Spring Boot Messaging with RabbitMQ

Description: Spring Boot is an open source, micro service-based Java web framework. The Spring Boot framework creates a fully production-ready environment that is completely configurable using its prebuilt code within its own codebase.

Project: Build a Spring Boot application setting up a RabbitMQ AMQP server that publishes and subscribes to messages. The application publishes a message by using Spring AMQP's RabbitTemplate and subscribes to the message on a POJO by using MessageListenerAdapter. This project is based on Messaging with RabbitMQ and just expands on the screenshots and step by step instructions.

Technology: This project uses the following technology:

Integrated Development Environment (IDE):

Spring Tool Suite 4 (Version: 4.15.0.RELEASE)

Java Development Kit (JDK):

Oracle's JDK 8 (1.8)

Other technology:

RabbitMQ (3.10.n) - an open-source message broker.

Requirement:

RabbitMQ installed on the local machine. See document title "Environment Setup 06 RabbitMQ" for instructions.

Table of Contents

Glossary of Terminology

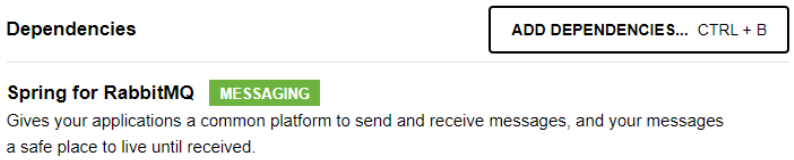
For a list of key terms and definitions used throughout this and various Spring Boot demo documents see the document titled “Appendix 01 Glossary”.

Generate Spring Boot Download

Follow the instructions in the document title “Appendix 02 Spring Initializr” to generate a spring boot download for this project.

When the document talks about adding dependencies add only the following:

Spring for RabbitMQ



When the document talks about the items in the “**Project Metadata**” use the values shown below:

“**Group**” use “**com.example**”
“**Artifact**” use “**messaging-rabbitmq**”
“**Name**” use “**messaging-rabbitmq**”
“**Package name**” use “**com.example.messaging-rabbitmq**”

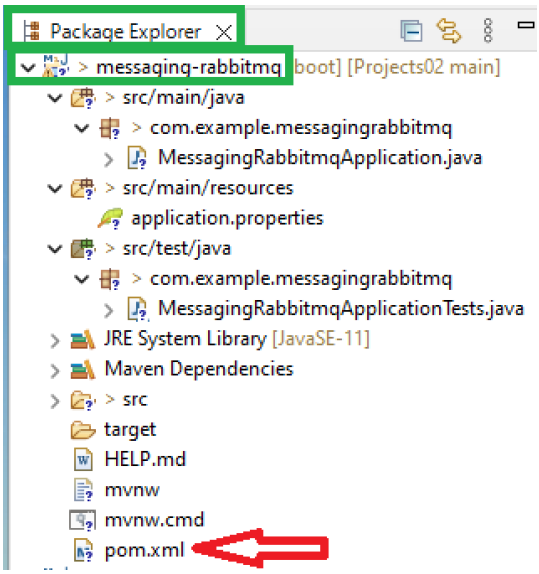
For other items in the “**Project Metadata**” use the defaults. Follow the instructions to extract the files from the zip file into the Sprint Tool Suite 4 workspace.

Import the Spring Boot Download

Open the Spring Tool Suite 4 IDE.

Import the project: “messaging-rabbitmq”

Follow the instructions in the document title “Appendix 03 Import Spring Tool Suite Project” and import the “**messaging-rabbitmq**” project that was created with Spring Initializr. After importing the project, it should look like the following using the IDE “Package Explorer”.



Look at the pom.xml and find the two dependencies for this project.

Project messaging-rabbitmq Discussion

"With any messaging-based application, you need to create a receiver that responds to published messages. The following listing (from `src/main/java/com.example.messagingrabbitmq/Receiver.java`)" -- [Messaging with RabbitMQ]

Main Class: MessagingRabbitmqApplication

The MessagingRabbitmqApplication class created when using the Spring Initializr is used with modification as the last section before testing. This class contains the "`public static void main(String[] args) {}`" method that starts the application listening when the internal web server is started. This class is modified in another section.

Create RabbitMQ Message Listener Class

A receiver class is a listener as with any messaging-based application. As a receiver it will listen for messages on the queue, read and process the message.

Create Listener Class: Receiver

Right click on package "com.example.messagingrabbitmq"
Right highlight "New" ➔ select "Class"
Enter class "Name:" "Receiver"
Click "Finish"

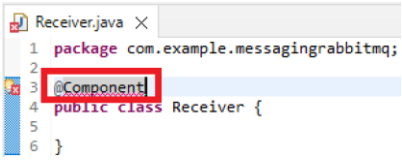
The new class opens automatically in an edit window.

Update Listener class: Receiver

If you are experienced with RabbitMQ concepts and correcting errors using the IDE you can skip to the section "[Copy and paste source code for class: Receiver]".

Add component annotation. @Component is an annotation that allows Spring to automatically detect our custom beans.

```
@Component
```



Resolve the error.

```
@Component
Component cannot be resolved to a type
5 quick fixes available:
Import 'Component' (org.springframework.stereotype)
```

Add private class variable, constructor, and method.

```
private CountDownLatch latch = new CountDownLatch(1);

public void receiveMessage(String message) {
  System.out.println("Received <" + message + ">");
  latch.countDown();
}

public CountDownLatch getLatch() {
  return latch;
}
```

```
public class Receiver {

    private CountDownLatch latch = new CountDownLatch(1);

    public void receiveMessage(String message) {
      System.out.println("Received <" + message + ">");
      latch.countDown();
    }

    public CountDownLatch getLatch() {
      return latch;
    }
```

Resolve the errors.

```
private CountDownLatch latch = new CountDownLatch(1);

public v
  System
  latch.
}
CountDownLatch cannot be resolved to a type
6 quick fixes available:
Import 'CountDownLatch' (java.util.concurrent)
```

The completed class.

```
Receiver.java
package com.example.messagingrabbitmq;

import java.util.concurrent.CountDownLatch;

import org.springframework.stereotype.Component;

@Component
public class Receiver {

    private CountDownLatch latch = new CountDownLatch(1);

    public void receiveMessage(String message) {
      System.out.println("Received <" + message + ">");
      latch.countDown();
    }

    public CountDownLatch getLatch() {
      return latch;
    }

}
```

"The `Receiver` is a POJO that defines a method for receiving messages. When you register it to receive messages, you can name it anything you want.

For convenience, this POJO also has a `CountDownLatch`. This lets it signal that the message has been received. This is something you are not likely to implement in a production application." -- Messaging with RabbitMQ

Copy and paste source code for class: Receiver

As an alternative you can copy the code below and replace all code in the class if you used the same package name.

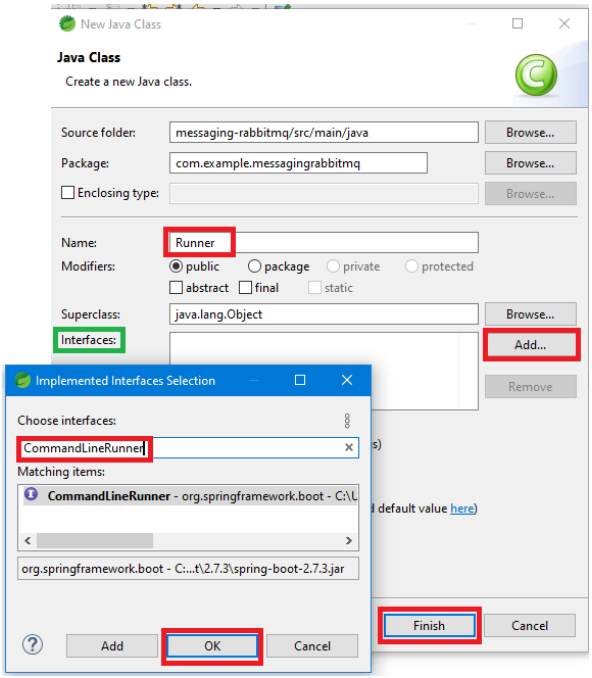
```
package com.example.messagingrabbitmq;

import java.util.concurrent.CountDownLatch;
import org.springframework.stereotype.Component;

@Component
public class Receiver {

  private CountDownLatch latch = new CountDownLatch(1);

  public void receiveMessage(String message) {
    System.out.println("Received <" + message + ">");
    latch.countDown();
  }

  public CountDownLatch getLatch() {
    return latch;
  }

}
```

Create Send Message Class: Runner

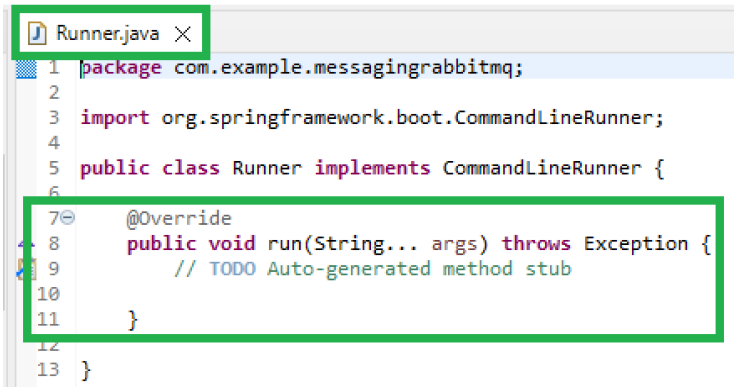
> "In this sample, test messages are sent by a `CommandLineRunner`, which also waits for the latch in the receiver and closes the application context." -- Messaging with RabbitMQ

Right click on package "com.example.messagingrabbitmq"
Right highlight "New" ➔ select "Class"

Enter class "Name:" "Runner"
Click "Extended interfaces:" "Add…" button
Enter "Chooses interface": "CommandLineRunner"
Click "OK"
Click "Finish"



The interface opens automatically in an edit window with an override interface method defined automatically.

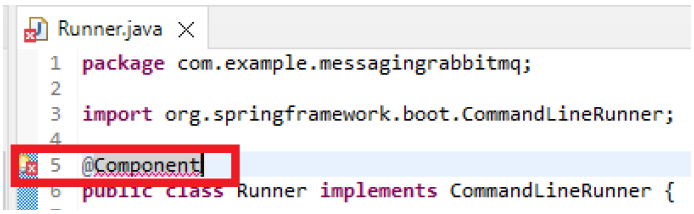


Update Send Message Class: Runner

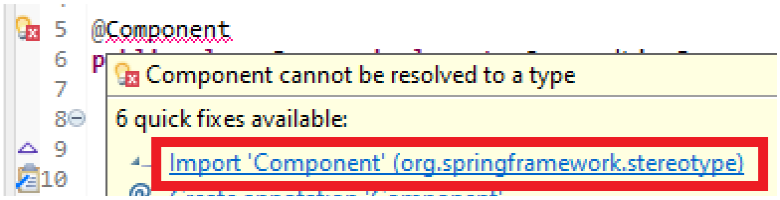
If you are experienced with RabbitMQ concepts and correcting errors using the IDE you can skip to the section "Copy and paste source code for class: Runner".

Add component annotation. @Component is an annotation that allows Spring to automatically detect our custom beans.

```
@Component
```

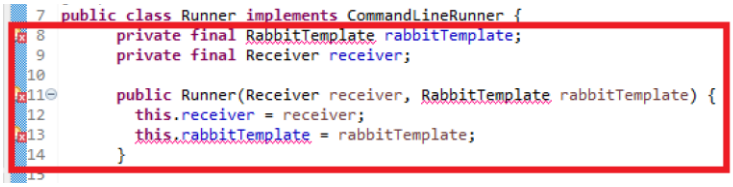


Resolve the error.

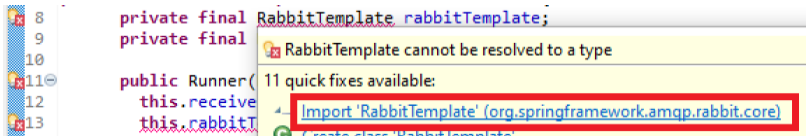


Add class variables and a constructor with arguments.

```
private final RabbitTemplate rabbitTemplate;
private final Receiver receiver;

public Runner(Receiver receiver, RabbitTemplate rabbitTemplate) {
  this.receiver = receiver;
  this.rabbitTemplate = rabbitTemplate;
}
```



Resolve the errors.



Complete the override method.

```
System.out.println("Sending message...");
rabbitTemplate.convertAndSend(MessagingRabbitmqApplication.topicExchangeName, "foo.bar.baz",
                "Hello from RabbitMQ!");
receiver.getLatch().await(10000, TimeUnit.MILLISECONDS);
```

```
17    @Override
18    public void run(String... args) throws Exception {
19        System.out.println("Sending message...");
20        rabbitTemplate.convertAndSend(MessagingRabbitmqApplication.topicExchangeName, "foo.bar.baz",
21                "Hello from RabbitMQ!");
22        receiver.getLatch().await(10000, TimeUnit.MILLISECONDS);
23    }
```

Resolve the errors.
The MessagingRabbitmqApplication.topicExchangeName error will be resolved when the main application class is update in the next section. For now resolve the TimeUnit import error.

```
22        receiver.getLatch().await(10000, TimeUnit.MILLISECONDS);
23    }
24
25 }
26
```

TimeUnit cannot be resolved to a variable
9 quick fixes available:
Import 'TimeUnit' (java.util.concurrent)

The completed class.

Runner.java

```
1  package com.example.messagingrabbitmq;
2
3  import java.util.concurrent.TimeUnit;
4
5  import org.springframework.amqp.rabbit.core.RabbitTemplate;
6  import org.springframework.boot.CommandLineRunner;
7  import org.springframework.stereotype.Component;
8
9  @Component
10 public class Runner implements CommandLineRunner {
11     private final RabbitTemplate rabbitTemplate;
12     private final Receiver receiver;
13
14     public Runner(Receiver receiver, RabbitTemplate rabbitTemplate) {
15         this.receiver = receiver;
16         this.rabbitTemplate = rabbitTemplate;
17     }
18
19     @Override
20     public void run(String... args) throws Exception {
21         System.out.println("Sending message...");
22         rabbitTemplate.convertAndSend(MessagingRabbitmqApplication.topicExchangeName, "foo.bar.baz",
23                 "Hello from RabbitMQ!");
24         receiver.getLatch().await(10000, TimeUnit.MILLISECONDS);
25     }
26
27 }
```

Copy and paste source code for class: Runner

As an alternative you can copy the code below and replace all code in the class if you used the same package name.

```
package com.example.messagingrabbitmq;

import java.util.concurrent.TimeUnit;

import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

@Component
public class Runner implements CommandLineRunner {

  private final RabbitTemplate rabbitTemplate;
  private final Receiver receiver;

  public Runner(Receiver receiver, RabbitTemplate rabbitTemplate) {
    this.receiver = receiver;
    this.rabbitTemplate = rabbitTemplate;
  }
```

```
  @Override
  public void run(String... args) throws Exception {
    System.out.println("Sending message...");
    rabbitTemplate.convertAndSend(MessagingRabbitmqApplication.topicExchangeName, "foo.bar.baz", "Hello from RabbitMQ!");
    receiver.getLatch().await(10000, TimeUnit.MILLISECONDS);
  }

}
```

Update Main Class: MessagingRabbitmqApplication

Like mentioned earlier the MessagingRabbitmqApplication class created when using the Spring Initializr is used with modification. This section walks through the modification of this class.

"Spring AMQP's `RabbitTemplate` provides everything you need to send and receive messages with RabbitMQ. However, you need to:

- Configure a message listener container.
- Declare the queue, the exchange, and the binding between them.
- Configure a component to send some messages to test the listener.

Spring Boot automatically creates a connection factory and a RabbitTemplate, reducing the amount of code you have to write.

You will use `RabbitTemplate` to send messages, and you will register a `Receiver` with the message listener container to receive messages. The connection factory drives both, letting them connect to the RabbitMQ server." -- Messaging with RabbitMQ

If you are experienced with the concepts above and correcting errors using the IDE you can skip to the section "Copy and paste source code for class: MessagingRabbitmqApplication".

The class Wizard generated the following basic main class.

MessagingRabbitmqApplication.java

```
1  package com.example.messagingrabbitmq;
2
3  import org.springframework.boot.SpringApplication;
5
6  @SpringBootApplication
7  public class MessagingRabbitmqApplication {
8
9      public static void main(String[] args) {
10         SpringApplication.run(MessagingRabbitmqApplication.class, args);
11     }
12
13 }
```

Add class variable and a Bean.
Add a method with @Bean annotation.
The annotation is applied on a method to specify that it returns a bean. The bean is managed by Spring context. The method is responsible for creating the instance.

```
        static final String topicExchangeName = "spring-boot-exchange";

        static final String queueName = "spring-boot";

        @Bean
        Queue queue() {
                return new Queue(queueName, false);
        }
```

```
6  @SpringBootApplication
7  public class MessagingRabbitmqApplication {
8
9      static final String topicExchangeName = "spring-boot-exchange";
10
11     static final String queueName = "spring-boot";
12
13     @Bean
14     Queue queue() {
15         return new Queue(queueName, false);
16     }
```

Resolve the errors:

```
13     @Bean
14     Q
15
16     }
17
18     p
```

Bean cannot be resolved to a type

5 quick fixes available:

- Import 'Bean' (org.springframework.context.annotation)

```
15     Queue queue() {
16
17
18
19
20
21
22
23 }
24 {
```

Queue cannot be resolved to a type

16 quick fixes available:

- Import 'Queue' (com.rabbitmq.client.AMQP)
- Import 'Queue' (com.rabbitmq.client.impl.AMQImpl)
- Import 'Queue' (java.util)
- Import 'Queue' (org.springframework.amqp.core)

Add four more bean methods with @Bean annotation.

```
        @Bean
        TopicExchange exchange() {
                return new TopicExchange(topicExchangeName);
        }

        @Bean
        Binding binding(Queue queue, TopicExchange exchange) {
                return BindingBuilder.bind(queue).to(exchange).with("foo.bar.#");
        }

        @Bean
        SimpleMessageListenerContainer container(ConnectionFactory connectionFactory,
                        MessageListenerAdapter listenerAdapter) {
                SimpleMessageListenerContainer container = new SimpleMessageListenerContainer();
                container.setConnectionFactory(connectionFactory);
                container.setQueueNames(queueName);
                container.setMessageListener(listenerAdapter);
```

```
            return container;
        }

        @Bean
        MessageListenerAdapter listenerAdapter(Receiver receiver) {
            return new MessageListenerAdapter(receiver, "receiveMessage");
        }
```

```
20  @Bean
21  TopicExchange exchange() {
22      return new TopicExchange(topicExchangeName);
23  }
24
25  @Bean
26  Binding binding(Queue queue, TopicExchange exchange) {
27      return BindingBuilder.bind(queue).to(exchange).with("foo.bar.#");
28  }
29
30  @Bean
31  SimpleMessageListenerContainer container(ConnectionFactory connectionFactory,
32          MessageListenerAdapter listenerAdapter) {
33      SimpleMessageListenerContainer container = new SimpleMessageListenerContainer();
34      container.setConnectionFactory(connectionFactory);
35      container.setQueueNames(queueName);
36      container.setMessageListener(listenerAdapter);
37      return container;
38  }
39
40  @Bean
41  MessageListenerAdapter listenerAdapter(Receiver receiver) {
42      return new MessageListenerAdapter(receiver, "receiveMessage");
43  }
```

Resolve the errors:

```
21  TopicExchange exchange() {
22
23
24
25
26
```

TopicExchange cannot be resolved to a type

9 quick fixes available:

Import 'TopicExchange' (org.springframework.amqp.core)

```
27  Binding binding(Queue queue, TopicExchange ex
28
29
30
31
32
33
```

Binding cannot be resolved to a type

24 quick fixes available:

Import 'Binding' (javax.naming)

Import 'Binding' (org.springframework.amqp.core)

```
29      return BindingBuilder.bind(queue).to(exchange).with("foo.b
30  }
31
32  @Bean
33  SimpleMessa
34          Mes
```

BindingBuilder cannot be resolved

109 quick fixes available:

Import 'BindingBuilder' (org.springframework.amqp.core)

```
34  SimpleMessageListenerContainer container(ConnectionFactory connectionFact
35
36
37
38
39
```

SimpleMessageListenerContainer cannot be resolved to a type

14 quick fixes available:

Import 'SimpleMessageListenerContainer' (org.springframework.amqp.rabbit.listener)

```
35    SimpleMessageListenerContainer container(ConnectionFactory connectionFactory,
36            MessageListenerAdapter listenerAd
37        SimpleMessageListenerContainer contai
38        container.setConnectionFactory(connec
39        container.setQueueNames(queueName);
40        container.setMessageListener(listener
41        return container;
```

ConnectionFactory cannot be resolved to a type

41 quick fixes available:

Import 'ConnectionFactory' (com.rabbitmq.client)

Import 'ConnectionFactory' (org.springframework.amqp.rabbit.connection)

```
37            MessageListenerAdapter listenerAdapter) {
38        Simp
39        cont
40        cont
41        cont
42        retu
```

MessageListenerAdapter cannot be resolved to a type

11 quick fixes available:

Import 'MessageListenerAdapter' (org.springframework.amqp.rabbit.listener.adapter)

The completed class code.

MessagingRabbitmqApplication.java

```java
import org.springframework.amqp.core.Binding;
import org.springframework.amqp.core.BindingBuilder;
import org.springframework.amqp.core.Queue;
import org.springframework.amqp.core.TopicExchange;
import org.springframework.amqp.rabbit.connection.ConnectionFactory;
import org.springframework.amqp.rabbit.listener.SimpleMessageListenerContainer;
import org.springframework.amqp.rabbit.listener.adapter.MessageListenerAdapter;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

@SpringBootApplication
public class MessagingRabbitmqApplication {

    static final String topicExchangeName = "spring-boot-exchange";

    static final String queueName = "spring-boot";

    @Bean
    Queue queue() {
        return new Queue(queueName, false);
    }

    @Bean
    TopicExchange exchange() {
        return new TopicExchange(topicExchangeName);
    }

    @Bean
    Binding binding(Queue queue, TopicExchange exchange) {
        return BindingBuilder.bind(queue).to(exchange).with("foo.bar.#");
    }

    @Bean
    SimpleMessageListenerContainer container(ConnectionFactory connectionFactory,
            MessageListenerAdapter listenerAdapter) {
        SimpleMessageListenerContainer container = new SimpleMessageListenerContainer();
        container.setConnectionFactory(connectionFactory);
        container.setQueueNames(queueName);
        container.setMessageListener(listenerAdapter);
        return container;
    }

    @Bean
    MessageListenerAdapter listenerAdapter(Receiver receiver) {
        return new MessageListenerAdapter(receiver, "receiveMessage");
    }

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

}
```

Copy and paste source code for class: MessagingRabbitmqApplication

As an alternative you can copy the code below and replace all code in the class if you used the same package name.

```java
package com.example.messagingrabbitmq;
```

```java
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

@SpringBootApplication
public class MessagingRabbitmqApplication {

  private static final Logger log = LoggerFactory.getLogger(MessagingrabbitmqApplication.class);

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

  @Bean
  public CommandLineRunner demo(CustomerRepository repository) {
    return (args) -> {
      // save a few customers
      repository.save(new Customer("Jack", "Bauer"));
      repository.save(new Customer("Chloe", "O'Brian"));
      repository.save(new Customer("Kim", "Bauer"));
      repository.save(new Customer("David", "Palmer"));
      repository.save(new Customer("Michelle", "Dessler"));

      // fetch all customers
      log.info("Customers found with findAll():");
      log.info("-------------------------------");
      for (Customer customer : repository.findAll()) {
        log.info(customer.toString());
      }
      log.info("");

      // fetch an individual customer by ID
      Customer customer = repository.findById(1L);
      log.info("Customer found with findById(1L):");
      log.info("--------------------------------");
      log.info(customer.toString());
      log.info("");

      // fetch customers by last name
      log.info("Customer found with findByLastName('Bauer'):");
      log.info("--------------------------------------------");
      repository.findByLastName("Bauer").forEach(bauer -> {
        log.info(bauer.toString());
      });
      // for (Customer bauer : repository.findByLastName("Bauer")) {
      //  log.info(bauer.toString());
      // }
      log.info("");
    };
  }

}
```

Run the Project

Testing the project can be with an executable Java Archive (JAR) file as the way to run and test the program. Alternatively testing can be performed using the IDE running the main class "MessagingRabbitmqApplication" as "Sprint Boot App" inside the IDE.

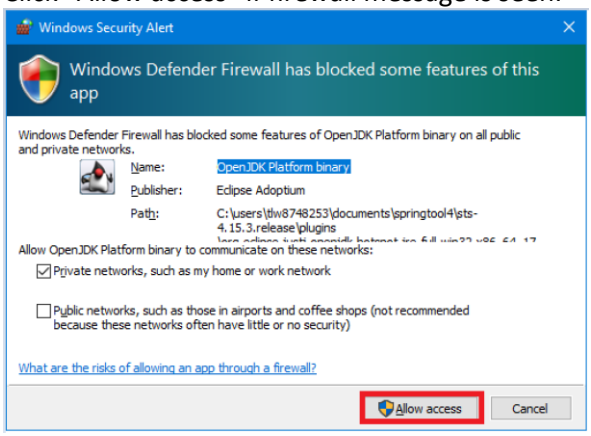
In the "Appendix 04 Run Spring Initializr Project" document follow the instructions in section title "Executable JAR File Lifecycle". Follow the instructions in sub sections title "Build the Executable JAR" and "Run the Executable JAR File" and stop there. Makes sure to build in this project's folder.

This is a simple application with minimal testing and output. The following runs the application in the IDE and show the results.

Start the IDE Tomcat Server

Right click inside the main class "MessagingRabbitmqApplication" select "Run As" ➔ "Sprint Boot App".

Click “Allow access” if firewall message is seen.



The Tomcat server bundled with the Spring Boot application starts.
The application displays two messages.

```
Sending message...
Received <Hello from RabbitMQ!>
```

The Runner class display the “Sending message…” and initiates the RabbitMQ message.

```
19    @Override
20    public void run(String... args) throws Exception {
21        System.out.println("Sending message...");
22        rabbitTemplate.convertAndSend(MessagingRabbitmqApplication.topicExchangeName, "foo.bar.baz",
23                "Hello from RabbitMQ!");
24        receiver.getLatch().await(10000, TimeUnit.MILLISECONDS);
25    }
```

The Reciever class displays the message received off the queue.

```
12        public void receiveMessage(String message) {
13            System.out.println("Received <" + message + ">");
14            latch.countDown();
15        }
```

The main class MessagingRabbitmqApplication maps the Receiver class as the listener.

```
50    @Bean
51    MessageListenerAdapter listenerAdapter(Receiver receiver) {
52        return new MessageListenerAdapter(receiver, "receiveMessage");
53    }
```

Adding additional print statement to the code, shows the flow of execution.
MessagingRabbitmqApplication main() starts execution and runs through setup of RabbitMQ elements.
A message is sent from class Runner and is displayed by class Reciever.

```
 :: Spring Boot ::                (v2.7.3)

2022-08-29 10:11:51.281  INFO 21572 --- [
2022-08-29 10:11:51.284  INFO 21572 --- [
queue() entered
exchange() entered
binding() entered
listenerAdapter() recieved message
container() entered
2022-08-29 10:11:52.498  INFO 21572 --- [
2022-08-29 10:11:52.558  INFO 21572 --- [
2022-08-29 10:11:52.563  INFO 21572 --- [
2022-08-29 10:11:52.622  INFO 21572 --- [
Sending message...
Received <Hello from RabbitMQ!>
```