Clase 9

Spring

Spring

- Librería de código abierto
- Orientado al desarrollo de aplicaciones eficientes y escalables
- Brinda herramientas que simplifican y aceleran el desarrollo
- Se busca la implementación de buenas prácticas del desarrollo
- Modular (de acuerdo a las necesidades del proyecto)
    - Otros proyectos de Spring
    - Otros proyectos populares
        - Hibernate (ORM)
        - JPA (Java Persistence) / JDBC
        - JMS (Java Message Service)

Componentes

Spring Boot

Takes an opinionated view of building Spring applications and gets you up and running as quickly as possible.

Spring Framework

Provides core support for dependency injection, transaction management, web apps, data access, messaging, and more.

Spring Session

Provides an API and implementations for managing a user's session information.

Spring Data

Provides a consistent approach to data access – relational, non-relational, map-reduce, and beyond.

Spring Cloud

Provides a set of tools for common patterns in distributed systems. Useful for building and deploying microservices.

Spring HATEOAS

Simplifies creating REST representations that follow the HATEOAS principle.

Spring Cloud Data Flow

Provides an orchestration service for composable data microservice applications on modern runtimes.

Spring Security

Protects your application with comprehensive and extensible authentication and authorization support.

Spring Batch

Simplifies and optimizes the work of processing high-volume batch operations.

Spring Authorization Server

Provides a secure, light-weight, and customizable foundation for building OpenID Connect 1.0 Identity Providers and OAuth2 Authorization Server products.

Spring for GraphQL

Spring for GraphQL provides support for Spring applications built on GraphQL Java.

Spring Flo

Provides a JavaScript library that offers a basic embeddable HTML5 visual builder for pipelines and simple graphs.

Componentes

Spring Integration

Supports the well-known Enterprise Integration Patterns through lightweight messaging and declarative adapters.

Spring LDAP

Simplifies the development of applications that use LDAP by using Spring's familiar template-based approach.

Spring Shell

Makes writing and testing RESTful applications easier with CLI-based resource discovery and interaction.

Spring REST Docs

Lets you document RESTful services by combining hand-written documentation with auto-generated snippets produced with Spring MVC Test or REST Assured.

Spring Statemachine

Provides a framework for application developers to use state machine concepts with Spring applications.

Spring Web Flow

Supports building web applications that feature controlled navigation, such as checking in for a flight or applying for a loan.

Spring AMQP

Applies core Spring concepts to the development of AMQP-based messaging solutions.

Spring Web Services

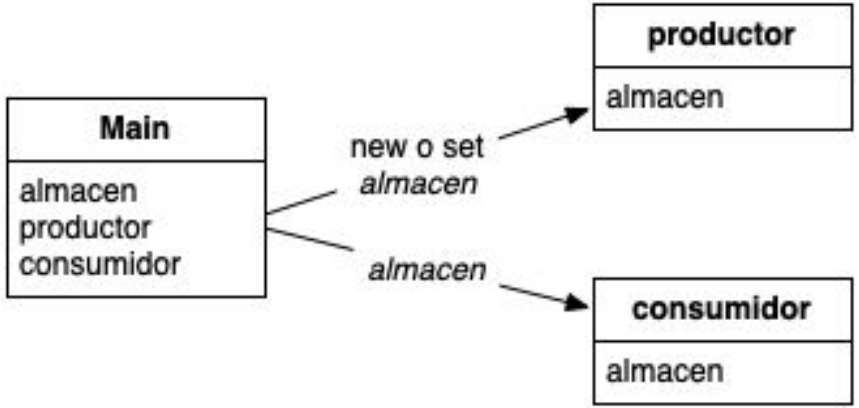
Facilitates the development of contract-first SOAP web services.

Spring for Apache Kafka

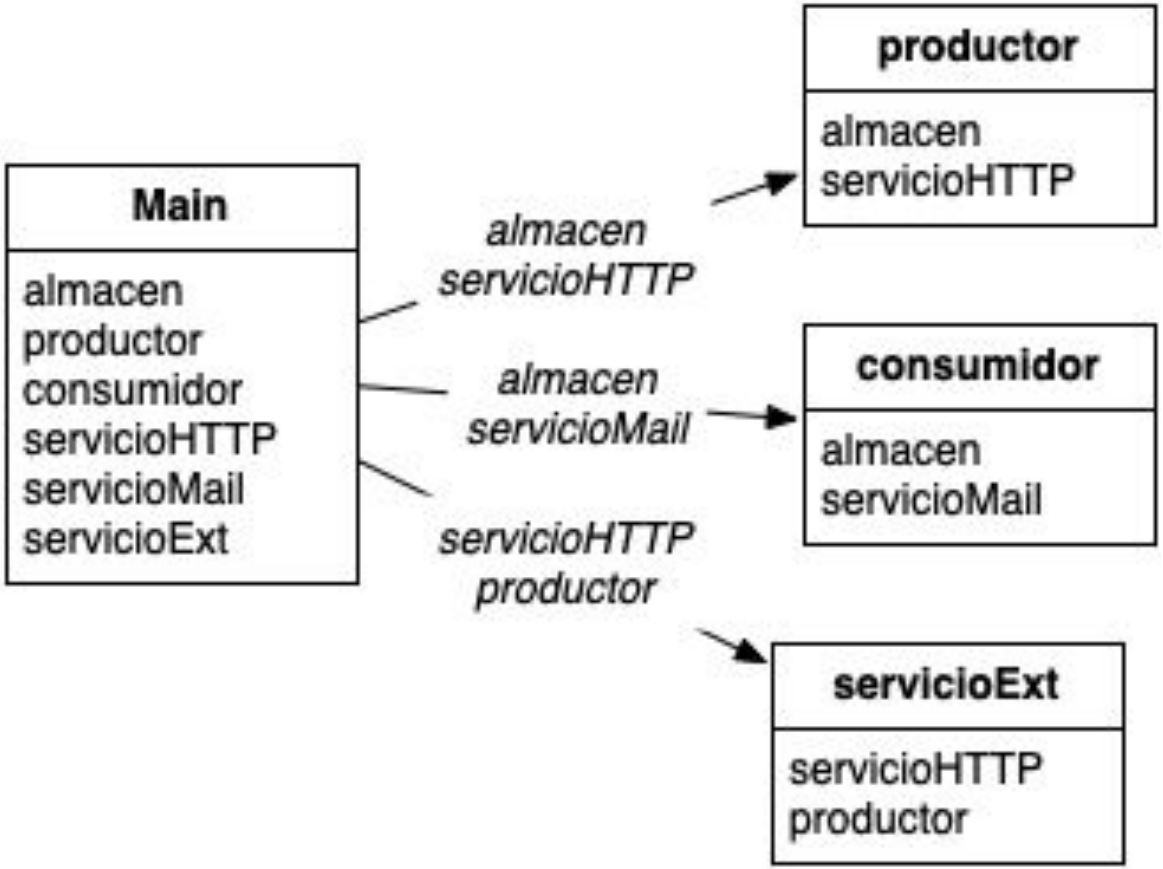
Provides Familiar Spring Abstractions for Apache Kafka.

Objetos y servicios

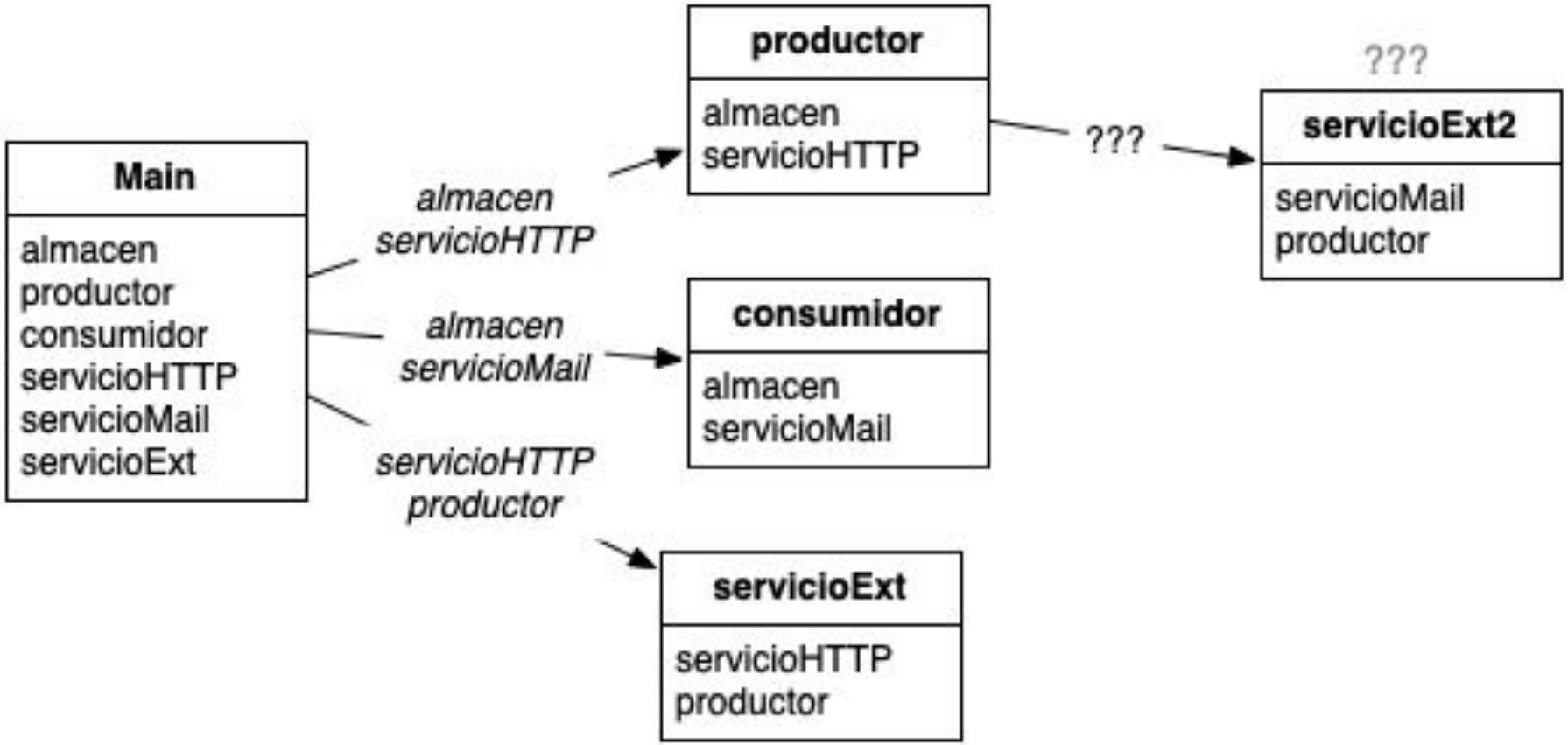
```java
public class ArranquePC {

    Run | Debug
    public static void main(String[] args) {
        ArranquePC arranque = new ArranquePC();
        arranque.arranque();
    }

    public void arranque() {
        // Creamos una instancia del almacenamiento compartido
        Almacenamiento almacenamiento = new Almacenamiento();

        // Creamos una instancia del productor y lo iniciamos
        Productor productor = new Productor(almacenamiento);
        Thread hiloProductor = new Thread(productor);
        hiloProductor.start();

        // Creamos una instancia del consumidor y lo iniciamos
        Consumidor consumidor = new Consumidor(almacenamiento);
        Thread hiloConsumidor = new Thread(consumidor);
        hiloConsumidor.start();
    }
}
```



Objetos y servicios



Objetos y servicios



IoC - Inversión de control

Se delega la creación de objetos a un Contenedor de objetos (Application Context) en vez de realizarlo a mano.

Se utiliza:

- inyección de dependencias (DI - Dependency Injection) - normalmente

  Se logra mediante la inyección de:

  - Constructor
  - Setters
  - Inyección por campo (@Autowired)
- Factorías (factory)
- Templates

IoC - Beneficios

- Desacoplamiento

  Se reduce el acoplamiento entre componentes
- Reutilización

  Delegar la creación de objetos al contenedor hace que sea más sencilla la configuración/reconfiguración del código o clases existentes
- Pruebas unitarias y de integración

  Separar las dependencias hace más sencillo crear objetos simulados al realizar las pruebas

Agregando Spring

```
<properties>
   <maven.compiler.source>17</maven.compiler.source>
   <maven.compiler.target>17</maven.compiler.target>
   <spring.version>5.3.27</spring.version>
</properties>
<dependencies>
   <dependency>
       <groupId>org.springframework</groupId>
       <artifactId>spring-context</artifactId>
       <version>${spring.version}</version>
   </dependency>
</dependencies>
```

Clases necesarias

- ApplicationConfig.java (@Configuration) donde:
  - se crean los beans necesarios para referenciar en la aplicación
  - se indica en dónde se va a escanear los componentes
- Main.java donde:
  - se inicia la aplicación
  - se crea el ApplicationContext
- ApplicationContextProvider.java donde:
  - Se guarda referencia del applicationContext para trabajar con hilos
- Arranque.java donde:
  - Se encuentra el primer llamado, que es el lugar para que arranque la aplicación
- Otras clases

Main.java

```
public class Main {

    public static void main(String[] args) {
        ApplicationContext context =
            new AnnotationConfigApplicationContext(ApplicationConfig.class);
        Arranque arranque = context.getBean(Arranque.class);
        ApplicationContextProvider applicationContextProvider =
            new ApplicationContextProvider();
        applicationContextProvider.setApplicationContext(context);
        arranque.arranque();
    }
}
```

ApplicationConfig.java

```java
@Configuration
@ComponentScan("ar.edu.um.programacion2.anio2023.spring")
public class ApplicationConfig {

    @Bean
    public Arranque arranque() {
        return new Arranque();
    }
}
```

ApplicationContextProvider.java

```
public class ApplicationContextProvider implements ApplicationContextAware {

    private static ApplicationContext context;

    @Override
    public void setApplicationContext(ApplicationContext applicationContext)
            throws BeansException {
        context = applicationContext;
    }

    public static ApplicationContext getContext() {
        return context;
    }
}
```

Arranque.java

```
public class Arranque {

    @Autowired
    ServicioPrueba servicioPrueba;

    @Autowired
    Almacenamiento almacenamiento;
    public void arranque() {
        System.out.println("Arranque");
        this.servicioPrueba.llamado1();

        // Creamos una instancia del productor y lo iniciamos
        Productor productor = new Productor();

        // Creamos una instancia del consumidor y lo iniciamos
        Consumidor consumidor = new Consumidor();

        ExecutorService es = Executors.newFixedThreadPool(2);
        es.submit(productor);
        es.submit(consumidor);
    }
}
```

ServicioPrueba.java

```java
@Service
public class ServicioPrueba {

    public void llamado1() {
        System.out.println("Llamado 1");
    }

}
```

Almacenamiento.java

```java
@Service
public class Almacenamiento {
    private Queue<Integer> cola;
    private int capacidad;

    public Almacenamiento() {
        cola = new LinkedList<>();
        capacidad = 5;
    }

    public synchronized void agregar(int valor) {
        while (cola.size() == capacidad) {
            try {
                wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        cola.add(valor);
        System.out.println("Productor agrega " + valor + " en la cola.");
        notifyAll();
    }
```

Consumidor.java

```
public class Consumidor implements Runnable{
    private Almacenamiento almacenamiento;

    public Consumidor() {
        ApplicationContext context = ApplicationContextProvider.getContext();
        this.almacenamiento = context.getBean(Almacenamiento.class);
    }

    public Consumidor(Almacenamiento almacenamiento) {
        this.almacenamiento = almacenamiento;
    }

    public void run() {
        for (int i = 1; i <= 10; i++) {
            int valor = almacenamiento.retirar();
            System.out.println("Retirado: "+valor);
            try {
                Thread.sleep(2000);
            } catch (InterruptedException e) {
                e.printStackTrace();
```

Configuración

Se puede configurar una aplicación usando archivos externos

src/main/resources/config.properties

```
clase9.conf1.parametro1=valor1
servicioHTTP.url=http://www.pepe.com
```

```
@Configuration
@Data
public class PruebaConfiguration {
    @Value("${clase9.conf1.parametro1}")
    protected String parametro1;

    @Value("${servicioHTTP.url}")
    protected String url;
}
```