ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

"ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ"

Факультет ИСП

Кафедра ПИ им Л.П. Фельдмана

Лабораторная работа №4

по курсу: «Профессиональная практика программной инженерии»

по теме: «Создание самодокументирующегося кода»

Выполнил:

ст. гр. ПИ-19б

Решетняк Я. А.

Приняла:

ассистент каф. ПИ

Дмитрюк Т.Г.

ДОНЕЦК – 2023

*Лабораторная работа 4*

**Создание самодокументирующегося кода**

**Цель работы:** научиться добавлять в программный код специальным образом оформление докблок-комментарии, для последующей автоматической генерации API reference, а также познакомиться с форматом оформления документации DocBook.

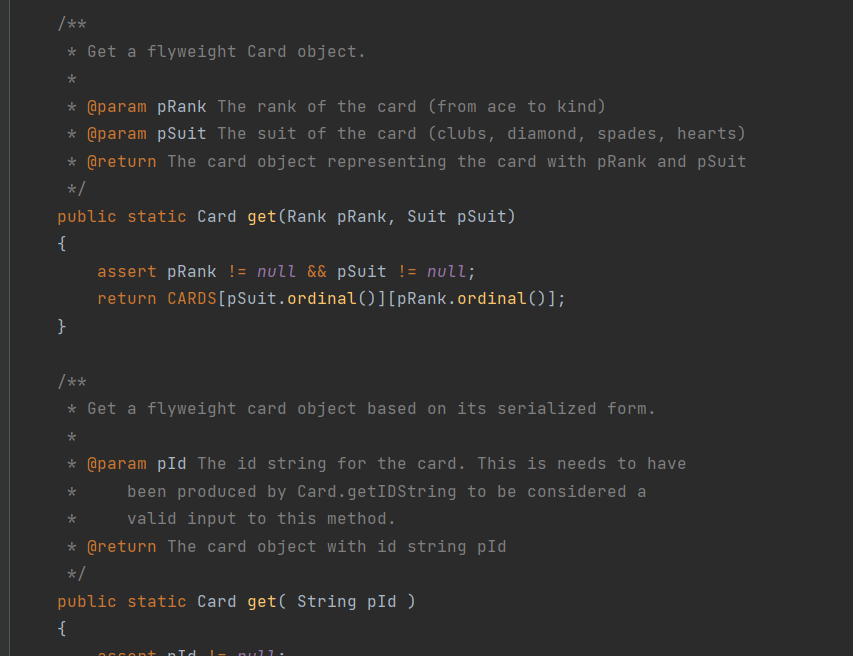
**Задание к лабораторной работе**

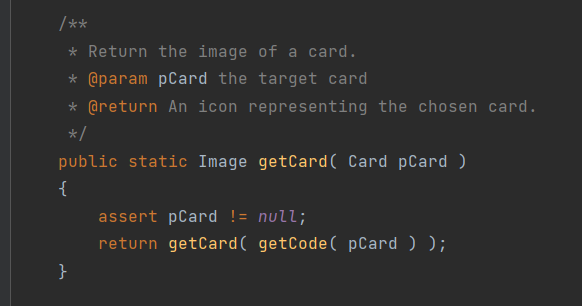
Внедрить в ранее разработанный программный код докблоки.

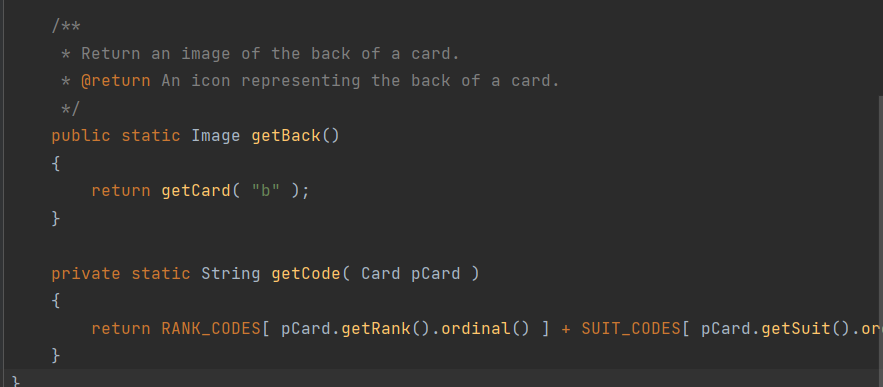
Разработать DocBook и проверить корректность созданного DocBook файла с помощью средств редактора или одного из онлайн валидаторов.

**Порядок выполнения работы**

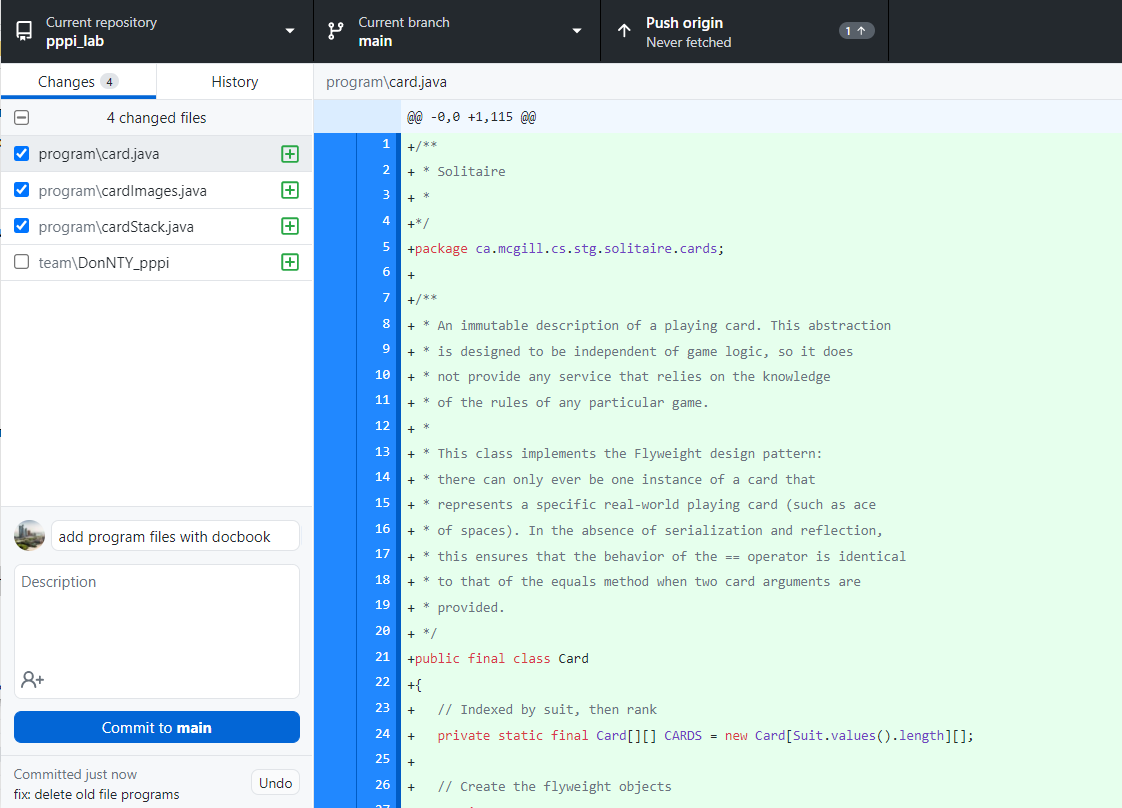
1. Во всех файлах с исходным кодом добавить докблоки.







2. Закоммитить изменения в репозиторий.



3. Содержимое docbook.xml

<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.5//EN"

"http://www.oasis-open.org/docbook/xml/4.5/docbookx.dtd">

<book>

<bookinfo>

<title>Пасьянс</title>

<author>

<firstname>Yaroslav</firstname>

<surname>Reshetnyak</surname>

<affiliation>

<orgname>yar-resh@yandex.com</orgname>

</affiliation>

</author>

<pubdate>10-03-2023</pubdate>

</bookinfo>

<chapter>

<title>Документация к игре Пасьянс</title>

<section>

<title>Общее описание</title>

<para>

<emphasis>Пасьянс с возможнотью игры по сети</emphasis>

</para>

</section>

</chapter>

<chapter>

<title>Работа с картами</title>

<section>

<title>Создание карт</title>

<mediaobject>

<textobject>

<phrase>Вариант использования №1 – создание карт при начале игры</phrase>

</textobject>

</mediaobject>

<programlisting>

public final class Card

{

// Indexed by suit, then rank

private static final Card[][] CARDS = new Card[Suit.values().length][];

// Create the flyweight objects

static

{

for( Suit suit : Suit.values() )

{

CARDS[suit.ordinal()] = new Card[Rank.values().length];

for( Rank rank : Rank.values() )

{

CARDS[suit.ordinal()][rank.ordinal()] = new Card(rank, suit);

}

}

}

private final Rank aRank;

private final Suit aSuit;

private Card(Rank pRank, Suit pSuit )

{

aRank = pRank;

aSuit = pSuit;

}

/\*\*

\* Get a flyweight Card object.

\*

\* @param pRank The rank of the card (from ace to kind)

\* @param pSuit The suit of the card (clubs, diamond, spades, hearts)

\* @return The card object representing the card with pRank and pSuit

\*/

public static Card get(Rank pRank, Suit pSuit)

{

assert pRank != null &amp;&amp; pSuit != null;

return CARDS[pSuit.ordinal()][pRank.ordinal()];

}

/\*\*

\* Get a flyweight card object based on its serialized form.

\*

\* @param pId The id string for the card. This is needs to have

\* been produced by Card.getIDString to be considered a

\* valid input to this method.

\* @return The card object with id string pId

\*/

public static Card get( String pId )

{

assert pId != null;

int id = Integer.parseInt(pId);

return get(Rank.values()[id % Rank.values().length],

Suit.values()[id / Rank.values().length]);

}

/\*\*

\* Obtain the rank of the card.

\* @return An object representing the rank of the card.

\*/

public Rank getRank()

{

return aRank;

}

/\*\*

\* Return the id string for this card.

\*

\* @return A string uniquely representing this card. The string

\* format is not specified except that it is fully compatible

\* with the format expected by Card.get(String).

\*/

public String getIDString()

{

return Integer.toString(getSuit().ordinal() \* Rank.values().length + getRank().ordinal());

}

/\*\*

\* Obtain the suit of the card.

\* @return An object representing the suit of the card

\*/

public Suit getSuit()

{

return aSuit;

}

/\*\*

\* @see java.lang.Object#toString()

\*/

@Override

public String toString()

{

return aRank + " of " + aSuit;

}

}

</programlisting>

<para>Изображение карт</para>

<mediaobject>

<textobject>

<phrase>Вариант использования №2 – отображение изображений карт</phrase>

</textobject>

</mediaobject>

<programlisting>

package ca.mcgill.cs.stg.solitaire.cards;

import java.util.HashMap;

import java.util.Map;

import javafx.scene.image.Image;

/\*\*

\* A class to store and manage images of the 52 cards.

\*/

public final class CardImages

{

private static final String IMAGE\_LOCATION = "";

private static final String IMAGE\_SUFFIX = ".gif";

private static final String[] RANK\_CODES = {"a", "2", "3", "4", "5", "6", "7", "8", "9", "t", "j", "q", "k"};

private static final String[] SUIT\_CODES = {"c", "d", "h", "s"};

private static Map&lt;String, Image&gt; aCards = new HashMap&lt;String, Image&gt;();

private CardImages()

{}

/\*\*

\* Return the image of a card.

\* @param pCard the target card

\* @return An icon representing the chosen card.

\*/

public static Image getCard( Card pCard )

{

assert pCard != null;

return getCard( getCode( pCard ) );

}

private static Image getCard( String pCode )

{

Image image = (Image) aCards.get( pCode );

if( image == null )

{

image = new Image(CardImages.class.getClassLoader().getResourceAsStream( IMAGE\_LOCATION + pCode + IMAGE\_SUFFIX ));

aCards.put( pCode, image );

}

return image;

}

/\*\*

\* Return an image of the back of a card.

\* @return An icon representing the back of a card.

\*/

public static Image getBack()

{

return getCard( "b" );

}

private static String getCode( Card pCard )

{

return RANK\_CODES[ pCard.getRank().ordinal() ] + SUIT\_CODES[ pCard.getSuit().ordinal() ];

}

}

</programlisting>

</section>

<section>

<title>Создание стека карт</title>

<mediaobject>

<textobject>

<phrase>Вариант использования №3 – сложить карты в стопку</phrase>

</textobject>

</mediaobject>

<programlisting>

package ca.mcgill.cs.stg.solitaire.cards;

import java.util.ArrayList;

import java.util.Iterator;

import java.util.List;

/\*\*

\* Represents a general-purpose stack of cards. New CardStack

\* instances are initially empty.

\*/

public class CardStack implements Iterable

&lt;Card&gt;

{

private final List

&lt;Card&gt; aCards;

/\*\*

\* Creates an empty CardStack.

\*/

public CardStack()

{

aCards = new ArrayList&lt;&gt;();

}

/\*\*

\* Creates a CardStack that contains all the cards

\* in pCard, in the iteration order, from bottom to top.

\*

\* @param pCards The cards to initialize the stack with.

\*/

public CardStack(Iterable

&lt;Card&gt; pCards)

{

this();

for( Card card : pCards )

{

aCards.add(card);

}

}

/\*\*

\* Pushes pCard onto the stack.

\*

\* @param pCard The card to push.

\* @pre pCard != null;

\* @pre !aCards.contains(pCard)

\*/

public void push(Card pCard)

{

assert pCard != null &amp;&amp; !aCards.contains(pCard);

aCards.add(pCard);

}

/\*\*

\* Removes the card on top of the stack and returns it.

\*

\* @return The card on top of the stack.

\* @pre !isEmpty()

\*/

public Card pop()

{

assert !isEmpty();

return aCards.remove(aCards.size()-1);

}

/\*\*

\* @return The card at the top of the stack.

\* @pre !isEmpty();

\*/

public Card peek()

{

assert !isEmpty();

return aCards.get(aCards.size()-1);

}

/\*\*

\* @param pIndex The index to peek in the stack.

\* @return The card at the position indicated by pIndex

\* @pre pIndex &gt;= 0 &amp;&amp; pIndex &lt; size();

\*/

public Card peek(int pIndex)

{

assert pIndex &gt;= 0 &amp;&amp; pIndex &lt; size();

return aCards.get(pIndex);

}

/\*\*

\* @return The number of cards in the stack.

\*/

public int size()

{

return aCards.size();

}

/\*\*

\* Removes all the cards in the stack.

\*/

public void clear()

{

aCards.clear();

}

/\*\*

\* @return True if and only if the stack has no cards in it.

\*/

public boolean isEmpty()

{

return aCards.size() == 0;

}

@Override

public Iterator&lt;Card&gt; iterator()

{

return aCards.iterator();

}

}

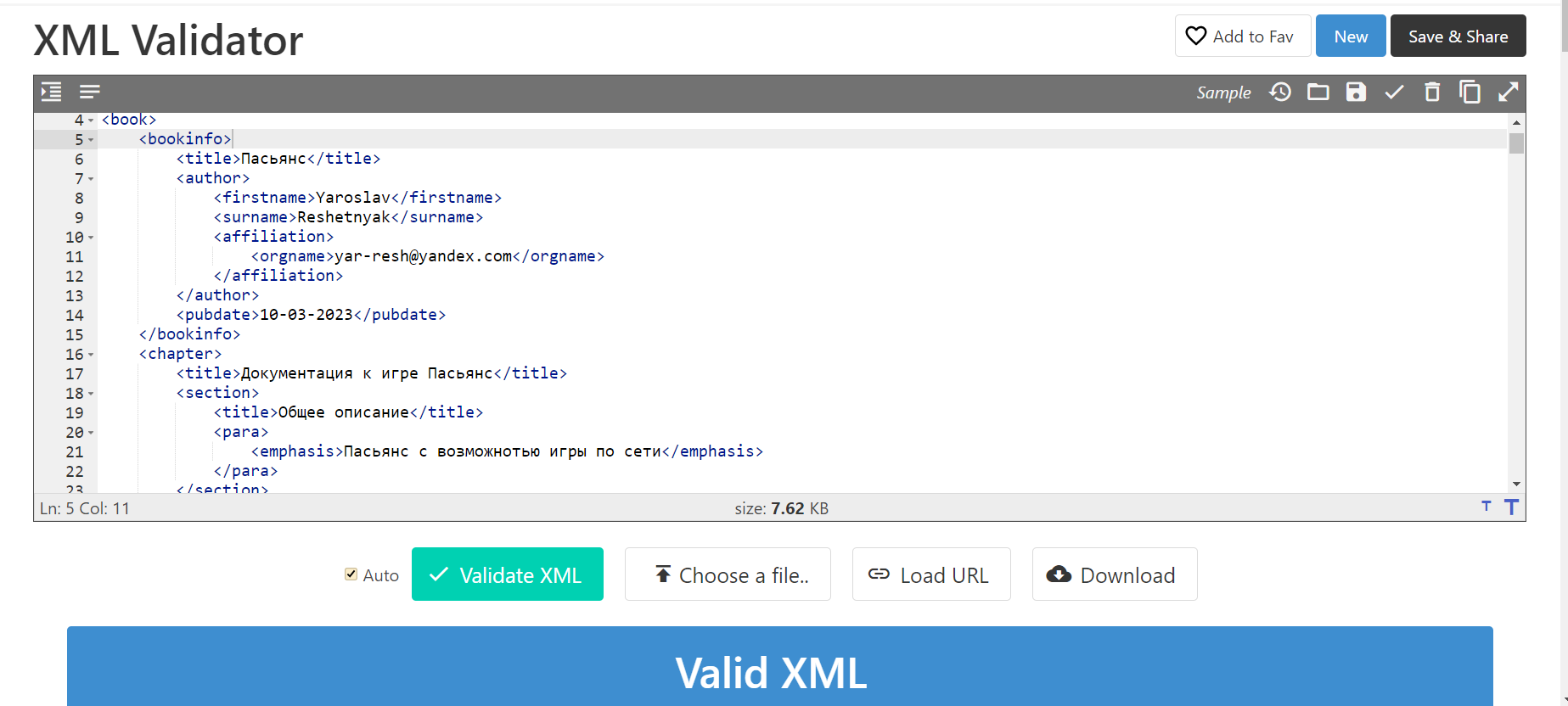
</programlisting>

</section>

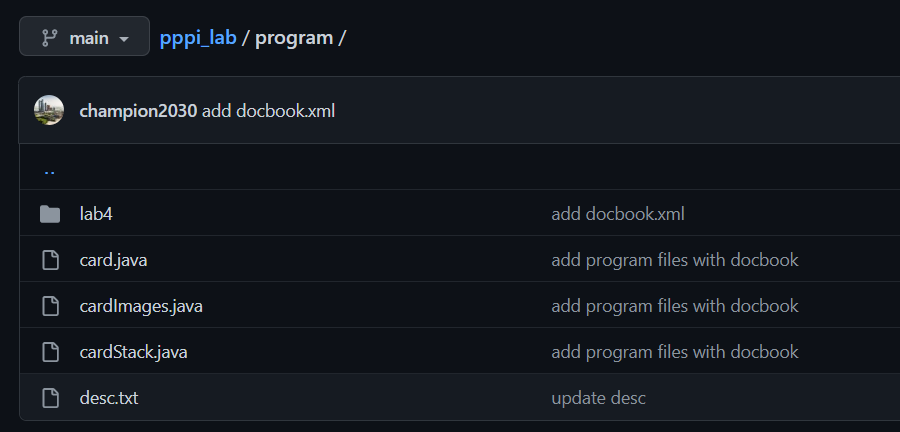
</chapter>

</book>

4. Проверить корректность созданного DocBook файла с помощью средств редактора или одного из онлайн валидаторов.



Xml валиден.



**Выводы:** в ходе лабораторной работы мы научились добавлять в программный код докблок-коментарии, а также познакомиться с форматом оформления документации DocBook.