Университет ИТМО

Факультет ПИиКТ

Кафедра ВТ

Лабораторная работа №1

По дисциплине

«Тестирование программного обеспечения»

Выполнил:

Студент 4-го курса

Группа Р3400

Толстов Д.Д

Преподаватель:

Харитонова А. Е.

Санкт - Петербург

2020 год

# Изображение выглядит как текст Автоматически созданное описаниеЗадание

# Выполнение

## Функция arcsin(x)

Код функции:

***public class*** Taylor *{* ***public static final double*** *eps* = 1E-9;  
 ***public static double*** arcsin*(****double*** x*) {* ***double*** curr = x;  
 ***double*** result = 0.0;  
 ***int*** n = 1;  
 ***if*** *(*x == 1.0*) {* ***return*** Math.*PI*/2;  
 *}* ***else if*** *(*x == -1.0*) {* ***return*** -Math.*PI*/2;  
 *}* ***else if*** *(*Math.*abs(*x*)* < 1*) {* ***while*** *(*Math.*abs(*curr*)* >= *eps*/10*) {* result += curr;  
 curr = curr \* x \* x \* *(*2 \* n - 1*)* \* *(*2 \* n - 1*)* / *((*2 \* n*)* \* *(*2 \* n + 1*))*;  
 n++;  
 *}  
 }* ***else*** *{* ***throw new*** IllegalArgumentException*()*;  
 *}* ***return*** result;  
 *}  
}*

Тестирование:

***public class*** TaylorTest *{* @Test  
 ***public void*** LowBorderReturnsCorrect*() {  
 assertEquals(*-Math.*PI*/2, Taylor.*arcsin(*-1*))*;  
 *}* @Test  
 ***public void*** HighBorderReturnsCorrect*() {  
 assertEquals(*Math.*PI*/2, Taylor.*arcsin(*1*))*;  
 *}* @Test  
 ***public void*** RightOutOfBoundsThrowsException*() {  
 assertThrows(*IllegalArgumentException.***class***, *()* -> Taylor.*arcsin(*2*))*;  
 *}* @Test  
 ***public void*** LeftOutOfBoundsThrowsException*() {  
 assertThrows(*IllegalArgumentException.***class***, *()* -> Taylor.*arcsin(*-2*))*;  
 *}* @Test  
 ***public void*** ReturnsCorrect*() {  
 assertEquals(*Math.*asin(*0.33*)*, Taylor.*arcsin(*0.33*)*, Taylor.*eps)*;  
 *assertEquals(*Math.*asin(*0.88*)*, Taylor.*arcsin(*0.88*)*, Taylor.*eps)*;  
 *assertEquals(*Math.*asin(*-0.33*)*, Taylor.*arcsin(*-0.33*)*, Taylor.*eps)*;  
 *assertEquals(*Math.*asin(*-0.88*)*, Taylor.*arcsin(*-0.88*)*, Taylor.*eps)*;  
 *}* @Test  
 ***public void*** YAxisInterceptionReturnsCorrect*() {  
 assertEquals(*Math.*asin(*0*)*, Taylor.*arcsin(*0*)*, Taylor.*eps)*;  
 *}  
  
}*

## 2. Б-Дерево

Код реализации Б-Дерева

***public class*** BTree*<*Key ***extends*** Comparable*<*Key*>*, Value*> {* ***private static final int*** *M* = 4;  
  
 ***private*** Node root;  
 ***private int*** height;  
 ***private int*** elemsCount;  
 ***public*** List*<*BTreeActions*>* actionsLog = ***new*** LinkedList*<>()*;  
  
 ***private static final class*** Node *{* ***private int*** childrenCount;  
 ***private*** Entry*[]* children = ***new*** Entry*[M]*;  
  
 ***private*** Node*(****int*** k*) {* childrenCount = k;  
 *}  
 }* ***private static class*** Entry*<*Key ***extends*** Comparable*<*Key*>*, Value*> {* ***private*** Key key;  
 ***private*** Value val;  
 ***private*** Node next;  
  
 ***public*** Entry*(*Key key, Value val, Node next*) {* ***this***.key = key;  
 ***this***.val = val;  
 ***this***.next = next;  
 *}  
 }* ***public*** BTree*() {* root = ***new*** Node*(*0*)*;  
 *}* ***public int*** size*() {* ***return*** elemsCount;  
 *}* ***public int*** height*() {* ***return*** height;  
 *}* ***public*** Value get*(*Key key*) {* ***if*** *(*key == ***null****)* ***throw new*** IllegalArgumentException*(*"Argument to get() cannot be null"*)*;  
 ***return*** search*(*root, key, height*)*;  
 *}* ***private*** Value search*(*Node x, Key key, ***int*** ht*) {* Entry*[]* children = x.children;  
  
 ***if*** *(*ht == 0*) {* actionsLog.add*(*BTreeActions.*externalNodeTraverse)*;  
 ***for*** *(****int*** j = 0; j < x.childrenCount; j++*) {* ***if*** *(*eq*(*key, children*[*j*]*.key*)) {* actionsLog.add*(*BTreeActions.*nodeFound)*;  
 ***return*** *(*Value*)* children*[*j*]*.val;  
 *}  
 }  
 }* ***else*** *{* actionsLog.add*(*BTreeActions.*internalNodeTraverse)*;  
 ***for*** *(****int*** j = 0; j < x.childrenCount; j++*) {* ***if*** *(*j + 1 == x.childrenCount || less*(*key, children*[*j + 1*]*.key*))* ***return*** search*(*children*[*j*]*.next, key, ht - 1*)*;  
 *}  
 }* actionsLog.add*(*BTreeActions.*nodeNotFound)*;  
 ***return null***;  
 *}* ***public void*** put*(*Key key, Value val*) {* ***if*** *(*key == ***null****)* ***throw new*** IllegalArgumentException*(*"Argument key to put() cannot be null"*)*;  
 Node u = insert*(*root, key, val, height*)*;  
 elemsCount++;  
 ***if*** *(*u == ***null****)* ***return***;  
  
 actionsLog.add*(*BTreeActions.*rootSplit)*;  
 Node t = ***new*** Node*(*2*)*;  
 t.children*[*0*]* = ***new*** Entry*(*root.children*[*0*]*.key, ***null***, root*)*;  
 t.children*[*1*]* = ***new*** Entry*(*u.children*[*0*]*.key, ***null***, u*)*;  
 root = t;  
 height++;  
 *}* ***private*** Node insert*(*Node node, Key key, Value val, ***int*** ht*) {* ***int*** j;  
 Entry entry = ***new*** Entry*(*key, val, ***null****)*;  
  
 ***if*** *(*ht == 0*) {* actionsLog.add*(*BTreeActions.*externalNodeTraverse)*;  
 ***for*** *(*j = 0; j < node.childrenCount; j++*) {* ***if*** *(*less*(*key, node.children*[*j*]*.key*))* ***break***;  
 *}  
 }* ***else*** *{* actionsLog.add*(*BTreeActions.*internalNodeTraverse)*;  
 ***for*** *(*j = 0; j < node.childrenCount; j++*) {* ***if*** *((*j + 1 == node.childrenCount*)* || less*(*key, node.children*[*j + 1*]*.key*)) {* Node u = insert*(*node.children*[*j++*]*.next, key, val, ht - 1*)*;  
 ***if*** *(*u == ***null****)* ***return null***;  
 entry.key = u.children*[*0*]*.key;  
 entry.val = ***null***;  
 entry.next = u;  
 ***break***;  
 *}  
 }  
 }* ***for*** *(****int*** i = node.childrenCount; i > j; i--*)* node.children*[*i*]* = node.children*[*i - 1*]*;  
 node.children*[*j*]* = entry;  
 node.childrenCount++;  
 actionsLog.add*(*BTreeActions.*nodeInserted)*;  
 ***if*** *(*node.childrenCount < *M)* ***return null***;  
 ***else return*** split*(*node*)*;  
 *}* ***private*** Node split*(*Node h*) {* actionsLog.add*(*BTreeActions.*nodeSplit)*;  
 Node t = ***new*** Node*(M* / 2*)*;  
 h.childrenCount = *M* / 2;  
 ***for*** *(****int*** j = 0; j < *M* / 2; j++*)* t.children*[*j*]* = h.children*[M* / 2 + j*]*;  
 ***return*** t;  
 *}* ***private boolean*** less*(*Comparable k1, Comparable k2*) {* ***return*** k1.compareTo*(*k2*)* < 0;  
 *}* ***private boolean*** eq*(*Comparable k1, Comparable k2*) {* ***return*** k1.compareTo*(*k2*)* == 0;  
 *}  
}*

Тестирование:

***public class*** BTreeTest *{* ***public static*** BTree*<*Integer, String*> tree*;  
  
 @BeforeEach  
 ***public void*** prepare*() {  
 tree* = ***new*** BTree*<>()*;  
 *}* @Test  
 ***public void*** rootSplits*() {  
 tree*.put*(*1, "1"*)*;  
 *tree*.put*(*2, "2"*)*;  
 *tree*.put*(*3, "3"*)*;  
 *tree*.put*(*4, "4"*)*;  
 *assertEquals(*List.*of(* BTreeActions.*externalNodeTraverse*,  
 BTreeActions.*nodeInserted*,  
 BTreeActions.*externalNodeTraverse*,  
 BTreeActions.*nodeInserted*,  
 BTreeActions.*externalNodeTraverse*,  
 BTreeActions.*nodeInserted*,  
 BTreeActions.*externalNodeTraverse*,  
 BTreeActions.*nodeInserted*,  
 BTreeActions.*nodeSplit*,  
 BTreeActions.*rootSplit  
 )*, *tree*.actionsLog*)*;  
 *}* @Test  
 ***public void*** nodeInsertSuccess*() {  
 tree*.put*(*1, "1"*)*;  
 *tree*.put*(*3, "3"*)*;  
 *tree*.put*(*2, "2"*)*;  
 *tree*.put*(*5, "5"*)*;  
 *tree*.put*(*7, "7"*)*;  
 *tree*.put*(*6, "6"*)*;  
 *assertEquals(*List.*of(* BTreeActions.*externalNodeTraverse*,  
 BTreeActions.*nodeInserted*,  
 BTreeActions.*externalNodeTraverse*,  
 BTreeActions.*nodeInserted*,  
 BTreeActions.*externalNodeTraverse*,  
 BTreeActions.*nodeInserted*,  
  
 BTreeActions.*externalNodeTraverse*,  
 BTreeActions.*nodeInserted*,  
 BTreeActions.*nodeSplit*,  
 BTreeActions.*rootSplit*,  
  
 BTreeActions.*internalNodeTraverse*,  
 BTreeActions.*externalNodeTraverse*,  
 BTreeActions.*nodeInserted*,  
  
 BTreeActions.*internalNodeTraverse*,  
 BTreeActions.*externalNodeTraverse*,  
 BTreeActions.*nodeInserted*,  
 BTreeActions.*nodeSplit*,  
 BTreeActions.*nodeInserted  
 )*, *tree*.actionsLog*)*;  
 *}* @Test  
 ***public void*** searchSuccess*() {  
 tree*.put*(*1, "1"*)*;  
 *tree*.put*(*2, "2"*)*;  
 *tree*.put*(*3, "3"*)*;  
 *tree*.put*(*4, "4"*)*;  
 *tree*.actionsLog.clear*()*;  
 String result = *tree*.get*(*4*)*;  
 *assertEquals(*List.*of(* BTreeActions.*internalNodeTraverse*,  
 BTreeActions.*externalNodeTraverse*,  
 BTreeActions.*nodeFound  
 )*, *tree*.actionsLog*)*;  
 *assertEquals(*result, "4"*)*;  
 *}* @Test  
 ***public void*** searchFails*() {  
 tree*.put*(*1, "1"*)*;  
 *tree*.put*(*2, "2"*)*;  
 *tree*.put*(*3, "3"*)*;  
 *tree*.put*(*4, "4"*)*;  
 *tree*.actionsLog.clear*()*;  
 String result = *tree*.get*(*9*)*;  
 *assertEquals(*List.*of(* BTreeActions.*internalNodeTraverse*,  
 BTreeActions.*externalNodeTraverse*,  
 BTreeActions.*nodeNotFound  
 )*, *tree*.actionsLog*)*;  
 *assertNull(*result*)*;  
 *}* @Test  
 ***public void*** validation*() {  
 assertThrows(*IllegalArgumentException.***class***, *()* -> *tree*.put*(****null***, ***null****))*;  
 *assertThrows(*IllegalArgumentException.***class***, *()* -> *tree*.get*(****null****))*;  
 *}  
}*

## 3. Доменная модель

Код реализации доменной модели:

***public enum*** BrainStatus *{  
 STABLE*,  
 *BLOWN  
}*

***public enum*** EngineStatus *{  
 OFF*,  
 *WORKING*,  
 *BROKEN  
}*

***public enum*** Mood *{  
 GLAD*,  
 *SHOCKED*,  
 *BORED  
}*

***public interface*** ICar *{* ***void*** startEngine*()*;  
 ***void*** stopEngine*()*;  
 ***void*** changeGear*(****int*** gear*)*;  
 ***void*** overtake*()*;  
 EngineStatus checkEngine*()*;  
 ***int*** checkGear*()*;  
 IPerson getDriver*()*;  
*}*

***public interface*** IPerson *{* Mood getCurrentMood*()*;  
 ***void*** brainBlow*()*;  
 ***void*** reactToOvertake*()*;  
 BrainStatus getBrainStatus*()*;  
*}*

***public class*** Car ***implements*** ICar*{* ***public*** IPerson driver;  
 ***private*** EngineStatus engineStatus = EngineStatus.*OFF*;  
 ***private int*** currentGear = 0;  
  
 ***public*** Car*(*IPerson driver*) {* ***this***.driver = driver;  
 *}* @Override  
 ***public void*** startEngine*() {* ***if*** *(*engineStatus == EngineStatus.*WORKING) {* ***return***;  
 *}* ***if*** *(*engineStatus == EngineStatus.*BROKEN){* ***throw new*** EngineDiedException*(*"Engine is broken"*)*;  
 *}* ***if*** *(*currentGear > 0*){* ***throw new*** EngineDiedException*(*"You tried to start engine with not neutral gear selected"*)*;  
 *}* engineStatus = EngineStatus.*WORKING*;  
 *}* @Override  
 ***public void*** stopEngine*(){* engineStatus = EngineStatus.*OFF*;  
 *}* @Override  
 ***public void*** changeGear*(****int*** gear*) {* ***if*** *(*gear > 5 || gear < 0*){* ***throw new*** IllegalArgumentException*(*"Gear can be 0-5"*)*;  
 *}* ***if*** *(*gear == 0*){* currentGear = gear;  
 *}* ***else if*** *(*Math.*abs(*currentGear - gear*)* > 1 && engineStatus == EngineStatus.*WORKING) {* engineStatus = EngineStatus.*BROKEN*;  
 driver.brainBlow*()*;  
 *}* ***else*** *{* currentGear = gear;  
 *}  
 }* @Override  
 ***public void*** overtake*() {* ***if*** *(*engineStatus == EngineStatus.*WORKING* && currentGear > 0*)* driver.reactToOvertake*()*;  
 ***else throw new*** CannotOvertakeException*(*"Cannot overtake while standing"*)*;  
 *}* @Override  
 ***public*** EngineStatus checkEngine*() {* ***return*** engineStatus;  
 *}* @Override  
 ***public int*** checkGear*() {* ***return*** currentGear;  
 *}* @Override  
 ***public*** IPerson getDriver*() {* ***return*** driver;  
 *}  
}*

***public class*** Person ***implements*** IPerson*{* ***private*** Mood mood = Mood.*BORED*;  
 ***private*** BrainStatus brain = BrainStatus.*STABLE*;  
  
 @Override  
 ***public*** Mood getCurrentMood*() {* ***return*** mood;  
 *}* @Override  
 ***public void*** brainBlow*() {* brain = BrainStatus.*BLOWN*;  
 mood = Mood.*SHOCKED*;  
 *}* @Override  
 ***public void*** reactToOvertake*() {* mood = Mood.*GLAD*;  
 *}* @Override  
 ***public*** BrainStatus getBrainStatus*() {* ***return*** brain;  
 *}  
}*

Тестирование:

***public class*** DomainTest *{* ICar car;  
  
 @BeforeEach  
 ***public void*** prepare*(){* car = ***new*** Car*(****new*** Person*())*;  
 *}* @Test  
 ***public void*** changingGearsSuccess*(){* car.startEngine*()*;  
 car.changeGear*(*1*)*;  
 car.changeGear*(*2*)*;  
 car.changeGear*(*3*)*;  
 car.changeGear*(*4*)*;  
 car.changeGear*(*5*)*;  
 *assertEquals(*EngineStatus.*WORKING*, car.checkEngine*())*;  
 *assertEquals(*5, car.checkGear*())*;  
 *}* @Test  
 ***public void*** cannotStartCarWithNonNeutralGear*(){* car.changeGear*(*1*)*;  
 *assertThrows(*EngineDiedException.***class***, *()* -> car.startEngine*())*;  
 *}* @Test  
 ***public void*** cannotChangeToSixthsGear*(){* car.startEngine*()*;  
 car.changeGear*(*1*)*;  
 car.changeGear*(*2*)*;  
 car.changeGear*(*3*)*;  
 car.changeGear*(*4*)*;  
 car.changeGear*(*5*)*;  
 *assertThrows(*IllegalArgumentException.***class***, *()* -> car.changeGear*(*6*))*;  
 *}* @Test  
 ***public void*** wrongGearChangingCausesBrainBlowingAndEngineBreaking*(){* car.startEngine*()*;  
 car.changeGear*(*1*)*;  
 car.changeGear*(*5*)*;  
 *assertEquals(*EngineStatus.*BROKEN*, car.checkEngine*())*;  
 *assertEquals(*Mood.*SHOCKED*, car.getDriver*()*.getCurrentMood*())*;  
 *assertEquals(*BrainStatus.*BLOWN*, car.getDriver*()*.getBrainStatus*())*;  
 *}* @Test  
 ***public void*** overtakingMakesDriverGlad*(){* car.startEngine*()*;  
 car.changeGear*(*1*)*;  
 car.overtake*()*;  
 *assertEquals(*Mood.*GLAD*, car.getDriver*()*.getCurrentMood*())*;  
 *}* @Test  
 ***public void*** cannotOvertakeWhileStandingInPlace*(){  
 assertThrows(*CannotOvertakeException.***class***, *()* -> car.overtake*())*;  
 car.startEngine*()*;  
 *assertThrows(*CannotOvertakeException.***class***, *()* -> car.overtake*())*;  
 *}* @Test  
 ***public void*** cannotStartBrokenEngine*(){* car.startEngine*()*;  
 car.changeGear*(*1*)*;  
 car.changeGear*(*5*)*;  
 *assertThrows(*EngineDiedException.***class***, *()* -> car.startEngine*())*;  
 *}  
}*

# Вывод

В ходе выполнения работы я научился составлять модульные тесты по принципах черного ящика (тестирование математической функции и доменной модели) и белого ящика (тестирование Б-Дерева)