Kolejkowanie wiadomości Standard MQ (JMS)

Kolejkowanie wiadomości

Standard wymiany informacji – wiadomości (ang. message) między procesami (mogą być rozproszone)

Przykładowe rozwiązania:

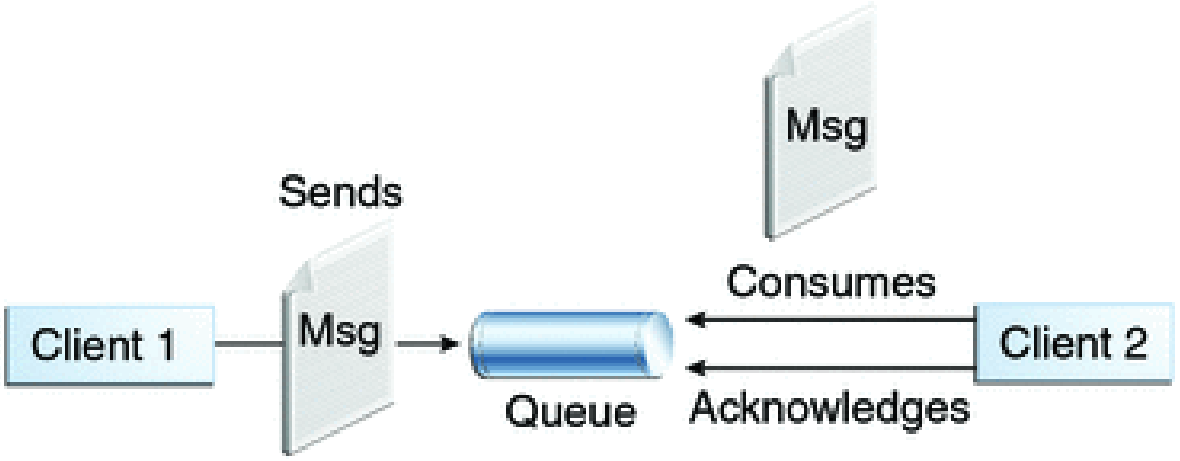
- RabbitMQ
- ActiveMQ
- ZeroMQ
- MsMQ
- Dwie filozofie działania:
- - Point to point
- - Publisher - subscriber

Architektura JMS API

- **JMS provider** - system komunikacyjny implementujący standard JMS i dostarczający funkcjonalności do administracji i kontroli
- **JMS client** - program lub komponent napisany w Javie który produkuje i lub konsumuje wiadomości. Każdy komponent aplikacji Java EE może działać jako klient JMS.
- **Messages (Wiadomości)** – obiekty które przesyłają informacje pomiędzy klientami JMS
- **Administrowane obiekty** – są pre-konfigurowanymi obiektami JMS, stworzonymi przez administratora do użytku przez klientów. Istnieją dwa typy obiektów Administracyjnych w JMS: fabryki połaczeń i punktów docelowych destinations and connection factories.

Filozofia JMS
point-to-point

- Każda wiadomość ma jednego konsumenta
- Nie ma zależności czasowych pomiędzy nadawcą i odbiorcą wiadomości – nadawca i odobiorca działają niezależnie
- Odbiorca musi przesłać potwierdzenie



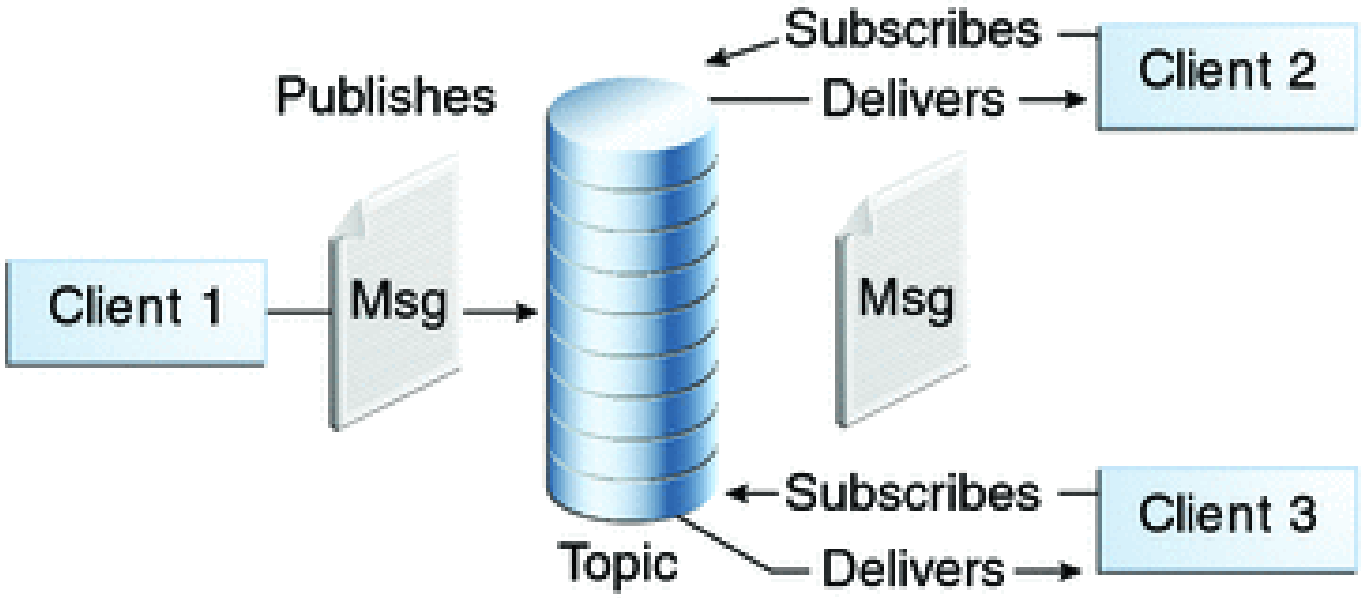
Filozofia JMS Publish/Subscribe

W filozofi publikatorzy/subskrybenci klient adresuje wiadomość do „tematu” który stanowi tablicę informacyjną. Klienci – każdy dostaje każdą wiadomość. „Tematy” przechowują wiadomości tak długo aż subskrybenci ich nie odbiorą

- Każda wiadomość ma wielu odbiorców
- Publikator i subskryber posiadają zależnosci czasowe – do subsklrybenta trafiają tylko te wiadomości, które zostały wysłane jak był aktywny. Dobierze tylko te wiadomości przy których był aktywny

Filozofia JMS Publish/Subscribe

W filozofi publikatorzy/subskrybenci klient adresuje wiadomość do „tematu” który stanowi tablicę inf
wi ści tak dłu



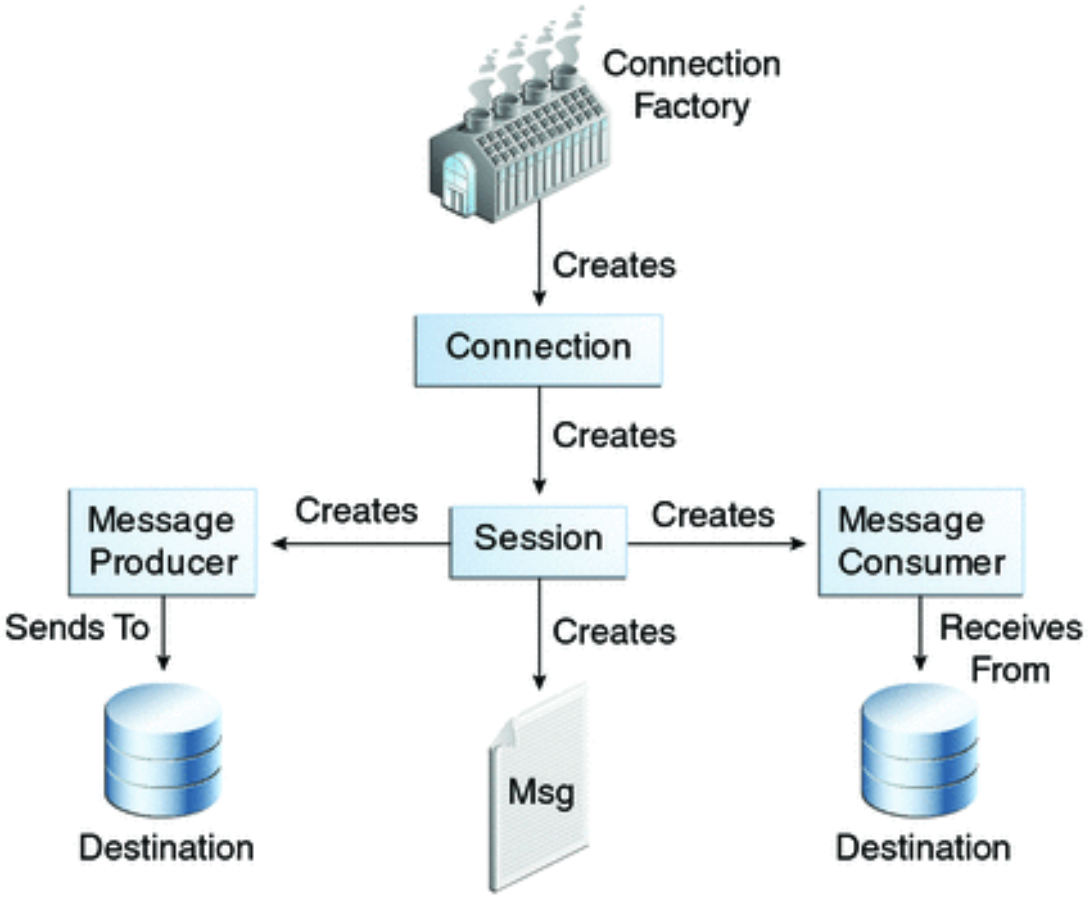
- k
- F
  c
  v ktywny.
  D ył
  aktywny

Klienci

Z założenia JMS jest rozwiązaniem asynchronicznym ale wszystko zależy od implementacji rozwiązania:

- Rozwiązanie synchroniczne - metoda recieve() jest sychroniczna tj. blokuje odbiór wiadomości i czeka aż nadejdą nowe wiadomości
- Rozwiązanie asynchroniczne – rejestracja tzw. **message listener** który obsługuje całość asynchronicznie i wywoływany jest w momencie pojawienia się wiadomości

JMS API model



Connection

Cele dla których używamy obiektu Connection:

- Enkapsulacja otwartego połączenia z dostarczycielem usług JMS. Zawiera otwarte gniazdo (socket typu TCP/IP) pomiędzy klientem a serwerem
- Zawiera informacje dot. autentykacji klienta
- Może w sposób unikatowy identyfikować klienta
- Dostarcza obiekt ConnectionMetaData
- Wspiera opcjonalny objekt ExceptionListener.

Session

Cele dla obiektu sesji:

- Sesje są przeznaczone dla jednego wątku
- Fabryka dla producentów i odbiorców wiadomości
- Zapewnia zoptymalizowaną pod kątem providera fabrykę wiadomości
- Jest fabryką dla TemporaryTopics i TemporaryQueues.
- Dostarcza narzędzi do stworzenia kolejek i tematów, dla klientów którzy potrzebują dynamicznego nimi zarządzania
- Zapewnia porządek dostarczania i dobierania wiadomości
- Zachowuje wiadomości do czasu ich potwierdzenia
- Serializuje przetwarzanie wiadomości przez zarejestrowany listener
- Fabryka dla QueueBrowsers.

Producent

Stworzenie połączenia i jego uruchomienie

```
main(String[] args) throws JMSException{

    String url = ActiveMQConnection.DEFAULT_BROKER_URL;
    String subject = "test";
    ConnectionFactory conncectionFactory = new ActiveMQConnectionFactory(url);
    Connection connection = conncectionFactory.createConnection();
    connection.start();

    Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE); //

    Destination destination = session.createQueue(subject);
    MessageProducer producer = session.createProducer(destination);

    boolean messageToSend = true;
    while(messageToSend){
        TextMessage message = session.createTextMessage("hej
        producer.send(message);
        messageToSend = false;
    }
    connection.close();
}
```

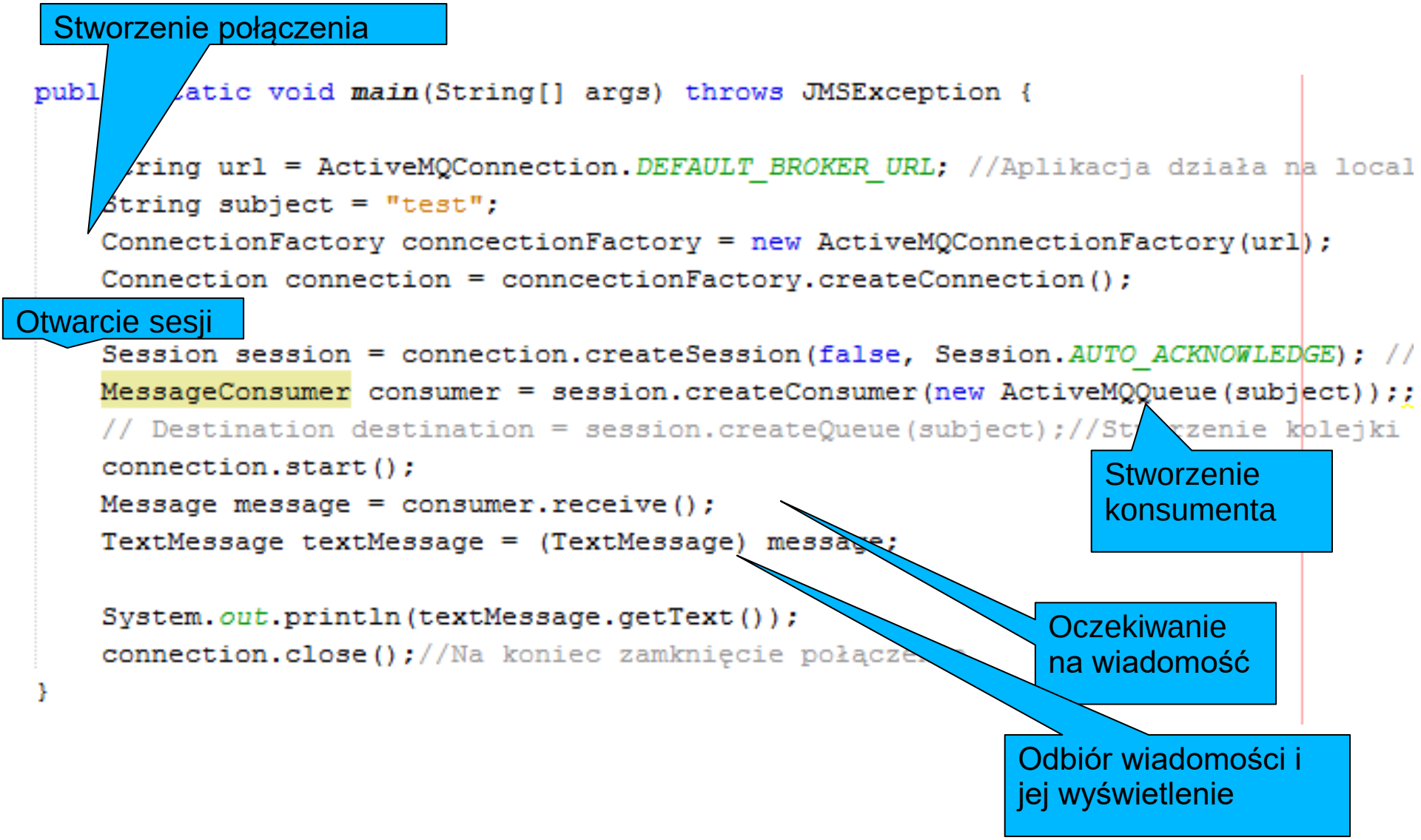
Stworzenie sesji

Stworzenie elementu docelowego i podpięcie się niego

Wysyłanie wiadomości

Zamknięcie połaczenia

Konsument




```
publ  atic void main(String[] args) throws JMSException {

    tring url = ActiveMQConnection.DEFAULT_BROKER_URL; //Aplikacja działa na local
    String subject = "test";
    ConnectionFactory conncectionFactory = new ActiveMQConnectionFactory(url);
    Connection connection = conncectionFactory.createConnection();
    Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE); //
    MessageConsumer consumer = session.createConsumer(new ActiveMQQueue(subject));;
    // Destination destination = session.createQueue(subject);//St rzenie kolejki
    connection.start();
    Message message = consumer.receive();
    TextMessage textMessage = (TextMessage) messa e;

    System.out.println(textMessage.getText());
    connection.close();//Na koniec zamknięcie połącze
}
```

Konsument 2

```
public static void main(String[] args) throws JMSException {

    String url = ActiveMQConnection.DEFAULT_BROKER_URL; //Aplikacja działa na localhost
    String subject = "test";
    ConnectionFactory conncectionFactory = new ActiveMQConnectionFactory(url);
    Connection connection = conncectionFactory.createConnection();

    Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE); //

    MessageConsumer consumer = session.createConsumer(new ActiveMQQueue(subject));;//Stworzenie do
    consumer.setMessageListener(new MessageListener() {
        @Override
        public void onMessage(Message msg) {
            if (msg instanceof TextMessage) {
                TextMessage txtMsg = (TextMessage) msg;
                try {
                    System.out.println(txtMsg.getText());
                } catch (JMSException ex) { }
        }  }
    });
    connection.start();
    connection.close();//Na koniec zamknięcie p
}
```

Dodanie MessageListenera

Dopiero po dodaniu listenera tworzymy połączenie