ПРАКТИЧНА РОБОТА 1-2

ЖИТТЕВИЙ ЦИКЛ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА МЕТОДОЛОГІЇ РОЗРОБКИ.

1 Теоретична частина

1.1 Стиль документації Java

Комплект розробки Java (Java Development Kit) (JDK) поставляється з інструментом під назвою JavaDoc. Цей інструмент буде генерувати документацію для вихідного коду Java з коментарями, написаними відповідно до стилю документації Java. Нижче наведені посилання на корисні WEB-сторінки в JavaDoc:

1) http://docs.oracle.com/javase/8/docs/technotes/tools/windows/javadoc.html (***lab1/javadoc.pdf***);

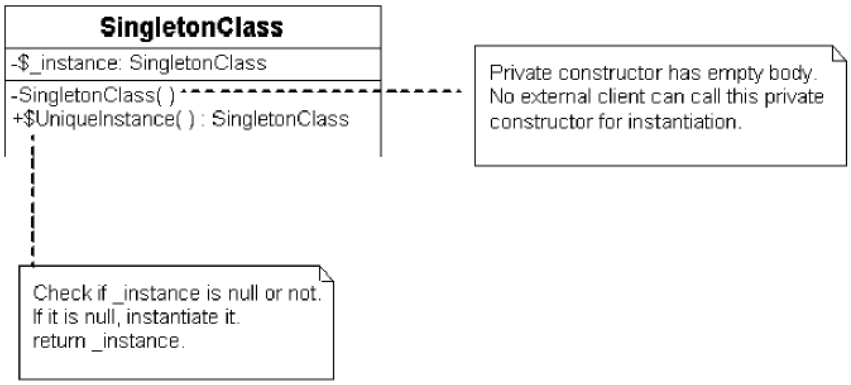
2) http://www.oracle.com/technetwork/articles/java/index-137868.html (***lab1/writingDocComments.pdf***);

3) http://www.oracle.com/technetwork/java/codeconventions-150003.pdf, http://lars-lab.jpl.nasa.gov/jpl-java-standard.pdf (***lab1/JavaCodeConventions.pdf, lab1/JavaCodingStandart.pdf***).

Створіть каталог lab_1/src/ для цієї лабораторної роботи та скопіюйте файл lab1/ *.java у новий каталог. Там повинно бути 3 вихідні файли java: FunList.java, Empty.java та Cons.java.

1.2 Singleton Pattern

Клас Empty являє собою "порожній список". Концептуально, в цьому світі є лише один порожній список. Поняття близьке до порожнього набору (empty set): існує лише один порожній набір. Як ми можемо гарантувати, що протягом усього життя програми можна створити лише один екземпляр Empty? Існує можливість розробки класу для забезпечення такої властивості. Вона називається шаблоном дизайну Singleton. Наступна UML-діаграма описує шаблон.




```
/**
* Singleton pattern in Java.
*/
```

```
public class SingletonClass {
        private static SingletonClass _instance = new SingletonClass();
        /**
        * Accessible only from within this class.
        * Body is blank because there is nothing to initialize.
        */
        private SingletonClass(){}
        public static SingletonClass UniqueInstance(){
            return _instance; // static methods can access static fields.
   }
}
```

Зверніть увагу, що поле _instance та метод UniqueInstance () статичні.
Нижче наведено посилання на корисні джерела:
1) http://www.buyya.com/254/Patterns/Singleton-2pp.pdf
(***lab1/Singleton-2pp.pdf***)
2) https://en.wikipedia.org/wiki/Singleton_pattern
(***lab1/Singleton pattern - Wikipedia.pdf***)

2 Завдання

2.1. Додати коментарі до існуючих файлів та запустити утиліту javadoc.
Використовуйте редактор Notepad ++ для відображення та зміни коду Java.

2.1.1 Додайте коментарі до всіх класів, полів та методів в FunList.java, Cons.java та Empty.java, згідно JavaDoc. Для кожного класу використовуйте тег **@author** та тег **@since**. Для кожного методу функції використовуйте тег **@return**. Для методів та конструкторів, що мають параметри, використовуйте тег **@param**.

2.1.2 Тепер ви готові запустити утиліту javadoc. Виконайте команду:

javadoc * .java.

Цю команду можна записати у командному рядку або у bat файлі.
Будуть створені відповідні HTML-файли.
Які створені HTML-файли? Використовуйте браузер, щоб переглянути їх. Чи відображаються приватні поля та методи? Тепер виконайте команду:

javadoc -private * .java.

Ви бачите різницю?

2.1.3 Виконати команду:

javadoc

Ви повинні побачити короткий опис використання javadoc. Коротко погляньте на пояснення прапорів.

2.2. Впровадьте модель Singleton і перевірте її.

2.2.1 Впровадьте шаблону Singleton для класу Empty (рішення).

Empty.UniqueInstance ()

тепер являє собою унікальний порожній список.

2.2.2 Додайте другий конструктор

Cons(int i)

до класу Cons, який приймає єдиний int як аргумент і створює FunList, що містить цей елемент і порожній список як його хвіст. Цей конструктор повинен використовувати статичний метод Empty.UniqueInstance (). Оскільки конструктор класу Empty є приватним, для Cons (int i) не існує способу викликати new Empty().

2.2.3 Додати метод

FunList append(FunList other)

до класу FunList (і всіх його варіантів), які повертають FunList, що містить елементи цього, а потім елементи іншого (o the class FunList (and all of its variants) that returns a FunList containing the elements of this followed by elements of other.). Новий метод не повинен змінювати це і повинен використовувати статичний метод Empty.UniqueInstance ().

2.2.4 Створіть клас AppendTest з main, щоб протестувати свій новий конструктор singleton і добавлений метод.

Компіляція за допомогою команди

javac AppendTest.java

Запустити за допомогою команди

java AppendTest

2.2.5 Додайте метод

FunList insertInOrder (int i)

до класу FunList (і всіх його варіантів), що задовольняє наступній специфікації. Передумова (Pre-condition): цей FunList відсортовано в non-descending порядку. Post-conditon: FunList, що містить елементи **this *FunList***, і «i» вставлений у відповідному порядку повертається.

Новий метод не повинен змінювати **this *FunList***.

2.2.6 Створіть клас InOrderTest з main для перевірки вашого нового методу insertInOrder.

2.2.7 Додайте метод ***FunList sort()*** до класу FunList (і всіх його варіантів), який повертає список у відсортованому (non-descending) порядку, який містить ті самі елементи, що і this. Підказка: скористайтеся ***insertInOrder()***..

2.2.8 Створіть клас ***SortTest*** з main, щоб перевірити свій новий метод сортування.

2.3 Збережіть screen shots, що містять результати та відповідний код.

```
/**
* Mimics functional lists.
* @author Moskalenko Vyacheslav
```

```
* @since JDK1.4
*/
public abstract class FunList {
    /**
    * @return the first int in the list object
    */
    public abstract int car();
    /**
    * @return the tail (all but the first element) of the list object
    */
    public abstract FunList cdr();
    /**
    * NOTE: toString () method is NOT abstract. It calls, toStringHelp() , an abstract method.
    * It represents what we call an "invariant" behavior for <code>FunList</code>.
    * It is an example of the "Template Method Pattern". A "template method" is a method that
    * makes calls to at least one abstract method in its own class.
    */
    public String toString(){
        return "(" + toStringHelp() + " ) ";
    }
    /**
    * @return a String description of the list object
    */
    abstract String toStringHelp();
}
```

```
/**
* Mimics fundamental function cons for constructs
* memory objects which hold two values or pointers to values
* @author Vyacheslav Moskalenko
* @since JDK1.4
*/
public class Cons extends FunList {
    private int _dat;
    private FunList _cdr;

    public Cons(int carDat, FunList cdr){
        _dat = carDat;
        _cdr = cdr;
    }
    /**
    * @param i a left side (head) of the list.
    */
    public Cons(int i){
        _dat = i;
        _cdr = new Empty();
    }
    /**
    * @return the first int in the list object
    */
    public int car(){
        return _dat;
    }
    /**
    * @return the tail (all but the first element) of the list object
    */
    public FunList cdr(){
        return _cdr;
    }
    /**
    * @return a String description of the list object
    */
    String toStringHelp(){
        return " " + _dat + _cdr.toStringHelp();
    }
}
```

```
/**
* Mimics functional empty list.
* @author Vyacheslav Moskalenko
* @since JDK1.4
*/
public class Empty extends FunList{
   public Empty(){}

   public int car(){
      throw new java.util.NoSuchElementException("car requires a non Empty Funlist");
   }

   public FunList cdr(){
      throw new java.util.NoSuchElementException("cdr requires a non Empty Funlist");
   }

   String toStringHelp(){
      return "";
   }
}
```