Домашнее задание № 2.

Тема задания: структуры данных и их применение.

Предисловие

В данном задании используются некоторые идеи (с их сильным упрощением) языков описания виртуальной реальности (Virtual Reality Modeling Language (VRML) и Extensible 3D (X3D))[1], которые связаны с моделированием геометрических объектов в пространстве. Вопросы моделирования объектов в пространстве подразумевают определение понятий «*пространство»* и «*система координат»* в нем, что имеет долгую историю - от теории относительности Галилея, до работ А.Пуанкаре и А.Эйнштейна[2]. В данном задании иллюстрируются и используются лишь некоторые упрощенные аспекты этих вопросов.

Предварительные замечания

В принципе, мы можем поставить задачу применительно к пространствам трех различных размерностей: одномерному (1D), двумерному (2D) или трехмерному (3D). Можно было бы попытаться обобщить эти понятия пространств различной размерности при программировании с использованием обобщенных типов (generics). Однако, недостатки имеющихся в Java средств реализации обобщенных типов весьма затрудняют этот подход. Альтернативой является явная передача понятия пространства конкретной размерности к зависящим от этого объектам, либо использование для этого механизма *dependency injection*. Но мы оставим эти вопросы за рамками данного задания и – для определенности – будем считать, что мы работаем только с двумерным пространством.

Более того, положение объекта и его изменения в пространстве определенной размерности определяются не просто его координатами, соответствующими размерности пространства, но и ориентацией объекта в этом пространстве и другими подробностями. Для упрощения задания, мы исключим из рассмотрения вопросы, связанные с ориентацией, поворотами и масштабированием по осям координат, и будем считать, что все перемещения являются поступательными, а перемещаемые объекты являются только точками, положение которых в пространстве полностью определяется их координатами[3].

Основные понятия

1. Для выбранного нами двумерного пространства (2D) координаты определяются парами чисел x и y формата double и представлены инстансами immutable-класса Coord2D:

```
public class Coord2D {
    private final double x;
    private final double y;
    // … implement as appropriate…
}
```

   Инстансы этого класса можно рассматривать как «математические точки» 2D-пространства: они не способны к собственным перемещениям (изменениям своих координат), но могут использоваться для определения перемещения по ним физических точек, значениями позиций

[1] Международные стандарты ISO по 3D-графике. См: www.web3d.org
[2] См., например, https://www.marxists.org/reference/subject/philosophy/works/fr/poincare.htm
[3] Мы максимально упрощаем transformation hierarchy, которая используется в VRML и X3D.

которых они являются. Тип Coord2D – это т.н. *value type*. Соответственно, идентичность математических точек определяется совпадением значений их координат, что необходимо учесть при их реализации.

2. Физические точки определяются классом Point, в котором определяется property «**position**», значением которой является инстанс Coord2D, показывающий координаты текущей позиции физической точки относительно некоторой системы координат. Узнать позицию физической точки можно getter'ом: getPositon(), возвращающим соответствующий объект Coord2D. Изменение значения координат означает перемещение данной физической точки в другую математическую точку и осуществляется setter'ом: setPosition(Coord2D newValue).

3. Каждая система координат определяется своей точкой начала координат – Origin, которая имеет property «**position**» со значением типа Coord2D, показывающим смещение этого начала координат относительно некоторой другой системы координат. Как и физическая точка, точка начала координат (Origin) может перемещаться, изменяя свою позицию.

   Кроме собственной позиции система координат имеет свойство «**children**», значением которого является перечень (множество) систем координат и/или физических точек, позиции которых отсчитываются относительно данной системы координат. Узнать и/или задать значение property «**children**» можно с использованием соответствующего getter'а / setter'а.

4. Мы рассматриваем направленные ациклические графы (**D**irected **A**cyclic **G**raphs - **DAG**s), состоящие из узлов двух типов:
   - Физических точек (объектов типа Point) – это leaf nodes;
   - Систем координат (объекты типа Origin) – это parent nodes.

   Дугами графа (edges) являются связи между его узлами, которые определяются значениями свойств «**children**» в родительских узлах. Эти значения содержат множества ссылок на узлы, позиции которых отсчитываются относительно данного узла, и определяют исходящие дуги графа. Оконечные узлы (физические точки) исходящих дуг не имеют.

   Один и тот же узел графа (любого из двух типов) может присутствовать в наборах children разных родительских узлов графа (иметь несколько входящих дуг графа). Смысл такого присутствия узла означает его повторное использование (*reuse*) в нескольких разных контекстах. Однако, циклы в графе не допускаются. Мы это требуем для того, чтобы можно было обойти все его узлы за конечное число «*проходов в глубину*» по графу ссылок. Процесс обхода узлов графа в глубину мы называем *траверсированием* графа.

   Весть наш граф (DAG) представлен объектом типа Space, который является корневым родительским узлом графа и определяет для всего графа «мировую систему координат», относительно которой отсчитываются позиции всех остальных узлов.

   Ниже, на Рис.1, показан пример простого графа и его интерпретация в 2D-пространстве.

5. Ограничивающий прямоугольник (BoundBox). Физические точки, расположенные относительно некоторой системы координат, определяют в этой системе координат минимальный объем пространства, в который попадают они все – их BoundBox.

   В частности, для двумерного пространства, BoundBox – это прямоугольник, со сторонами, параллельными осям координат, определяемый координатами своей диагонали (точек с минимальными и максимальными координатами), и охватывающий координаты всех ограниченных им точек (включая те, что возможно находятся на его границах). BoundBox единственной точки является «вырожденным»: начало и конец его «диагонали» имеют координаты самой этой точки.

Система координат, представленная объектом типа Origin, имеет property «**bounds**» (read only, setter отсутствует). Тип значения property «**bounds**» - BoundBox. Это – immutable value type, значения которого генерируются по требованию. При вычислении значения bounds данного Origin учитываются все его children: в его BoundBox объединяются BoundBox всех его точек и (рекурсивно) BoundBox'ы дочерних систем координат (все - выраженные в системе координат данного объекта Origin).

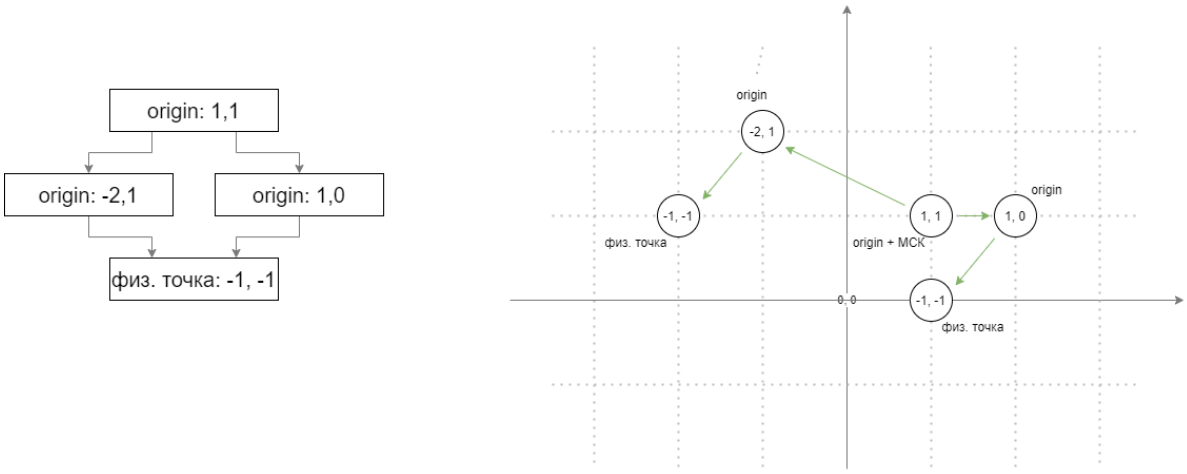

Рис. 1. Пример DAG-графа и его геометрическая интерпретация на плоскости[4].

Постановка задачи

В данном задании требуется реализовать описанные выше понятия в виде библиотеки, обеспечив возможность построения произвольного DAG'а с использованием предоставленного и реализованного API для создания объектов и манипулирования ими. При реализации допускается расширение разрабатываемой библиотеки дополнительными необходимыми средствами.

В частности, необходимо:

1. Реализовать класс Coord2D с операциями, необходимыми для относительных перемещений объектов Point и Origin.
2. Реализовать класс BoundBox с необходимыми операциями для их вычислений, как указано выше.
3. Реализовать класс Point в соответствии с указанным выше описанием, имеющий properties:
   - «**position**», тип значения Coord2D, getter и setter;
   - «**bounds**», тип значения BoundBox, только getter.
4. Реализовать класс Origin в соответствии с указанным выше описанием, имеющий properties:
   - «**position**», тип значения Coord2D, getter и setter;
   - «**bounds**», тип значения BoundBox (с описанной выше семантикой);
   - «**children**», тип значения – параметризация Set<…> для необходимых манипуляций с наборами children;
     Важно: все изменения в children должны контролироваться и не должны приводить к появлению циклов в графе ссылок. При обнаружении нарушения ацикличности в DAG необходимо генерировать exception (DAGConstraintException).
5. Реализовать класс Space как корневой узел DAG, определяющий мировую систему координат.
6. Опционально, для получения оценки, превышающей оценку 8, реализовать утилитный класс (DAGUtils) с методами:

[4] Иллюстрация любезно предоставлена А.Мишуткиным.

- exportAsString (Space o) - для сериализации (сохранения состояния) DAG'a с заданным корнем в строку;
- importFromString (String s) – для десериализации (восстановления состояния) DAG'a из заданной строки.

Формат описания состояния графа в строке выбрать самостоятельно и задокументировать.

Прочие требования

7. Все классы должны сопровождаться своими тестами, реализованными средствами JUnit5. Все public–методы должны быть протестированы. В целом покрытие кода тестами не должно быть менее 75%.
8. Для проверки должен быть предоставлен архив с именем <ФИО>_<группа>_HW2.zip, в котором заархивирована директория maven-проекта с pom.xml и всеми необходимыми для него каталогами, но без папки target (хранящей результаты компиляции и сборки). Распакованный проект должен собираться и тестироваться на машине проверяющего с установленным для проверки JDK17 без дополнительных манипуляций (связанных с подключением библиотек, тестов и т.п.).
9. Требования к кодированию и документированию – стандартные (не использовать default-package, соблюдать java code style, использовать javadocs, соблюдать читаемость кода, мыть руки перед едой и т.п.).

Архив с выполненным заданием загружается в SmartLMS строго до 23:00 3 декабря 2021. Результаты выполнения задания, предоставленные иным способом, не оцениваются (0).