Internet Applications Design and Implementation

(Lecture 4 - MVC & Persistence : JPA & Hibernate)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)

(with previous participations of Jácome Cunha (jacome@fct.unl.pt) and João Leitão (jc.leitao@fct.unl.pt))

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

Outline

- Server-side MVC Architecture
- Data Sources in MVC
- Object Relational Mapping
- Spring & Data Abstraction

Internet Applications Design and Implementation 2020 - 2021

(Lecture 4 - Part 1 - Server-side MVC Architecture)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

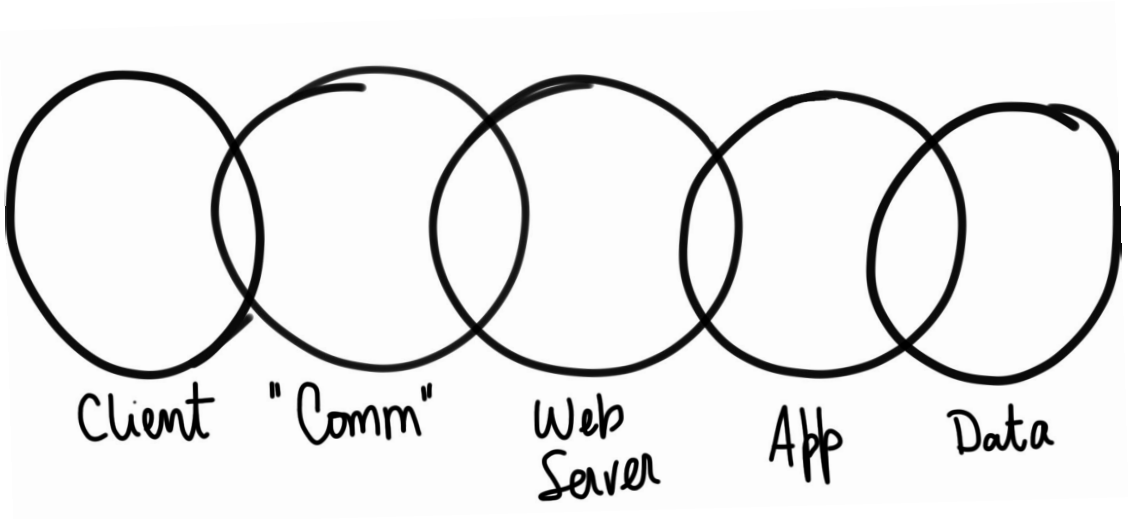
João Costa Seco (joao.seco@fct.unl.pt)

(with previous participations of Jácome Cunha (jacome@fct.unl.pt) and João Leitão (jc.leitao@fct.unl.pt))

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

Web architectures, patterns and styles

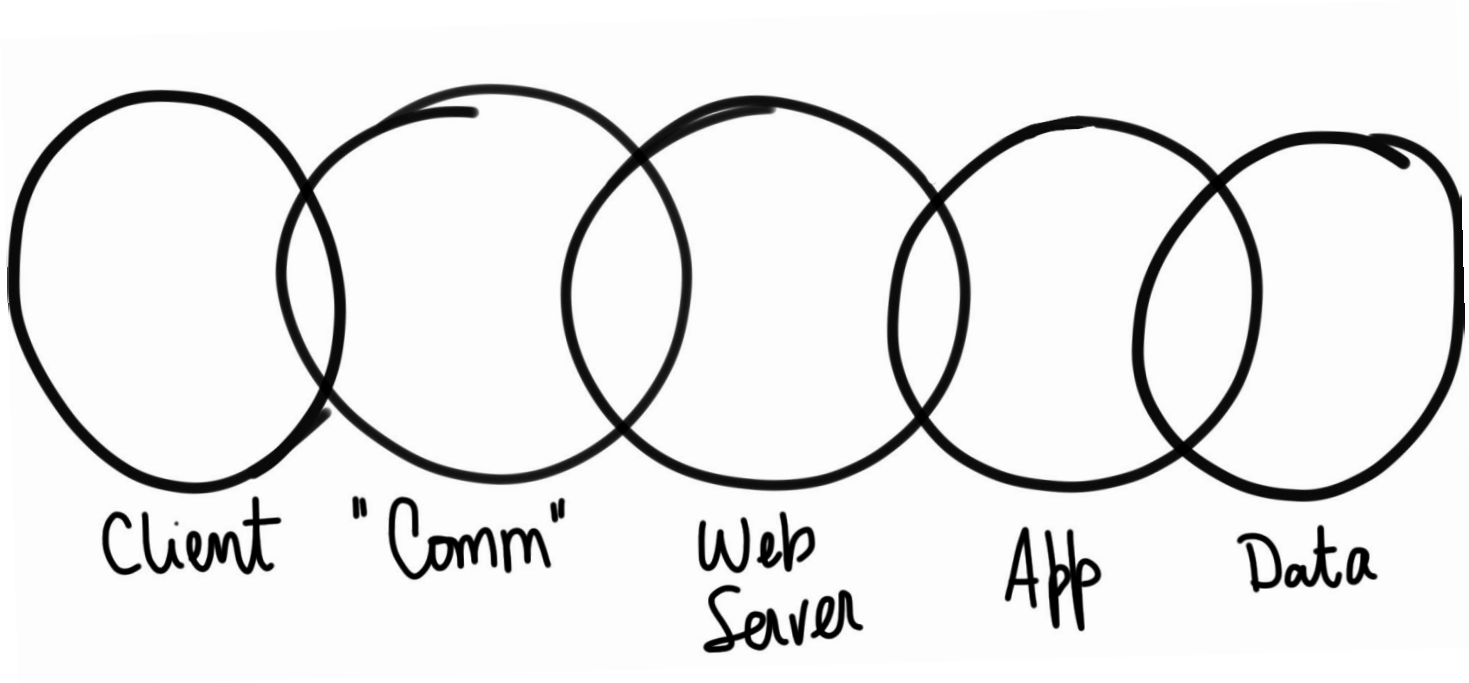
- Web applications usually follow a MVC architectural pattern.
  - Model layer - isolate the representation of persistent data and its operations, validations and conditions
  - Controller - contains the core application logic implementing the application interface (e.g. ad-hoc URL mapping, REST convention)
  - View - defines the way in which responses are formed (e.g. HTML, JSON)



Summary - Web Frameworks

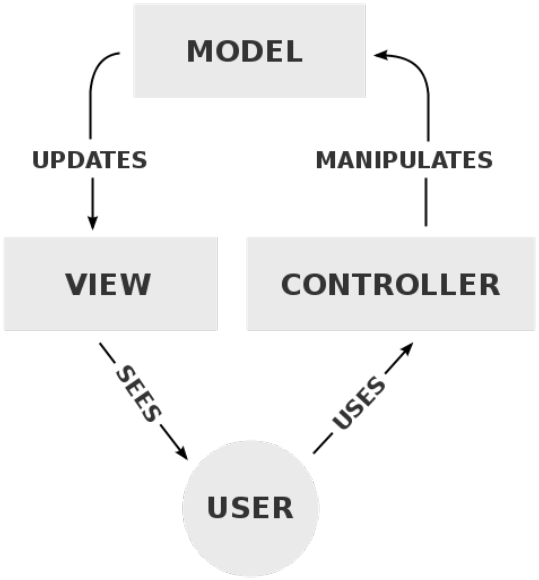
RECAP

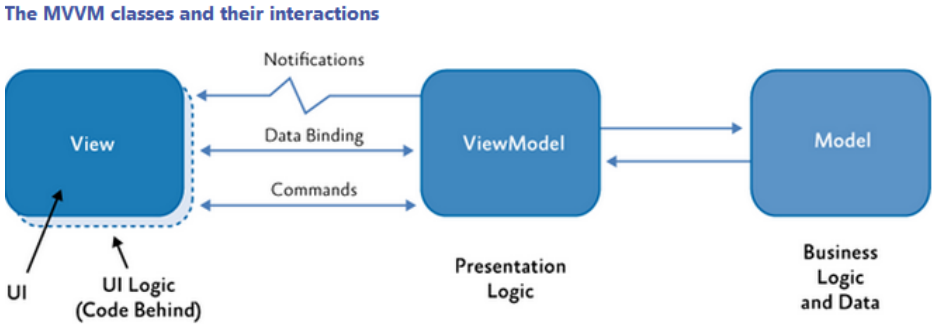
- Web Frameworks are “languages” that carry libraries and abstractions that get compiled to run on the “web virtual machine”.



The classic MVC design pattern

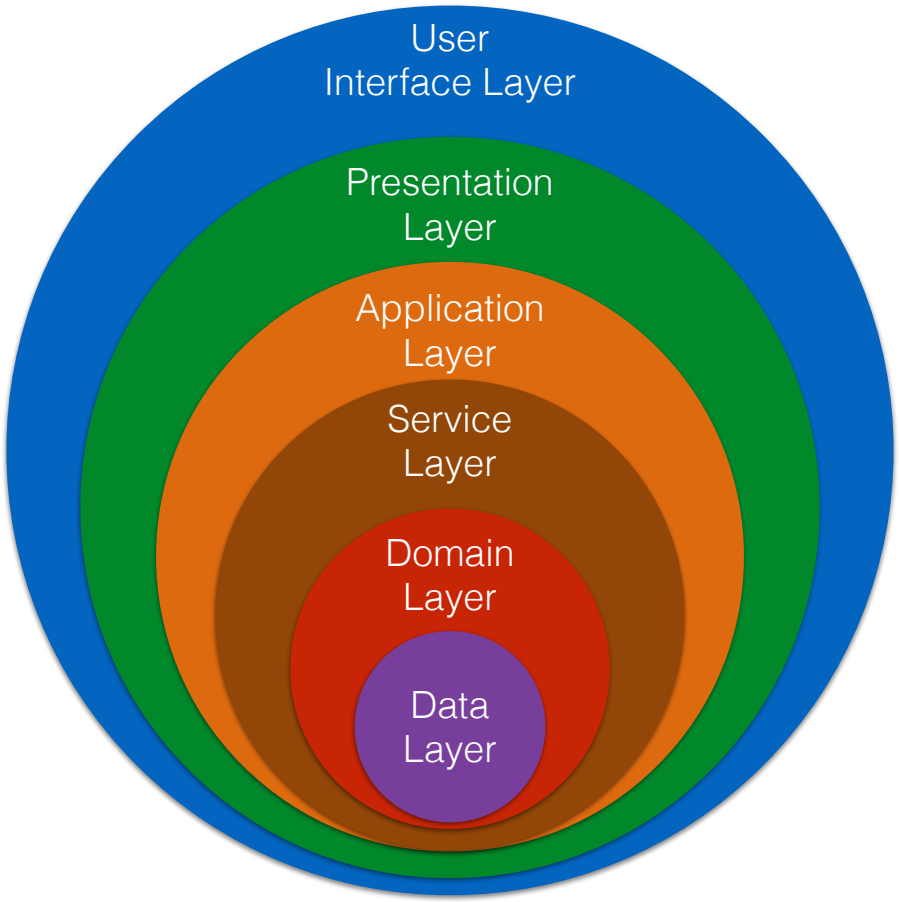
- The Model-View-Controller (Reenskaug'79, JOT'88)
  - designed to develop GUI
  - popular in web applications' context
- Variants of the MVC Architecture (Separation of Concerns)
  - MVP, PM (Fowler), MVVM (Microsoft)





https://manojjaggavarapu.wordpress.com/2012/05/02/presentation-patterns-mvc-mvp-pm-mvvm/

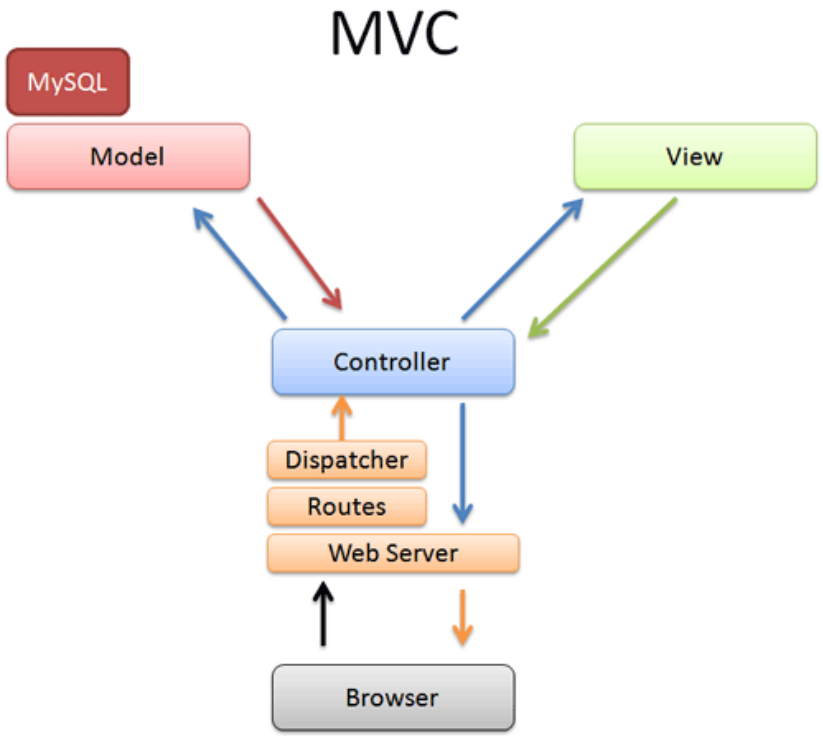
Internet Applications are Data-Centric



https://dzone.com/articles/layered-architecture-is-good

Frameworks and MVC Architecture

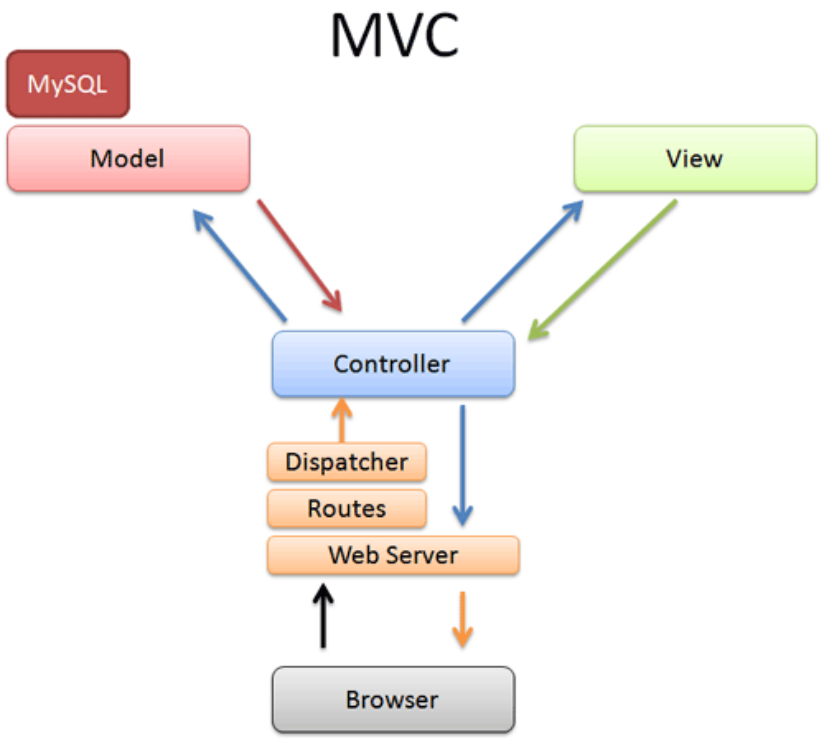
- Frameworks help to implement and maintain architectures.
- Rails (2005):
  - conventions on folder, file, and class names
  - A flexible OO prog language (Ruby) supports data sharing between model, controller, and view objects.



https://betterexplained.com/articles/intermediate-rails-understanding-models-views-and-controllers/

Frameworks and MVC Architecture

- Frameworks help to implement and maintain architectures.
- Django (2005):
  - views are controllers
  - templates are views
  - models are models



https://betterexplained.com/articles/intermediate-rails-understanding-models-views-and-controllers/

Frameworks and MVC Architecture

- Java Spring is a component-based programming framework (based on configuration).
- It does the “plumbing," and lets components implement the “logic” of applications.
- How spring implements the MVC pattern
  - Dependency Injection (inversion of control)
  - Aspect-Oriented Programming including Spring's declarative transaction management
  - Spring MVC web application and RESTful web service framework
  - Foundational support for JDBC, JPA, JMS
  - …

https://spring.io/guides/

Inversion of Control

- Design pattern where user-defined code fragment receives the flow of control from a generic framework.
  - Context: object-oriented programming
  - Found in: Frameworks, Event handlers, Callbacks
- Dependency Injection
  - An instance of inversion of control to build object networks
  - Centralised broker that maps types to implementations
  - Java Spring: Pool of beans/components, auto-wiring of object networks

Without inversion of Control

- Explicit initialisation of references

```
@RestController
@RequestMapping("/")
class EmpController(val employees:EmployeeService) {

   @GetMapping("/api/departments/{id}/employees")
    fun employeesOfDepartment(
            @PathVariable id:String,
            @RequestParam search:String?
    )
    = listOf(
                Employee("John Oliver",40,"New York"),
                Employee("John Gleese", 60, "London")
        )

   @GetMapping( "/api/projects/{id}/team")
    fun teamMembersOfProject(
            @PathVariable id:String
    )
    = employees.teamMembersOfProject(id)


}
```

```
@Service
class EmployeeService(val employees:EmployeeRepository) {
    fun teamMembersOfProject(id:String) = employees.findAll()
}

interface EmployeeRepository : CrudRepository<Employee, Long>
```

```
fun someMethod() {
    EmpController(
            EmployeeService(
                    EmployeeRepositoryImp(
                            DBConnection("...")
                    )
            )
    )
}
```

Without inversion of Control

- Explicit initialisation of references

```
package pt.unl.fct.demo.controllers;

import org.springframework.web.bind.annotation.*;
import pt.unl.fct.demo.model.Company;
import pt.unl.fct.demo.services.CompaniesService;

@RestController
@RequestMapping(value="/companies")
public class CompaniesController {

    CompaniesService companies;

    public CompaniesController(CompaniesService companies) {
        this.companies = companies;
    }

    @GetMapping("")
    Iterable<Company> getAllCompanies(@RequestParam(required=false) String search) {
        // Do some extra checking on the request, and then...
        return companies.getAllCompanies(search);
    }

    @PostMapping("")
    void addNewCompany(@RequestBody Company company) {
        // Do some extra checking on the request, and then...
        companies.addCompany(company);
    }
}
```

Spring uses annotations to indicate the kind of class, and where to plug it in

Without inversion of Control

- Explicit initialisation of references

```
package pt.unl.fct.demo.controllers;

import org.springframework.web.bind.annotation.*;
import pt.unl.fct.demo.model.Company;
import pt.unl.fct.demo.services.CompaniesService;

@RestController
@RequestMapping(value="/companies")
public class CompaniesController {

    CompaniesService companies;

    public CompaniesController(CompaniesService companies) {
        this.companies = companies;
    }

    @GetMapping("")
    Iterable<Company> getAllCompanies(@RequestParam(required=false) String search) {
        // Do some extra checking on the request, and then...
        return companies.getAllCompanies(search);
    }

    @PostMapping("")
    void addNewCompany(@RequestBody Company company) {
        // Do some extra checking on the request, and then...
        companies.addCompany(company);
    }
}
```

Instead of doing it explicitly ->>

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;

public class DemoServ extends HttpServlet{
    public void doGet(HttpServletRequest req,
                      HttpServletResponse res)
            throws ServletException,IOException
    {
        res.setContentType("text/html");
        PrintWriter pw=res.getWriter();

        String name=req.getParameter("name");
        pw.println("Welcome "+name);

        pw.close();
    }
}
```

from: https://www.javatpoint.com/servletrequest

Without inversion of Control

- Explicit initialisation of references

```java
package pt.unl.fct.demo.controllers;

import org.springframework.web.bind.annotation.*;
import pt.unl.fct.demo.model.Company;
import pt.unl.fct.demo.services.CompaniesService;

@RestController
@RequestMapping(value="/companies")
public class CompaniesController {

    CompaniesService companies;

    public CompaniesController(CompaniesService companies) {
        this.companies = companies;
    }

    @GetMapping("")
    Iterable<Company> getAllCompanies(@RequestParam(required=false) String search) {
        // Do some extra checking on the request, and then...
        return companies.getAllCompanies(search);
    }

    @PostMapping("")
    void addNewCompany(@RequestBody Company company) {
        // Do some extra checking on the request, and then...
        companies.addCompany(company);
    }
}
```

resources in ruby'nrails a whole DSL to define >> routes

```
resources :photos
```

creates seven different routes in your application, all mapping to the `Photos` controller:

HTTP Verb	Path	Controller#Action	Used for
GET	/photos	photos#index	display a list of all photos
GET	/photos/new	photos#new	return an HTML form for creating a new photo
POST	/photos	photos#create	create a new photo
GET	/photos/:id	photos#show	display a specific photo
GET	/photos/:id/edit	photos#edit	return an HTML form for editing a photo
PATCH/PUT	/photos/:id	photos#update	update a specific photo
DELETE	/photos/:id	photos#destroy	delete a specific photo

https://guides.rubyonrails.org/routing.html

Using Dependency Injection

- Explicitly tell what things are, let spring do the wiring

```
@RestController
@RequestMapping("/")
class EmpController() {

    @Autowired
    lateinit var employees:EmployeeService

    @GetMapping("/api/departments/{id}/employees")
    fun employeesOfDepartment(
            @PathVariable id:String,
            @RequestParam search:String?
    )
```

Declare dependencies explicitly to be initialised by Spring

Using Dependency Injection

- Explicitly tell what things are, let spring do the wiring

```
@RestController
@RequestMapping("/")
class EmpController(val employees:EmployeeService) {

    @GetMapping("/api/departments/{id}/employees")
    fun employeesOfDepartment(
            @PathVariable id:String,
            @RequestParam search:String?
    )
```

Declare constructor dependencies and let Spring initialise correctly

Frameworks and MVC Architecture

- Java Spring is a component-based programming framework (based on configuration).
- It does the “plumbing," and lets components implement the “logic” of applications.
- How spring implements the MVC:

```
@SpringBootApplication
class McqApplication

fun main(args: Array<String>) {
    runApplication<McqApplication>(*args)
}
```

Frameworks and MVC Architecture

- Java Spring is a configuration and programming framework.
- It does the “plumbing”, and lets the components implement the “logic” of applications.
- How spring implements the MVC: (in Java)

```java
@Configuration
@EnableAutoConfiguration
@EnableWebMvc
@ComponentScan
public class Application {

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

`@Configuration`

Annotation specifies that the class has Bean definition methods

Frameworks and MVC Architecture

- Java Spring is a configuration and programming framework.
- It does the “plumbing”, and lets the components implement the “logic” of applications.
- How spring implements the MVC

```
@Configuration
@EnableAutoConfiguration
@EnableWebMvc
@ComponentScan
public class Application {

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

```
@EnableAutoConfiguration
```

Attempts to guess and configure beans that you are likely to need (doc.spring.io)

Frameworks and MVC Architecture

- Java Spring is a configuration and programming framework.
- It does the “plumbing”, and lets the components implement the “logic” of applications.
- How spring implements the MVC

```
@Configuration
@EnableAutoConfiguration
@EnableWebMvc
@ComponentScan
public class Application {

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

`@ComponentScan`

Configures component scanning.If specific packages are not defined, scanning will occur from the package of the class that declares this annotation.

Frameworks and MVC Architecture

- Java Spring is a configuration and programming framework.
- It does the “plumbing”, and lets the components implement the “logic” of applications.
- How spring implements the MVC (Here the view is an HTML (thymeleaf) template)

```
@Controller
public class GreetingController {

    private static final String template = "Hello, %s! %d”;
    private final AtomicLong counter = new AtomicLong();

    @RequestMapping("/greeting")
    public String greeting(@RequestParam(value="name", defaultValue="World") String name,
                           Model model) {
        long c = counter.incrementAndGet();
        model.addAttribute(“message”, String.format(template, name, c));
        return “greeting”;
    }
}
```

Frameworks and MVC Architecture

- Java Spring is a configuration and programming framework.
- It does the “plumbing”, and lets the components implement the “logic” of applications.
- How spring implements the MVC (Here the view is a JSON object formatter)

```
@RestController
public class GreetingController {

    private static final String template = "Hello, %s!";
    private final AtomicLong counter = new AtomicLong();

    @RequestMapping("/greeting")
    public Greeting greeting(@RequestParam(value="name", defaultValue="World") String name) {
        return new Greeting(counter.incrementAndGet(),
                            String.format(template, name));
    }
}
```

Add-ons to the MVC Framework

- Resource control: DB connection & transactions

```
@Component
public class BookingService {

    private final static Logger logger = LoggerFactory.getLogger(BookingService.class);

    private final JdbcTemplate jdbcTemplate;

    public BookingService(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }

    @Transactional
    public void book(String... persons) {
        for (String person : persons) {
            logger.info("Booking " + person + " in a seat...");
            jdbcTemplate.update("insert into BOOKINGS(FIRST_NAME) values (?)", person);
        }
    }

    public List<String> findAllBookings() {
        return jdbcTemplate.query("select FIRST_NAME from BOOKINGS",
                (rs, rowNum) -> rs.getString("FIRST_NAME"));
    }

}
```

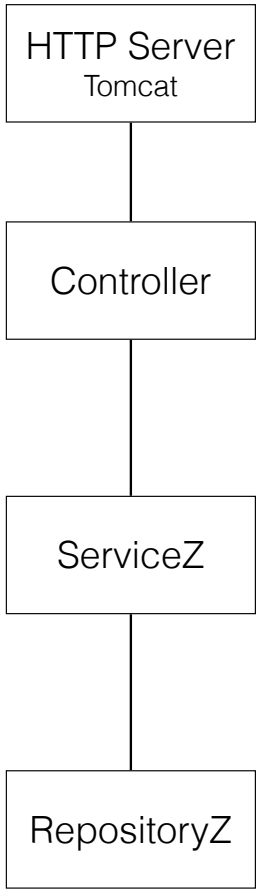
Add-ons to the MVC Framework

- Across application concerns: security

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests()
                .antMatchers("/", "/home").permitAll()
                .anyRequest().authenticated()
                .and()
            .formLogin()
                .loginPage("/login")
                .permitAll()
                .and()
            .logout()
                .permitAll();
    }

    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
        auth
            .inMemoryAuthentication()
                .withUser("user").password("password").roles("USER");
    }
}
```

Example of an Architecture built with Spring

RECAP



- Spring is a component framework
- Resolves component dependencies by dependency injection
- Uses annotations to configure components

```
@RestController
@RequestMapping("/")
class EmpController(val employees:EmployeeService) {

   // http GET :8080/api/projects/2/team
    @GetMapping( "/api/projects/{id}/team")
    fun teamMembersOfProject(
            @PathVariable id:String
    )
    = employees.teamMembersOfProject(id)


}

@Service
class EmployeeService(val employees:EmployeeRepository) {
    fun teamMembersOfProject(id:String) = employees.findAll()
}

interface EmployeeRepository : CrudRepository<Employee, Long>
```

Architecture to the rescue of testers

- Unit tests should test components in isolation
- Defining the context for a component (correctly) is laborious and error prone
- Difficult to do with persistent data (must prepare tests)
- Impossible to do in tightly coupled structures
- Component frameworks allow mocking of dependencies

```
@RunWith(SpringRunner::class)
@SpringBootTest
@AutoConfigureMockMvc
open class RESTApplicationTests() {

    @Autowired lateinit var mvc: MockMvc
    @MockBean  lateinit var questions:QuestionRepository

    @Test
    fun `basic REST test`() {
        Mockito.`when`(questions.findAll()).thenReturn(l)

        mvc.perform(get(questionsURL))
                .andExpect(status().isOk)
   }
```

Architecture to the rescue of testers

```
@RunWith(SpringRunner::class)
@SpringBootTest
@AutoConfigureMockMvc
open class RESTApplicationTests() {

    @Autowired lateinit var mvc: MockMvc
    @MockBean  lateinit var questions:QuestionRepository

    @Test
    fun `basic REST test`() {
        Mockito.`when`(questions.findAll()).thenReturn(l)

        mvc.perform(get(questionsURL))
                .andExpect(status().isOk)
    }
```

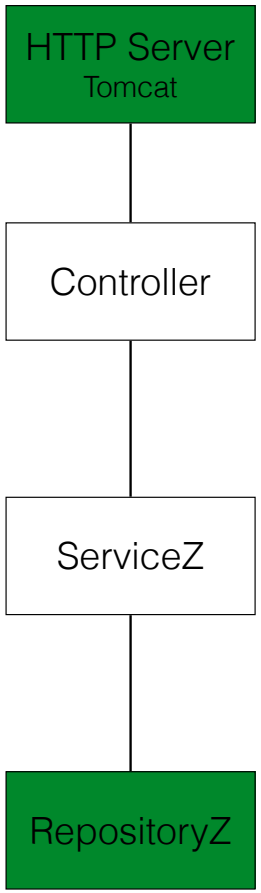
Replaces web server

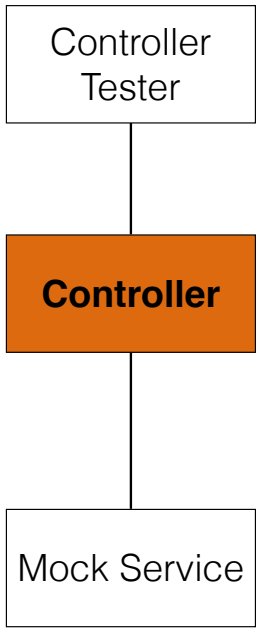
Replaces component with Mock object

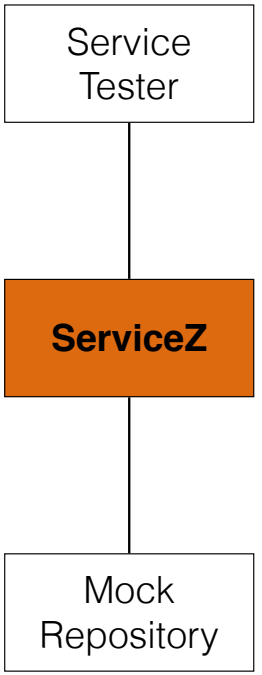
Replaces call results with expected values

Architecture to the rescue of testers

- Unit tests should test components in isolation
- Unit tests simulate inputs and compare outputs
- Mock components create a controlled context for each test or set of tests.







Questions?

Internet Applications Design and Implementation 2020 - 2021

(Lecture 4 - Part 2 - Data Sources in MVC)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)

(with previous participations of Jácome Cunha (jacome@fct.unl.pt) and João Leitão (jc.leitao@fct.unl.pt))

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

Data Abstraction: The M in MVC

- An application layer that abstracts how information is stored, related, and protected.
- Examples of database languages, libraries and frameworks
  - JDBC
  - LINQ (in MVC ASP.NET)
  - ORMs (ActiveRecord in Rails, Hibernate in Java*)
  - NoSQL: MongoDB
  - External Web-services

JDBC

```
Connection conn = DriverManager.getConnection(
     "jdbc:somejdbcvendor:other data needed by some jdbc vendor",
     "myLogin",
     "myPassword" );
try {
     /* you use the connection here */
} finally {
    //It's important to close the connection when you are done with it
    try { conn.close(); } catch (Throwable ignore) { /* Propagate the original exception
instead of this one that you may want just logged */ }
}
```

```
try (Statement stmt = conn.createStatement();
    ResultSet rs = stmt.executeQuery( "SELECT * FROM MyTable" )
) {
    while ( rs.next() ) {
        int numColumns = rs.getMetaData().getColumnCount();
        for ( int i = 1 ; i <= numColumns ; i++ ) {
           // Column numbers start at 1.
           // Also there are many methods on the result set to return
           //  the column as a particular type. Refer to the Sun documentation
           //  for the list of valid conversions.
           System.out.println( "COLUMN " + i + " = " + rs.getObject(i) );
        }
    }
}
```

JDBC

- Basic API for Java defining an access to a database
- Does not know the database schema
- Programmer needs to “manually” translate between data formats

```
try (Statement stmt = conn.createStatement();
    ResultSet rs = stmt.executeQuery( "SELECT * FROM MyTable" )
) {
    while ( rs.next() ) {
        int numColumns = rs.getMetaData().getColumnCount();
        for ( int i = 1 ; i <= numColumns ; i++ ) {
           // Column numbers start at 1.
           // Also there are many methods on the result set to return
           //  the column as a particular type. Refer to the Sun documentation
           //  for the list of valid conversions.
           System.out.println( "COLUMN " + i + " = " + rs.getObject(i) );
        }
    }
}
```

LINQ .NET

- Language based (integrated)
- Works in memory, xml, databases, etc
  - Based on the notion of **Provider**, it is extensible
- Programmers need to explicitly build objects that matches the database schema

```
var results =  from c in SomeCollection
               where c.SomeProperty < 10
               select new {c.SomeProperty, c.OtherProperty};

foreach (var result in results)
{
        Console.WriteLine(result);
}
```

LINQ .NET

- Language based (integrated)
- Works in memory, xml, databases, etc
  - Based on the notion of **Provider**, it is extensible
- Programmers need to explicitly build objects that matches the database schema

```
[Table(Name="Customers")]
public class Customer
{
      [Column(IsPrimaryKey = true)]
      public int CustID;

      [Column]
      public string CustName;
}
```

LINQ .NET

- Language based (integrated)
- Allows navigation using associations

```
// Query for customers who have placed orders.
var custQuery =
    from cust in Customers
    where cust.Orders.Any()
    select cust;

foreach (var custObj in custQuery)
{
    Console.WriteLine("ID={0}, Qty={1}", custObj.CustomerID,
        custObj.Orders.Count);
}
```

ActiveRecord in Rails

- Follows the active record pattern to implement an ORM. AR objects link to persistent data and define behaviour.
- Well integrated with the Model in the Rails MVC pattern
- Completely abstracts the database management by:
  - Object-mappings
  - Inheritance
  - Associations
  - Validations
  - Migrations (w/ Rails)

```
def create
  @author = Author.new(author_params)

  respond_to do |format|
    if @author.save
      format.html { redirect_to @author,
      format.json { render action: 'show'
    else
      format.html { render action: 'new'
      format.json { render json: @author.
    end
  end
end
```

NoSQL databases

- Not only SQL.
  - Document-based, Chave-Valor, Graph-based
- Desenhadas para escalonamento horizontal (replicação)
- Compromisso na consistência dos dados
- Limited transactional support (real concurrency control)… use of chards
- Very good for querying, harder to get right on “writes”, indexes, etc.
- Queries are written in JavaScript, Java, SPARK, Map reduce algorithms…

Data abstraction in Internet and Web Apps

- Abstraction over the actual data model
  - Hides the actual database engine running beneath
  - Integrates smoothly with the programming model (objects instead of string-based results).
- Independent configuration modes
  - Allows different execution modes with the same code (e.g. in-memory, transactional, replicated, etc.)
- Does not really cover all data models smoothly:
  - SQL model (e.g. Hibernate, ActiveRecord (Rn'R))
  - NoSQL model (e.g. google app engine framework)

Abstraction levels

- Connectivity (e.g. JDBC)
  - abstracts the connectivity and execution of queries.
  - you have to build queries, and parse and translate results
- Data translation (e.g. JPA, ActiveRecord, LINQ)
  - Translates data formats between program and database.
  - Integrates the language values typefully.
- Implementation and execution modes
  - Example: Google Cloud Datastore is a NoSQL document based storage that allows different implementations (cassandra, mongodb)

https://cloud.google.com/appengine/docs/java/datastore/

ORM

Object-relational impedance mismatch

- Most databases are based on relational algebra
- Applications define object oriented representations (object graphs)

346 systems in ranking, October 2018

Rank			DBMS	Database Model	Score		
Oct 2018	Sep 2018	Oct 2017			Oct 2018	Sep 2018	Oct 2017
1.	1.	1.	Oracle	Relational DBMS	1319.27	+10.15	-29.54
2.	2.	2.	MySQL	Relational DBMS	1178.12	-2.36	-120.71
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1058.33	+7.05	-151.99
4.	4.	4.	PostgreSQL	Relational DBMS	419.39	+12.97	+46.12
5.	5.	5.	MongoDB	Document store	363.19	+4.39	+33.79
6.	6.	6.	DB2	Relational DBMS	179.69	-1.38	-14.90
7.	↑ 8.	↑ 9.	Redis	Key-value store	145.29	+4.35	+23.24
8.	↓ 7.	↑ 10.	Elasticsearch	Search engine	142.33	-0.28	+22.09
9.	9.	↓ 7.	Microsoft Access	Relational DBMS	136.80	+3.41	+7.35
10.	10.	↓ 8.	Cassandra	Wide column store	123.39	+3.83	-1.40

https://db-engines.com/en/ranking

Object-relational impedance mismatch

- Most databases are based on relational algebra
- Applications define object oriented representations (object graphs)

355 systems in ranking, October 2019

Rank			DBMS	Database Model	Score		
Oct 2019	Sep 2019	Oct 2018			Oct 2019	Sep 2019	Oct 2018
1.	1.	1.	Oracle	Relational, Multi-model	1355.88	+9.22	+36.61
2.	2.	2.	MySQL	Relational, Multi-model	1283.06	+3.99	+104.94
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1094.72	+9.66	+36.39
4.	4.	4.	PostgreSQL	Relational, Multi-model	483.91	+1.66	+64.52
5.	5.	5.	MongoDB	Document	412.09	+2.03	+48.90
6.	6.	6.	IBM Db2	Relational, Multi-model	170.77	-0.79	-8.91
7.	7.	↑ 8.	Elasticsearch	Search engine, Multi-model	150.17	+0.90	+7.85
8.	8.	↓ 7.	Redis	Key-value, Multi-model	142.91	+1.01	-2.38
9.	9.	9.	Microsoft Access	Relational	131.18	-1.53	-5.62
10.	10.	10.	Cassandra	Wide column	123.22	-0.18	-0.17

https://db-engines.com/en/ranking

Object-Relational Impedance Mismatch

- What's the difference?
  - Encapsulation
  - Interface, class, polymorphism
  - Mapping relational concepts
  - Data type differences
  - Structural and integrity differences
  - Transactional differences

http://blogs.tedneward.com/post/the-vietnam-of-computer-science/

https://dev.to/alagrede/why-i-dont-want-use-jpa-anymore-fl

Object-Relational Impedance Mismatch

- Identity
  - Object: a == b and a.equals(b)
  - Relational: primary key based
- Inheritance
  - Object: natural relation
  - Relational: does not exist, idioms are necessary
- Accessing data
  - Object: through the object interface
  - Relational: select queries (with joins)
- Associations/Navigation
  - Object: unidirectional references through objects' interface
  - Relational: through foreign keys
- Granularity
  - In some cases there may be a difference in the granularity level

Object-Relational Impedance Mismatch

- Identity based on Primary Keys,
- We must define method `equals` (or utils like Lombok).

```
@Entity
public class Person {

    @Id
    @GeneratedValue
    private long id;

    private String name;

    public Person() {}

    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

Object-Relational Impedance Mismatch

- Inheritance: optional support for inheritance is provided by some frameworks

```
@Entity
@Inheritance(strategy= InheritanceType.JOINED)
@JsonSubTypes({
       @JsonSubTypes.Type(value = Professor.class, name = "PROFESSOR"),
        @JsonSubTypes.Type(value = Staff.class, name = "STAFF"),
        @JsonSubTypes.Type(value = Student.class, name = "STUDENT")
})
public class User {

   @Id
   @GeneratedValue
   private Long id_login;

   private String name;
…
}
```

```
@Entity
public class Student extends User {

    @Column
    private int number;

    public Student() {
        super();
    }

    public Student(String login,
                   String password,
                   String name,
                   String tel,
                   String email,
                   String address,
                   String type,
                   int number) {

        super( login,
               password,
               name,
               tel,
               email,
               address,
               type);
        this.number = number;
    }

    public int getNumber() {
        return number;
    }
}
```

Object-Relational Impedance Mismatch

- Inheritance: optional support for inheritance is provided by some frameworks

```
@Entity
@Inheritance(strategy= InheritanceType.JOINED)
@JsonSubTypes({
       @JsonSubTypes.Type(value = Professor.class, name = "PROFESSOR"),
        @JsonSubTypes.Type(value = Staff.class, name = "STAFF"),
        @JsonSubTypes.Type(value = Student.class, name = "STUDENT")
})
public class User {

   @Id
   @GeneratedValue
   private Long id login;

   private
…
}
```

```
@Entity
public class Student extends User {

    @Column
    private int number;

    public Student() {
        super();
    }

    public Student(String login,
                   String password,
                   String name,
                   String tel,
                   String email,
                   String address,
                   String type,
                   int number) {

        super( login,
               password,
               name,
               tel,
               email,
               address,
```

- *MappedSuperclass* – the parent classes, can't be entities
- Single Table – the entities from different classes with a common ancestor are placed in a single table
- Joined Table – each class has its table and querying a subclass entity requires joining the tables
- Table-Per-Class – all the properties of a class, are in its table, so no join is required

https://www.baeldung.com/hibernate-inheritance

Object-Relational Impedance Mismatch

- Access to data by object navigation
- All fields are retrieved to memory instead of explicitly selected
- May result in navigation queries

```
Organization o = organizationRepository.findById(id).get();
System.out.println(o.getName() + o.getContactInfo());
```

```
@Entity
public class Organization {

    @Id
    @GeneratedValue
    private Long id_entity;

    @Column
    private String name;

    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "contact_id")
    private ContactInfo contactInfo;

    public Organization(){ }

   public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public ContactInfo getContactInfo() {
        return contactInfo;
    }

    public void setContactInfo(ContactInfo contactInfo) {
        this.contactInfo = contactInfo;
    }
    …
}
```

ActiveRecord Example

```
class Album < ActiveRecord::Base
    has_many :tracks
end

class Track < ActiveRecord::Base
    belongs_to :album
end

album = Album.create(:title => 'Black and Blue', :performer => 'The Rolling Stones')
album.tracks.create(:track_number => 1, :title => 'Hot Stuff')
album.tracks.create(:track_number => 2, :title => 'Hand Of Fate')
album.tracks.create(:track_number => 3, :title => 'Cherry Oh Baby ')
album.tracks.create(:track_number => 4, :title => 'Memory Motel ')
album.tracks.create(:track_number => 5, :title => 'Hey Negrita')
album.tracks.create(:track_number => 6, :title => 'Fool To Cry')
album.tracks.create(:track_number => 7, :title => 'Crazy Mama')
album.tracks.create(:track_number => 8, :title => 'Melody (Inspiration By Billy
Preston)')

album = Album.create(:title => 'Sticky Fingers',:performer => 'The Rolling Stones')
album.tracks.create(:track_number => 1, :title => 'Brown Sugar')
album.tracks.create(:track_number => 2, :title => 'Sway')
album.tracks.create(:track_number => 3, :title => 'Wild Horses')
album.tracks.create(:track_number => 4,:title => 'Can\'t You Hear Me Knocking')
album.tracks.create(:track_number => 5, :title => 'You Gotta Move')
album.tracks.create(:track_number => 6, :title => 'Bitch')
album.tracks.create(:track_number => 7, :title => 'I Got The Blues')
album.tracks.create(:track_number => 8, :title => 'Sister Morphine')
album.tracks.create(:track_number => 9, :title => 'Dead Flowers')
album.tracks.create(:track_number => 10, :title => 'Moonlight Mile')
```

https://dzone.com/articles/simple-ruby-activerecord

Object-Relational Mapping

- In Java you navigate the object network.
- Not efficient to retrieve data from a RDBMS.
- Minimize the number of SQL queries by using JOINs and selecting the targeted entities from the start (pre-fetching).

```
from Cat as cat
    inner join cat.mate as mate
    left outer join cat.kittens as kitten
```

https://docs.jboss.org/hibernate/orm/3.3/reference/en/html/queryhql.html#queryhql-joins

JPA

Java Persistance API (JPA)

JPA provides a POJO persistence model for ORM

```
@Entity
public class Customer {

    private int id;
    private String name;
    private Collection<Order> orders;

  // no-args constructor necessary
   public Costumer () {}

  // primary key required
    @Id  // property access is used
    public int getId() {
        return id;
    }

  // gets and sets required
    public void setId(int id) {
        this.id = id;
    }
   …
```

```
…
public String getName() {
    return name;
}

 public void setName(String name) {
     this.name = name;
 }

// also OneToOne, ManyToOne, and ManyToMany
  @OneToMany(cascade=ALL,
         mappedBy="customer")
  public Collection<Order>
                    getOrders() {
     return orders;
  }

  public void setOrders(
     Collection<Order> newValue) {
     this.orders = newValue;
  }
```

Java Persistance API (JPA)

```
@Entity
@Table(name="ORDER_TABLE")
public class Order {

    private int id;
    private String address;
    private Customer customer;

    @Id
    @Column(name="ORDER_ID")
    public int getId() {
        return id;
    }

    public void setId(int id){
        this.id = id;
    }
  …
```

```
  …

    @Column(name="SHIPPING_ADDRESS")
    public String getAddress() {
        return address;
    }
    public void setAddress(
                String address) {
        this.address = address;
    }

    @ManyToOne()
  // foreign key: column used for join
    @JoinColumn(name="CUSTOMER_ID")
    public Customer getCustomer() {
        return customer;
    }

    public void setCustomer(
              Customer customer) {
        this.customer = customer;
    }
```

JDBC

Internet Applications Design and Implementation 2020 - 2021

(Lecture 4 - Part 3 - Object Relational Mapping)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

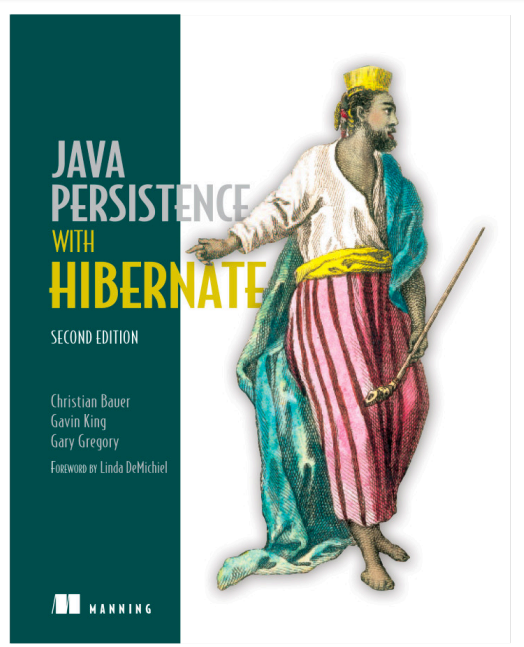
João Costa Seco (joao.seco@fct.unl.pt)

(with previous participations of Jácome Cunha (jacome@fct.unl.pt) and João Leitão (jc.leitao@fct.unl.pt))

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

Object-Relational Mapping

- Manages persistence of data in the OO realm
- Abstracts the use of SQL in connection to RDBMSs
- Covers most injection attacks on queries (except with APIs that allow creation with string)
- Specified by a common and standard API (JPA)
- Implementations differ by JPA providers




```
public List<AccountDTO> unsafeJpaFindAccountsByCustomerId(String customerId) {
    String jql = "from Account where customerId = '" + customerId + "'";
    TypedQuery<Account> q = em.createQuery(jql, Account.class);
    return q.getResultList()
      .stream()
      .map(this::toAccountDTO)
      .collect(Collectors.toList());
}
```

Object-Relational Impedance Mismatch

RECAP

- Identity
  - Object: a == b and a.equals(b)
  - Relational: primary key based
- Inheritance
  - Object: natural relation
  - Relational: does not exist, idioms are necessary
- Accessing data
  - Object: through the object interface
  - Relational: select queries (with joins)
- Associations/Navigation
  - Object: unidirectional references through objects' interface
  - Relational: through foreign keys
- Granularity
  - In some cases there may be a difference in the granularity level

http://hibernate.org/orm/what-is-an-orm/

JPA - Java Persistence API

- Java Persistence API (Jakarta Persistence since Set 2019)
- Provides a OO interface to a relational database
- Defines JPQL (Java Persistence Query Language)
- The reference implementation is EclipseLink.
- Hibernate is a JPA provider (extended API and query language HQL)

Data Abstraction layers

- Advantages
  - Hides the complexity of a particular query language
  - Allows the portability of database engines (not really models)
  - Prevents attacks such as SQL injection, or XSS
  - Avoids runtime errors in the construction of queries
  - May isolate efficient implementations (previously, with prepared and compiled parameterised SQL queries)
  - Allows scalability via customised runtime configurations (distribution, transactional behaviour, indexing, …)
  - Avoids early optimization pitfalls, develop first, configure later.
- Pitfalls
  - Lack of access to proprietary features of providers
  - May lead to inefficient data transmissions: more queries (N+1 queries), more data than needed.

OpenJPA

JDO™
Java™ Data Objects

CakePHP

eclipselink

Microsoft® .NET
Entity
Framework

Internet Applications Design and Implementation 2020 - 2021

(Lecture 4 - Part 4 - Spring & Data Abstraction)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)

(with previous participations of Jácome Cunha (jacome@fct.unl.pt) and João Leitão (jc.leitao@fct.unl.pt))

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

JDBC & Spring

- SpringBoot provides direct support for JDBC
- As in any JDBC setting, it is necessary to define a DataSource (DB, CSV file, etc.)
- SpringBoot offers the JdbcTemplate class to assist programmers
- Spring easily integrates JPA support (later)

JDBC & Spring & In-memory DBs

- JdbcTemplate handles the setup and connection to the DataSource based on the POM dependencies (when possible)

- For instance, for H2 in-memory database (Spring also supports HSQL and Derby):

```
    <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
      </dependency>
```

- This handles the connection and exceptions

JDBC & Spring

- It also supports “regular” relational databases adding the necessary properties to the **`application.properties`** file, e.g.:

```
spring.datasource.url=jdbc:mysql://localhost/test
spring.datasource.username=dbuser
spring.datasource.password=dbpass
```

JDBC & Spring

```
@Component
public class Hotels {

    private JdbcTemplate jdbc;

    @Autowired
    public Hotels(JdbcTemplate jdbc) {
        this.jdbc = jdbc;
        createTable();
    }

    public Hotels(){}

    public void createTable() {
        jdbc.execute("drop table tablea if exists");
        jdbc.execute("create table tablea(id SERIAL, attributea VARCHAR(16))");
    }

    public List<Map<String,Object>> select() {
        return jdbc.query("select * from tablea", new ColumnMapRowMapper());
    }

    public int save(String att) {
        return jdbc.update("insert into tablea(attributea) values (?)", att);
    }
}
```

JPA & Spring

- Spring-boot-starter-data-jpa provides the necessary dependencies:
  - Hibernate — One of the most popular JPA implementations
  - Spring Data JPA — Makes it easy to implement JPA-based repositories
  - Spring ORMs — Core ORM support from the Spring Framework

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

Declare an Entity (in Java)

```java
@Entity
public class Customer {
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private Long id;
    private String firstName;
    private String lastName;

    protected Customer() {}

    public Customer(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }
    @Override
    public String toString() {
        return String.format(
                "Customer[id=%d, firstName='%s', lastName='%s']",
                id, firstName, lastName);
    }
}
```

Declare a Repository and plug it in… (in Java)

```
public interface CustomerRepository extends CrudRepository<Customer, Long> {

    List<Customer> findByLastName(String lastName);
}
```

```
@Controller
public class CustomerController
{
    @Autowired
    CustomerRepository customers;

   … findByLastName(“Smith”);…
}
```

Declare an Entity and a Repository and plug it in…

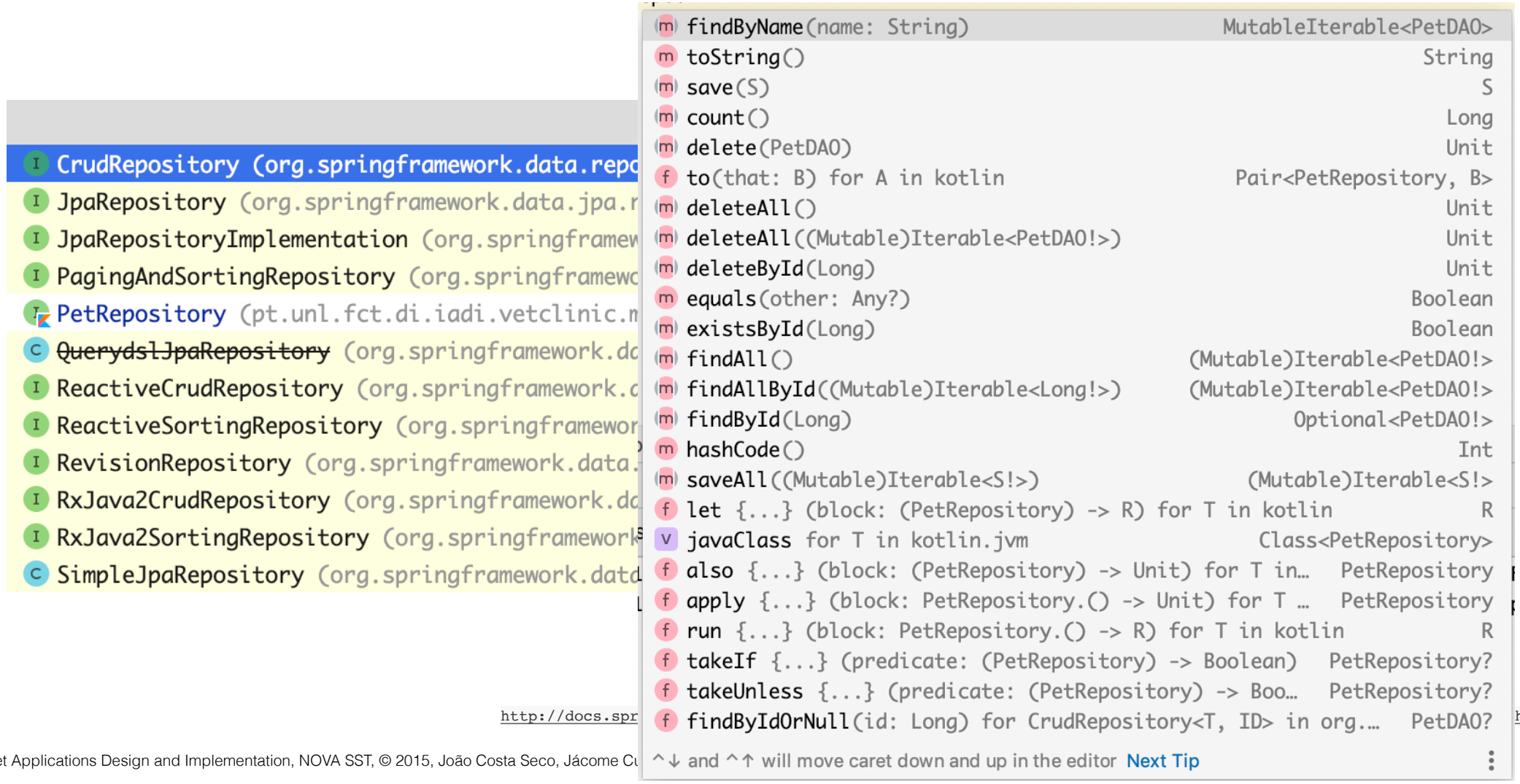
```kotlin
@Entity
data class PetDAO(
        @Id @GeneratedValue val id:Long,
        var name: String,
        var species: String
)
```

```kotlin
interface PetRepository : CrudRepository<PetDAO, Long> {
   fun findByName(name:String): MutableIterable<PetDAO>
}
```

```kotlin
@Autowired
lateinit var repo:PetRepository

fun someFunction() {

…repo.findByName("pantufas")…

}
```

```
findByName(name: String)                          MutableIterable<PetDAO>
toString()                                                         String
save(S)                                                                S
count()                                                             Long
delete(PetDAO)                                                      Unit
to(that: B) for A in kotlin                        Pair<PetRepository, B>
deleteAll()                                                         Unit
deleteAll((Mutable)Iterable<PetDAO!>)                               Unit
                                                                    Unit
                                                                 Boolean
                                                                 Boolean
                                                 (Mutable)Iterable<PetDAO!>
                                                 (Mutable)Iterable<PetDAO!>
                                                    Optional<PetDAO!>
hashCode()                                                          Int
saveAll((Mutable)Iterable<S!>)                   (Mutable)Iterable<S!>
let {...} (block: (PetRepository) -> R) for T in kotlin                R
javaClass for T in kotlin.jvm                    Class<PetRepository>
also {...} (block: (PetRepository) -> Unit) for T in…   PetRepository
apply {...} (block: PetRepository.() -> Unit) for T …   PetRepository
run {...} (block: PetRepository.() -> R) for T in kotlin               R
takeIf {...} (predicate: (PetRepository) -> Boolean)    PetRepository?
takeUnless {...} (predicate: (PetRepository) -> Boo…    PetRepository?
findByIdOrNull(id: Long) for CrudRepository<T, ID> in org.…   PetDAO?
^↓ and ^↑ will move caret down and up in the editor  Next Tip
```

interface CrudRepository<T , ID extends Serializable>



An Example

```
@Entity
@NamedQuery(name = "User.findByTheUsersName",
            query = "from User u where u.username = ?1")
class User(
            @Column(unique = true)
            val username:String,
            val firstname:String,
            val lastname:String
)
```

```
interface SimpleUserRepository : CrudRepository<User, Long> {

    fun findByTheUsersName(username:String):User

    fun findByLastname(lastname:String):List<User>

    @Query("select u from User u where u.firstname = ?")
    fun findByFirstname(firstname:String):List<User>

    @Query("select u from User u where u.firstname = :name or u.lastname = :name")
    fun findByFirstnameOrLastname(@Param("name") name:String):List<User>
}
```

Keyword	Sample	JPQL snippet
And	findByLastnameAndFirstname	… where x.lastname = ?1 and x.firstname
Or	findByLastnameOrFirstname	… where x.lastname = ?1 or x.firstname =
Is,Equals	findByFirstname,findByFirstnameIs,findByFirstnameEquals	… where x.firstname = 1?
Between	findByStartDateBetween	… where x.startDate between 1? and ?2
LessThan	findByAgeLessThan	… where x.age < ?1
LessThanEqual	findByAgeLessThanEqual	… where x.age ⇐ ?1
GreaterThan	findByAgeGreaterThan	… where x.age > ?1
GreaterThanEqual	findByAgeGreaterThanEqual	… 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
EndingWith	findByFirstnameEndingWith	… where x.firstname like ?1 (parameter
Containing	findByFirstnameContaining	… where x.firstname like ?1 (parameter
OrderBy	findByAgeOrderByLastnameDesc	… where x.age = ?1 order by x.lastname
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
IgnoreCase	findByFirstnameIgnoreCase	… where UPPER(x.firstame) = UPPER(?

Spring Data (and other ORM Implementations)

SUMMARY

- Simple declaration of generic queries
- Specific declaration of custom queries (JPQL)
- Richer Behaviour from Repositories (e.g. paged)

```
public interface StudentRepository extends PagingAndSortingRepository<Student, Long> {

    List<Student> findByName(String name);

    @Query("select s from Student s where s.name like CONCAT(?,'%')")
    List<Student> search(String name);

    Page<Student> findByName(String name, Pageable pageable);
}
```

- Dependency Injection and component assembly

```
public class StudentController {

    @Autowired
    StudentRepository students;
```

Spring Data (and other ORM Implementations)

SUMMARY

- Simple declaration of generic queries
- Specific declaration of custom queries (JPQL)
- Richer Behaviour from Repositories (e.g. paged)

```
public interface StudentRepository extends PagingAndSortingRepository<Student, Long> {

    List<Student> findByName(String name);
```

```
@RequestMapping(value= "/page")
public @ResponseBody Page<Student> getStudentsPaged(
      @RequestParam(required=false, defaultValue =  "0") Integer page,
      @RequestParam(required=false, defaultValue = "3") Integer size) {

   return students.findAll(new PageRequest(page, size));
}
```

- Depe

```
@Autowired
StudentRepository students;
```

Spring Data (and other ORM Imp

SUMMARY

- Simple declaration of generic queries
- Specific declaration of custom queries (JP
- Richer Behaviour from Repositories (e.g. p
- Depe

```
public interface StudentRepository extends Pag

    List<Student> findByName(String name);
```

```
@RequestMapping(value= "/page")
public @ResponseBody Page<Student> ge
      @RequestParam(required=false, de
      @RequestParam(required=false, de

   return students.findAll(new PageRe
}
```

```
@Autowired
StudentRepository students;
```

```
{
  "content": [
    {
      "id": 1,
      "name": "Ingrid Daubechies",
      "age": 19
    },
    {
      "id": 2,
      "name": "Jacqueline K. Barton",
      "age": 18
    },
    {
      "id": 3,
      "name": "Jane Goodall",
      "age": 20
    }
  ],
  "last": false,
  "totalElements": 6,
  "totalPages": 2,
  "size": 3,
  "number": 0,
  "sort": null,
  "first": true,
  "numberOfElements": 3
}
```

Spring Data - Features

- Powerful repository and custom object-mapping abstractions
- Dynamic query derivation from repository method names
- Implementation domain base classes providing basic properties
- Support for transparent auditing (created, last changed)
- Possibility to integrate custom repository code
- Easy Spring integration via JavaConfig and custom XML namespaces
- Advanced integration with Spring MVC controllers
- Experimental support for cross-store persistence

Spring Data

- Spring Data Commons - Core Spring concepts underpinning every Spring Data project.
- Spring Data JPA - Makes it easy to implement JPA-based repositories.
- Spring Data MongoDB - Spring based, object-document support and repositories for MongoDB.
- Spring Data Redis - Provides easy configuration and access to Redis from Spring applications.
- Spring Data Solr - Spring Data module for Apache Solr.
- Spring Data Gemfire - Provides easy configuration and access to GemFire from Spring applications.
- Spring Data REST - Exports Spring Data repositories as hypermedia-driven RESTful resources.

Community Modules

- Spring Data Cassandra - Spring Data module for Apache Cassandra.
- Spring Data Couchbase - Spring Data module for Couchbase.
- Spring Data DynamoDB - Spring Data module for DynamoDB.
- Spring Data Elasticsearch - Spring Data module for Elasticsearch.
- Spring Data Neo4j - Spring based, object-graph support and repositories for Neo4j.

MongoDB Example

```
public class MongoApp {

  private static final Log log = LogFactory.getLog(MongoApp.class);

  public static void main(String[] args) throws Exception {

    MongoOperations mongoOps = new MongoTemplate(new Mongo(), "database");
    mongoOps.insert(new Person("Joe", 34));

    log.info(mongoOps.findOne(new Query(where("name").is("Joe")), Person.class));

    mongoOps.dropCollection("person");
  }
}
```

Other database representations

- Objectify: gives easy and full access to the Google Cloud Datastore
- Dynamic / Reflexion based
- Alternative to JPA interface

```
@Entity
class Car {
    @Id String vin; // Can be Long, long, or String
    String color;
}

ofy().save().entity(new Car("123123", "red")).now();
Car c = ofy().load().type(Car.class).id("123123").now();
ofy().delete().entity(c);
```

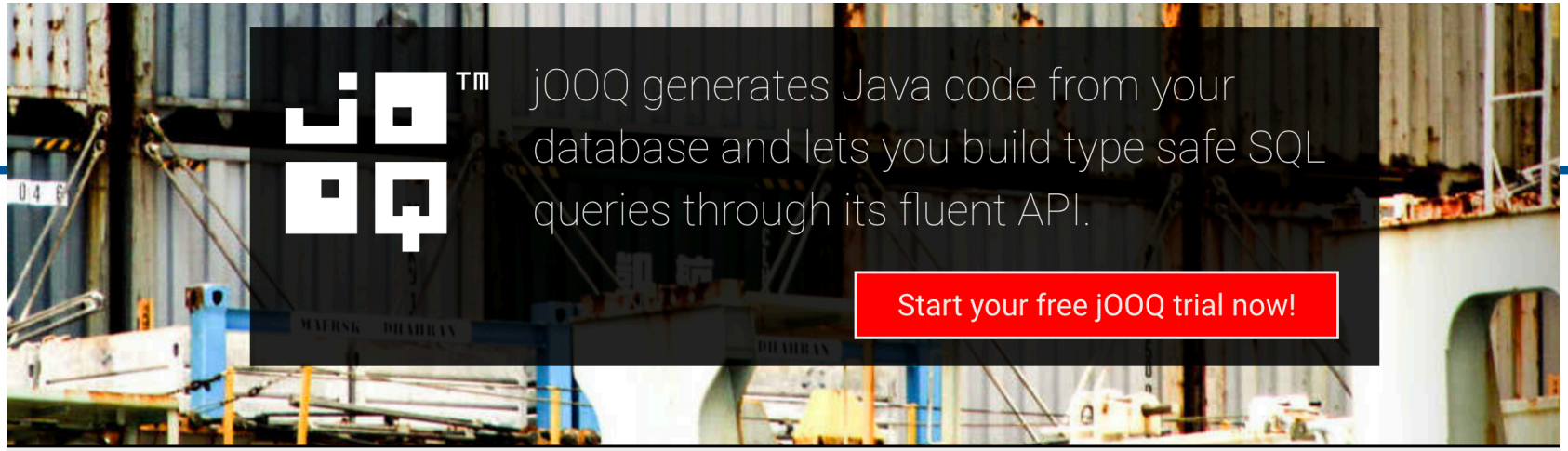
https://github.com/objectify/objectify

Other database representations

- Objectify: gives easy and full access to the Google Cloud Datastore
- Dynamic / Reflexion based
- Alternative to JPA interface

- Objectify surfaces **all native datastore features**, including batch operations, queries, transactions, asynchronous operations, and partial indexes.
- Objectify provides **type-safe key and query classes** using Java generics.
- Objectify provides a **human-friendly query interface**.
- Objectify can automatically **cache your data in memcache** for improved read performance.
- Objectify can store polymorphic entities and perform **true polymorphic queries**.
- Objectify provides a simple, **easy-to-understand transaction model**.
- Objectify provides built-in facilities to **help migrate schema changes** forward.
- Objectify provides **thorough documentation** of concepts as well as use cases.
- Objectify has an **extensive test suite** to prevent regressions.

https://github.com/objectify/objectify



Great Reasons for Using jOOQ

Our customers spend most time on *their* business-logic.
Because jOOQ takes care of all their Java/SQL infrastructure problems.

https://www.jooq.org/

Database First

Tired of ORMs driving your database model?

Whether you design a new application or integrate with your legacy, your database holds your most important asset: your data.

jOOQ is SQL-centric. Your database comes "first".

Typesafe SQL

Fed up with detecting SQL syntax errors in production?

SQL is a highly expressive and type safe language with a rich syntax. jOOQ models SQL as an internal DSL and uses the Java compiler to compile your SQL syntax, metadata and data types.

Code Generation

Bored with renaming table and column names in your Java code?

jOOQ generates Java classes from your database metadata. Your Java compiler will tell you when your code is out of sync with your schema.

Active Records

Annoyed by the amount of SQL you write for CRUD?

jOOQ lets you perform CRUD and POJO mapping directly on Active Records, which are also generated from the code generator.