**Lab work 6**

Тема урока: More about containers - LinkedList

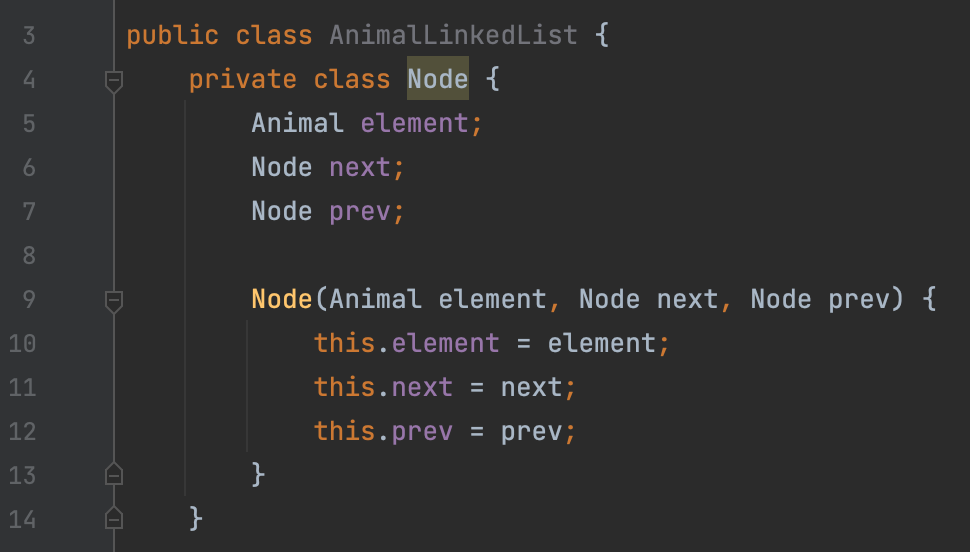
Цель урока:

* Научиться объявлять и имплементировать интерфейсы,
* Ознакомиться со второй реализацией списка - LinkedList,
* Ознакомиться с использованием внутренних классов;

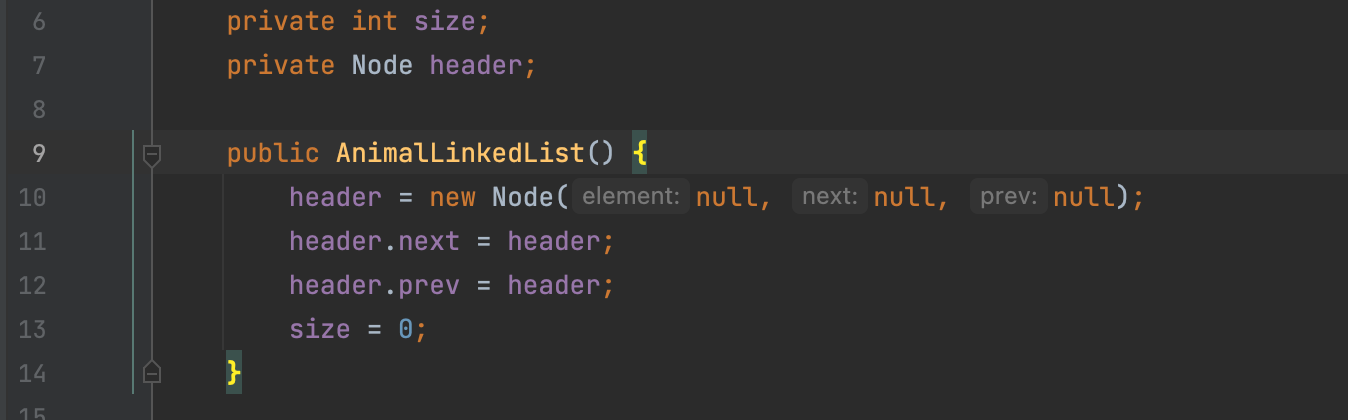
Обобщенный класс LinkedList представляет структуру данных в виде связанного списка. Он реализует интерфейсы List (реализацию ArrayList мы делали на прошлой лабораторной работе), Dequeue и Queue. То есть он соединяет функциональность работы со списком и функциональность очереди. Особенность LinkedArrayList-а в том что он представляет собой цепочку сущностей, где каждый элемент имеет в себе ссылку на следующий и предыдущий элементы.

План урока:

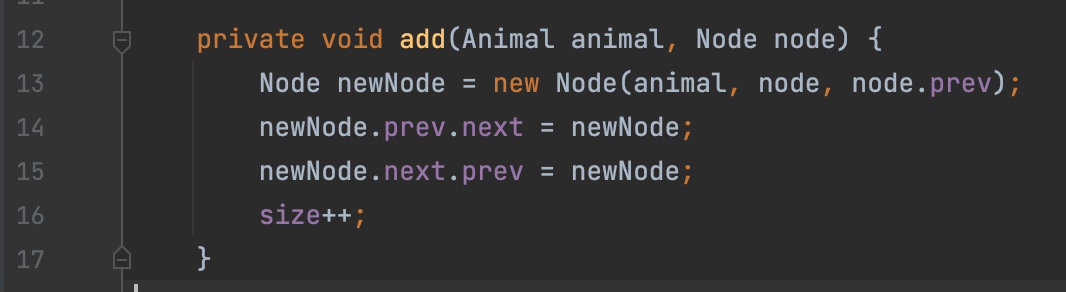
1. Написать реализацию *LinkedList* для объектов класса *Animal* - создать новый класс *AnimalLinkedList*,
2. Создать приватный внутренний класс *Node* внутри класса *AnimalLinkedList*,
   1. Объявить внутри *Node* параметры:
      1. Параметр с именем *element* класса *Animal* (для хранения самого элемента),
      2. Параметр с именем *prev* класса *Animal* (для хранения ссылки на предыдущий элемент),
      3. Параметр с именем *next* класса *Animal* (для хранения ссылки на следующий элемент);
   2. Описать конструктор со всеми параметрами этого класса;



1. Описать параметр обозначающий длину *LinkedList*-а - *private int size;*
2. Описать параметр указывающий на первый элемент в связке нашего *LinkedList*-а;
3. Описать дефолтный пустой конструктор в котором:
   1. Создать объект класса *Node* (корневой) и приравнять его к нашему корневому, вызвав его конструктор и передав в качестве элементов - *null-ы* (так как список у нас еще пустой) и указать в виде ссылки на предыдущий и следующий элементы - себя же,
   2. Указать размер списка равным нулю;



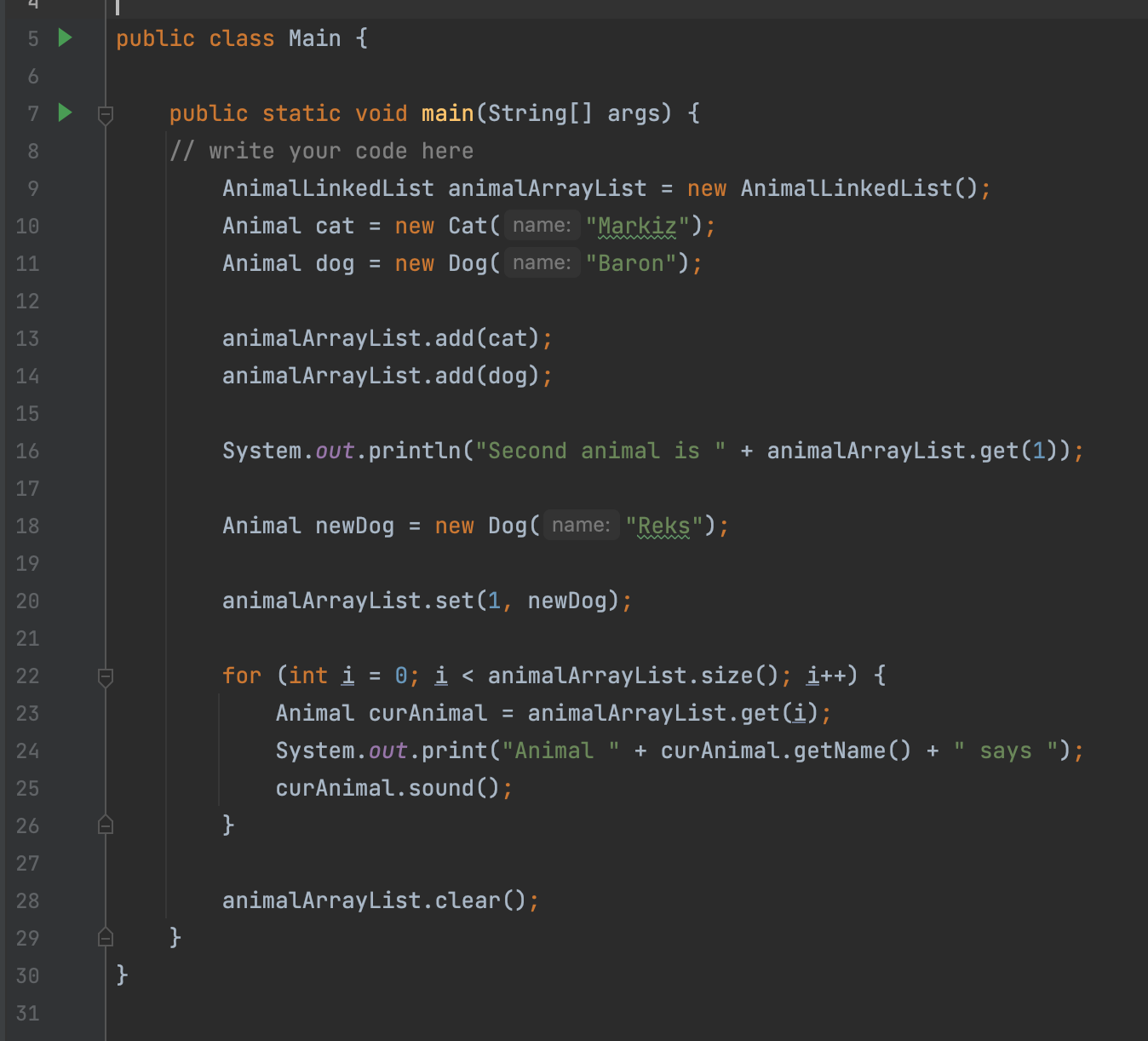
1. Описать метод *private void add(Animal animal, Node node)*, который добавляет новый элемент (*animal*) в цепочку после элементом *node.* Для этого:
   1. Создать новый объект класса *Node*, где в конструкторе указать предыдущим элементом - элемент *node*, а последующим элементом - следующий элемент у *node*. Таким образом вы создали ссылки у нового элемента на другие элементы.
   2. Теперь же вам надо создать ссылки в других элементах на ваш новый элемент для полной связки в вашей цепочке;
   3. Увеличить размерность цепочки на 1;



1. Описать метод *public void add(Animal animal)*, который добавляет новый элемент (*animal*) в цепочку после элемента *header* (параметр вашего класса). Примечание - использовать внутри этого метода вызов описанного вами метода выше *private void add(Animal animal, Node node)*.
2. Описать метод *public void addAll(AnimalArrayList arrayList)*, который добавляет все элементы из входного списка (*arrayList*) в цепочку после элемента *header* (параметр вашего класса). Примечание - использовать внутри этого метода вызов описанного вами метода выше *private void add(Animal animal, Node node)*.
3. Описать метод *private Node findNodeByIndex(int i)*, который будет идти по ссылкам внутри цепочки *i*-раз и возвращать *i*-тый элемент в цепочке (чтобы пробежать по всей цепочку надо вызывать параметр *next* каждого элемента начиная с корневого - *header*).

**Дополнительно** (для оценки 100): оптимизировать метод с поиском элемента если индекс находится во второй половине цепочки.

1. Описать метод *public Animal get(int index)*, который будет вызывать приватный метод f*indNodeByIndex(int i)* и вернуть его внутреннее значение типа *Animal.*
2. Написать метод: *public void set(int index, Animal element)*, который будет сохранять в индексе *index* значение *element*, для этого надо проверить есть ли у вас элемент по индексу *index*, и если находится, заменить его на новый элемент (восстановив связку в цепочке, указав у элемента *index - 1*, *next = element*, и у элемента по индексу *index* указать *prev* = *element*, также у нового элемента сделать связку *prev* и *next* к этим двум соседним элементам);
3. Написать метод возвращающий размер size: *public int size()*;
4. Написать метод проверки на пустоту: *public boolean isEmpty()*, проверяющий заполненность и возвращающий true если наша цепочка (список) пустая;
5. Написать метод: *public void clear()*, который будет очищать наш список (цепочку);
6. Написать в классе Main:



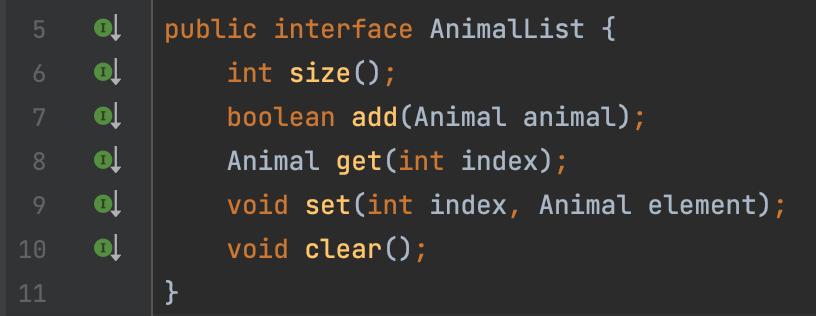
Обратите внимание что код в Main классе схож с кодом прошлой лабораторной работы, мы реализовали точно такие же методы (названия и входные параметры), т.е. мы можем сказать что поведение этих двух классов для конечного пользователя одинаково и мы можем вывести описание их поведений в отдельный ***интерфейс***.

*Механизм наследования очень удобен, но он имеет свои ограничения. В частности мы можем наследовать только от одного класса, в отличие, например, от языка С++, где имеется множественное наследование.*

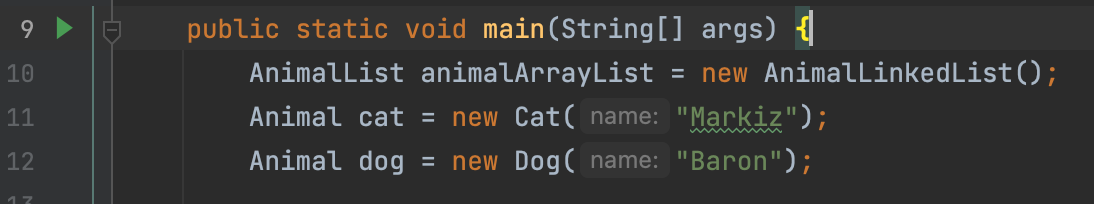
*В языке Java подобную проблему частично позволяют решить интерфейсы. Интерфейсы определяют некоторый функционал, не имеющий конкретной реализации, который затем реализуют классы, применяющие эти интерфейсы. И один класс может применить множество интерфейсов.*

1. Создайте новый файл (*interface*) - *public interface AnimalList*. Напишите описание всех общих публичных методов (тип ответа, название, входные параметры),

Class AnimalArrayList implements AnimalList { …



1. Измените первую строку в методе main():



Теперь вы можете обращаться с *animalArrayList* как с реализацией интерфейса *AnimalList*;

1. Весь проект заархивировать (zip) и отправить на почту преподавателя - [askhayerga@gmail.com](mailto:askhayerga@gmail.com), подписать имя, фамилию, группу, номер лабораторной, дедлайн до следующей недели.