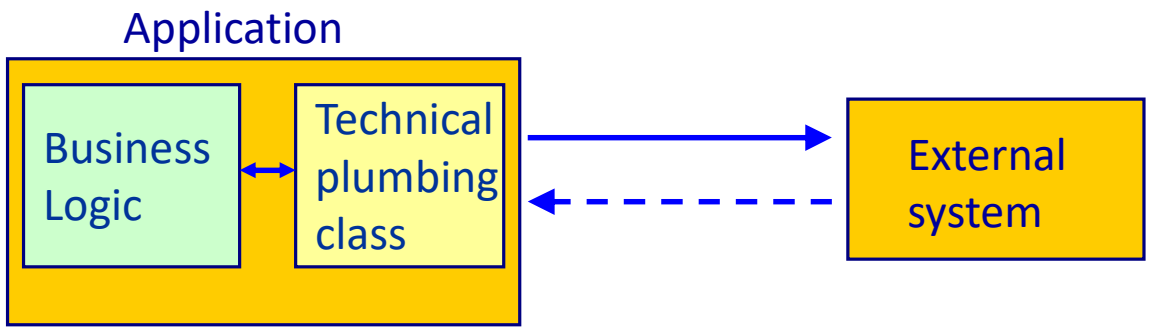
Data Access Object (DAO)/Repository

- Object that knows how to access the database
- Contains all database related logic
- Also called repository

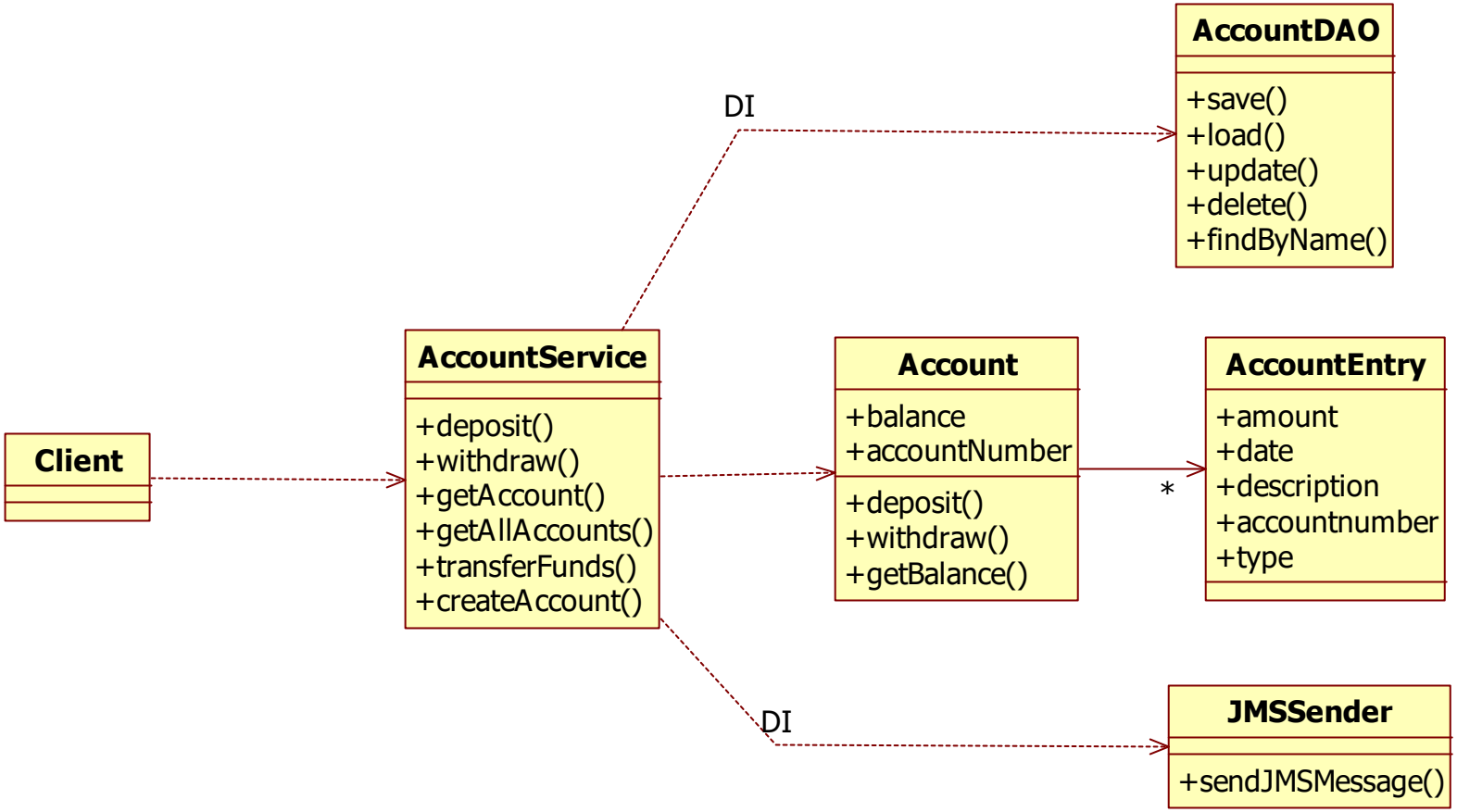


Technical plumbing classes

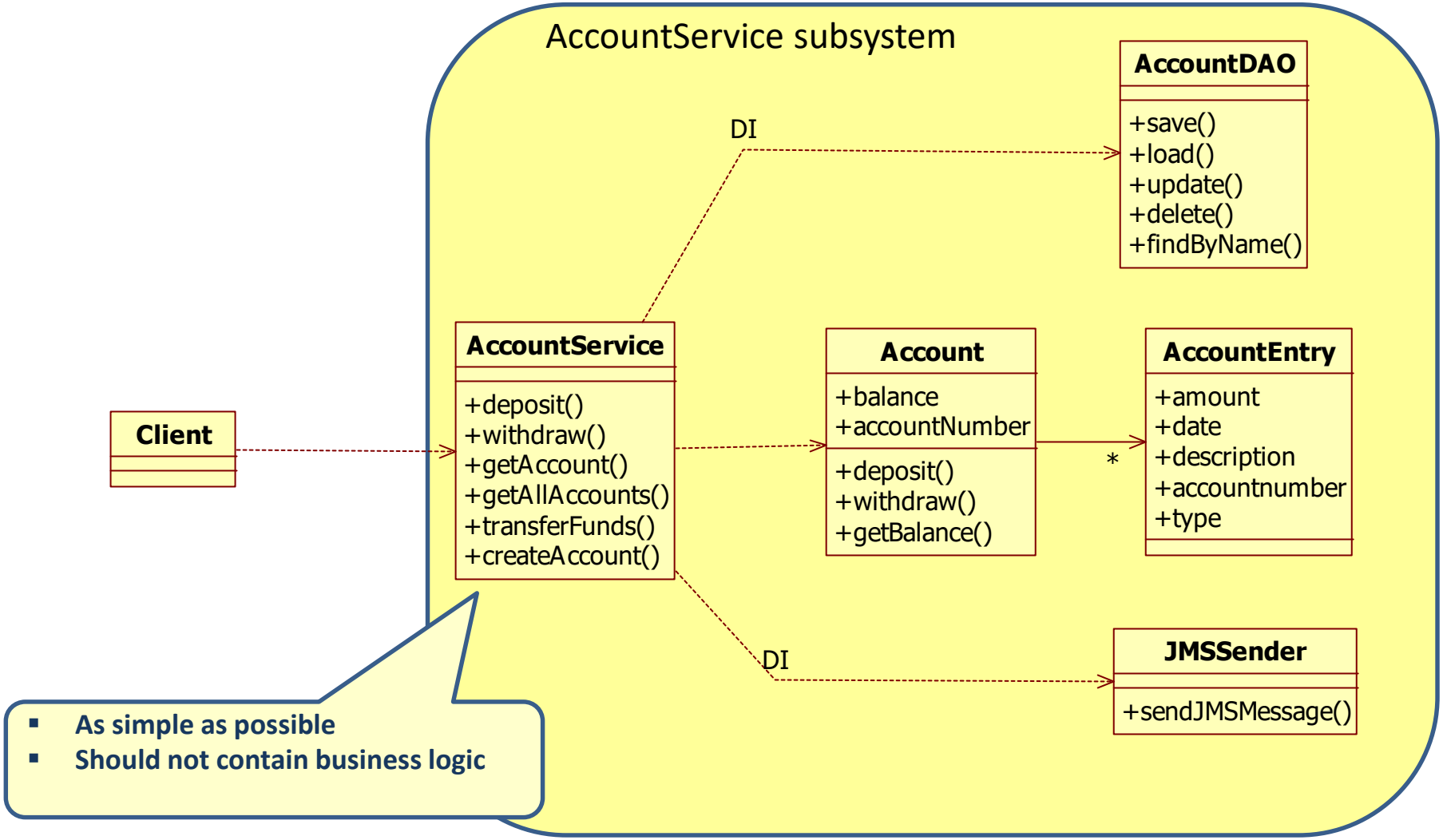
- Single responsibility
  - Web service
  - Remote calls
  - Messaging
  - Email
  - Logging



Service Object

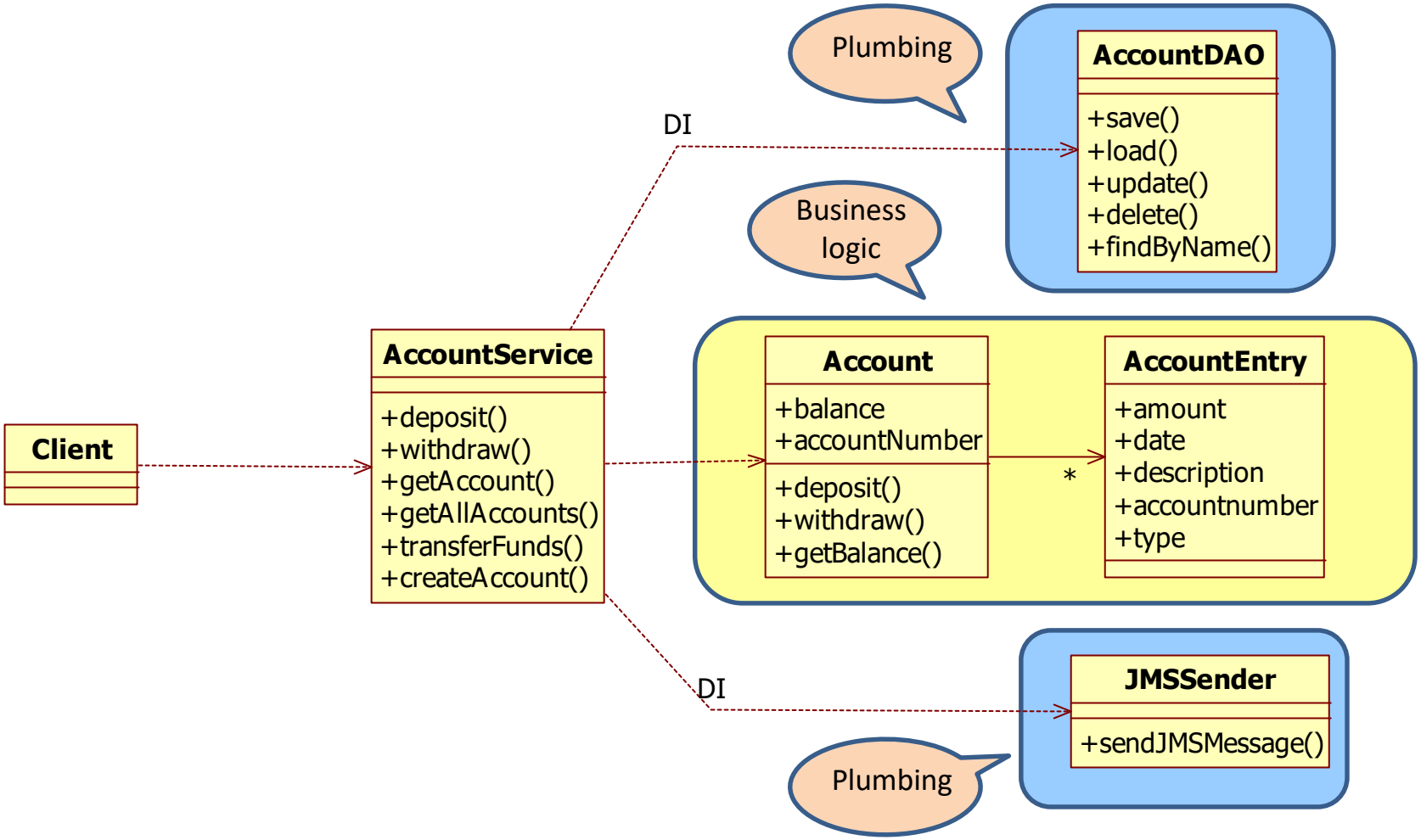


Entry of a complex subsystem

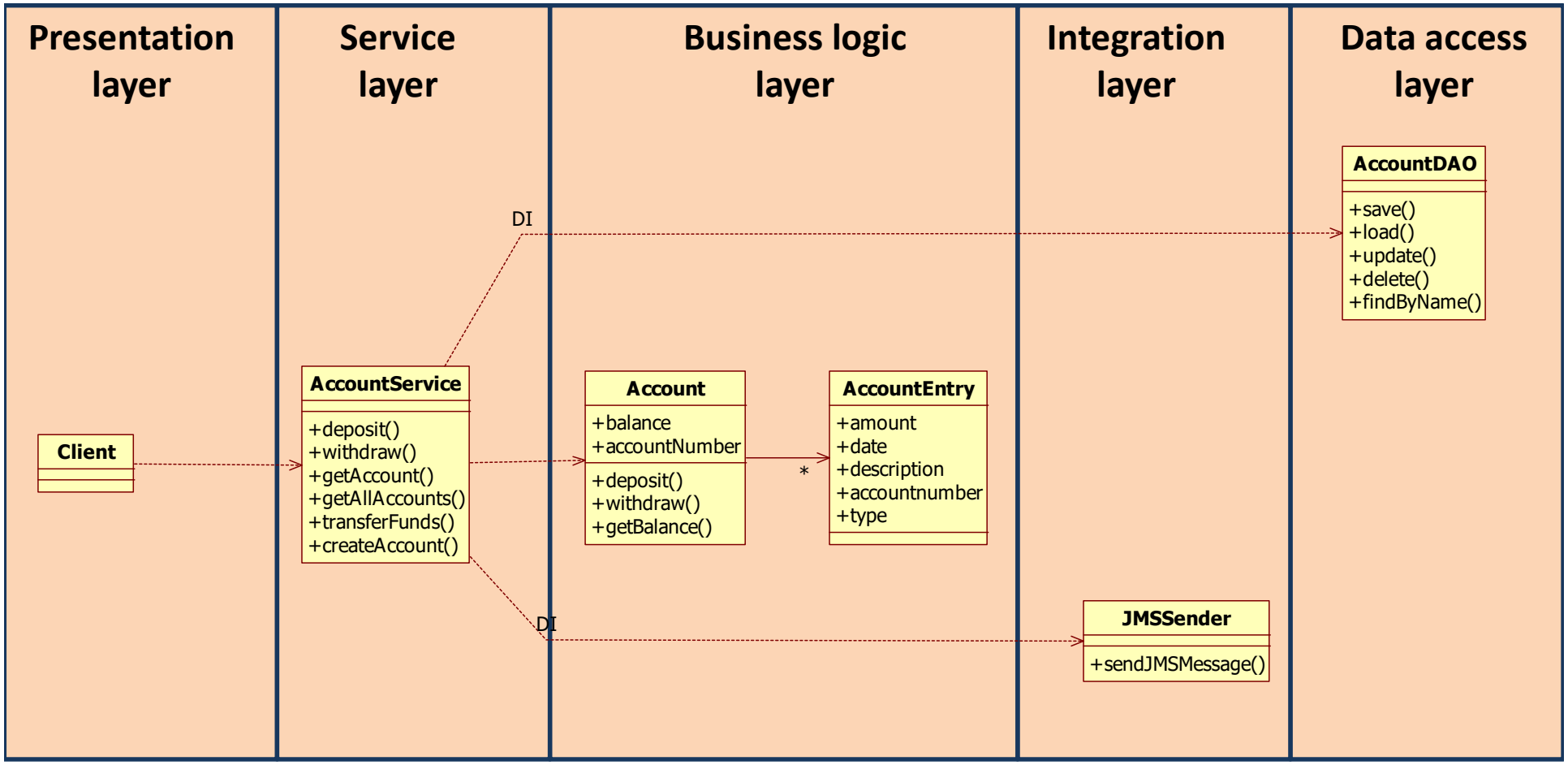


- As simple as possible
- Should not contain business logic

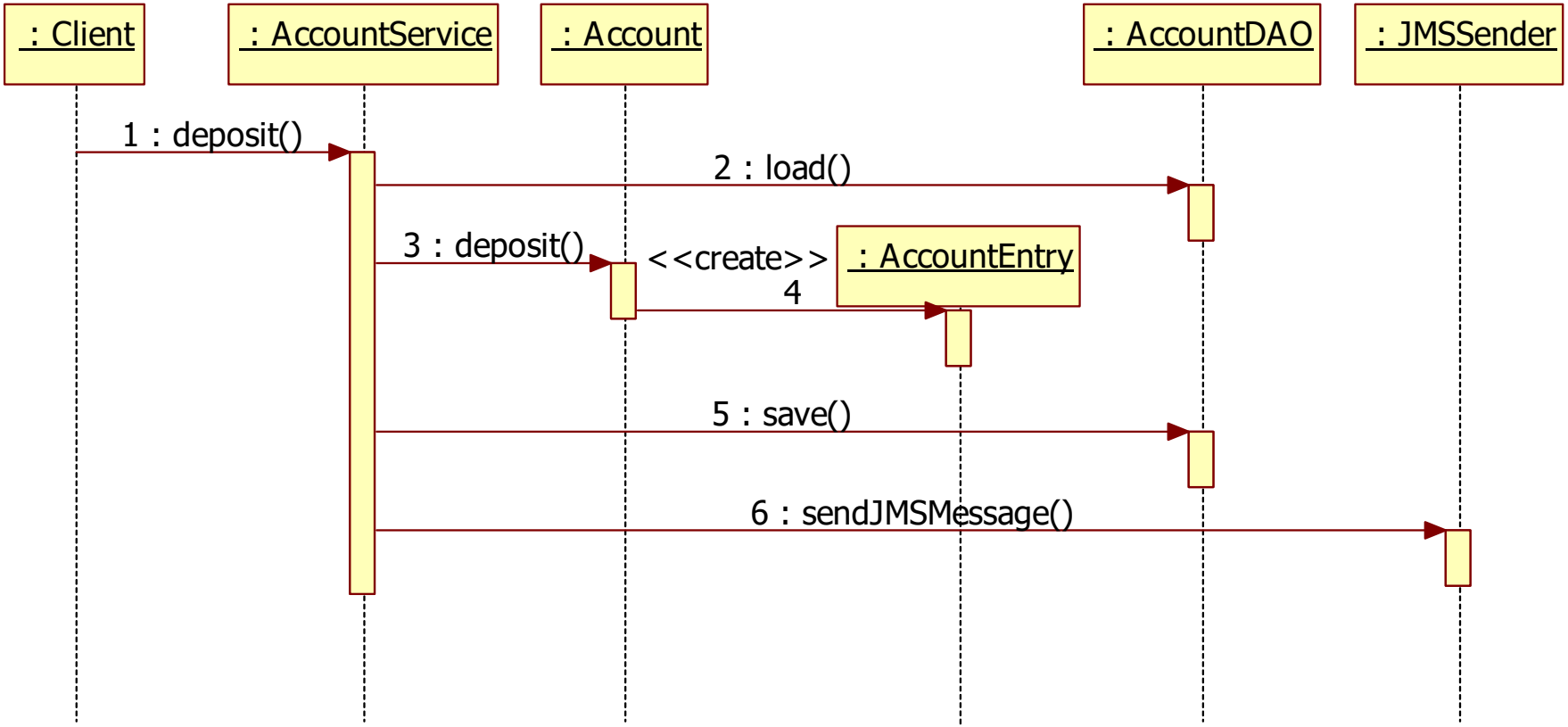
Separation of concern



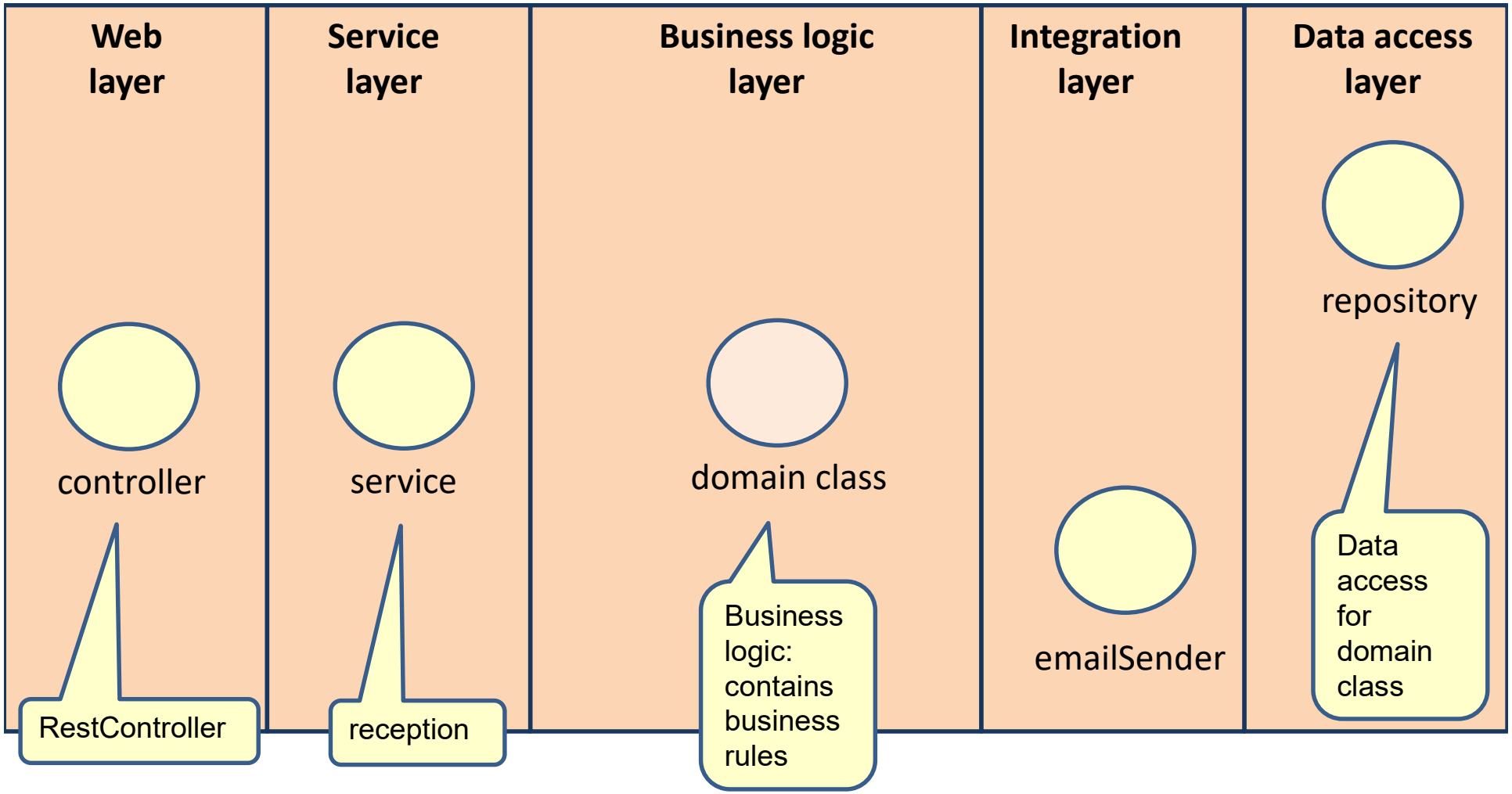
Application layers



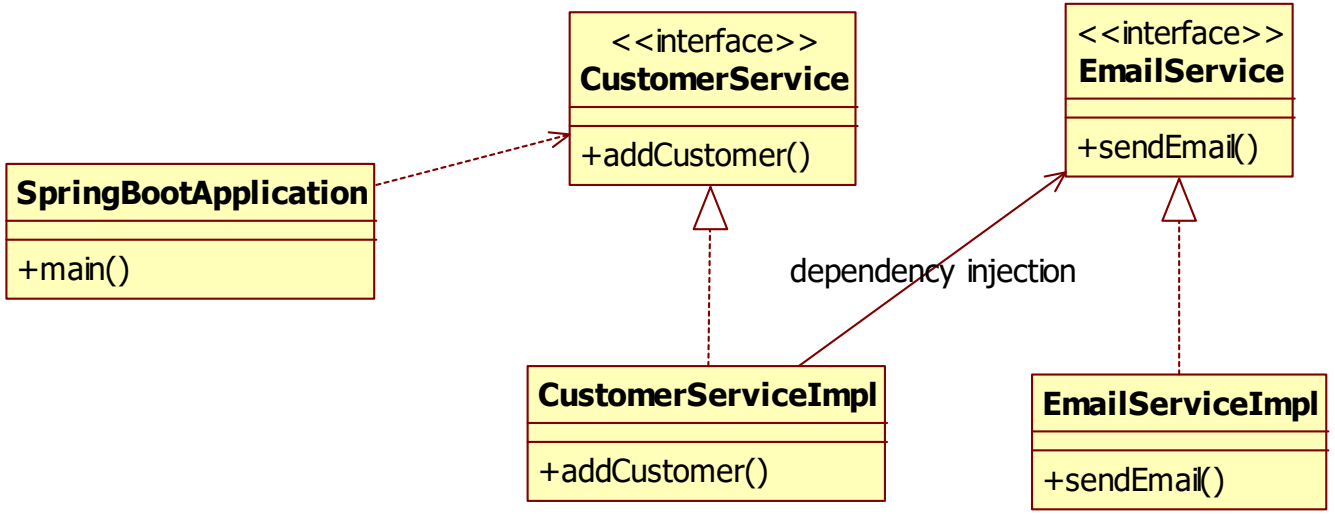
Service object

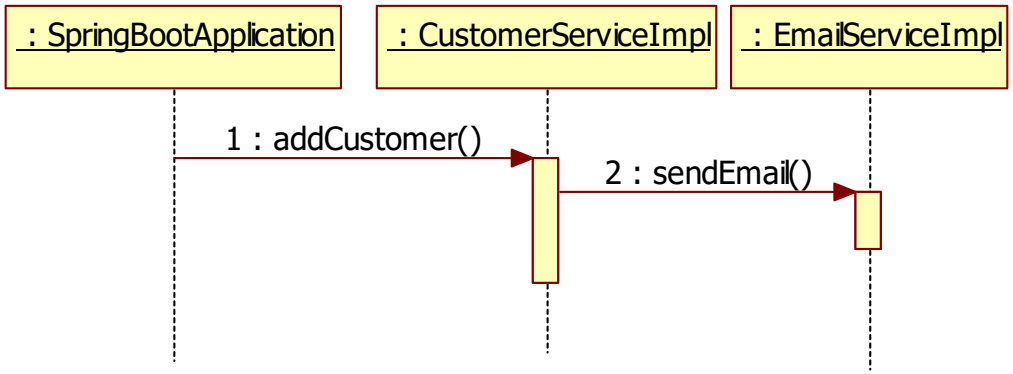


Layered architecture



Dependency injection





Dependency injection: Setter injection

```
@Service
public class CustomerServiceImpl implements CustomerService {

  private EmailService emailService;

  @Autowired
  public void setEmailService(EmailService emailService) {
    this.emailService = emailService;
  }


  public void addCustomer() {
    emailService.sendEmail();
  }
}
```

Setter injection

```
@Service
public class EmailServiceImpl implements EmailService{
  public void sendEmail() {
    System.out.println("Sending email");
  }
}
```

Dependency injection: Customer injection

```
@Service
public class CustomerServiceImpl implements CustomerService {

  private EmailService emailService;

  @Autowired
  public CustomerService(EmailService emailService) {
    this.emailService = emailService;
  }


  public void addCustomer() {
    emailService.sendEmail();
  }
}
```

Customer injection

```
@Service
public class EmailServiceImpl implements EmailService{
  public void sendEmail() {
    System.out.println("Sending email");
  }
}
```

Dependency injection: Field injection

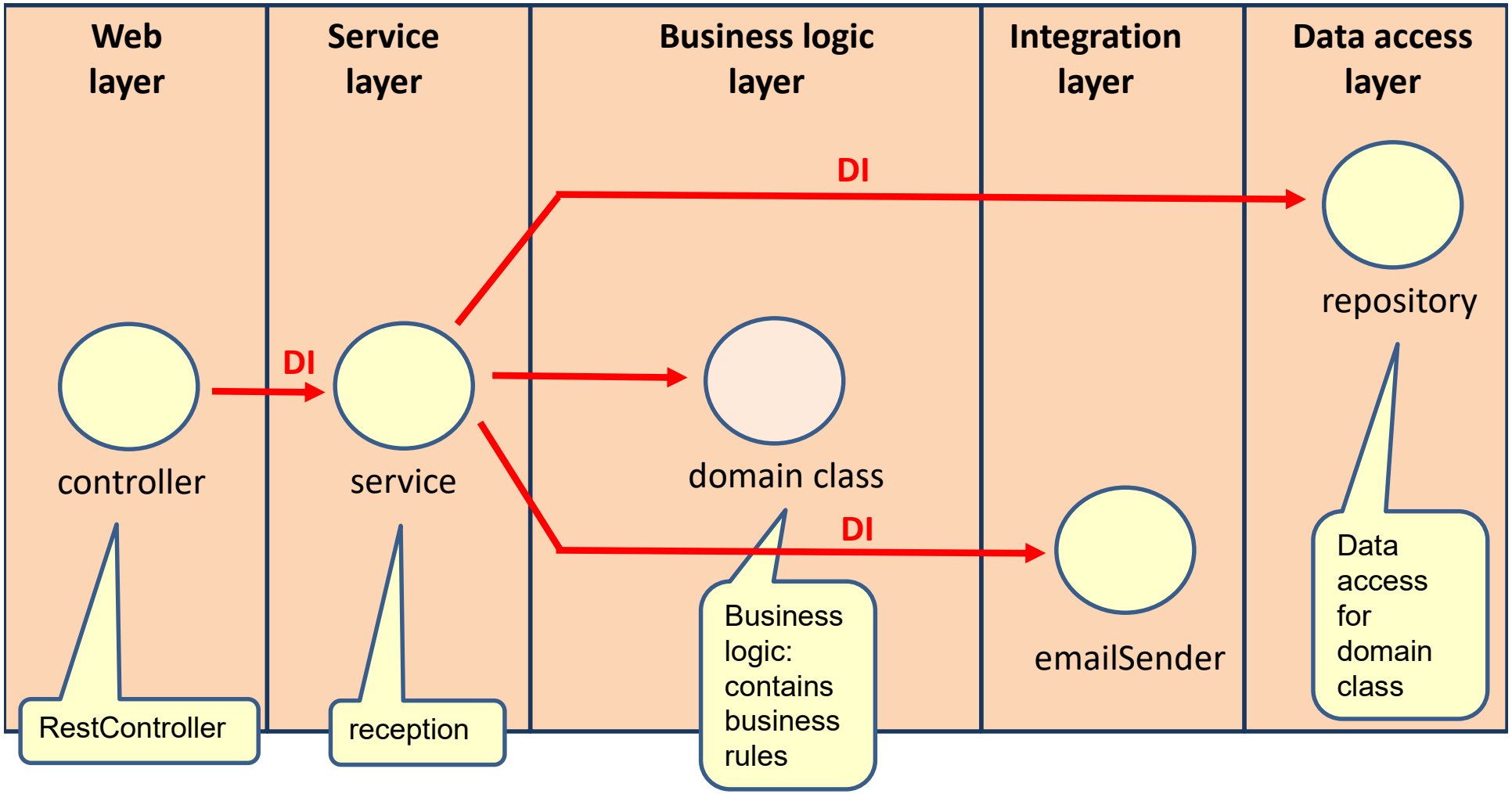
```java
@Service
public class CustomerServiceImpl implements CustomerService {
  @Autowired
  private EmailService emailService;

  public void addCustomer() {
    emailService.sendEmail();
  }
}
```

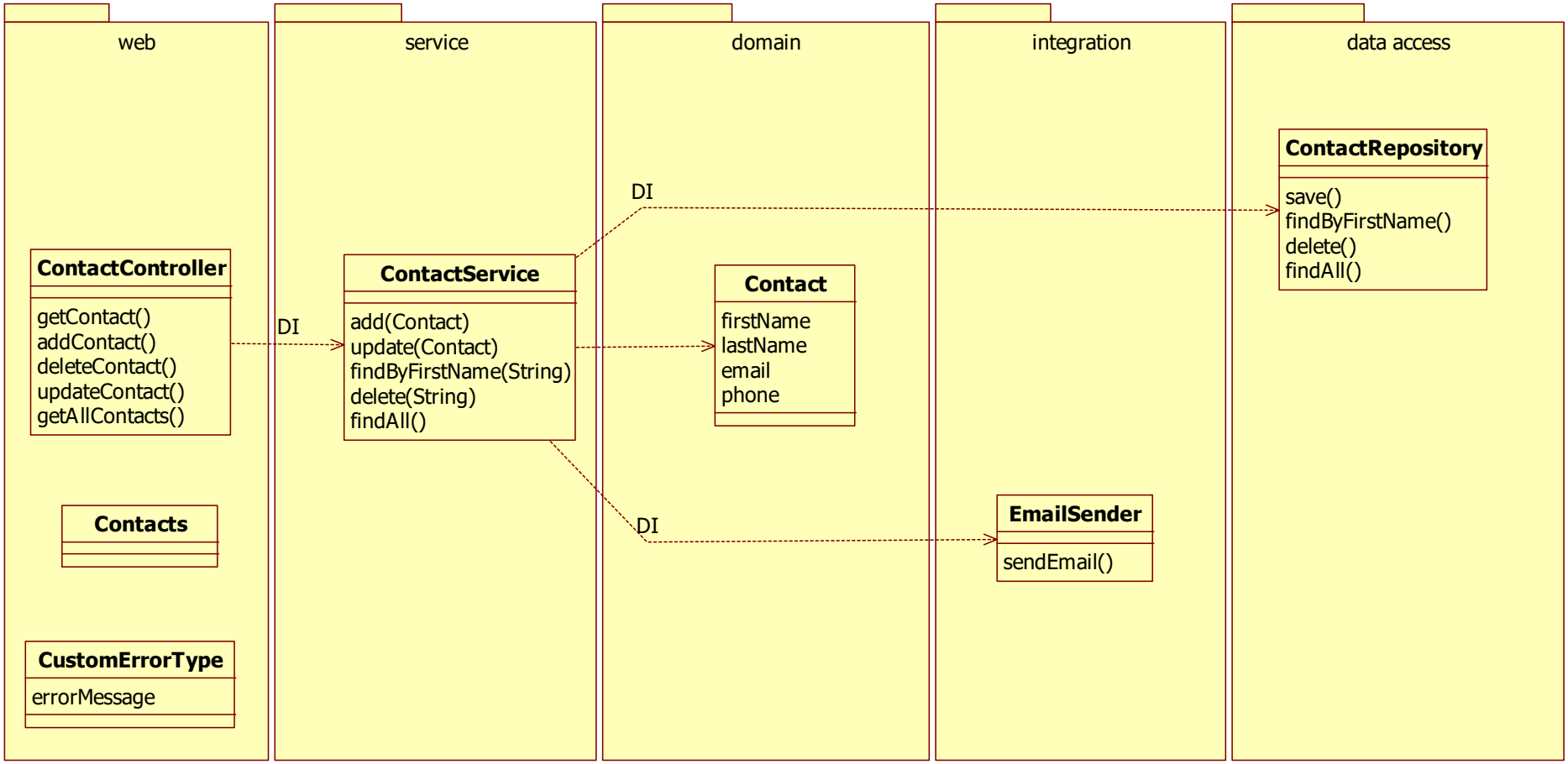
Field injection

```java
@Service
public class EmailServiceImpl implements EmailService{
  public void sendEmail() {
    System.out.println("Sending email");
  }
}
```

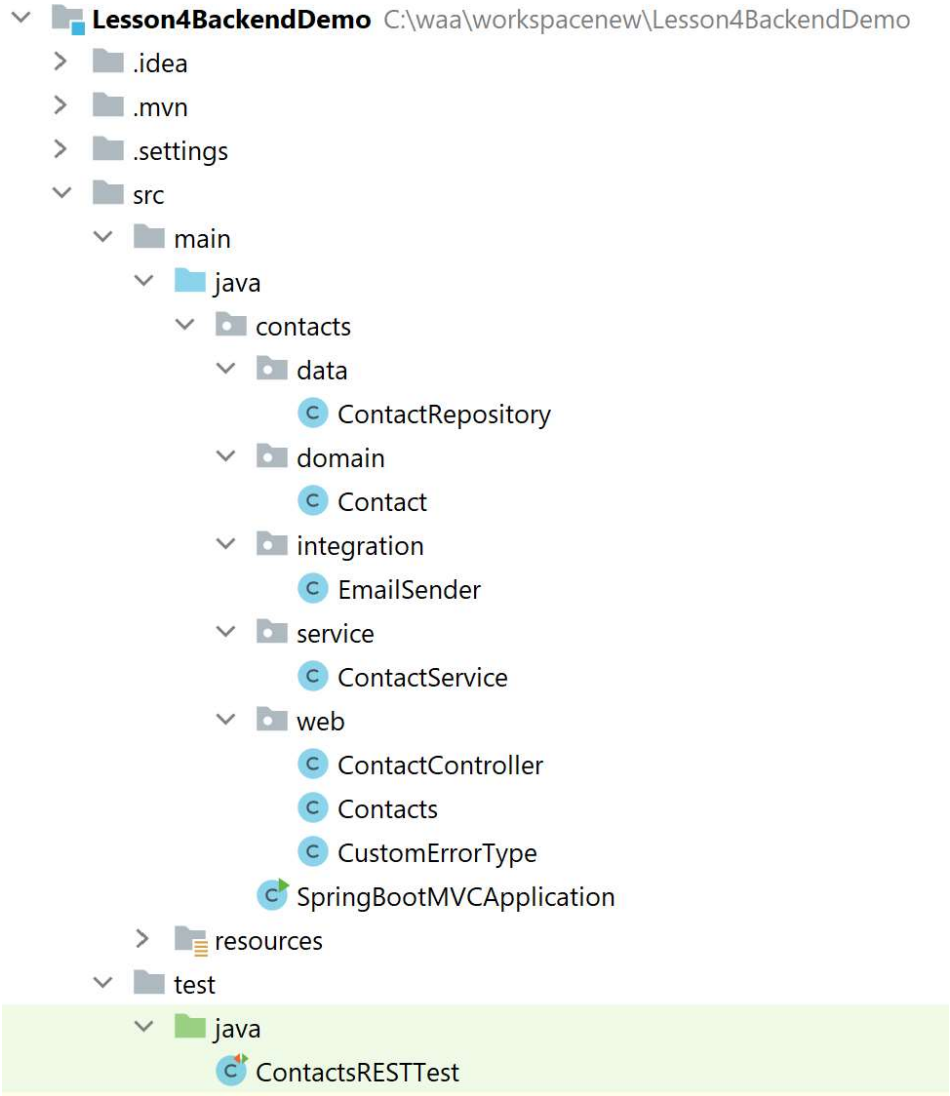
Layered architecture



Spring Boot example



Spring Boot example



Repository

@Repository

```
@Repository
public class ContactRepository {
   private Map<String, Contact> contacts = new HashMap<String, Contact>();

   public void save(Contact contact){
      contacts.put(contact.getFirstName(),contact);
   }

   public Contact findByFirstName(String firstName){
      return contacts.get(firstName);
   }

   public void delete(String firstName){
      contacts.remove(firstName);
   }

   public Collection<Contact> findAll(){
      return contacts.values();
   }
}
```

EmailSender

@Component

```
@Component
public class EmailSender {
   public void sendEmail (String message, String emailAddress){
      System.out.println("Send email message '"+ message+"' to"+emailAddress);
   }
}
```

Service

@Service

@Autowired

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

   public void add(Contact contact){
      contactRepository.save(contact);
      emailSender.sendEmail(contact.getEmail(), "Welcome");
   }

   public void update(Contact contact){
      contactRepository.save(contact);
   }

   public Contact findByFirstName(String firstName){
      return contactRepository.findByFirstName(firstName);
   }

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

   public Collection<Contact> findAll(){
      return contactRepository.findAll();
   }
}
```

Controller(1/2)

@RestController

@Autowired

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


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

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

…
```

Controller(2/2)

```
@RestController
public class ContactController {
   …

   @DeleteMapping("/contacts/{firstName}")
   public ResponseEntity<?> deleteContact(@PathVariable String firstName) {
      Contact contact = contactService.findByFirstName(firstName);
      if (contact == null) {
         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 Contact contact) {
      contactService.update(contact);
      return new ResponseEntity<Contact>(contact, HttpStatus.OK);
   }

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

Main point

- An enterprise back-end system is typically divided in different layers. *Life is found in layers*.

Connecting the parts of knowledge with the wholeness of knowledge

1. Layering is a powerful technique to separate different aspects of a system
2. The service class is the connection point between the different layers

---

3. **Transcendental consciousness** is the direct experience of pure consciousness, the unified field of all the laws of nature.
4. **Wholeness moving within itself:** In unity consciousness, one appreciates the inherent underlying unity that underlies all the diversity of creation.