1-entity(consumer & producer)

2-etc(consumer & producer)

Entitymanagerprovider class(

```
public static EntityManager getEntityManager() {
    EntityManagerFactory emf =
Persistence.createEntityManagerFactory("default");
    return emf.createEntityManager();
}
```

3-repository

For customer:

```
public void save(Customer customer) {
    EntityManager entityManager = EntityManagerProvider.getEntityManager();
    EntityTransaction transaction = entityManager.getTransaction();
    transaction.begin();
    entityManager.persist(customer);
    transaction.commit();
    entityManager.close();

}

public Optional<Customer> findById(int id) {
    EntityManager entityManager = EntityManagerProvider.getEntityManager();
    Optional<Customer> customer =
Optional.ofNullable(entityManager.find(Customer.class, id));
    entityManager.close();
    return customer;
}
```

4-carrier(consumer & producer)

5-service

For deposit producer

```
public class DepositServiceProducer {

    private  DepositRepository depositRepository = new DepositRepository();
    private  MessageSender messageSender = new MessageSender();
    public void define(CustomerDefineCarrier carrier) {
        Deposit deposit = new Deposit(carrier.fullName());
        repository.save(deposit);
    public void withdrawAndSend(String shabaNumber, double amount) {
        Optional<Deposit> depositOptional =
depositRepository.findByShabaNumber(shabaNumber);

        if (depositOptional.isPresent()) {
            Deposit deposit = depositOptional.get();
            if (deposit.getBalance() >= amount) {
                deposit.setBalance(deposit.getBalance() - amount);
                depositRepository.update(deposit);
                messageSender.sendWithdrawalMessage(shabaNumber, amount);
 }}  }

    public void closeMessagingConnection() {
        messageSender.closeConnection();
```

for customer(producer & consumer)

```
public class CustomerService {

    private final CustomerRepository customerRepository = new
CustomerRepository();

    public void registerCustomer(Customer customer) {

        if (customer.getFirstName() == null ||
customer.getFirstName().isEmpty()) {
            throw new IllegalArgumentException("نام مشتری نباید خالی باشد.");
        }
        if (customer.getLastName() == null ||
customer.getLastName().isEmpty()) {
            throw new IllegalArgumentException("نام خانوادگی مشتری نباید خالی
باشد.");
        }
        if (customer.getNationalCode() == null ||
customer.getNationalCode().isEmpty()) {
            throw new IllegalArgumentException("کد ملی مشتری نباید خالی
باشد.");
        }

        customerRepository.save(customer);
    }
```

for consumer

```
public class DepositServiceConsumer {

    private  DepositRepository depositRepository = new DepositRepository();
public void define(CustomerDefineCarrier carrier) {
        Deposit deposit = new Deposit(carrier.fullName());
        repository.save(deposit);
public void processDepositMessage(TextMessage message) {
        try {
            String shabaNumber = message.getStringProperty("shabaNumber");
            double amount = message.getDoubleProperty("amount");

            Optional<Deposit> depositOptional =
depositRepository.findByShabaNumber(shabaNumber);
            if (depositOptional.isPresent()) {
                Deposit deposit = depositOptional.get();
                deposit.setBalance(deposit.getBalance() + amount);
                depositRepository.update(deposit);
            }
        } catch (JMSException e) {
```

6- MessageSender for produser

```
public class MessageSender {

    private final String activeMqUrl = "tcp://localhost:61616";
    private final String queueName = "transferQueue";
    private ConnectionFactory connectionFactory;
    private Connection connection;
    private Session session;
    private MessageProducer producer;
    private Destination destination;

    public MessageSender() {
        try {
            connectionFactory = new ActiveMQConnectionFactory(activeMqUrl);
            connection = connectionFactory.createConnection();
            connection.start();
            session = connection.createSession(false,
Session.AUTO_ACKNOWLEDGE);
            destination = session.createQueue(queueName);
            producer = session.createProducer(destination);
        } catch (JMSException e) {
            e.printStackTrace();
        }
    }
    public void sendWithdrawalMessage(String shabaNumber, double amount) {
        try {
            TextMessage message = session.createTextMessage();
            message.setStringProperty("shabaNumber", shabaNumber);
            message.setDoubleProperty("amount", amount);
            message.setStringProperty("transactionType", "WITHDRAWAL");
            producer.send(message);
        } catch (JMSException e) {
            e.printStackTrace();
        }
    }

    public void closeConnection() {
        try {
            if (producer != null) producer.close();
            if (session != null) session.close();
            if (connection != null) connection.close();
        } catch (JMSException e) {
            e.printStackTrace();
        }
    }
}
```

TransactionMessageConsumer

```
public class TransactionMessageConsumer {

    private final String activeMqUrl = "tcp://localhost:61616";
    private final String queueName = "transferQueue";
    private ConnectionFactory connectionFactory;
```

```java
    private Connection connection;
    private Session session;
    private MessageConsumer consumer;
    private Destination destination;
    private final DepositServiceConsumer depositService = new
DepositServiceConsumer();

    public TransactionMessageConsumer() {
        try {
            connectionFactory = new ActiveMQConnectionFactory(activeMqUrl);
            connection = connectionFactory.createConnection();
            connection.start();
            session = connection.createSession(false,
Session.AUTO_ACKNOWLEDGE);
            destination = session.createQueue(queueName);
            consumer = session.createConsumer(destination);
            while (true) {
                Message message = consumer.receive();
                if (message instanceof TextMessage) {
                    TextMessage textMessage = (TextMessage) message;
                    depositService.processDepositMessage(textMessage);
                } else if (message != null) {
                    System.out.println("شد دریافت متنی غیر پیام: " +
message.getClass().getName());
                }
            }

        } catch (JMSException e) {
            e.printStackTrace();
            throw new RuntimeException("پیام دریافت یا اتصال در خطا", e);
        } finally {
            closeConnection();
        }
    }

    public void closeConnection() {
        try {
            if (consumer != null) consumer.close();
            if (session != null) session.close();
            if (connection != null) connection.close();
        } catch (JMSException e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        new TransactionMessageConsumer();
    }
}
```