Практичне завдання до теми 3

1. Вимоги щодо оформлення звіту

Звіт подається у вигляді документу MS Word (будь-якої версії), що містить :

1) варіант завдання;

2) програмний код класу тестів;

3) пояснення щодо вибору тестуючих комбінацій (класу еквівалентності).

Назва документу складається з фамилії та ініціалів студента, номера групи.

2. Варіанти завдань

Варіант 1.

Дано код програми торгового кошика **`ShoppingCart`**.

Необхідно описати класи еквівалентності та здійснити модульне тестування для методу **`addItem,`** для якого сформульовано наступні вимоги:

1) назва товару повинна бути не пустою і не перевищувати 32 символи;

2) ціна повинна бути більшою нуля і меншою 1000;

3) кількість товару повинна бути більшою нуля, але не більшою 1000;

4) тип товару повинен бути одним з допустимих типів, описаних константами **`Item.Type;`**

5) при додаванні 100-го товару повинно викинутися виключення **`IndexOutOfBoundsException;`**

6) при неправильному параметрі товару повинно викинутися виключення **`IllegalArgumentException.`**

Необхідно описати класи еквівалентності та розробити параметризовані модульні тести для методу **`ShoppingCart`**.calculateDiscount, який розраховує знижку (Discount) за наступними правилами:

1) для REGULAR товаров знижки немає;

2) для SECOND товарів знижка становить 50%, якщо їх купують більше одного;

3) для DISCOUNT товарів знижка становить 10% і по 10% за кожний повний десяток товарів, але не більше 50% сумарно;

4) для SALE товарів знижка становить 90%;

5) крім того за кожну повну сотню товарів додається 10% знижки, але так, щоб сумарна знижка не перевищувала 80%.

Повна ціна розраховується як ціна товару × кількість × знижку.

Варіант 2.

Дано код програми торгового кошика ShoppingCart.

Необхідно описати класи еквівалентності та здійснити модульне тестування для методу toString, для якого сформульовано такі вимоги:

1) Якщо товарів в кошику немає, то повертається рядок "No items.";

2) Якщо товари в кошику є, то інформація виводиться у табличній формі в наступному форматі:

```
 # Item                   Price Quan. Discount      Total
-------------------------------------------------------
 1 Some title              $.30    2         -       $.60
 2 Some very long ti... $100.00    1       50%     $50.00
...
31 Item 42              $999.00 1000         - $999000.00
-------------------------------------------------------
31                                              $999050.60
```

де всі числа вирівняні по правому краю; назва обрізається, якщо перевищує 20 символів; замість нульової скидки виводиться прочерк.

Необхідно описати класи еквівалентності та розробити параметризовані модульні тести для методу **`ShoppingCart`**.calculateDiscount, який розраховує знижку (Discount) за наступними правилами:

1) для REGULAR товаров знижки немає;

2) для SECOND товарів знижка становить 50%, якщо їх купують більше одного;

3) для DISCOUNT товарів знижка становить 10% і по 10% за кожний повний десяток товарів, але не більше 50% сумарно;

4) для SALE товарів знижка становить 90%;

5) крім того за кожну повну сотню товарів додається 10% знижки, але так, щоб сумарна знижка не перевищувала 80%.

Повна ціна розраховується як ціна товару × кількість × знижку.

3. Контрольний приклад

Варіант 1. Тестування методу `addItem`.

```
import org.junit.*;
import static org.junit.Assert.*;

public class AddItemTest {

    private ShoppingCart cart;

    @Before
    public void createCart(){
        cart = new ShoppingCart();
    }

    @Test(expected = IllegalArgumentException.class)
    public void zeroPrice(){
        cart.addItem("Title", 0.00f, 1, Item.Type.REGULAR);
    }

    @Test(expected = IllegalArgumentException.class)
    public void zeroPrice(){
        cart.addItem("Title", 0.00f, 1, Item.Type.REGULAR);
    }
....
}
```

1) Тест AddItemTest.zeroPrice() – перевірка значення за межею допустимого діапазону (перевірка робастності).

2) .....

Варіант 1. Тестування методу `ShoppingCart`.calculateDiscount

Створимо три тестові масиви:

Q_VALUES – масив варіантів кількості одиниць товару;

T_VALUES – масив варіантів типу товару;

D_VALUES – масив контрольних(очікуваних) значень дисконту (знижки).

Код тесту матиме вигляд:

```
import org.junit.*;
import static org.junit.Assert.*;
import java.util.*;
import org.junit.runners.*;
import org.junit.runner.*;
```

```
@RunWith(Parameterized.class)
public class DiscountTest
{
    private static final int[] Q_VALUES = {1, 9, 10, 19};

    private static final Item.Type[] T_VALUES =
    {
        Item.Type.SECOND,
        Item.Type.REGULAR,
        Item.Type.SALE,
        Item.Type.DISCOUNT
    };

    private static final int[][] D_VALUES =
    {
        {  0 },
        {  0,  0,  0,  0},
        { 90 },
        { 10, 10, 20, 20}
    };

    @Parameterized.Parameters
    @SuppressWarnings("unchecked")
    public static Collection getTypeQuantityPairs()
    {
        Collection pairs = new ArrayList();
        for (int q = 0; q < Q_VALUES.length; q++)
            for (int t = 0; t < T_VALUES.length; t++)
                pairs.add(new Object[] {
                    T_VALUES[t],
                    Q_VALUES[q],
                    (D_VALUES[t].length > q)
                        ? D_VALUES[t][q]
                        : D_VALUES[t][D_VALUES[t].length - 1]
                });
        return pairs;
    }

    private Item.Type _type;
    private int _quantity;
    private int _discount;

    public DiscountTest(Item.Type type, int quantity, int discount){
        _type = type;
        _quantity = quantity;
        _discount = discount;
    }

    @Test
    public void discountTest(){
        assertEquals("type: "+ _type + ", quantity: " + _quantity,
                _discount,
                ShoppingCart.calculateDiscount(
                        new Item("Title", 5.99f, _type, _quantity)
                ),
                0.01f
        );
    }
}
```

1) Значення _quantity з масиву {1, 9, 10, 19} перевіряє граничні умови

а) більше одного;

б) повний десяток.

2) Значення _type з масиву { SECOND, REGULAR, SALE, DISCOUNT} перевіряє всі можливі значення типу товару.

3) Значення _discount містить для кожного типу товару очікувані значення знижок при різних його кількостях.

Нижче приведено програмний код класу ShoppingCart.

```
import java.util.*;
import java.text.*;

/**
 * Containing items and calculating price.
 */
public class ShoppingCart
{
    /**
     * Tests all class methods.
     */
    public static void main(String[] args)
    {
        // TODO: add tests here
        ShoppingCart cart = new ShoppingCart();
        cart.addItem("Apple", 0.99, 5, Item.Type.REGULAR);
        cart.addItem("Banana", 20.00, 4, Item.Type.DISCOUNT);
        cart.addItem("A long piece of toilet paper", 17.20, 1, Item.Type.SALE);
        cart.addItem("Nails", 2.00, 500, Item.Type.REGULAR);
        System.out.println(cart.toString());
    }

    /**
     * Adds new item.
     *
     * @param title     item title 1 to 32 symbols
     * @param price     item ptice in cents, > 0, < 1000
     * @param quantity  item quantity, from 1 to 1000
     * @param type      item type, on enum Item.Type
     *
     * @throws IndexOutOfBoundsException if total items added over 99
     * @throws IllegalArgumentException if some value is wrong
     */
    public void addItem(String title, double price, int quantity, Item.Type type)
    {
        if (title == null || title.length() == 0 || title.length() > 32)
            throw new IllegalArgumentException("Illegal title");

        if (price < 0.01 || price >= 1000.00)
            throw new IllegalArgumentException("Illegal price");

        if (quantity <= 0 || quantity > 1000)
            throw new IllegalArgumentException("Illegal quantity");

        if (items.size() == 99)
            throw new IndexOutOfBoundsException("No more space in cart");

        Item item = new Item();
        item.title    = title;
        item.price    = price;
        item.quantity = quantity;
        item.type     = type;

        items.add(item);
    }

    /**
     * Formats shopping price.
     *
     * @return  string as lines, separated with \n,
     *          first line:   # Item                    Price Quan. Discount      Total
     *          second line: ------------------------------------------------------------
     *          next lines:  NN Title                 $PP.PP    Q       DD%     $TT.TT
     *                        1 Some title              $.30    2         -       $.60
     *                        2 Some very long ti... $100.00    1       50%     $50.00
     *                       ...
     *                       31 Item 42              $999.00 1000         - $999000.00
     *          end line:    ------------------------------------------------------------
     *          last line:   31                                             $999050.60
     *
     *          Item title is trimmed to 20 chars adding '...'
     *
```

```
     *          if no items in cart returns "No items." string.
     */
    public String toString()
    {
        StringBuffer sb = new StringBuffer();
        if (items.size() == 0)
            return "No items.";

        double total = 0.00;

        sb.append(" # Item                   Price Quan. Discount      Total\n");
        sb.append("-------------------------------------------------------\n");

        for (int i = 0; i < items.size(); i++) {
            Item item = (Item) items.get(i);
            int discount = calculateDiscount(item);
            double itemTotal = item.price * item.quantity * (100.00 - discount) / 100.00;

            appendPaddedRight(sb, String.valueOf(i + 1), 2);
            sb.append(" ");
            appendPaddedLeft(sb, item.title, 20);
            sb.append(" ");
            appendPaddedRight(sb, MONEY.format(item.price), 7);
            sb.append(" ");
            appendPaddedRight(sb, String.valueOf(item.quantity), 4);
            sb.append("  ");
            if (discount == 0)
                sb.append("       -");
            else {
                appendPaddedRight(sb, String.valueOf(discount), 7);
                sb.append("%");
            }
            sb.append(" ");
            appendPaddedRight(sb, MONEY.format(itemTotal), 10);
            sb.append("\n");

            total += itemTotal;
        }
        sb.append("-------------------------------------------------------\n");
        appendPaddedRight(sb, String.valueOf(items.size()), 2);
        sb.append("                                          ");
        appendPaddedRight(sb, MONEY.format(total), 10);

        return sb.toString();
    }

    // --- private section ----------------------------------------------------

    private static final NumberFormat MONEY;
    static {
        DecimalFormatSymbols symbols = new DecimalFormatSymbols();
        symbols.setDecimalSeparator('.');
        MONEY = new DecimalFormat("$#.00", symbols);
    }

    /**
     * Adds to string buffer given string, padded with spaces.
     * @return "    str".length() == width
     */
    private static void appendPaddedRight(StringBuffer sb, String str, int width)
    {
        for (int i = str.length(); i < width; i++)
            sb.append(" ");
        sb.append(str);
    }

    /**
     * Adds string to buffer, wills spaces to width.
     * If string is longer than width it is trimmed and ends with '...'
     */
    private static void appendPaddedLeft(StringBuffer sb, String str, int width)
    {
        if (str.length() > width) {
            sb.append(str.substring(0, width-3));
            sb.append("...");
        }
        else {
```

```
            sb.append(str);
            for (int i = str.length(); i < width; i++)
                sb.append(" ");
        }
    }

    /**
     * Calculates item's discount.
     * For Item.Type.REGULAR discount is 0%;
     * For Item.Type.SECOND discount is 50% if quantity > 1
     * For Item.Type.DISCOUNT discount is 10% + 10% for each full 10 items, but not more than 50%
     *     total
     * For Item.Type.SALE discount is 90%
     * For each full 100 items item gets additional 10%, but not more than 80% total
     */
    private static int calculateDiscount(Item item)
    {
        int discount = 0;
        switch (item.type) {
            case Item.Type.SECOND:
                if (item.quantity > 1)
                    discount = 50;
                break;

            case Item.Type.DISCOUNT:
                discount = 10 + item.quantity / 10 * 10;
                if (discount > 50)
                    discount = 50;
                break;

            case Item.Type.SALE:
                discount = 90;
        }
        discount += item.quantity / 100 * 10;
        if (discount > 80)
            discount = 80;

        return discount;
    }

    /** Container for added items */
    private List items = new ArrayList();
}

public class Item {
    public enum Type { SECOND, REGULAR, SALE, DISCOUNT };
    public String title;
    public float  price;
    public int quantity;
    public Type type;

    public Item(String title, float price, Item.Type type, int quantity){
        this.title = title;
        this.price = price;
        this.type = type;
        this.quantity = quantity;
    }
}
```

ПРАКТИЧНЕ ЗАВДАННЯ №2

Сформувати для заданого коду модульні тести (з використання JUnit), покрити ними наступні методи (вимоги описані в коментарях до методів):

ShoppingCart.appendFormatted(StringBuilder,String,int,int)
ShoppingCart.calculateDiscount(ItemType,int)

Провести рефакторинг коду, дописуючи тести в разі необхідності з метою зменшення імовірності появи помилок редагування. Варто покривати тестами метод, перед його розбиванні на складові частини.

Рефакторинг необхідно проводити поетапно, після успішного застосування кожного методу рефакторинга (наприклад, виділення методу чи заміни локальної змінної викликом метода) необхідно робити копію папки з програмними кодами (створіть папку commits, куди будите копіювати папку з програмними кодами, переіменовуючи її в 0001, 0002, і т.д.). Рефакторинг повинен супроводжуватися пояснювальною запискою, яка описує зміни, які відбуваються з кодом. В цій записці необхідно вказувати на недоліки коду, способи усунення цих недоліків та ім'я папки, в якій міститься. Наприклад:

1. Занадто великий метод ShoppingCart.formatTicket(), оскільки він виконує відразу декілька функцій (*вказати яких*), виділяємо з цього методу код, що виконує пов'язані підфункції і виносимо в окремі методи. Результат в папці 0028 (це *означає, що виправлення цього недоліку займає 28 кроків*).
2. …

Програмний код приведений нижче.

```
import java.util.*;
import java.text.*;

/**
 * Containing items and calculating price.
 */
public class ShoppingCart
{
    public static enum ItemType { NEW, REGULAR, SECOND_FREE, SALE };

    /**
     * Tests all class methods.
     */
    public static void main(String[] args)
    {
        // TODO: add tests here
        ShoppingCart cart = new ShoppingCart();
        cart.addItem("Apple", 0.99, 5, ItemType.NEW);
        cart.addItem("Banana", 20.00, 4, ItemType.SECOND_FREE);
        cart.addItem("A long piece of toilet paper", 17.20, 1, ItemType.SALE);
        cart.addItem("Nails", 2.00, 500, ItemType.REGULAR);
        System.out.println(cart.formatTicket());
    }

    /**
     * Adds new item.
     *
     * @param title     item title 1 to 32 symbols
     * @param price     item ptice in USD, > 0
     * @param quantity  item quantity, from 1
     * @param type      item type
     *
     * @throws IllegalArgumentException if some value is wrong
     */
    public void addItem(String title, double price, int quantity, ItemType type)
```

```
{
    if (title == null || title.length() == 0 || title.length() > 32)
        throw new IllegalArgumentException("Illegal title");

    if (price < 0.01)
        throw new IllegalArgumentException("Illegal price");

    if (quantity <= 0)
        throw new IllegalArgumentException("Illegal quantity");

    Item item = new Item();
    item.title    = title;
    item.price    = price;
    item.quantity = quantity;
    item.type     = type;

    items.add(item);
}

/**
 * Formats shopping price.
 *
 * @return  string as lines, separated with \n,
 *          first line:   # Item                    Price Quan. Discount      Total
 *          second line: ------------------------------------------------------------
 *          next lines:  NN Title                 $PP.PP    Q       DD%     $TT.TT
 *                        1 Some title              $.30    2         -       $.60
 *                        2 Some very long       $100.00    1       50%     $50.00
 *                       ...
 *                       31 Item 42              $999.00 1000         - $999000.00
 *          end line:    ------------------------------------------------------------
 *          last line:   31                                            $999050.60
 *
 *          if no items in cart returns "No items." string.
 */
public String formatTicket()
{
    if (items.size() == 0)
        return "No items.";

    List<String[]> lines = new ArrayList<String[]>();

    String[] header = {"#","Item","Price","Quan.","Discount","Total"};
    int[] align = new int[] { 1, -1, 1, 1, 1, 1 };

    // formatting each line
    double total = 0.00;
    int    index = 0;
    for (Item item : items) {
        int discount = calculateDiscount(item.type, item.quantity);
        double itemTotal = item.price * item.quantity * (100.00 - discount) / 100.00;

        lines.add(new String[]{
            String.valueOf(++index),
            item.title,
            MONEY.format(item.price),
            String.valueOf(item.quantity),
            (discount == 0) ? "-" : (String.valueOf(discount) + "%"),
            MONEY.format(itemTotal)
        });
        total += itemTotal;
    }
    String[] footer = { String.valueOf(index),"","","","",
        MONEY.format(total) };

    // formatting table

    // column max length
    int[] width = new int[]{0,0,0,0,0,0};
    for (String[] line : lines)
        for (int i = 0; i < line.length; i++)
            width[i] = (int) Math.max(width[i], line[i].length());
    for (int i = 0; i < header.length; i++)
        width[i] = (int) Math.max(width[i], header[i].length());
    for (int i = 0; i < footer.length; i++)
        width[i] = (int) Math.max(width[i], footer[i].length());
```

```
    // line length
    int lineLength = width.length - 1;
    for (int w : width)
        lineLength += w;

    StringBuilder sb = new StringBuilder();

    // header
    for (int i = 0; i < header.length; i++)
        appendFormatted(sb, header[i], align[i], width[i]);
    sb.append("\n");

    // separator
    for (int i = 0; i < lineLength; i++)
        sb.append("-");
    sb.append("\n");

    // lines
    for (String[] line : lines) {
        for (int i = 0; i < line.length; i++)
            appendFormatted(sb, line[i], align[i], width[i]);
        sb.append("\n");
    }
    if (lines.size() > 0) {
        // separator
        for (int i = 0; i < lineLength; i++)
            sb.append("-");
        sb.append("\n");
    }

    // footer
    for (int i = 0; i < footer.length; i++)
        appendFormatted(sb, footer[i], align[i], width[i]);

    return sb.toString();
}

// --- private section -------------------------------------------------------

private static final NumberFormat MONEY;
static {
    DecimalFormatSymbols symbols = new DecimalFormatSymbols();
    symbols.setDecimalSeparator('.');
    MONEY = new DecimalFormat("$#.00", symbols);
}

/**
 * Appends to sb formatted value.
 * Trims string if its length > width.
 * @param align -1 for align left, 0 for center and +1 for align right.
 */
public static void appendFormatted(StringBuilder sb, String value, int align, int width)
{
    if (value.length() > width)
        value = value.substring(0,width);
    int before = (align == 0)
        ? (width - value.length()) / 2
        : (align == -1) ? 0 : width - value.length();
    int after = width - value.length() - before;
    while (before-- > 0)
        sb.append(" ");
    sb.append(value);
    while (after-- > 0)
        sb.append(" ");
    sb.append(" ");
}

/**
 * Calculates item's discount.
 * For NEW item discount is 0%;
 * For SECOND_FREE item discount is 50% if quantity > 1
 * For SALE item discount is 70%
 * For each full 10 not NEW items item gets additional 1% discount,
 * but not more than 80% total
 */
public static int calculateDiscount(ItemType type, int quantity)
{
```

```
        int discount = 0;
        switch (type) {
            case NEW:
                return 0;

            case REGULAR:
                discount = 0;
                break;

            case SECOND_FREE:
                if (quantity > 1)
                    discount = 50;
                break;

            case SALE:
                discount = 70;
                break;
        }
        if (discount < 80) {
            discount += quantity / 10;
            if (discount > 80)
                discount = 80;
        }

        return discount;
    }

    /** item info */
    private static class Item
    {
        String   title;
        double   price;
        int      quantity;
        ItemType type;
    }

    /** Container for added items */
    private List<Item> items = new ArrayList<Item>();
}
```