

# Spring ve AMQP



# AMQP Nedir?

- AMQP'nin açılımı “advanced message queuing protocol”dür
- Farklı mimari ve teknolojilerle geliştirilen uygulamaların birbirleri ile **asen kron haberleşmelerini** sağlayan **açık ve cross-platform bir spesifikasyondur**
- Mesajlaşma olduğu için Java dünyasında çoğunlukla **JMS'e benzetilir** ve onunla kıyaslanır

# Neden JMS Değil?

- JMS'de **mesaj formatı standart değildir**
- JMS'de **mesajların nasıl yapılandırılıp, transfer edileceği ile ilgili bir standart yoktur**
- AMQP'de ise buralar standart olduğu için **farklı teknolojilerde geliştirilen sistemlerin birbirleri ile haberleşmeleri mümkündür**

# Neden Spring AMQP?

- AMQP temelli mesajlaşma çözümleri üretmek için **Spring merkezli** bir yaklaşım sunar
- **Farklı broker'larda** mesaj gönderme ve alma için “**template**” **tabanlı** bir AMQP'nin üstünde ayrı bir abstraction sağlar
- **Message Driven POJO**'lar oluşturmaya imkan tanır

# Spring-Rabbit Nedir?

- **Spring AMQP abstract modülünün** spesifik bir **implemantasyonudur**
- **RabbitMQ Broker'ı** için geliştirilmiştir

# Spring AMQP

## Bileşenleri: Message

- AMQP spesifikasyonunda **ayrı bir mesaj sınıfı veya arayüzü yoktur**
- Mesaj içeriği `byte[]`, diğer özellikleri ise ayrı metot parametreleri şeklinde gönderilir
- Spring AMQP ise **ayrı bir Message sınıfı tanımlar**
- Amacı **mesaj içeriğini ve özelliklerini bir arada encapsule edebilmek** ve API'yi basitleştirmektir

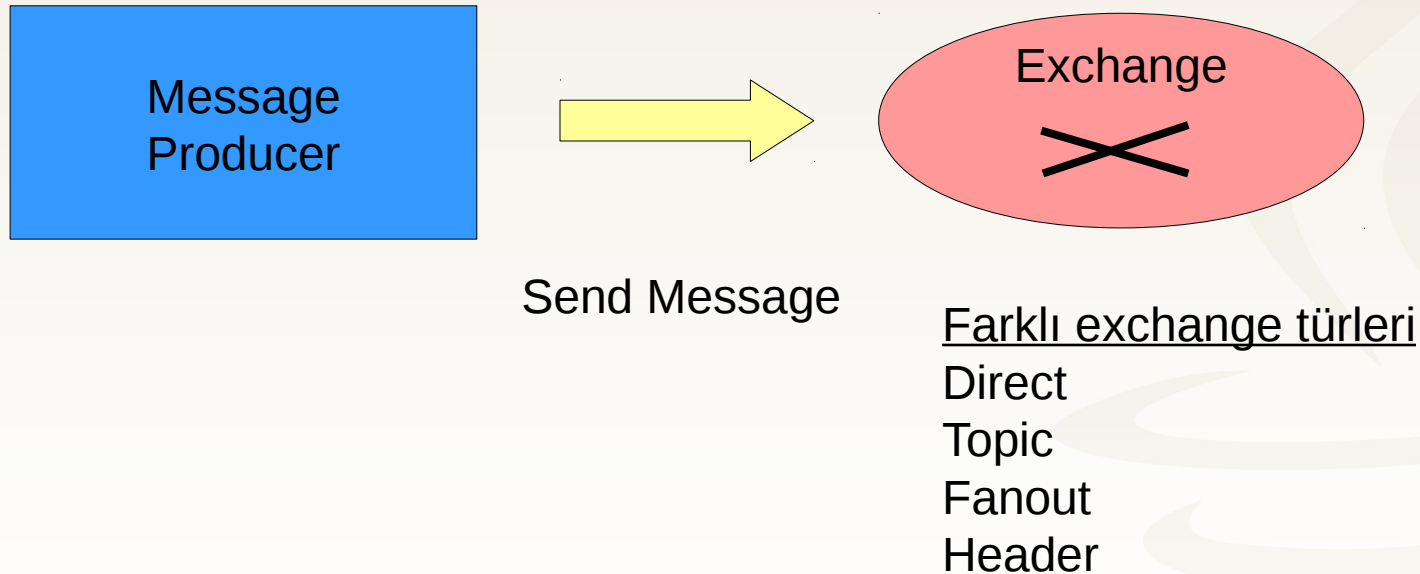
# Spring AMQP

## Bileşenleri: Exchange

- Mesaj üreticisinin (message producer) **mesajı gönderdiği yapıdır**
- Bir AMQP broker'ındaki her bir virtual host altında exchange nesneleri **benzersiz isime** sahiptir
- Değişik **exchange tipleri** vardır
  - Direct, Topic, Fanout, Headers
  - Bu tiplerin davranışları **queue binding'in nasıl yapıldığına** göre değişiklik gösterir

# Spring AMQP

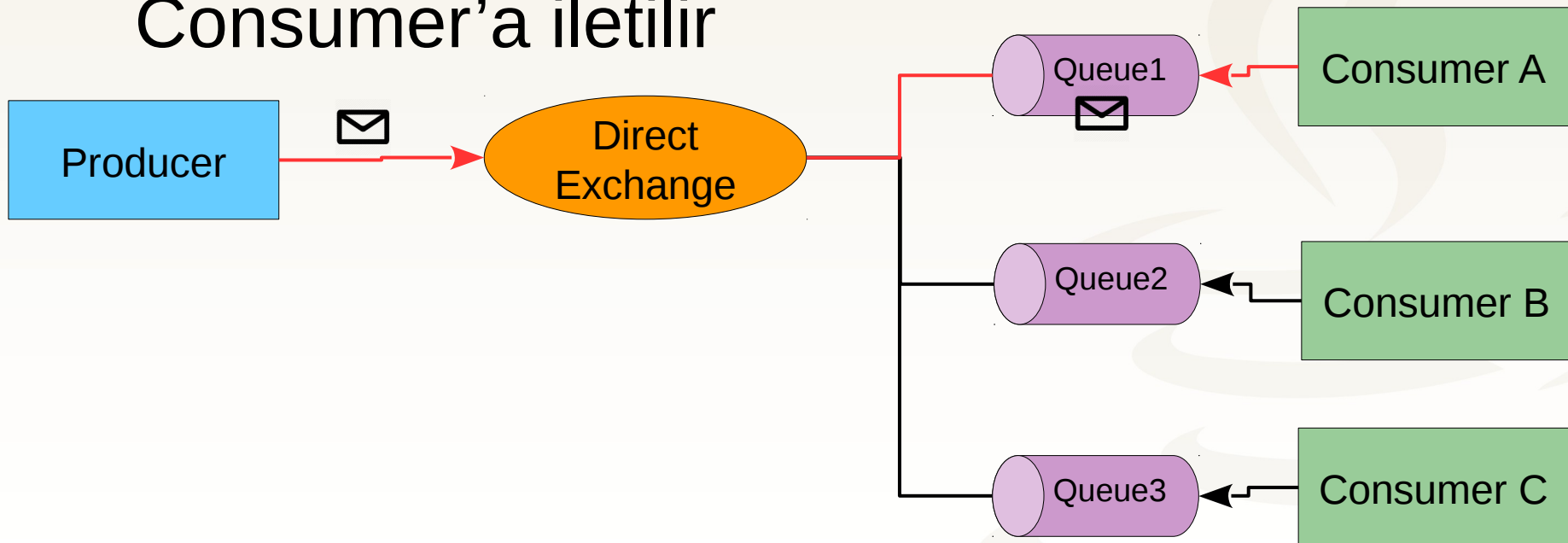
## Bileşenleri: Exchange





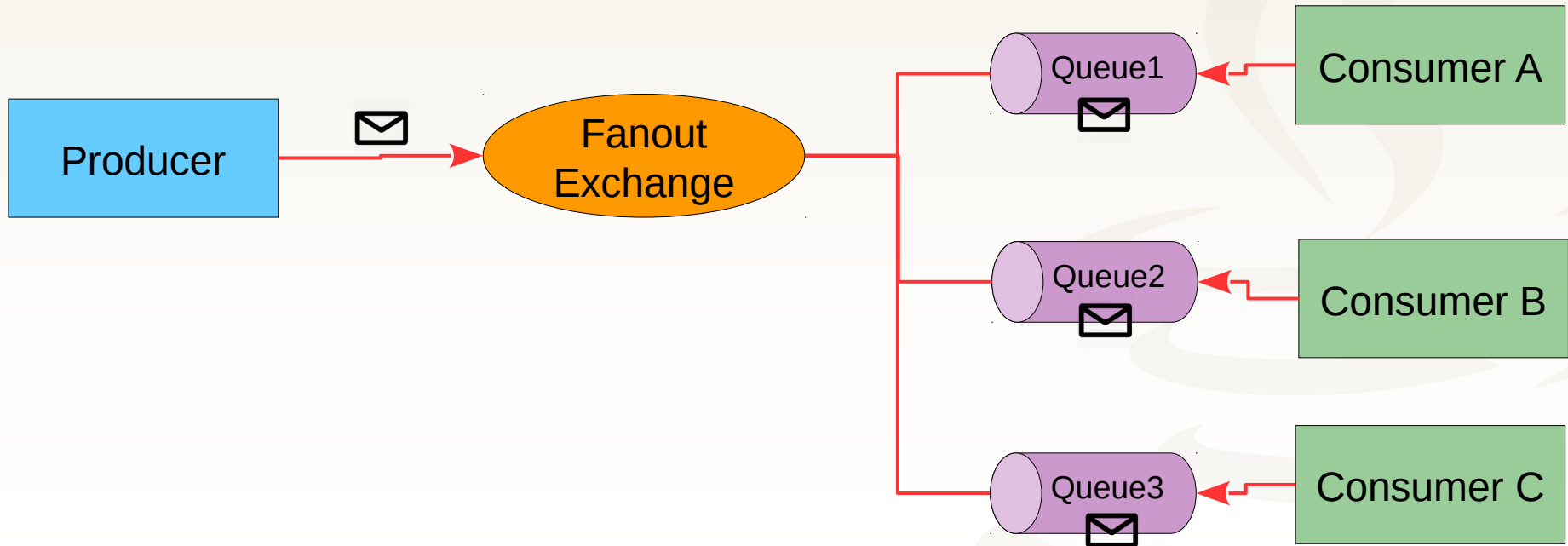
# Exchange Türleri: Direct

- Producer ve Consumer arasında **bire bir (point-to-point)** bir mesajlaşma sağlar
- Mesaj key eşleşmesine göre sadece tek bir Consumer'a iletilir



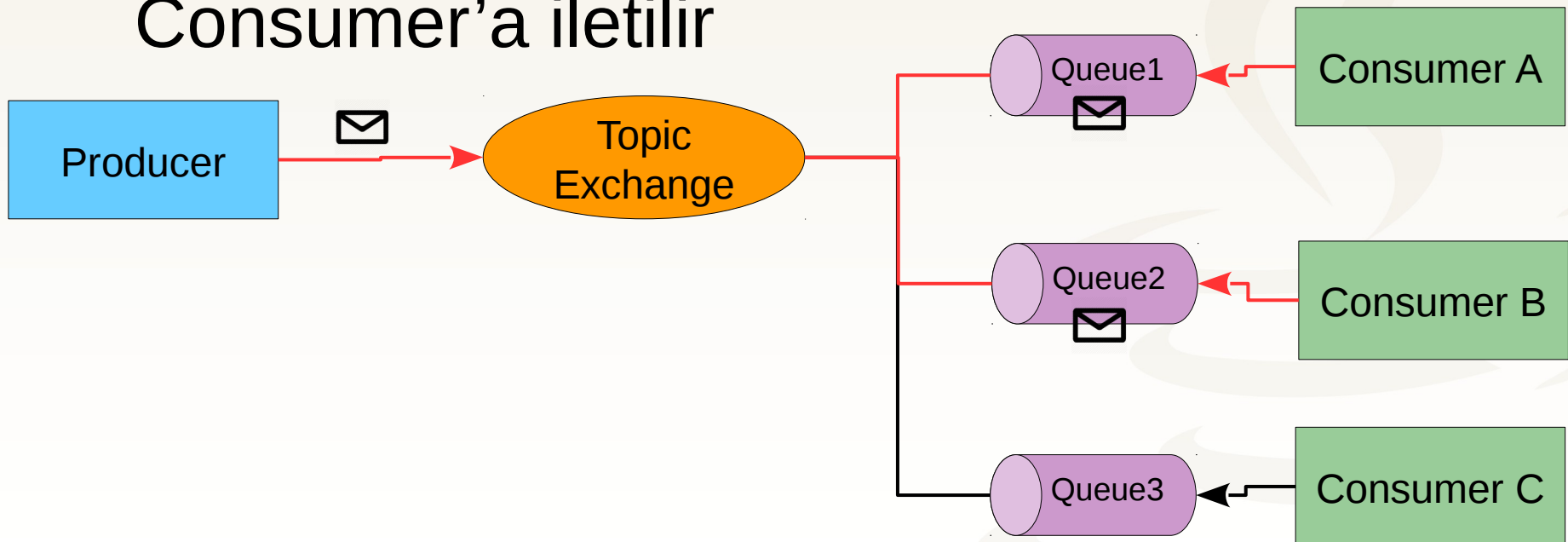
# Exchange Türleri: Fanout

- Producer için **broadcast mesajlaşma** imkanı sunar
- Mesaj bütün Consumer'lara iletilir



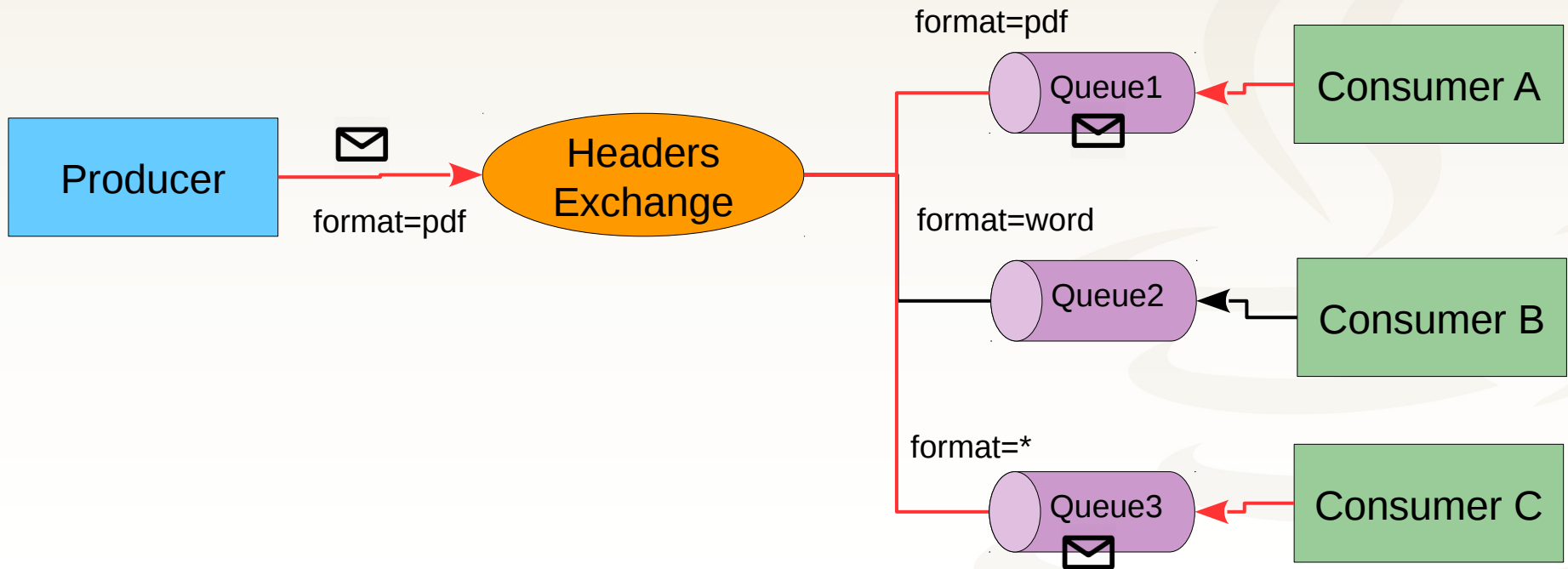
# Exchange Türleri: Topic

- Producer için multicast mesajlaşma imkanı sağlar
- Mesaj key eşleşmesine göre bir grup Consumer'a iletilir



# Exchange Türleri: Headers

- Mesaj yönlendirmesi için mesaj key yerine mesaj header değerlerine bakılır



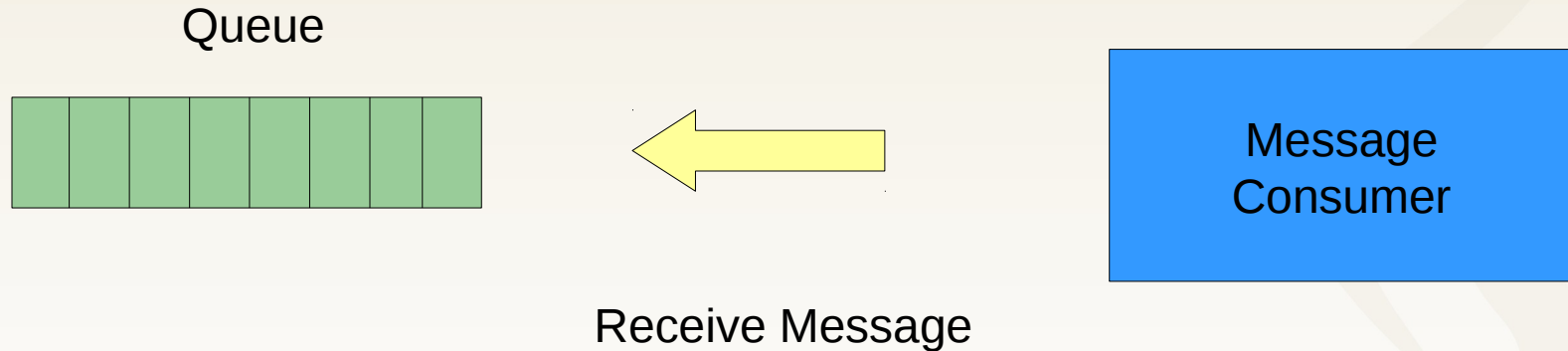
# Spring AMQP

## Bileşenleri: Queue

- Mesaj tüketicisi (message consumer)'nin **mesajı aldığı yapıdır**
- Bir queue nesnesi **durable, exclusive** ve **auto-delete** özelliklerine sahip olabilir

# Spring AMQP

## Bileşenleri: Queue



# Spring AMQP

## Bileşenleri: Binding



Exchange ve Queue yapılarının birbirleri ile bağlantısının kurulması gerekir

# Spring AMQP

## Bileşenleri: Binding

- Exchange ve queue yapıları **arasındaki bağlantıyı** ifade eder
- **Bağlantı ile ilgili bir veri** tutmaktadır, herhangi bir davranış sergilemez
- Queue-exchange **binding işlemi farklı opsiyonlarla** gerçekleştirilebilir



# Spring AMQP Bileşenleri

- Spring AMQP,
  - AMQP ile ilgili **bean tanımlarını otomatik** olarak üreten
  - **Queue, Exchange ve Binding** tanımlarını tespit eden
  - Ve bütün bunları startup aşamasında **AMQP broker'da tanımlayan** bir altyapı sunmaktasunardır

# ConnectionFactory

- Broker bağlantılarını (**Connection**) yöneten **merkezi arayüzdür**
- RabbitMQ için **CachingConnectionFactory** implemantasyonu vardır
- CachingConnectionFactory üzerinden bir tane **proxy Connection nesnesi** yaratılır ve **uygulama genelinde paylaşılır**

# Connection ve Channel

- AMQP'de mesajlaşma işleminde “**unit-of-work**” channel'dır
- Connection'dan elde edilir
- CachingConnectionFactory **channel instance'larını cache'leyebilmektedir**
- Channel instance'larının **transactional olup olmamasına göre ayrı ayrı cache'ler vardır**

# ConnectionFactory, Connection ve Channel



# Spring Rabbit Namespace Desteği

- Spring; broker, connectionFactory, queue, exchange, binding, message listener container gibi AMQP **nesnelerinin yaratılması ve konfigürasyonu için bir namespace** sunmaktadır
- Bu namespace sayesinde AMQP konfigürasyonu **daha basit ve kolay** olmaktadır

# ConnectionFactory Konfigürasyonu

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:rabbit="http://www.springframework.org/schema/rabbit"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/rabbit
http://www.springframework.org/schema/rabbit/spring-rabbit.xsd">
```

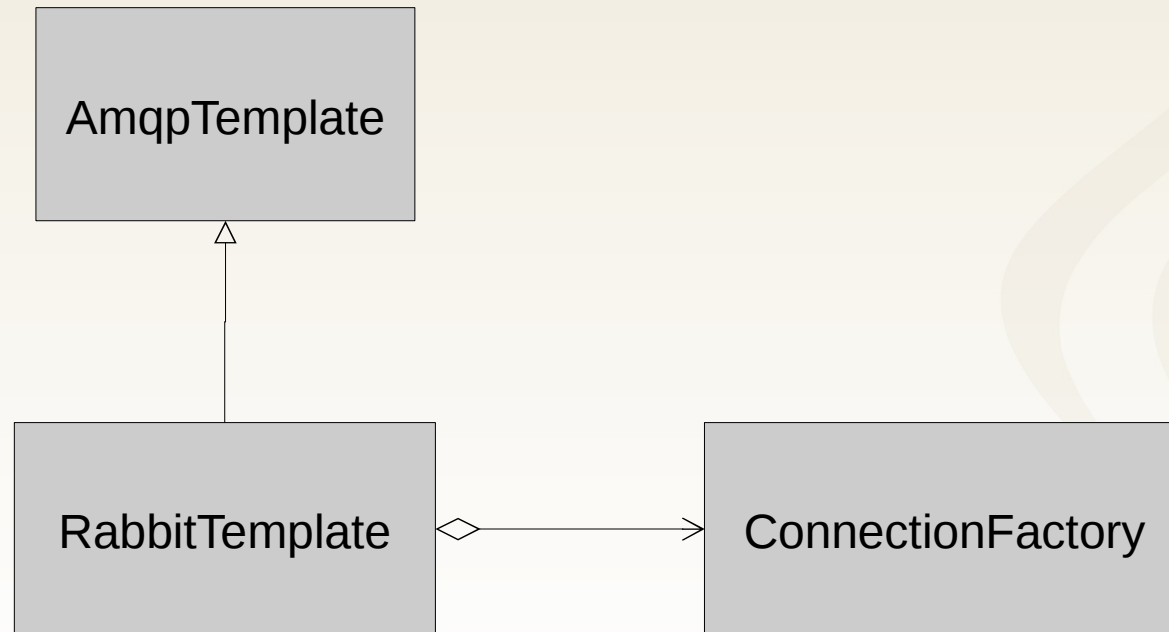
```
<rabbit:connection-factory id="rabbitConnectionFactory"
    port="5672"
    host="localhost"
    virtual-host="/"
    username="guest"
    password="guest"
    channel-cache-size="25" />
```

```
</beans>
```

# Template Tabanlı Yaklaşım

- Spring Application Framework'ün diğer pek çok modülündeki **template tabanlı yaklaşımın** aynısıdır
- AMQP'de mesaj gönderme ve alma işlemleri için merkezi yapı **AmqpTemplate** arayüzüdür
- **RabbitTemplate** bunun bir implemantasyonudur

# Template Tabanlı Yaklaşım





# RabbitTemplate Konfigürasyonu

```
<rabbit:template id="rabbitTemplate"  
    connection-factory="rabbitConnectionFactory"  
    exchange=""  
    queue=""  
    routing-key=""  
    channel-transacted="false"  
    mandatory="false"  
    encoding="utf-8"  
    receive-timeout="0" />
```

# Mesaj Gönderme

- Mesaj göndermek için **AmqpTemplate** aşağıdaki metotları sağlamaktadır

```
void send(String exchange, String routingKey, Message message)  
        throws AmqpException;
```

```
void send(String routingKey, Message message)  
        throws AmqpException;
```

```
void send(Message message) throws AmqpException;
```

Exchange ve routingKey parametreleri sürekli tekrarlıyorsa RabbitTemplate düzeyinde set edilerek doğrudan mesaj gönderimi de gerçekleştirilebilir

# Mesaj Alma

- Mesaj almak için **AmqpTemplate** aşağıdaki metotları sunmaktadır

```
Message receive(String queueName, long timeoutMillis)
                                throws AmqpException;
```

```
Message receive(long timeoutMillis) throws AmqpException;
```

```
Message receive(String queueName) throws AmqpException;
```

```
Message receive() throws AmqpException;
```

receiveTimeout, queue gibi değerler mesaj alımında tekrarlıyor ise RabbitTemplate düzeyinde set edilerek doğrudan mesaj alımı da gerçekleştirilebilir

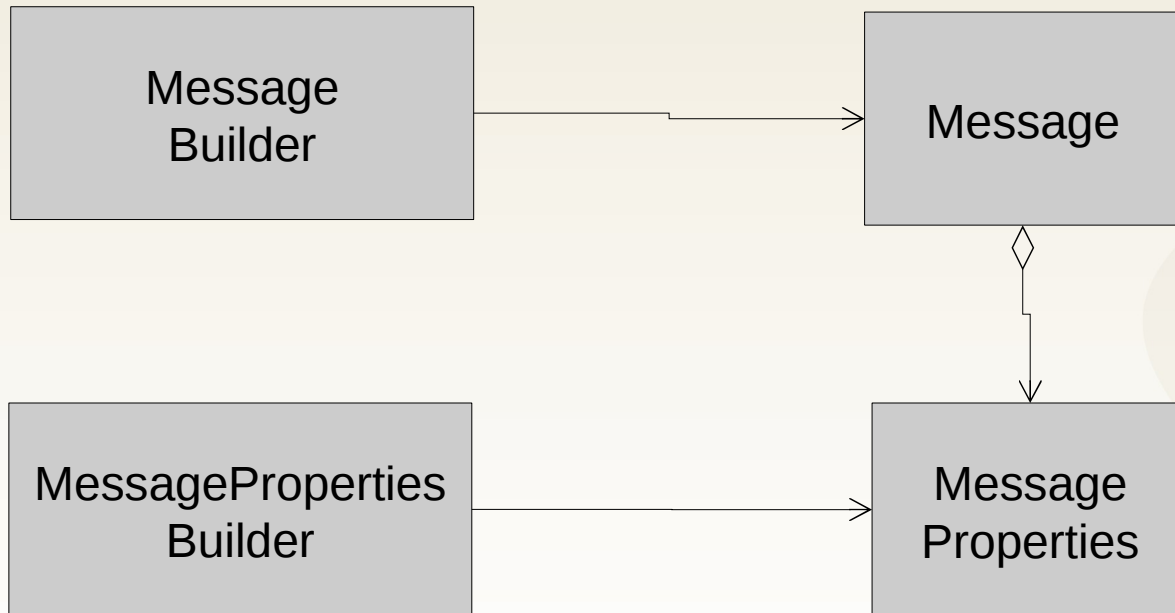
# MessageBuilder API

- **MessageBuilder** ve **MessagePropertiesBuilder** sınıfları vasıtası ile **Message** nesneleri “fluent bir API” ile **yaratılabilir**

```
MessageProperties props = MessagePropertiesBuilder
    .newInstance()
    .setContentType(MessageProperties.CONTENT_TYPE_TEXT_PLAIN)
    .setMessageId("123")
    .setHeader("confidentiality", "top-secret")
    .build();
```

```
Message message = MessageBuilder
    .withBody("top secret message content".getBytes())
    .andProperties(props)
    .build();
```

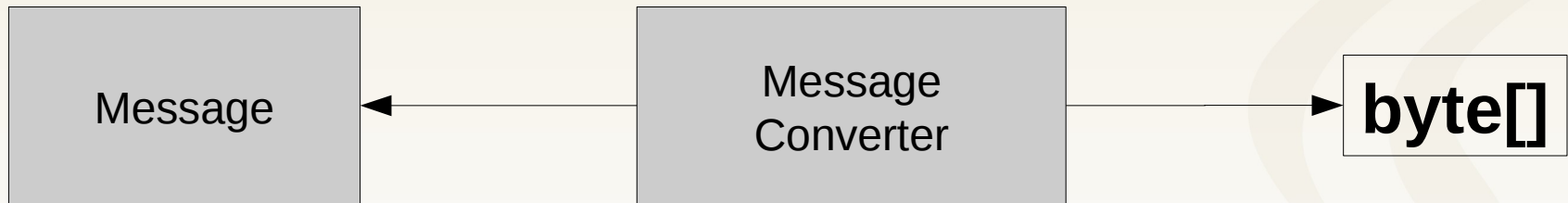
# MessageBuilder API



# MessageConverter


- AMQP mesaj içeriğini `byte[]` olarak ele almaktadır
- Mesaj gönderirken ve alırken nesne-mesaj dönüşümü için **MessageConverter**'lar kullanılmaktadır
- **RabbitTemplate** default olarak `String`, `byte[]` ve `Serializable` tipteki Java nesnelerini **SimpleMessageConverter** kullanarak dönüştürmektedir

# MessageConverter



# Broker Konfigürasyonu

```
<rabbit:admin  
  connection-factory="rabbitConnectionFactory"  
  auto-startup="true" />
```



ApplicationContext içerisinde tanımlı queue, exchange, binding nesnelerinin broker'da otomatik tanımlanmalarını sağlar



# Queue Konfigürasyonu

```
<rabbit:queue name="myqueue" />

<rabbit:queue name="myqueue2" />

<rabbit:queue name="myqueue3">
  <rabbit:queue-arguments>
    <entry key="key1" value="1"/>
    <entry key="key2" value="2"/>
  </rabbit:queue-arguments>
</rabbit:queue>
```

Queue nesnesi broker üzerinde tanımlanırken kullanılacak argümanları vermek için kullanılır

# Exchange ve Binding Konfigürasyonu

```
<rabbit:topic-exchange name="mytopic">
  <rabbit:bindings>
    <rabbit:binding pattern="speedy.*" queue="myqueue" />
  </rabbit:bindings>
</rabbit:topic-exchange>
```


Belirtilen pattern ile eşleşen routingKey ile gönderilen mesajlar bu queue'ya iletilir

```
<rabbit:direct-exchange name="mydirect">
  <rabbit:bindings>
    <rabbit:binding key="mykey" queue="myqueue2" />
  </rabbit:bindings>
</rabbit:direct-exchange>
```

Key ile birebir eşleşen routingKey ile gönderilen mesajlar bu queue'ya iletilir  
Belirtilmez ise key=queue name'dir

# Exchange ve Binding Konfigürasyonu

```
<rabbit:fanout-exchange name="myfanout">
  <rabbit:bindings>
    <rabbit:binding queue="myqueue1"/>
    <rabbit:binding queue="myqueue2"/>
    <rabbit:binding queue="myqueue3"/>
  </rabbit:bindings>
</rabbit:fanout-exchange>
```



Fanout exchange'e gönderilen mesajlar  
bind edilen bütün queue'lara iletilir

# MessageListener

- Asenkron mesaj alımlarında **MessageListener** arayüzü kullanılır

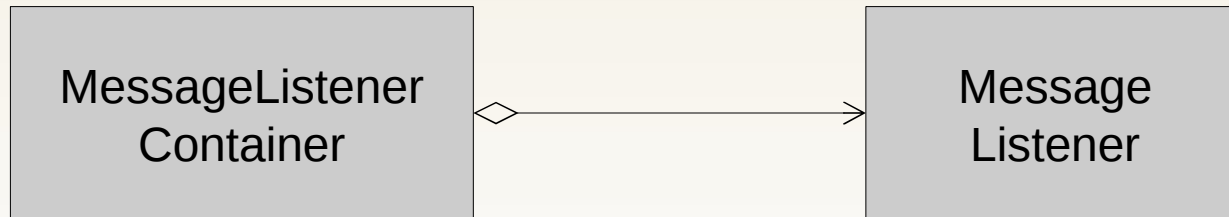
```
public interface MessageListener {  
    void onMessage(Message message);  
}
```

- Mesaj alımı yapacak sınıfın bu **arayüzü implement etmesi** gerekir

# MessageListener ve Listener Container

- Mesajları asenkron almak için **ApplicationContext** içerisinde konfigüre edilen **AmqpTemplate** dışında ayrı bir yapı kullanılır
- Bu yapı **MessageListenerContainer**'dır
- **SimpleMessageListenerContainer** bu arayüzün bir implementasyonudur

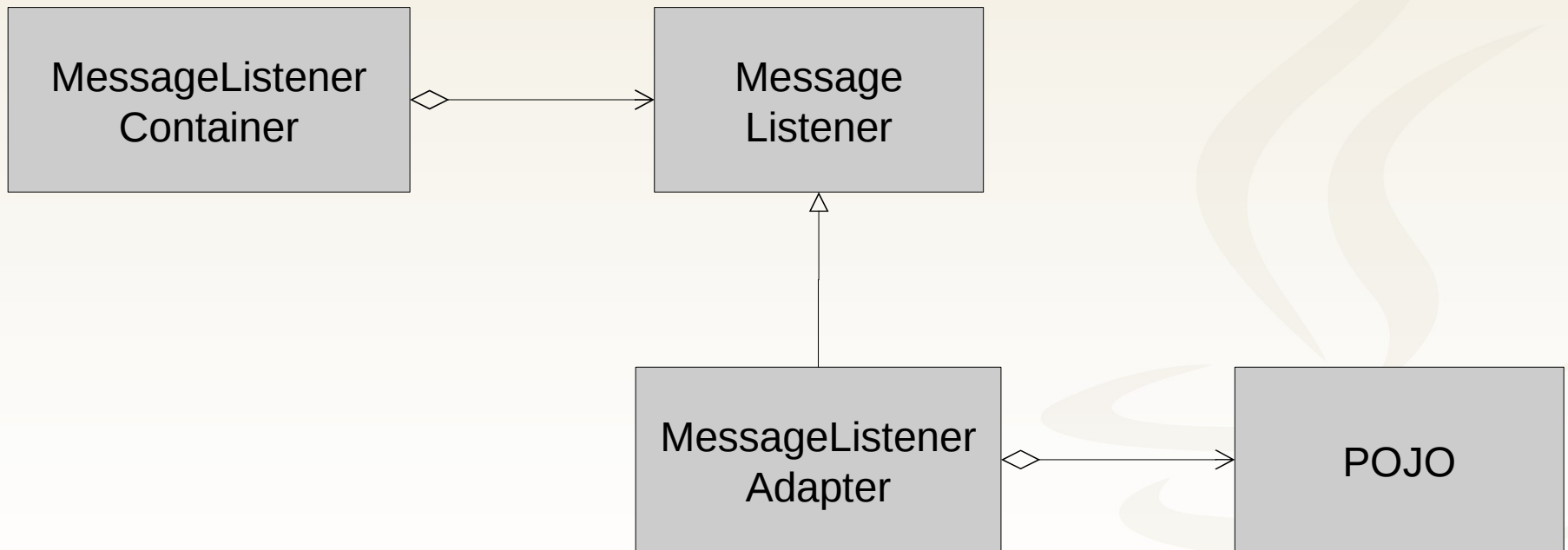
# MessageListener ve Listener Container



# MessageListenerAdapter ve Message Driven POJO

- **MessageListenerAdapter** sınıfı vasıtası ile MessageListener arayüzünü implement etmeden uygulamadaki **sıradan java nesneleri** de asenkron **mesaj alımlarında** kullanılabilir

# MessageListenerAdapter ve Message Driven POJO



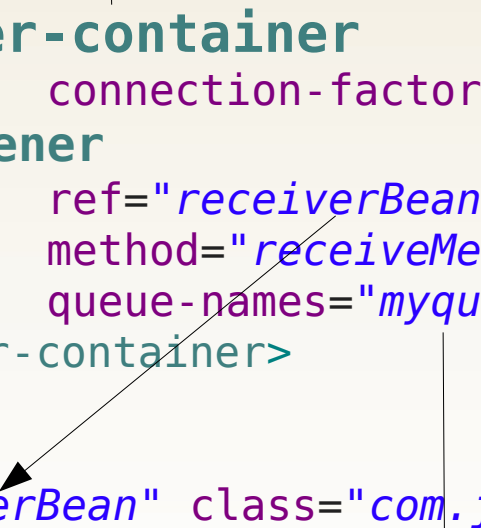


# Message Listener ve Container Konfigürasyonu

Arka tarafta SimpleMessageListenerContainer konfigüre eder

```
<rabbit:listener-container
    connection-factory="rabbitConnectionFactory">
  <rabbit:listener
    ref="receiverBean"
    method="receiveMessage"
    queue-names="myqueue" />
</rabbit:listener-container>

<bean id="receiverBean" class="com.javaegitimleri.amqp.Receiver" />
```



Listener'ın hangi queue(lar)'daki mesajları ele alacağı belirtilmelidir

# Transaction Yönetimi

- Mesaj gönderme ve alma işlemleri **aktif bir transaction** içerisinde gerçekleştirilebilir
- Bu işlemlerin sonucuna göre **TX commit veya rollback** edilebilir
- RabbitTemplate ve SimpleMessageListenerContainer sınıflarında **channelTransacted=true** olarak belirtilirse transactional bir channel kullanılır ve işlemler sonucunda commit/rollback gerçekleştirilir

# Transaction Yönetimi

- Transaction yönetiminin gerçekleşebilmesi için ApplicationContext'de **PlatformTransactionManager** tipinde bean'in kontrolünde **external bir TX'in** olması gerekir
- Alternatifi **RabbitTransactionManager**'dir

# RabbitTransactionManager Konfigürasyonu

```
<bean id="transactionManager"  
class="org.springframework.amqp.rabbit.transaction  
    .RabbitTransactionManager">  
    <property name="connectionFactory"  
        ref="rabbitConnectionFactory" />  
</bean>
```

PlatformTransactionManager arayüzünü  
implement eder  
XA desteği yoktur

# İletişim

- **Harezmi** Bilişim Çözümleri
- Kurumsal Java Eğitimleri
- <http://www.java-egitimleri.com>
- [info@java-egitimleri.com](mailto:info@java-egitimleri.com)

