SOA – laboratorium nr 7

Temat*: JMS oraz Message-Driven Beans.*

Część teoretyczna.

Przesyłanie komunikatów to metoda komunikacji między elementami oprogramowania i aplikacjami. **JMS (Java Message Service)** udokumentowane na stronie https://javaee.github.io/tutorial/jms-concepts.html#BNCDQ to API Javy umożliwiające aplikacjom tworzenie, wysyłanie, odbieranie i odczytywanie komunikatów.

Przesyłanie komunikatów różni się od innych standardowych protokołów, na przykład **RMI (Remote Method Invocation)** lub **HTTP (Hypertext Transfer Protocol)** w dwojaki sposób. Po pierwsze, w komunikacji pośredniczy serwer komunikatów, więc nie jest to bezpośrednia komunikacja dwukierunkowa. Po drugie, zarówno odbiorca, jak i nadawca muszą znać format komunikatu i jego cel, ale nic więcej. Różni się to od technologii ściślej wiążących odbiorcę i nadawcę, takich jak RMI, w których to aplikacja musi znać zdalnie wywoływane metody

Krótkie wprowadzenie do JMS

JMS definiuje niezależny od dostawcy (choć specyficzny dla Javy) zbiór interfejsów programistycznych do interakcji z systemami asynchronicznego przesyłania komunikatów. Takie rozwiązanie umożliwia rozproszoną komunikację luźno powiązanych elementów. Cały przesył komunikatów to proces dwuetapowy: jeden komponent wysyła komunikat do celu, a drugi komponent odbiera go z serwera JMS.

JMS służy do asynchronicznej komunikacji (można synchronicznie odbierać komunikaty lecz jest to niezalecane) pomiędzy systemami za pomocą szeroko pojętych komunikatów (plików, preparowanych wiadomości itp). Pozwala tworzyć, wysyłać, otrzymywać i czytać te komunikaty.

W JMS istnieją dwa rodzaje celów — tematy i kolejki.

W modelu **punkt-punkt** komunikaty przesyłane są od producentów do konsumentów przy użyciu **kolejki**. Kolejka może mieć wielu odbiorców, ale konkretny komunikat otrzyma tylko jeden z nich. Pierwszy odbiorca otrzyma komunikat, a pozostali nawet się nie dowiedzą, że istniał.

Z drugiej strony, komunikat wysłany do **tematu** może być odczytany przez wielu odbiorców. Komunikat wysłany pod konkretny temat trafi do wszystkich konsumentów, którzy zgłosili chęć otrzymywania tego rodzaju komunikatów (zarejestrowali się). Nietrwały subskrybent może otrzymywać komunikaty opublikowane tylko wtedy, gdy jest **aktywny**. Subskrypcja nietrwała nie gwarantuje dostarczenia komunikatu lub też może dostarczyć komunikat więcej niż jeden raz. Subskrypcja trwała daje pewność, że konsument otrzymał komunikat dokładnie jeden raz.

Konsumpcja komunikatu, choć sam system JMS jest w pełni asynchroniczny, może odbywać się na dwa sposoby wskazane w specyfikacji JMS.

Synchroniczny — subskrybent lub odbiorca jawnie pobiera komunikat z celu, wywołując metodę receive() dowolnej instancji MessageConsumer. Metoda receive()może zablokować działanie wątku, aż do momentu nadejścia komunikatu, lub też można wskazać maksymalny czas oczekiwania na komunikat.
Asynchroniczny — w trybie asynchronicznym klient musi zaimplementować interfejs javax.jms.MessageListener i

udostępnić metodę onMessage(). Gdy komunikat dotrze do celu, dostawca JMS dostarczy go do odbiorcy, wywołując kod metody onMessage.

Komunikat JMS składa się z nagłówka, właściwości i treści.

Nagłówki komunikatu to ściśle określone metadane opisujące komunikat, na przykład ustalające jego cel i pochodzenie. Właściwości to zestawy par klucz-wartość wykorzystywane do celów specyficznych dla aplikacji. Najczęściej używa się ich do szybkiej filtracji nadchodzących komunikatów. Treść komunikatu to właściwa zawartość komunikatu przesyłana od nadawcy do odbiorcy.

API JMS obsługuje dwa rodzaje dostarczania komunikatów, które określają, czy komunikaty zostaną zagubione, jeśli serwer JMS nie zadziała prawidłowo.

- Tryb **trwały** (stosowany domyślnie) instruuje dostawcę JMS, by podjął szczególne środki, żeby komunikat nie został utracony w przypadku błędu JMS. Dostawca JMS zapisuje otrzymany komunikat w trwałym magazynie danych.
- Tryb **nietrwały** nie wymaga od dostawcy JMS trwałego przechowywania komunikatu, więc nie gwarantuje jego dostarczenia w przypadku błędu serwera JMS.

Aplikacja JMS składa się z następujących elementów:

- obiektów administracyjnych — fabryk połączeń i celów,
- połączeń,
- sesji,
- producentów komunikatów,
- konsumentów komunikatów,
- komunikatów.

Obiekt **fabryki połączeń** zawiera zbiór parametrów konfiguracyjnych zdefiniowanych przez administratora. Klient używa go do utworzenia połączenia z dostawcą JMS. Fabryka połączeń ukrywa przed klientem szczegóły specyficzne dla dostawcy, udostępniając wszystkie dane w postaci standardowych obiektów Javy.

Cel to komponent używany przez klienta do określenia docelowej lokalizacji dla tworzonych lub odbieranych komunikatów. W przypadku komunikacji typu **punkt-punkt (PTP)** cel nazywa się kolejką; w przypadku systemów **publikuj-subskrybuj (pub-sub)** cel nazywa się tematem.

Połączenie zawiera wirtualne połączenie z dostawcą JMS. Może ono reprezentować otwarte gniazdo TCP/IP między klientem i dostawcą usługi. Połączenie służy do utworzenia jednej lub wielu sesji.

Sesja to jednowątkowy kontekst dotyczący produkcji i konsumpcji komunikatów. Sesji używa się do tworzenia komunikatów po stronie producenta, konsumpcji komunikatów, a także obsługi samych komunikatów. Sesje szeregują wykonanie metody odbioru komunikatów i zapewniają obsługę transakcyjności, czyli wysyłanie lub odbieranie komunikatów jako niepodzielnej jednostki zadaniowej.

Producent komunikatów to obiekt utworzony przez sesję i używany do wysyłania komunikatów do konkretnej lokalizacji docelowej. W komunikacji punkt-punkt jest to obiekt implementujący interfejs QueueSender. W komunikacji publikuj-subskrybuj jest to obiekt implementujący interfejs TopicPublisher.

Konsument komunikatów to obiekt utworzony przez sesję. Służy do odbierania komunikatów wysłanych pod konkretny adres. Obiekt konsumenta umożliwia klientowi JMS wskazanie swojego zainteresowania konkretnym celem u dostawcy JMS. Dostawca JMS zarządza dostarczeniem komunikatu z celu do zarejestrowanych konsumentów. W komunikacji punkt-punkt jest to obiekt implementujący interfejs QueueReceiver. W komunikacji publikuj-subskrybuj jest to obiekt implementujący interfejs TopicSubscriber.

Część praktyczna – konfigurowanie dostępu do JMS.

Aby rozpocząć tworzenie aplikacji wykorzystującej technologie JMS najpierw musimy przygotować infrastrukturę tj. Kolejki i Topiki do których nadawca będzie pisał a odbiorca czytał wiadomości.

Niestety w aktualnej wersji WildFly Subsystem obsługujący wymianę komunikatów znajduje się poza standardem. Oznacza to ze nie jest zdefiniowany w domyślnej wersji pliku standalone.xml. Na szczęście istnieje drugi plik standalone-full.xml, który zawiera odpowiednie zapisy.

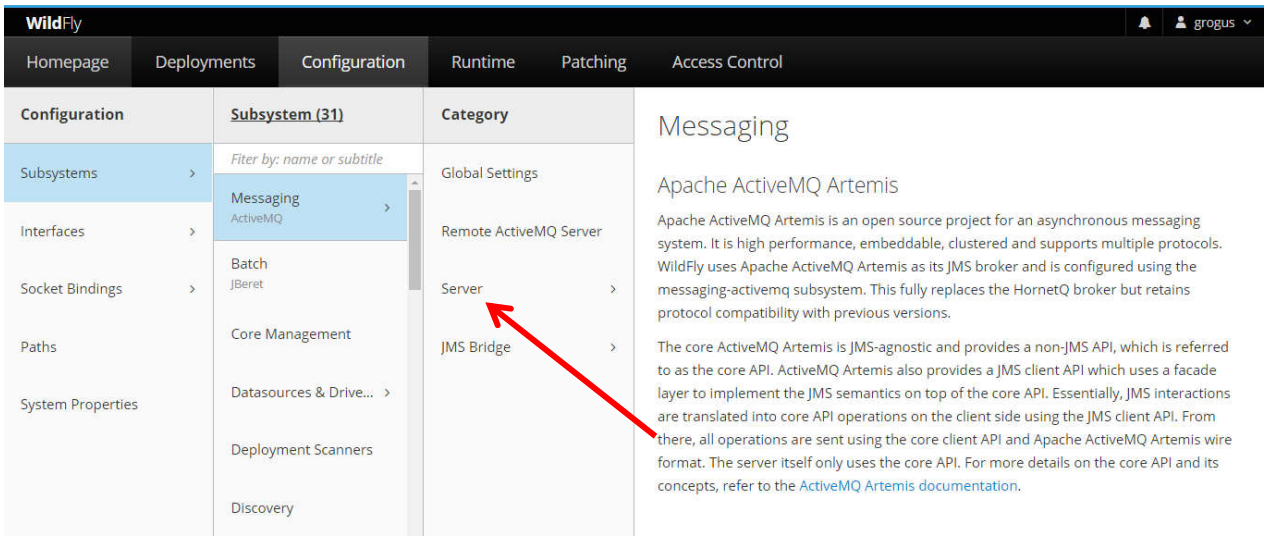
Proszę nadpisać plik standalone plikiem standalone-full. Sprawdzić czy w nowym pliku znajduje się moduł obsługujący messaging.

Krokiem następnym jest konfiguracja infrastruktury. Podobnie jak przy okazji konfiguracji JPA możemy to zrobić na 3 sposoby:

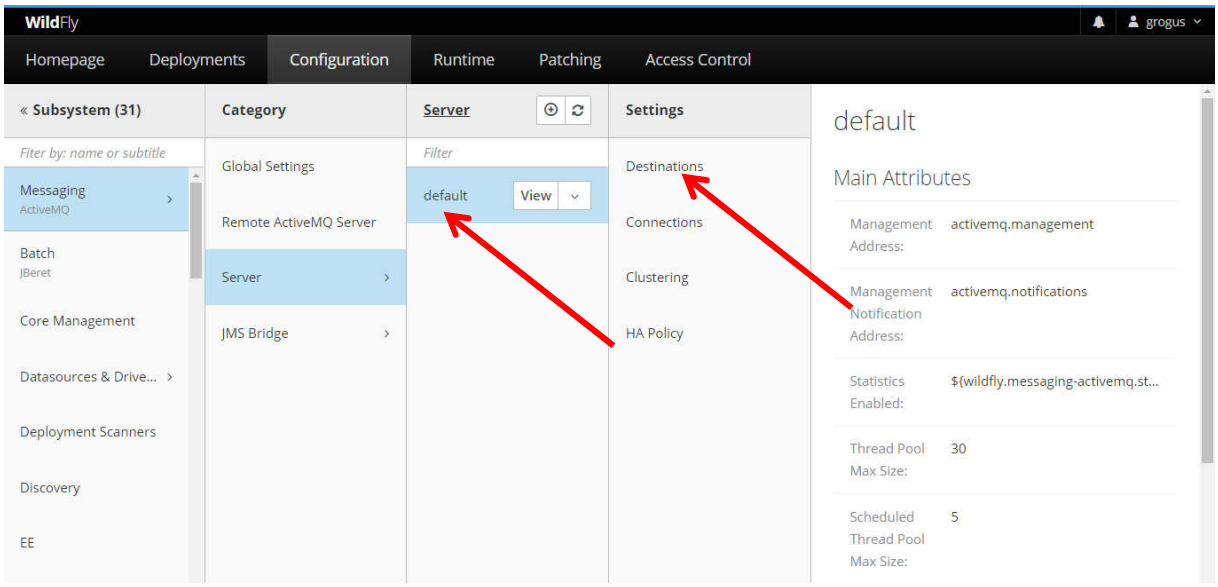
1. Za pomocą Web Admin Consoli
2. CLI
3. Ręcznie modyfikując plik standalone.xml

1. Konfiguracja za pomocą Web Admin Console

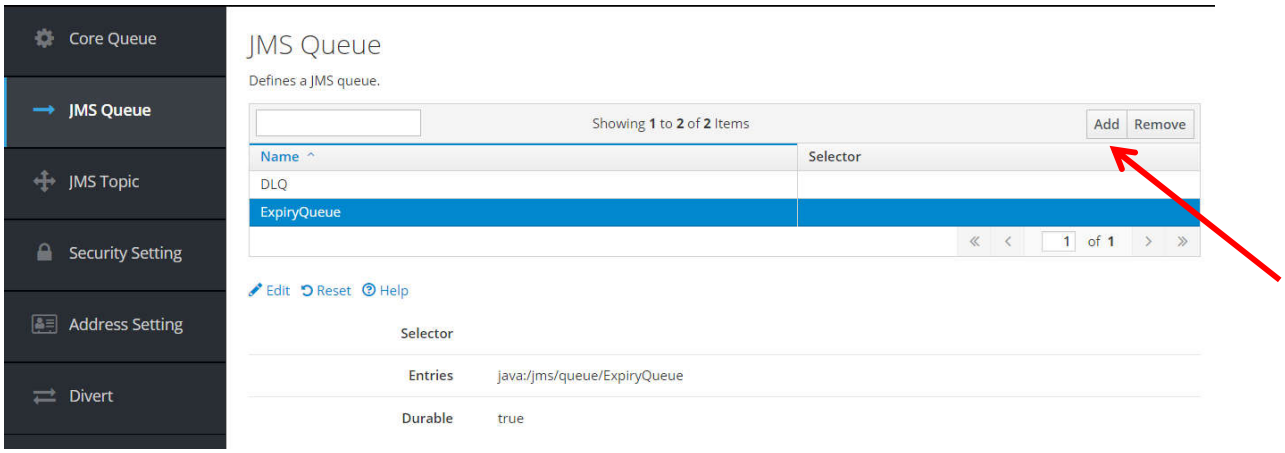
W sekcji Configuration znajdź podsystem Messaging



Konfigurować będziemy domyślny serwer wbudowany w WildFly. Jak będzie można zauważyć konfigurować można także dowolny zewnętrzny serwer obsługujący kolejki komunikatów.

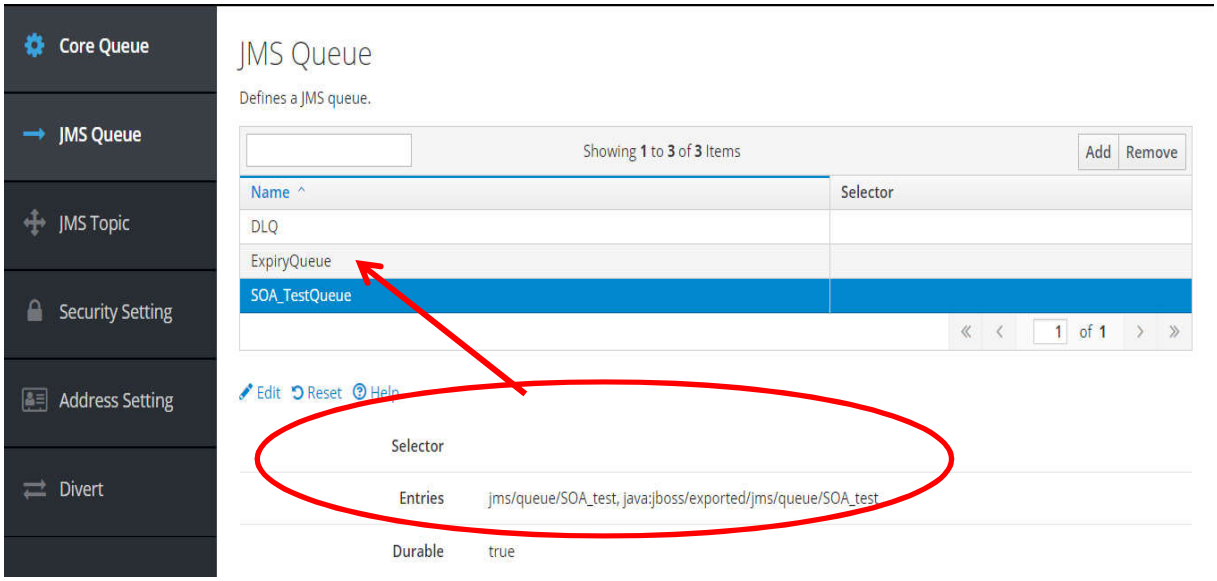


Wybieramy sekcje Destination w celu stworzenia nowej kolejki oraz topica, którego będziemy używali w naszej aplikacji testowej.

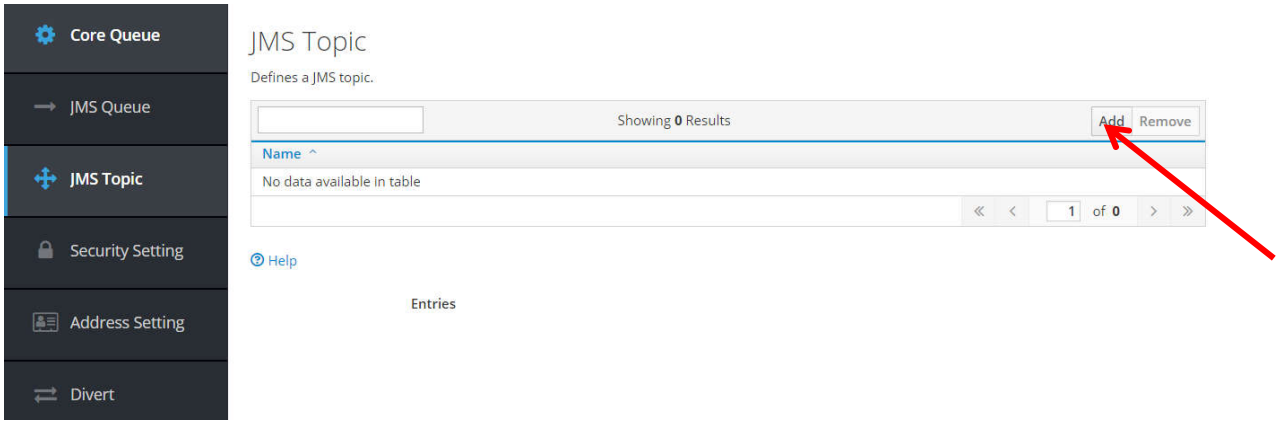


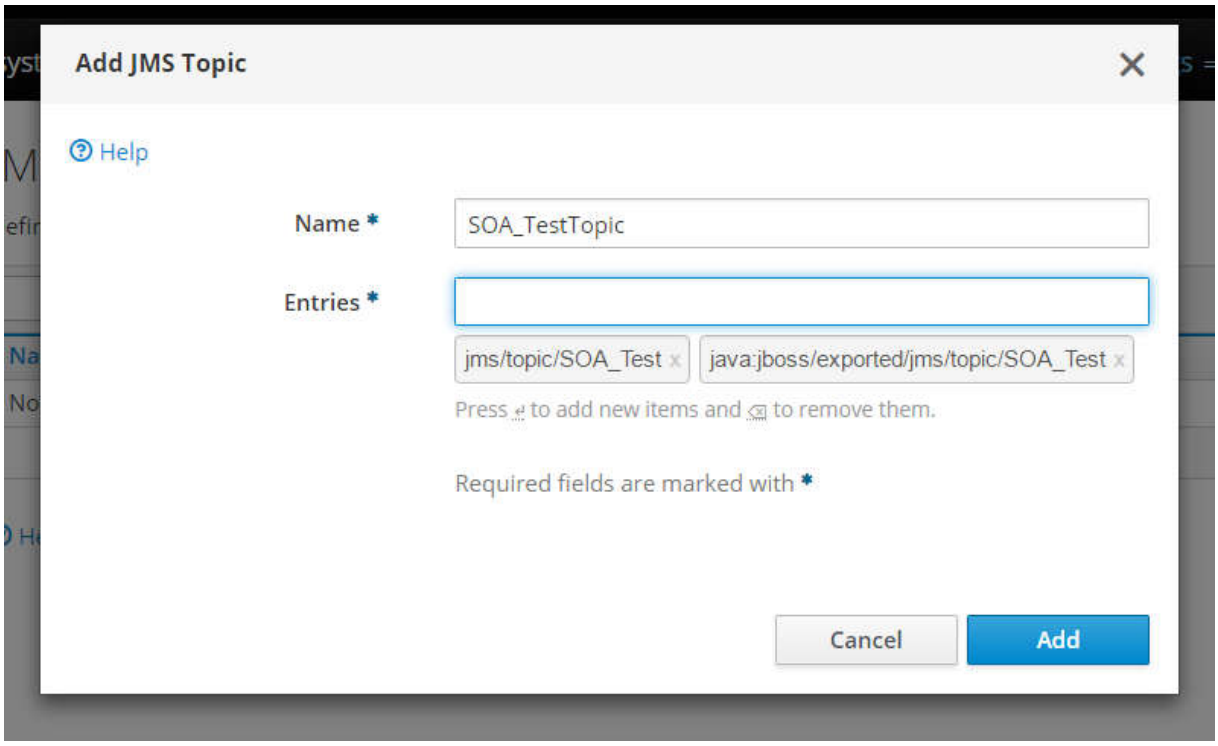
Dodajemy nową kolejkę o nazwie SOA_TestQueue

v



Teraz czas na nowy topic





Wynikiem tych operacji jest modyfikacja pliku standalone.xml w sekcji

<subsystem xmlns="urn:jboss:domain:messaging-activemq:6.0">

```
<jms-queue name="SOA_TestQueue" entries="jms/queue/SOA_test java:jboss/exported/jms/queue/SOA_test" durable="true"/>
<jms-topic name="SOA_TestTopic" entries="jms/topic/SOA_Test java:jboss/exported/jms/topic/SOA_Test"/>
```

Alternatywnym sposobem jest konfiguracja za pomocą CLI:

Przez CLI:

Dodanie kolejki JMS:

```
[standalone@localhost:9999/] jms-queue add --queue-address= SOA_Test
Queue --entries=java:/jms/queue/SOA_test,java:/jboss/exported/jms/qu
eue/SOA_test
```

Add JMS Topic:

```
[standalone@localhost:9999/] jms-topic add --topic-address=SOA_TestT
opic --entries=java:/jms/topic/SOA_Test,java:/jboss/exported/jms/top
ic/SOA_Test
```

Drugi adres JNDI *"java:/jboss/exported/jms/[destination]"* jest wykorzystywany w przypadku łączenia z zdalnym klientem JMS

Poprzez ręczna modyfkacje pliku standalone.xml:

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:6.0">
   <server name="default">
      . . . .
      <jms-queue name="SOA_Test" entries="java:/jms/queue/SOA_test
java:/jboss/exported/jms/queue/SOA_test"/>


      <jms-topic name="SOA_TestTopic" entries="java:/jms/topic/SOA_Test
java:/jboss/exported/jms/topic/SOA_Test"/>

   </server>
</subsystem>
```

Tworzenie i wykorzystanie fabryk połączeń

Zadaniem fabryki połączeń jest przechowywanie parametrów połączenia umożliwiających tworzenie nowych połączeń JMS. Fabryka połączeń korzysta z **JNDI (Java Naming Directory Index)** i może być wyszukiwana przez lokalne oraz zdalne klienty, jeśli tylko obsługują odpowiednie parametry środowiska. Ponieważ fabrykę połączeń można w kodzie stosować wielokrotnie, jest to obiekt, który warto umieścić w pamięci podręcznej klienta zdalnego lub ziarna sterowanego komunikatami.

Domyślnie dostępne są dwie fabryki połaczeń.

- InVmConnectionFactory — ta fabryka połączeń jest dostępna pod adresem java:/ConnectionFactory. Używa się jej, gdy serwer i klient stanowią część jednego procesu (czyli działają w jednej maszynie wirtualnej Javy).
- RemoteConnectionFactory — ta fabryka połączeń dotyczy sytuacji, w których połączenia JMS są zapewniane przez zdalny serwer. Do komunikacji używana jest Netty.

Tworzenie aplikacji:

Wymagane zależności:

```
<dependency>
    <groupId>org.jboss.spec.javax.jms</groupId>
    <artifactId>jboss-jms-api_2.0_spec</artifactId>
    <version>1.0.0.Final</version>
</dependency>
```

Oraz

```
<!-- https://mvnrepository.com/artifact/org.wildfly/wildfly-jms-client-bom -->
    <!-- https://mvnrepository.com/artifact/org.wildfly/wildfly-jms-client-bom -->
    <dependency>
      <groupId>org.wildfly</groupId>
      <artifactId>wildfly-jms-client-bom</artifactId>
      <version>10.1.0.Final</version>
      <type>pom</type>
    </dependency>
```

Przykłady kodu:

Wysyłanie wiadomości:

```
@Resource(mappedName="jms/ConnectionFactory")
private static ConnectionFactory cf;
@Resource(mappedName=”jms/Queue”)
Private static Queue queue;

Connection conn = cf.createConnection();
Session session = conn.createSession(false, Session.AUTO_ACKNOWLEDGE);

MessageProducer producer = session.createProducer(queue);
TextMessage msg = session.createTextMessage();
msg.setText("Komunikat testowy");
producer.send(msg);
```

lub tak: (JMS 2.0)

```
@Inject
private JMSContext context;
@Resource(mappedName=”jms/Queue”)
private static Queue queue;

TextMessage msg = context.createTextMessage("txt");
context.createProducer().send(queue, msg);
```

Odbiór komunikatu z wykozystaniem komponentu MDB

```
@MessageDriven(mappedName=”jms/Queue”)
public class MyMessageBean implements MessageListener {
        public void onMessage(Message msg) {
                TextMessage txtMsg = null;
                try {
                        if (msg instanceof TextMessage) {
                                txtMsg = (TextMessage) msg;
                                String txt = txtMsg.getText();
                        }
                } catch (JMSException e) {...}
        }
}
```

File: Nadawca.java

```
import javax.naming.*;
import javax.jms.*;
```

```
public class Nadawca {
   public static void main(String[] args) {
      try
      {  //Tworzenie połączenie z infrastruktura JMS
         InitialContext ctx=new InitialContext();
        QueueConnectionFactory f=(QueueConnectionFactory)ctx.lookup("QueueConnectionFactory" );
         QueueConnection con=f.createQueueConnection();
         con.start();
         //2) tworzymy sesje
         QueueSession ses=con.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);
         //3) dostep do kolejki
         Queue t=(Queue)ctx.lookup("SOA_TestQueue");
         QueueSender sender=ses.createSender(t);
         //5) Przygotowanie obiektu  do wysyłki
         TextMessage msg=ses.createTextMessage();
            msg.setText(„Ala ma kotka”);
            //7) wysyłka komunikatu
            sender.send(msg);
            System.out.println("Komunikat wysłany.");
         }
         //8) zamknięcie połączenia
         con.close();
      }catch(Exception e){System.out.println(e);}
   }
}
```

File: Odbiorca.java

```
import javax.jms.*;
import javax.naming.InitialContext;

public class MyReceiver {
      public static void main(String[] args) {
               try{
         //1) Tworzenie I inichalizacja połaczenia do JMS
         InitialContext ctx=new InitialContext();
        QueueConnectionFactory f=(QueueConnectionFactory)ctx.lookup("QueueConnectionFactory");
         QueueConnection con=f.createQueueConnection();
         con.start();
         //2) stworzenie secji dla  Queuen
```

```
        QueueSession ses=con.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);
        //3) Dostep do obiektu kolejki
        Queue t=(Queue)ctx.lookup("SOA_TestQueue ");
        //4)tworzenie  QueueReceiver
        QueueReceiver receiver=ses.createReceiver(t);

        //5) Tworzenie nasłuchu
        MyListener listener=new MyListener();

        //6 rejstracja funkji  nasłuchujacej
        receiver.setMessageListener(listener);

        System.out.println("Jestem gotowy, c zekam na komunikat...");
         while(true){
                Thread.sleep(1000);
        }
        }catch(Exception e){System.out.println(e);}
  }

}
```

File: Listener.java

```
import javax.jms.*;
public class MyListener implements MessageListener {

   public void onMessage(Message m) {
      try{
      TextMessage msg=(TextMessage)m;

      System.out.println("Otrzymałem:"+msg.getText());
      }catch(JMSException e){System.out.println(e);}
   }
}
```

Zadanie do realizacji

Zad 1. Napisz prostą aplikacje typu forum tematyczne. W ramach forum możliwe jest tworzenie nowych list tematycznych. Użytkownicy mogą subskrybować się do dowolnej ilości list tematycznych. W ramach istniejącej listy wysyłamy komunikaty do wszystkich subskrybentów lub powinna być możliwość wskazania tylko wybranego do którego chcemy napisać.

Zad 2. Rozszerz projekt biblioteka z lab 6 o otrzymywane potwierdzenia wykonywania każdej z operacji. Użyj do tego celu MDB. Możliwe jest również dodawania nowych pozycji książkowych do oferty biblioteki. W takim przypadku wysyłane są komunikaty powiadamiające o nowej pozycji do wszystkich użytkowników, którzy w procesie rejestracji zdefiniowali taka chęć otrzymywania powiadomień. Podobnie w przypadku zwrotu książki, wszystkie osoby które chciały pożyczyć ksiązke gdy była nie dostępna mają otrzymać powiadomienia. Przyjmijmy, że w naszej bibliotece każdy tytuł liczy tylko jedną pozycje.