Task2

На странице представлена информация по заданию Task2 (лабораторная работа №2).

Система содержит тест, который проверяет вывод, он должен быть ровно таким, каким представлен в примерах. Все методы main класов даны в заглушке (совпадают с тем, что есть в примерах).

Также система содержит модульные тесты, которые тестируют методы. В случае, если Jenkins не собирает задачу причину см. в логе сборки.

Входные данные

Проект заглушка

Проект заглушка Task2Stub находится в репозитории по адресу: **/examples/projects/Task2Stub**

Корневой пакет проекта

Корневой пакет для проекта: ua.nure.your_last_name.Task2

your_last_name заменить на свою фамилию (ваш логин, без последних трех букв - jff).

Задание

В каждом из трех классов (**Circle**, **Matrix**, **ListImpl**) должен быть метод **main**, который демонстрирует работу соответствующего класса. Все данные задавать внутри **main**.

Класс Circle

Поля класса:

1. координаты центра окружности:
   double x
   double y

2. радиус окружности:
   double r

Методы класса:

1. передвинуть окружность на **dx** и **dy**:
   public void move(double dx, double dy) { ... }

2. проверить попадание заданной точки внутрь данной окружности:
   public boolean isInside(double x, double y) { ... }

3. проверить попадание другой окружности внутрь данной:
   public boolean isInside(Circle c) { ... }

4. вывести на экран параметры окружности
   public void print() { ... }

Конструкторы:

1. **public CirIce(double x, double y, double r) { ... }**

Реализовать класс. Создать метод main, который демонстрирует работу всех методов класса (**код метода main взять из примера ниже**).

Пример метода Circle.main

Circle.main

```
public static void main(String[] args) {
    System.out.println("~~~ c");
    Circle c = new Circle(0, 0, 1);
    c.print();
    System.out.println("~~~ c.move(1, 1)");
    c.move(1, 1);
    c.print();
    System.out.println("~~~ c.isInside(1, 1)");
    System.out.println(c.isInside(1, 1));
    System.out.println("~~~ c.isInside(10, 1)");
    System.out.println(c.isInside(10, 1));
    System.out.println("~~~ c2");
    Circle c2 = new Circle(1, 1, 2);
    c2.print();
    System.out.println("~~~ c.isInside(c2)");
    System.out.println(c.isInside(c2));
    System.out.println("~~~ c2.isInside(c)");
    System.out.println(c2.isInside(c));
}
```

Результат (должен быть ровно таким)

```
~~~ c
Circle (0.0, 0.0, 1.0)
~~~ c.move(1, 1)
Circle (1.0, 1.0, 1.0)
~~~ c.isInside(1, 1)
true
~~~ c.isInside(10, 1)
false
~~~ c2
Circle (1.0, 1.0, 2.0)
~~~ c.isInside(c2)
false
~~~ c2.isInside(c)
true
```

Класс Matrix

Поля класса:

1. двумерный массив вещественных чисел:
   double[rows][cols] ar

2. количество строк и столбцов в матрице.
   int rows
   int cols

Методы класса:

1. сложение с другой матрицей:
   public **void add(Matrix m) { ... }**

2. умножение на число:
   public **void mul(double x) { ... }**

3. умножение на другую матрицу:
   public **void mul(Matrix m) { ... }**

4. транспонирование:
   public **void transpose()** { ... }

5. печать матрицы на экран:
   public void print() { ... }

Конструкторы:

1. **public Matrix(double[][] ar) { ... }**

Реализовать класс. Создать метод main, который демонстрирует работу всех методов класса (**код метода main взять из примера ниже**).

Пример метода Matrix.main

Matrix.main

```
public static void main(String[] args) {
    Matrix m = new Matrix(new double[][] {
        {1.0, 2.0, 3.0},
        {4.0, 5.0, 6.0}
    });

    Matrix m2 = new Matrix(new double[][] {
        {1.0, 2.0, 3.0},
        {4.0, 5.0, 6.0}
    });

    System.out.println("~~~ m");
    m.print();

    System.out.println("~~~ m2");
    m2.print();

    System.out.println("~~~ transpose m");
    m.transpose();
    m.print();

    System.out.println("~~~ mul m on m2");
    m.mul(m2);
    m.print();

    System.out.println("~~~ mul m2 on 2");
    m2.mul(2);
    m2.print();
}
```

Результат (должен быть ровно таким)

```
~~~ m
1.0 2.0 3.0
4.0 5.0 6.0
~~~ m2
1.0 2.0 3.0
4.0 5.0 6.0
~~~ transpose m
1.0 4.0
2.0 5.0
3.0 6.0
~~~ mul m on m2
17.0 22.0 27.0
22.0 29.0 36.0
27.0 36.0 45.0
~~~ mul m2 on 2
2.0 4.0 6.0
8.0 10.0 12.0
```

Класс ListImpl

1. Определить интерфейс **List** следующего содержания:

Interface List

```
public interface List {
    // Appends the specified element to the end of this list
    void add(Object el);

    // Appends all of the elements in the specified collection to the end of this list
    void addAll(List list);

    // Removes all of the elements from this list
    void clear();

    // Returns true if this list contains the specified element
    boolean contains(Object el);

    // Returns the element at the specified position in this list.
    Object get(int index);

    // Returns the index of the first occurrence of the specified element in this list
    int indexOf(Object el);

    // Removes the element at the specified position in this list, returns the element previously at the
specified position
    // Throws IndexOutOfBoundsException if the index is out of range
    Object remove(int index);

    // Removes the first occurrence of the specified element from this list, returns true if this list
contained the specified element
    boolean remove(Object el);

    // Returns the number of elements in this list
    int size();
}
```

⚠ Для определения равенства объекта obj объекту obj2 использовать метод **Object#equals**: **obj.equals(obj2)**

2. Создать класс **ListImpl**, который реализует интерфейс **List**:

Class ListImpl

```
public class ListImpl implements List {
    public ListImpl() { ... }
    ...
}
```

3. Определить интерфейс **Iterator**, унаследовав его от **java.util.Iterator<Object>**:

Interface Iterator

```
public interface Iterator extends java.util.Iterator<Object> {

    // Returns true if the iteration has more elements
    boolean hasNext();

    // Returns the next element in the iteration
    E next();

    // Removes the last element returned by this iterator
    // This method can be called only once per call to next()
    // Method throws IllegalStateException if:
    // (1) the next method has not yet been called, or
    // (2) the remove method has already been called after the last call to the next method
    void remove();
}
```

4. Добавить к интерфейсу **List** наследование интерфейса **java.lang.Iterable<Object>**:

List extends java.lang.Iterable<Object>

```
public interface List extends java.lang.Iterable<Object> {

    ...

    // Returns an iterator
    Iterator iterator();
}
```

5. Реализовать метод **iterator** в классе **ListImpl**.

6. Перекрыть в классе ListImpl метод toString, следующим образом:
если list содержит элементы E, E2, ..., En то list.toString возвращает строку [E.toString(), E2.toString(), ..., En.toString()]

7. Создать метод main, который демонстрирует работу всех методов класса **ListImpl**. (**Код метода main взять из примера ниже**).

Пример метода ListImpl.main

ListImpl.main

```
public static void main(String[] args) {
    System.out.println("~~~ list A B C");
    System.out.println("~~~ Result: [A, B, C]");
    List list = new ListImpl();
    list.add("A");
    list.add("B");
    list.add("C");
    System.out.println(list);

    System.out.println("~~~ list2: D E F");
    System.out.println("~~~ Result: [D, E, F]");
    List list2 = new ListImpl();
    list2.add("D");
    list2.add("E");
    list2.add("F");
    System.out.println(list2);

    System.out.println("~~~ list.addAll(list2)");
    System.out.println("~~~ Result: [A, B, C, D, E, F]");
    list.addAll(list2);
    System.out.println(list);

    System.out.println("~~~ list.add(C)");
    System.out.println("~~~ Result: [A, B, C, D, E, F, C]");
    list.add("C");
    System.out.println(list);

    System.out.println("~~~ list.clear()");
    System.out.println("~~~ Result: []");
    list.clear();
    System.out.println(list);

    System.out.println("~~~ list.addAll(list2)");
    System.out.println("~~~ Result: [D, E, F]");
    list.addAll(list2);
    System.out.println(list);

    System.out.println("~~~ list.contains(E)");
    System.out.println("~~~ Result: true");
    System.out.println(list.contains("E"));

    System.out.println("~~~ list.contains(C)");
    System.out.println("~~~ Result: false");
    System.out.println(list.contains("C"));

    System.out.println("~~~ list.indexOf(D)");
    System.out.println("~~~ Result: 0");
    System.out.println(list.indexOf("D"));

    System.out.println("~~~ list.get(2)");
    System.out.println("~~~ Result: F");
    System.out.println(list.get(2));

    System.out.println("~~~ list.indexOf(F)");
    System.out.println("~~~ Result: 2");
    System.out.println(list.indexOf("F"));

    System.out.println("~~~ list.size()");
    System.out.println("~~~ Result: 3");
    System.out.println(list.size());

    System.out.println("~~~ list");
    System.out.println("~~~ Result: [D, E, F]");
    System.out.println(list);

    System.out.println("~~~ list.remove(1)");
    System.out.println("~~~ Result: [D, F]");
    list.remove(1);
    System.out.println(list);

    System.out.println("~~~ list.remove(F)");
    System.out.println("~~~ Result: [D]");
    list.remove("F");
    System.out.println(list);
```

```
        System.out.println("~~~ list.size()");
        System.out.println("~~~ Result: 1");
        System.out.println(list.size());

        System.out.println("~~~ list.addAll(list2)");
        System.out.println("~~~ Result: [D, D, E, F]");
        list.addAll(list2);
        System.out.println(list);

        System.out.println("~~~ foreach list");
        System.out.println("~~~ Result: D D E F");
        for (Object el : list) {
            System.out.print(el + " ");
        }
        System.out.println();

        System.out.println("~~~ Iterator it = list.iterator()");
        Iterator it = list.iterator();

        System.out.println("~~~ it.next()");
        System.out.println("~~~ Result: D");
        System.out.println(it.next());

        System.out.println("~~~ it.next()");
        System.out.println("~~~ Result: D");
        System.out.println(it.next());

        System.out.println("~~~ it.remove()");
        System.out.println("~~~ Result: [D, E, F]");
        it.remove();
        System.out.println(list);

        System.out.println("~~~ it.next()");
        System.out.println("~~~ Result: E");
        System.out.println(it.next());

        System.out.println("~~~ it.remove()");
        System.out.println("~~~ Result: [D, F]");
        it.remove();
        System.out.println(list);

        System.out.println("~~~ it.next()");
        System.out.println("~~~ Result: F");
        System.out.println(it.next());

        System.out.println("~~~ it.remove()");
        System.out.println("~~~ Result: [D]");
        it.remove();
        System.out.println(list);

        System.out.println("~~~ list.remove(D)");
        System.out.println("~~~ Result: []");
        list.remove("D");
        System.out.println(list);

        System.out.println("~~~ list.addAll(list2)");
        System.out.println("~~~ Result: [D, E, F]");
        list.addAll(list2);
        System.out.println(list);

        System.out.println("~~~ foreach list");
        System.out.println("~~~ Result: D E F ");
        for (Object el : list) {
            System.out.print(el + " ");
        }
        System.out.println();
    }
```

Результат (должен быть ровно таким)

```
 ~~~ list A B C
~~~ Result: [A, B, C]
[A, B, C]
~~~ list2: D E F
~~~ Result: [D, E, F]
[D, E, F]
~~~ list.addAll(list2)
~~~ Result: [A, B, C, D, E, F]
[A, B, C, D, E, F]
~~~ list.add(C)
~~~ Result: [A, B, C, D, E, F, C]
[A, B, C, D, E, F, C]
~~~ list.clear()
~~~ Result: []
[]
~~~ list.addAll(list2)
~~~ Result: [D, E, F]
[D, E, F]
~~~ list.contains(E)
~~~ Result: true
true
~~~ list.contains(C)
~~~ Result: false
false
~~~ list.indexOf(D)
~~~ Result: 0
0
~~~ list.get(2)
~~~ Result: F
F
~~~ list.indexOf(F)
~~~ Result: 2
2
~~~ list.size()
~~~ Result: 3
3
~~~ list
~~~ Result: [D, E, F]
[D, E, F]
~~~ list.remove(1)
~~~ Result: [D, F]
[D, F]
~~~ list.remove(F)
~~~ Result: [D]
[D]
~~~ list.size()
~~~ Result: 1
1
~~~ list.addAll(list2)
~~~ Result: [D, D, E, F]
[D, D, E, F]
~~~ foreach list
~~~ Result: D D E F
D D E F
~~~ Iterator it = list.iterator()
~~~ it.next()
~~~ Result: D
D
~~~ it.next()
~~~ Result: D
D
~~~ it.remove()
~~~ Result: [D, E, F]
[D, E, F]
~~~ it.next()
~~~ Result: E
E
~~~ it.remove()
~~~ Result: [D, F]
[D, F]
~~~ it.next()
~~~ Result: F
F
~~~ it.remove()
~~~ Result: [D]
```

```
[D]
~~~ list.remove(D)
~~~ Result: []
[]
~~~ list.addAll(list2)
~~~ Result: [D, E, F]
[D, E, F]
~~~ foreach list
~~~ Result: D E F
D E F
```

Замечания

1. Проект в IDE (Eclipse/IDEA) должен иметь название Task2.
2. Все типы должны находится в корневом пакете.
3. Дополнительно должен быть класс Demo, который демонстрирует работу всех классов.
4. Jenkins после компилирования исходного кода и запуска на выполнения класса Demo **запускает набор тестов** (параметризованный Junit-тест, который тестирует методы классов).

Порядок выполнения работы

1. Используя заглушку (заглушку использовать не обязательно, можно писать все "с нуля"):
    a. Сделать checkout проекта заглушки из репозитория и отвязать его от узла /examples/projects/Task2Stub
    b. Переименовать корневой пакет проекта: your_last_name == > ваш логин, без последних трех букв, которые обозначают код проекта - jff.
    c. Переименовать проект Task2Stub ==> Task2.
2. Создать интерфейсы и реализовать все классы, которые требуются. Написать класс Demo.
3. Привязать проект к нужному узлу в репозитории и сделать коммит проекта в репозиторий.

> ⛔ Коммит не пройдет, если среди файлов, помещаемых в репозиторий, будет хотя бы один class-файл, т.е. class-файлы не коммитить.
>
> Также в обязательном порядке в репозиторий должны быть помещены метафайлы IDE (для Eclipse - .project; для IDEA - Task2.iml), иначе Jenkins проект не соберет.

4. Добиться сборки проекта в Jenkins (после каждого коммита в репозиторий Jenkins пересобирает проект, если проект не собрался, то причину можно посмотреть в логах сборки).
5. Оптимизировать метрики проекта в Sonar (Blocker/Critical/Major isuues должны быть по нулям, RCI как можно ближе к 100%).
6. Прийти на занятие и защитить свою работу.

> ⚠ **Необходимым** (*но не достаточным!*) условием получения итоговой оценки по предмету более 60 является на момент получения зачета для всех л.р.:
>
> 1. присутствие в репозитории
> 2. успешная сборка в Jenkins
> 3. выведенные в ноль issues в Sonar

Вопросы (допуск)

1. Что такое перекрытие метода, перегрузка.
2. Что такое полиморфизм, как его реализовать.
3. Существует ли множественное наследование в Java?
4. Какие типы могут быть унаследованы?
5. Что такое наследование, ключевые слова implements, extends.
6. Что такое инкапсуляция, для чего предназначена, как реализовать.
7. Ключевое слово final, контекст использования.
8. Конструкторы, их назначение и отличие от методов.
9. Ограничение при перекрытии метода.
10. Блоки инициализации, какие бывают.
11. Порядок вызова конструкторов, блоков инициализации при создании объекта.
12. Вложенные классы, анонимные классы, написать пример.

13. В чем отличие вложенных классов от внутренних.

Пример Demo

Demo.main

Demo.main

```
public static void main(String[] args) {
    System.out.println("===== Circle");
    Circle.main(args);

    System.out.println("===== Matrix");
    Matrix.main(args);

    System.out.println("===== ListImplTest");
    ListImpl.main(args);
}
```

Результат (должен быть ровно таким)

```
===== Circle
~~~ c
Circle (0.0, 0.0, 1.0)
~~~ c.move(1, 1)
Circle (1.0, 1.0, 1.0)
~~~ c.isInside(1, 1)
true
~~~ c.isInside(10, 1)
false
~~~ c2
Circle (1.0, 1.0, 2.0)
~~~ c.isInside(c2)
false
~~~ c2.isInside(c)
true
===== Matrix
~~~ m
1.0 2.0 3.0
4.0 5.0 6.0
~~~ m2
1.0 2.0 3.0
4.0 5.0 6.0
~~~ transpose m
1.0 4.0
2.0 5.0
3.0 6.0
~~~ mul m on m2
17.0 22.0 27.0
22.0 29.0 36.0
27.0 36.0 45.0
~~~ mul m2 on 2
2.0 4.0 6.0
8.0 10.0 12.0
===== ListImplTest
~~~ list A B C
~~~ Result: [A, B, C]
[A, B, C]
~~~ list2: D E F
~~~ Result: [D, E, F]
[D, E, F]
~~~ list.addAll(list2)
~~~ Result: [A, B, C, D, E, F]
[A, B, C, D, E, F]
~~~ list.add(C)
~~~ Result: [A, B, C, D, E, F, C]
[A, B, C, D, E, F, C]
~~~ list.clear()
~~~ Result: []
[]
~~~ list.addAll(list2)
~~~ Result: [D, E, F]
[D, E, F]
~~~ list.contains(E)
~~~ Result: true
true
~~~ list.contains(C)
~~~ Result: false
false
~~~ list.indexOf(D)
~~~ Result: 0
0
~~~ list.get(2)
~~~ Result: F
F
~~~ list.indexOf(F)
~~~ Result: 2
2
~~~ list.size()
~~~ Result: 3
3
~~~ list
~~~ Result: [D, E, F]
[D, E, F]
~~~ list.remove(1)
~~~ Result: [D, F]
```

```
[D, F]
~~~ list.remove(F)
~~~ Result: [D]
[D]
~~~ list.size()
~~~ Result: 1
1
~~~ list.addAll(list2)
~~~ Result: [D, D, E, F]
[D, D, E, F]
~~~ foreach list
~~~ Result: D D E F
D D E F
~~~ Iterator it = list.iterator()
~~~ it.next()
~~~ Result: D
D
~~~ it.next()
~~~ Result: D
D
~~~ it.remove()
~~~ Result: [D, E, F]
[D, E, F]
~~~ it.next()
~~~ Result: E
E
~~~ it.remove()
~~~ Result: [D, F]
[D, F]
~~~ it.next()
~~~ Result: F
F
~~~ it.remove()
~~~ Result: [D]
[D]
~~~ list.remove(D)
~~~ Result: []
[]
~~~ list.addAll(list2)
~~~ Result: [D, E, F]
[D, E, F]
~~~ foreach list
~~~ Result: D E F
D E F
```