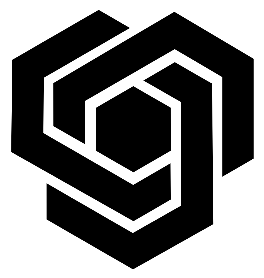
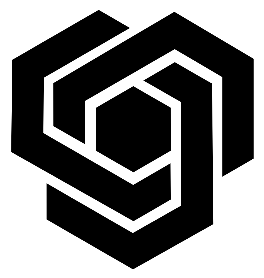
****

**ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ**

**КУРСОВ ПРОЕКТ**

**ПО**

**СЪВРЕМЕННИ JAVA**

**ТЕХНОЛОГИИ**

**ИЗГОТВИЛ: Християн Цветанов Зарков Проверил:** **гл ас. Явор Томов**

**Факултетен номер: 121319034 Подпис:……………... ДАТА: 01.02.2020**

Spring Framework е технологична рамка (framework) с [отворен код](https://bg.wikipedia.org/wiki/Отворен_код) (open source) за Java платформата. Spring framework предоставя много функции, които улесняват разработването на Java-базирани enterprise приложения.

Основното ядро в Spring Framework е неговият IoC (Inversion of Control) контейнер, който осигурява логически средства за конфигуриране и управление на Java обекти с помощта на reflection. Контейнерът е отговорен за управлението на жизнения цикъл на специфични обекти: създаването на тези обекти, извикването на инициализиращите им методи и конфигурирането им свързвайки ги заедно. Обектите, създадени от контейнера се наричат още управлявани обекти или бийнове (beans). Контейнерът може да бъде конфигуриран чрез зареждане на XML файлове или засичането на специфични Java анотации от конфигуриращите класове.

Обектите могат да бъдат получени чрез dependency lookup или dependency injection. Dependency injection е софтуерен шаблон за дизайн, който опростява Обектно-ориентираното програмиране (ООП) чрез намаляване на зависимостите между отделните класове. Така при създаването на един проект, използвайки някой обектно-ориентиран език се избягва зависимостта от конкретни имплементации, и се набляга повече на абстракции. Това се нарича още обръщане на контрола (inversion of control) – когато модулите от високо ниво не зависят от модули от по-ниско ниво.

При dependency injection обектите се проектират по начин, при който те получават инстанции на обекти от различен външен модул код, вместо да са конструирани вътре в него. Dependency injection включва най-общо три елемента:

* зависим елемент;
* декларация на зависимостите на елемента, дефинирани като интерфейси, абстрактни класове;
* контейнер на зависимостите (инжектор), който създава инстанции на класове, които реализират дадена интерфейсна зависимост при заявка.

Зависимият елемент определя на какъв софтуерен компонент зависи за да си свърши работата, а контейнерът решава какви конкретни класове отговарят на изискванията на зависимия обект, и ги предоставя като зависимости. По този начин контейнерът по свой избор, може да променя конкретните имплементации, които се използват по време на изпълнение на програмата (run-time), а не по време на компилация (compile-time). По време на изпълнение (run-time) могат да бъдат създадени множество различни инстанции на даден софтуерен компонент, без значение, каква конкретна е била инжектирана.

Това позволява кодът, който пишем да е много по модуларен и разкачен, което го прави в същото време и много по-тестваем.

Така наречените Spring frameworks за достъп на данни се отнасят за често срещани проблеми при работа с база данни, които разработчиците срещат. Както следват всички най-известни frameworks за достъп на данни в Java се поддържат: JDBC, iBatis/MyBatis, Hibernate, JDO, JPA, Oracle TopLink, Apache OJB, и Apache Cayenne и други.

За всички изброени до тук frameworks, Spring поддържа следните характеристики:

* Resource management – Автоматично поискване и получаване на ресурси от базата данни;
* Exception handling – превежда уникалните данните за достъп в Spring йерархия;
* Transaction participation – прозрачно участие в настоящи транзакции;
* Resource unwrapping – извличане на база данни обекти от pool wrappers към който е свързан;
* Abstraction – абстракция за BLOB и CLOB манипулации.

Всички тези възможности са достъпни когато се използват шаблонни класове (template classes) предоставени за всеки framework който Spring поддържа. Критиците казват, че тези шаблонни класове са натрапчиви и не предлагат предимства пред използването примерно на Hibernate API directly. В отговор на това, разработчиците на Spring направиха възможно използването на Hibernate и JPA APIs directly. Това от своя стана изисква използването на управление на прозрачни транзакции, тъй като кода на приложението вече не поддържа свойството за получаване и затваряне на ресурси от базата данни, и не поддържа уникални транзакции.

Заедно с управлението на Spring's транзакции, този framework предлага гъвкава абстракция за работа с frameworks за достъп на данни. Spring Framework не предлага стандартен API достъп на данни; вместо това цялата мощ на поддържащото API е оставена непокътната. Spring Framework е единствения framework достъпен за Java, който предлага управление на достъпа до данните извън сървъра или контейнера на приложението.

Model-View-Controller ([*MVC*](https://bg.wikipedia.org/wiki/MVC)) е архитектурен модел за изграждане на потребителски интерфейси. Той разделя дадено софтуерно приложение на три взаимносвързани части, така че да се отделят вътрешни представяния на информация от начините, по които информацията се представя на потребителя/или от страна на потребителя: *контролери*, *изгледи*, *модели*. Както и при други софтуерни модели, *MVC* изразява „core of the solution“ на проблем, докато позволява да бъде адаптиран за всяка система. Специфични *MVC* архитектури могат да се различават значително.

Елементи: Централният елемент на *MVC – The model* *моделът*, улавя поведението на приложението по отношение на своя домейн, независимо от потребителския интерфейс. Моделът директно управлява данните, логиката и правилата на приложението. *The view* *изгледът* може да бъде всеки изход за представяне на информация, например чрез графика или диаграма; множество изгледи към една и съща информация са възможни, като стълбчета за управление и табличен изглед за счетоводители.

Третата част *Controller* *контролера*, приема входа от потребителя и го конвертира в команди за модела или изгледа.

Взаимодействия: В допълнение за разделяне на приложението на трите вида елементи, дизайн на *model–view–controller* определя взаимодействието между тях. *The Controller* – Контролерът може да изпраща команди към модела за актуализиране състоянието на модела (например, редактирането на документ). Той също може да изпраща команди към свързаната с изгледа част, за да се промени представянето на модела (например, чрез скролиране на документа).

*The model* – Моделът уведомява свързаните части с изгледа и контролера, когато е налице промяна в състоянието му. Това уведомяване позволява на изгледа да изведе на изхода актуалната информация и контролера да променя разположението на наборите от команди.

*The view* – Изгледа изисква информация от модела, който използва, за да се генерира изходната информация за представяне на потребителя.

Използване в уеб приложения: Въпреки че първоначално е разработен за настолните компютри *Model-View-Controller* е широко приет като архитектура за *World Wide Web* приложения в голяма част от езици за програмиране. Няколко търговски и нетърговски уеб приложения са създадени, които прилагат тази схемата. Тези приложения се различават по своите интерпретации, основно в начина, по който отговорностите им *Model-View-Controller* са разделени между клиента и сървъра.