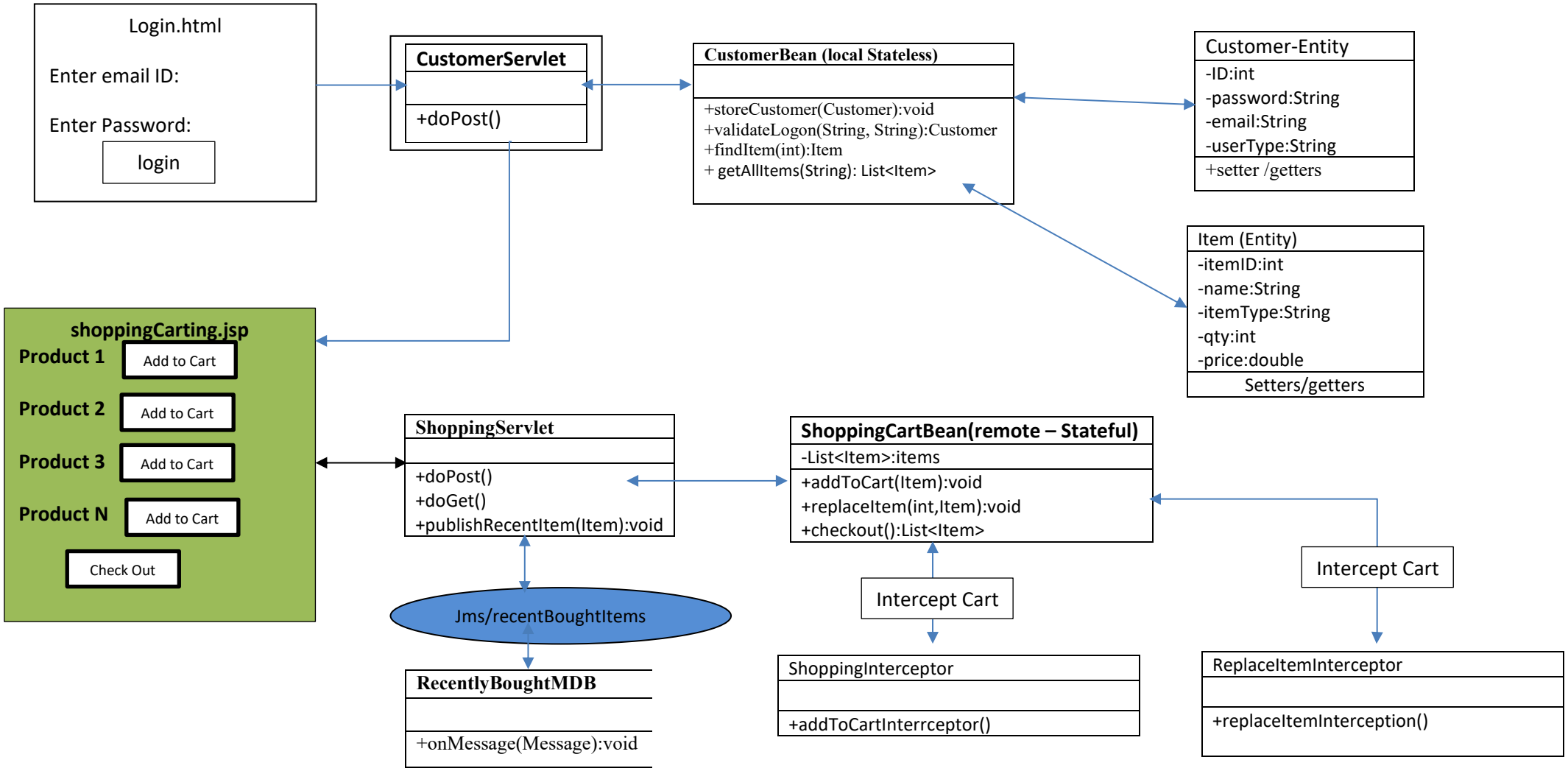
Tshwane University of Technology *We empower people* _____ Signature **I hereby subject myself to the examination rules and regulations of Tshwane University of Technology**	**Faculty of Information And Communication Technology (ICT)** **Department of Computer Science** **Subject: Distributed Programming** DSD117V **Semester Test 2**	
	11 October 2025 **Number of Pages: 4**	1st Examiner: Dr. ML. Gadebe
	Total Marks 70	
Group Number:	**Student Number:**	

Nooko wholesalers requested to develop an enterprise application that will allow all logon customers to view, add and replace items from a shopping cart online. The application will intercept all items added to a shopping cart to add a levy of 0.08 on items above R800, otherwise it will 0.03 on items less than R800. Also, the application will intercept the replace items to subtract R1.14 on all items replaced from a shopping cart. The application uses publish/subscribe communication to allows customers to publish all recently bought items on check out. See system architecture in the next page.


Login.html
Enter email ID:
Enter Password:
login
CustomerServlet
+doPost()
CustomerBean (local Stateless)
+storeCustomer(Customer):void
+validateLogon(String, String):Customer
+findItem(int):Item
+ getAllItems(String): List<Item>
Customer-Entity
-ID:int
-password:String
-email:String
-userType:String
+setter /getters
Item (Entity)
-itemID:int
-name:String
-itemType:String
-qty:int
-price:double
Setters/getters
shoppingCarting.jsp
Product 1
Add to Cart
Product 2
Add to Cart
Product 3
Add to Cart
Product N
Add to Cart
Check Out
ShoppingServlet
+doPost()
+doGet()
+publishRecentItem(Item):void
ShoppingCartBean(remote – Stateful)
-List<Item>:items
+addToCart(Item):void
+replaceItem(int,Item):void
+checkout():List<Item>
Intercept Cart
Intercept Cart
Jms/recentBoughtItems
RecentlyBoughtMDB
+onMessage(Message):void
ShoppingInterceptor
+addToCartInterrceptor()
ReplaceItemInterceptor
+replaceItemInterception()

Question 1 (40)

1. Create a persistence.xml that will interact with the nookoDbase which is mapped with entity managers. //1/

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">
<persistence-unit name="nookoDbase -ejbPU" transaction-type="JTA"> //1 mark
<jta-data-source>jdbc/sample</jta-data-source>
<exclude-unlisted-classes>false</exclude-unlisted-classes>
<properties>
<property name="javax.persistence.schema-generation.database.action" value="create"/>
</properties>
</persistence-unit>
```

2. Create two entities (*Customer and Item*) as shown on the system architecture. Map each entity to tblCustomer and tblItem respectively. //5/

```
@Entity/1 Mark
@Table(name="tblCustomer")/1 Mark
public class Customer implements Serializable
{
 @Id
 @GeneratedValue(strategy=GenerationType.AUTO) /1 Mark
 private int ID;
```

```
@Entity
@Table(name="tblItem")/1 Mark
public class Item implements Serializable
{
 @Id
 @GeneratedValue(strategy=GenerationType.AUTO) /1 Mark
 private int itemID;
```

```
 private String password;
 private String emails;
private String userType;
}
```

```
    private String name;
    private String itemType;
    private int qty;
    private double price;
}
```

3. Create a stateless local session bean called *CustomerBean* that will store and validate a Customer. The session bean will use email and password to validate logons and return an object of type Customer if the logons are valid otherwise return null. Finally, the session bean will return an Item using item id and it will a return list of all items using findItem() and getAllItems() respectively as shown on system architecture. //12/

```
public class CustomerBean implements CustomerBeanService
{
@PersistenceContext(unitName=" nookoDbase -ejbPU ")//1 mark
private EntityManager entityManager;
@Override
public void storeCustomer(Customer customer)
{
    entityManager.persist(customer); //1 mark
}
@Override
public Customer validateLogon(String email, String password)
{
String sql = "select customer from Customer customer where customer.email like '"+email+"' and customer.password
'"+password+"'";//2 mark
Customer cust = null;
Query query = entityManager.createQuery(sql); //2 mark
try
  {
    cust = (Customer) query.getSingleResult();//1 mark
  }
catch(Exception er)
```

```
        {
            cust = null;
        }
        return cust; //1 mark
    }
@Override
public Item getItem(int itemId)
{
    return entityManager.find(Item.class, itemId); //1 mark
}
public List<Item> getAllItems(String itemType)
{
   String sql = "select item from Item item where item.itemType = '" +itemType + "'";//1 mark
   List<Item> items;
   Query query = entityManager.createQuery(sql); //1 mark
  try
   {
      items = (List<Items>) query.getResultList();//1 mark
   }
catch(Exception er)
    {
       items = null;
    }
       return items;
  }
}
```

4. Create a stateful session bean called *ShoppingCartBean* that will allow a customer to add, replace and return all item on shopping cart. The replaceItem() method accepts itemID and a replacement Item; the method will search and replace existing item. //7/

```
@Stateful
public class ShoppingCartBean implements ShoppingCartBeanService {

    private List<Item> items;

    @Override
    @PostConstruct
    public void initialliseShopCart() {
        items = new ArrayList<Item>();
    }

    @Override
    @Interceptors(ShoppingInterceptor.class) //1 mark
    public void addItem(Item item) {
      products.add(product); //1 mark
    }

    @Override
    @Interceptors(ReplaceItemInterceptor.class) //1 mark
    public void replaceItem(int itemId, Item item)
    {
        int i = 0;
      for(Item item: items) //1 mark
       {
          if (item.getItemID() == itemId) //1 mark
          {
             items.set(i, Item); //1 mark
             break;
          }
         ++i;
       }
    }

    @Override
    public List<Product> getAllItem()
```

```
    {
        return products; //1 mark
    }
}
```

5. Create a reusable interceptor class named *ShoppingInterceptor* that will intercept the addToCart() methods The addToCartInterrceptor() method intercepts addToCart() method to add a levy of 0.08 or 0.03 according to item price. //8/

```
@Interceptor//1 mark
public class ShoppingInterceptor {
    @AroundInvoke
    public Object addToCartInterrceptor(InvocationContext cnt) throws Exception
    {
        Object[] parameters = cnt.getParameters();//1 mark
       if (parameters != null)
       {
          for(Object parameter: parameters ) //1 mark
           {
              if (parameter instanceof Item)
              {
                 Item item = (Item) parameter; //1 mark
                  if (item.getPrice() > 800) //1 mark
                     {
                          //Add Levy
                           item.setPrice(item.getPrice() + item.getPrice() * 0.08); //1 mark
```

```
                    }
                    else
                    {
                        item.setPrice(item.getPrice() + item.getPrice() * 0.03); //1 mark
                    }

                }
            }
            return cnt.proceed();//1 mark
        }
    }
```

6. Create another reusable interceptor class named *ReplaceItemInterceptor* that will intercept replaceItem() method. The replaceItemInterception() method intercepts replaceItem() method to subtract R1.14 from the replacement Item. //7/

```
@Interceptor//1 mark
public class ReplaceItemInterceptor {
    @AroundInvoke
    public Object replaceItemInterception (InvocationContext cnt) throws Exception//1 mark
    {
        Object[] parameters = cnt.getParameters();//1 mark
       if (parameters != null)
       {
          for(Object parameter: parameters ) //1 mark
           {
```

```
                if (parameter instanceof Item) //1 mark
                {
                    //Remove R 1.14 from item
                        item.setPrice(item.getPrice() – 1.14); //1 mark
                }
            }
        return cnt.proceed();//1 mark
    }
}
```

Question 2 (30)

1. Create a message driven bean called *RecentlyBoughtMDB* that subscribes to a topic named “Jms/recentBoughtItems”. The message driven bean will read any changes made on the to a topic named “Jms/recentBoughtItems” and display each bought item on a console screen. //11/

```
@MessageDriven(activationConfig = {
   @ActivationConfigProperty(propertyName = "clientId", propertyValue = "jms/ recentBoughtItems "), //1 mark
   @ActivationConfigProperty(propertyName = "destinationLookup", propertyValue = "jms/ recentBoughtItems"),
   @ActivationConfigProperty(propertyName = "subscriptionDurability", propertyValue = "Durable"), //1 mark
```

```
    @ActivationConfigProperty(propertyName = "subscriptionName", propertyValue = "jms/recentBoughtItems"),//1 mark
    @ActivationConfigProperty(propertyName = "destinationType", propertyValue = "javax.jms.Topic")//1 mark
})
public class RecentlyBoughtMDB implements MessageListener {//1 mark

    public RecentlyBoughtMDB () {
    }

    @Override
    public void onMessage(Message message) // 1 marks
    {
        if(message instanceof ObjectMessage) //1 mark
        {
            try {
                ObjectMessage objMsg = (ObjectMessage) message; //1 mark
                Item item = (List<Product>) objMsg.getObject();//1 mark
                System.out.println("Item recently bought ");
                System.out.println(item.getName() + " " + item.getQty()  + " " + item.getPrice());//2 mark
                }
            } catch (JMSException ex) {
                Logger.getLogger(RecentBoughtProductTopic.class.getName()).log(Level.SEVERE, null, ex);
            }

        }
    }

}
```

2. Create a controller class called LogonServlet that override the doPost method to do the following:

- The doPost method will accept the email address and password to validate a customer user; if the logons are valid the servlet will redirect the user to the shoppingCart.jsp with a list of items, otherwise will redirect the user to login.html. //6/

```
@WebServlet(urlPatterns = {"/LoginServlet"})
public class LoginServlet extends HttpServlet {
@EJB
private CustomerService service; //1 mark
 @Override
  protected void doPost(HttpServletRequest request, HttpServletResponse response)
          throws ServletException, IOException
    {
       String email, password, logons;
       email = request.getParameter("email");
       password = request.getParameter("password");
    try {
          Customer objCustomer service.validate(email, password);  //1 mark
          if (objCustomer != null) //1 mark
          {
              List<Item> items = service.getAllItems();
              request.setAttribute("items", items);    //1 mark
              RequestDispatcher dispatch = this.getServletContext().getRequestDispatcher("shoppingCarting.jsp");//1 mark
              dispatch.forward(request, response);
           }
      else
          {
               response.sendRedirect(login.html); //1 mark
           }
          }

    } catch (JMSException ex) {
       Logger.getLogger(LoginServlet.class.getName()).log(Level.SEVERE, null, ex);
```

```
        }

        }



}
```

3. Create another controller class called *ShoppingServlet* that override the doPost methods to do the following:

   a. The doPost method will accept the item id from the *shoppingCarting.jsp* and add it to the shopping cart and store it to a session and redirect the user back to *shoppingCarting.jsp*. //6/

   b. The doPost method will receive a command to "check out", then the method will publish each item from a shoppingCart list to a topic named "Jms/recentBoughtItems" topic using a publishRecentItem() method. //3/

   c. Create publishRecentItem(Item) method that will publish each item bought. //5/

```
@Resource(lookup="jms/FirstJmsConnectionFactory")
 ConnectionFactory factory;
 @Resource(lookup="jms/recentBoughtItems")
 Topic topic;
@Override
   protected void doPost(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException
   {

       try {

                  String decides = request.getParameter("select");
```

```
                        if (decides.equals("add to cart"))1 Mark
                          {
                            int itemID = Integer.parseInt(request.getParameter("itemID"));1 Mark
                            Item item = sessionBean.getItem(itemID); 1 Mark
                            if (item != null)
                                  {
                                    shoppingCart.addItem(item); 1 Mark
                                    response.sendRedirect("shoppingCarting.jsp");1 Mark
                                  }

                             else if (decides.equals("Check out"))1 Mark
                                  {
                                        for(Item item : shoppingCart.getAllItems())1 Mark
                                        {
                                              publishRecentItem(item); 1 Mark
                                        }
                                  }

                         }
                    catch (NamingException ex)
                    {
                        Logger.getLogger(ShoppingServlet.class.getName()).log(Level.SEVERE, null, ex);
                    }
            }


public void publishRecentItem(Item item)
{
      try
         {
            //Create to the connection
            Connection connection = factory.createConnection();1 Mark
            //Create session
            Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE); 1 Mark
            //Create Publisher
```

```
            MessageProducer producer = session.createProducer(topic); 1 Mark
            ObjectMessage objMsg = session.createObjectMessage(item); 1 Mark
            //send the object
            producer.send(objMsg); 1 Mark
            // Close session and connection
            session.close();
            connection.close();
        }
        catch(Exception ex)
        {

        }

}
```

Submit the following on EC:

a. *Persistence.xml.*

b. *Customer.java, Item.java and CustomerBean.java*

c. *ShoppingCartBean.java and ShoppingInterceptor.java*

d. *RecentlyBoughtMDB.java*

e. *ShoppingServlet.java, order.jsp.*