Университет ИТМО

Факультет программной инженерии и компьютерной техники

Направление подготовки 09.03.04 Программная инженерия

Дисциплина «Тестирование программного обеспечения»

Отчет

По лабораторной работе №1

Вариант 235717

Выполнил:
Зенин М.А.
P33101

Преподаватель:
Машина Е. А.

Санкт-Петербург, 2024 г.

Задание

Этапы

1. Для указанной функции провести модульное тестирование разложения функции в степенной ряд. Выбрать достаточное тестовое покрытие.
2. Провести модульное тестирование указанного алгоритма. Для этого выбрать характерные точки внутри алгоритма, и для предложенных самостоятельно наборов исходных данных записать последовательность попадания в характерные точки. Сравнить последовательность попадания с эталонной.
3. Сформировать доменную модель для заданного текста. Разработать тестовое покрытие для данной доменной модели

Вариант

1. Функция arcsin(x)
2. Программный модуль для работы с Фибоначчиевой кучей (Internal Representation, http://www.cs.usfca.edu/~galles/visualization/FibonacciHeap.html)
3. Описание предметной области:
   “*Еще мгновение люди сидели молча, а потом человек, громко смеявшийся, засмеялся снова. Девушка, которую он затащил с собой в бар, за последний час всем сердцем возненавидела его, и, возможно, была бы рада узнать, что через полторы минуты он испарится, превратившись в облачко водорода, озона и оксида углерода. Однако когда момент наступил, она была слишком занята собственным испарением, чтобы обратить на это внимание*.”

Диаграмма классов

Функция arcsin(x)

Тестовый класс выглядит так:

```
public class MyMathTest {

   @ParameterizedTest(name = "asin({0})")

   @DisplayName("Check dots from [-0.75; 0.75] with step 0.05")

   @MethodSource("genNotSoBigAnswerTestSource")

   public void notSoBigAnswerTest(Double arg) {
```

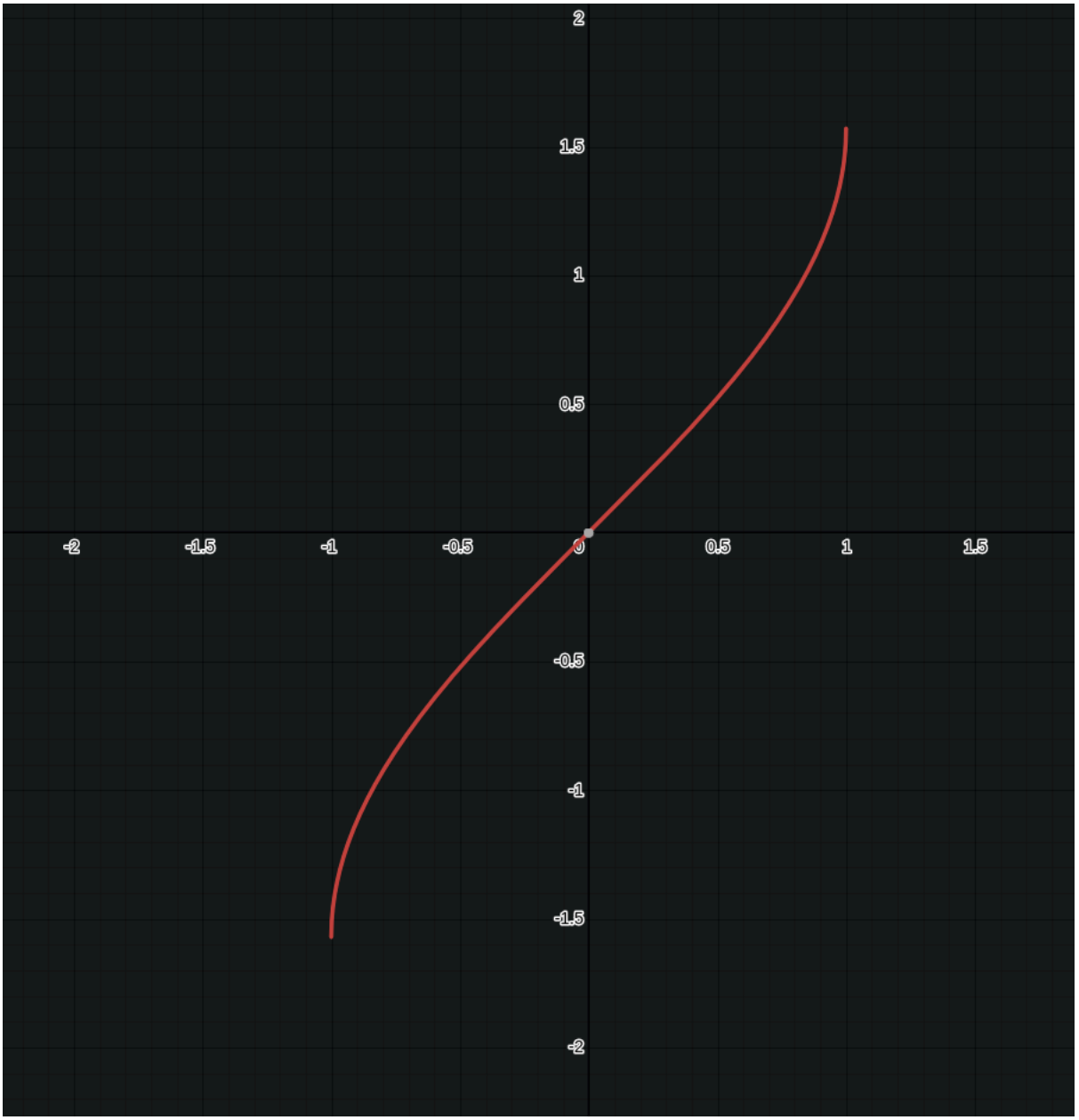
```
        assertEquals(Math.asin(arg), MyMath.asin(arg, 10), 0.001);
    }

    @ParameterizedTest(name = "asin({0})")
    @DisplayName("Check dots from [-1; -0.75] && [0.75; 1] with
step 0.05")
    @MethodSource("genSoBigAnswerTestSource")
    public void soBigAnswerTest(Double arg) {
        assertEquals(Math.asin(arg), MyMath.asin(arg, 10000),
0.01);
    }

    static Stream<Double> genSoBigAnswerTestSource() {
        double from = -1;
        double to = -0.75;
        double step = 0.05;
        int count = (int) Math.round((to - from) / step);
        var leftValuesStream =
ParamsFactory.genDotsInRangeSameDistanceBetween(from, to, count);
        from = 0.75;
        to = 1;
        var rightValuesStream =
ParamsFactory.genDotsInRangeSameDistanceBetween(from, to, count);
        return Stream.concat(leftValuesStream, rightValuesStream);
    }

    static Stream<Double> genNotSoBigAnswerTestSource() {
        double from = -0.75;
        double to = 0.75;
        double step = 0.05;
```

```
        int count = (int) Math.round((to - from) / step);

        return
ParamsFactory.genDotsInRangeSameDistanceBetween(from, to, count);

    }


    class ParamsFactory {

        static Stream<Double>
genDotsInRangeSameDistanceBetween(double from, double to, int
count) {

            var values = new LinkedList<Double>();

            var step = (to - from) / (count - 1);

            var current = from;

            for (double i = 0; i < count; i++, current += step) {

                values.add(current);

            }

            return values.stream();

        }

    }

}
```

Первые пара тестов проверяет в экстремумах функции — это значения 1 и -1.

Вторая пара тестов проверяет значения около разрывов.

Пятый тест проверяет значения в точках разрыва функции.



Как видно по графику, функция определена от -1 до 1, соответственно, нужно покрыть тестами этот промежуток

Тесты разделены на две части - одна в местах ф-ции, где она медленно возрастает, а вторая - где быстро. Это нужно для проверки результатов в рамках погрешности

Фибоначчиева куча

Тестовый класс выглядит так:

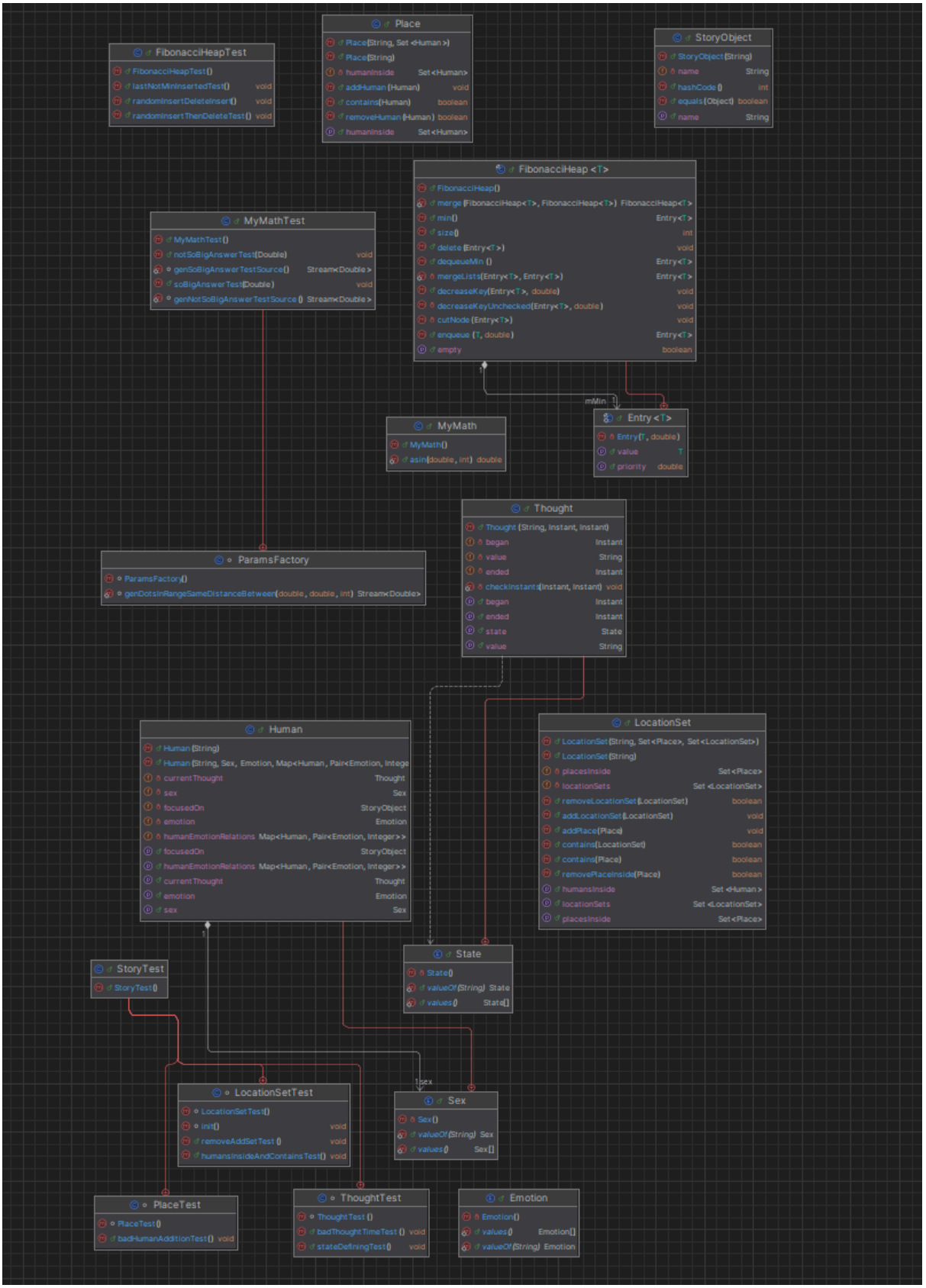
```
public class FibonacciHeapTest {


    @Test
    @DisplayName("last not min inserted")
    public void lastNotMinInsertedTest() {
        var lowerElementObject = new Object();
        FibonacciHeap<Object> heap = new FibonacciHeap<>();
        var prioritiesList = List.of(4, 3, 2);
        for (var priority : prioritiesList) {
            heap.enqueue(null, priority);
        }
        heap.enqueue(lowerElementObject, 1);
        heap.enqueue(null, 2);
        assertEquals(heap.size(), prioritiesList.size() + 1 + 1);
        assertEquals(heap.dequeueMin().getValue(),
lowerElementObject);
        assertEquals(heap.size(), prioritiesList.size() + 1);
    }


    @Test
    @DisplayName("Random insert then delete")
    public void randomInsertThenDeleteTest() {
        var elements = List.of(1, 5, 2, 8, 4, 6, 3, 7, 9);
        FibonacciHeap<Integer> heap = new FibonacciHeap<>();
        for (var element : elements) {
            heap.enqueue(element, element);
```

```
        }
        assertEquals(heap.size(), elements.size());
        var sorted = elements.stream().sorted().toList();
        for (var value : sorted) {
            assertEquals(heap.dequeueMin().getValue(), value);
        }
        assertEquals(heap.size(), 0);
    }

    @Test
    @DisplayName("Random insert then delete")
    public void randomInsertDeleteInsert() {
        var elements = List.of(1, 5, 2, 3, 4);
        FibonacciHeap<Integer> heap = new FibonacciHeap<>();
        for (var element : elements) {
            heap.enqueue(element, element);
        }
        assertEquals(heap.size(), elements.size());
        assertEquals(heap.dequeueMin().getValue(), 1);
        assertEquals(heap.dequeueMin().getValue(), 2);
        assertEquals(heap.dequeueMin().getValue(), 3);
        assertEquals(heap.size(), elements.size() - 3);
        heap.enqueue(1, 1);
        heap.enqueue(6, 6);
        heap.enqueue(3, 3);
        assertEquals(heap.dequeueMin().getValue(), 1);
        assertEquals(heap.dequeueMin().getValue(), 3);
        assertEquals(heap.dequeueMin().getValue(), 4);
```

```
        assertEquals(heap.size(), elements.size() - 3);

    }

}
```

В ходе тестов проверяется, что оно возвращает именно наименьший элемент, а не случайный / последний вставленный, затем аналогичный тест, но более полный - в случайном порядке вставляются значения и затем удаляются - каждый раз должен быть удален минимальный элемент. Затем то же самое, но уже с последующей после удаления вставкой - удаление не должно ломать структуру данных

Доменная модель



Тестовый класс выглядит так:

```
public class StoryTest {


    @Nested
    class ThoughtTest {
        @Test
        @DisplayName("Thought different state defining test")
        public void stateDefiningTest() {
            Thought thought = new Thought("", null, null);
            assertEquals(thought.getState(),
Thought.State.NOT_EVEN_BEGAN);
            thought = new Thought("", Instant.now(), null);
            assertEquals(thought.getState(),
Thought.State.IN_PROGRESS);
            thought = new Thought("", Instant.now(),
Instant.now());
            assertEquals(thought.getState(), Thought.State.ENDED);
        }


        @Test
        @DisplayName("Bad thought time test")
        public void badThoughtTimeTest() {
            assertThrows(IllegalStateException.class, () -> new
Thought("", null, Instant.now()));
            var notStartedThought = new Thought("", null, null);
            assertThrows(IllegalStateException.class, () ->
notStartedThought.setEnded(Instant.now()));
            Instant began = Instant.now();
            Instant badEnded =
Instant.now().minus(Duration.ofHours(1));
```

```
            assertThrows(IllegalStateException.class, () -> new
Thought("", began, badEnded));

            var startedThought = new Thought("", began, null);

            assertThrows(IllegalStateException.class, () ->
startedThought.setEnded(badEnded));

        }

    }


    @Nested

    class PlaceTest {


        @Test

        @DisplayName("Bad human addition test")

        public void badHumanAdditionTest() {

            Place place = new Place("");

            place.addHuman(new Human("Vasya"));

            place.addHuman(new Human("Petya"));

            assertThrows(IllegalStateException.class, () ->
place.addHuman(new Human("Vasya")));

        }

    }


    @Nested

    class LocationSetTest {


        private LocationSet parentLocationSet;

        private LocationSet childLocationSet;

        private Place childLocationSetPlace;

        private Place parentLocationSetPlace;
```

```
        private Human parentPlaceHuman;
        private Human childPlaceHuman;


        @BeforeEach
        void init() {
            parentLocationSet = new LocationSet("Parent");
            childLocationSet = new LocationSet("Child");
            childLocationSetPlace = new Place("Child.Place");
            parentLocationSetPlace = new Place("Parent.Place");
//
            // insert
            parentLocationSet.setLocationSets(new
HashSet<>(List.of(childLocationSet)));


            parentPlaceHuman = new Human("Parent.Place.Human");
            parentLocationSetPlace.addHuman(parentPlaceHuman);


            childPlaceHuman = new Human("Child.Place.Human");
            childLocationSetPlace.addHuman(childPlaceHuman);


            parentLocationSet.setPlacesInside(new
HashSet<>(List.of(parentLocationSetPlace)));
            childLocationSet.setPlacesInside(new
HashSet<>(List.of(childLocationSetPlace)));
        }


        @Test
        @DisplayName("Humans inside receiving and contains test")
        public void humansInsideAndContainsTest() {
```

```
            // checks

assertTrue(parentLocationSet.contains(parentLocationSetPlace));

assertTrue(parentLocationSet.contains(childLocationSet));

assertTrue(childLocationSet.contains(childLocationSetPlace));

assertTrue(parentLocationSet.contains(childLocationSetPlace));
            assertEquals(parentLocationSet.getHumansInside(),
                    new HashSet<>(List.of(parentPlaceHuman,
childPlaceHuman)));

assertFalse(parentLocationSet.contains(parentLocationSet));
            assertFalse(parentLocationSet.contains(new
LocationSet("")));

assertFalse(childLocationSet.contains(parentLocationSet));
            assertFalse(parentLocationSet.contains(new Place("")));

assertFalse(childLocationSet.contains(parentLocationSetPlace));
        }


        @Test
        @DisplayName("Remove-add-set test")
        public void removeAddSetTest() {
            parentLocationSet.removeLocationSet(childLocationSet);

assertFalse(parentLocationSet.contains(childLocationSet));

assertFalse(parentLocationSet.contains(childLocationSetPlace));
            parentLocationSet.addLocationSet(childLocationSet);
```

```
assertTrue(parentLocationSet.contains(childLocationSet));

assertTrue(parentLocationSet.contains(childLocationSetPlace));


            assertThrows(IllegalStateException.class,
                    () ->
parentLocationSet.removeLocationSet(parentLocationSet));
            assertThrows(IllegalStateException.class,
                    () ->
parentLocationSet.addLocationSet(parentLocationSet));


            var tempLocationSet = new LocationSet("tempLS");
            childLocationSet.addLocationSet(tempLocationSet);

assertTrue(parentLocationSet.contains(tempLocationSet));
            assertThrows(IllegalStateException.class,
                    () ->
parentLocationSet.addLocationSet(tempLocationSet));

assertTrue(parentLocationSet.removeLocationSet(tempLocationSet));

assertFalse(parentLocationSet.contains(tempLocationSet));

assertTrue(parentLocationSet.removeLocationSet(childLocationSet));

assertFalse(parentLocationSet.removeLocationSet(childLocationSet))
;
            parentLocationSet.addLocationSet(childLocationSet);


            var anotherLocationSetContainingChild = new
LocationSet("another");
```

```
anotherLocationSetContainingChild.addPlace(parentLocationSetPlace)
;

            anotherLocationSetContainingChild.setLocationSets(new
HashSet<>(List.of(childLocationSet)));

            assertThrows(IllegalStateException.class,

                    () -> parentLocationSet.setLocationSets(new
HashSet<>(List.of(anotherLocationSetContainingChild))));

            assertThrows(IllegalStateException.class,

                    () -> parentLocationSet.setLocationSets(new
HashSet<>(List.of(parentLocationSet))));



            assertThrows(IllegalStateException.class,

                    () -> parentLocationSet.setPlacesInside(new
HashSet<>(List.of(childLocationSetPlace))));



            assertThrows(IllegalStateException.class,

                    () ->
parentLocationSet.addPlace(childLocationSetPlace));

        }

    }

}
```

Тесты делятся на 3 группы: мысли, места и локации
Мысль:

1. Проверяется возврат ее состояния по параметрам конструктора (текущим внутренним состоянием)
2. Проверяются ненормальные состояния объекта (должна проброситься ошибка): установка времени окончания мысли без времени ее начала и установка времени окончания мысли, которое раньше времени начала

Место:

1. Проверка сохранения людей в месте

Локация:

1. Проверка сохранения людей, локаций, мест там (в том числе людей там не находящихся + учет рекурсии)

2. То же, но с удалением людей, локаций, мест (с проверкой удаления не добавленных экземпляров)

Вывод

Во время выполнения данной лабораторной работы мы познакомились с реализацией модульного тестирования на языке java, а именно с библиотекой junit. Эти знания кажутся нам очень полезными и пригодятся нам в будущем.