Множество.

Один математик на пенсии увлекается коллекционированием билетиков. Его мечта - собрать все номера билетов с 1 и до 999999. У него уже довольно многочисленная коллекция и ему трудно проверять новые билеты на уникальность. Реализуйте ему программу, с помощью которой он мог бы это делать легко.

Класс содержит в себе целые числа. Работает по принципу стека и не содержит в себе повторяющиеся числа. Реализовать функции:

\* Добавление элемента

\* Просмотра всех элементов в списке

\* Удаления элемента

\* Проверки на существование элемента в списке

\* Проверки на то, можно ли из элементов множества составить непрерывную цепочку подряд идущих элементов.

\* Вывода количества билетов, необходимых для полноты коллекции.

Реализация.

Запустив среду программирования, я создал новый Java Project назвал его "Set(Ds)". Далее я создал новый пакет "Set" и в нем два класса "Main" и "Set".

В классе "Set" будет реализован сам список. В задании указано, что список надо реализовать по принципу стека. Перед реализацией немного о стеке. **Стек** (англ. *stack* — стопка) — структура данных с методом доступа элементам **LIFO** (англ. Last In — First Out, «последним пришёл — первым вышел»). Наиболее распространены две принципа реализации стека: на базе массива и на базе связанного списка. В первом случае для хранения значений в памяти отводится сплошной массив ячеек, которые используются по мере необходимости. Во втором — для каждого элемента стека заказывается блок памяти, достаточный для хранения значения и ссылок на предыдущий и следующий элементы стека. Реализация на базе массива проще, эффективнее и экономичнее по расходу памяти, но она требует заранее знать предельный размер стека и может приводить к трудно выявляемым ошибкам. Реализация на базе списков более надежна, но менее эффективна. В моем задании я выбрал второй способ реализации. Для того чтобы реализовать связанный список, создаем внутренний класс "Node" , который содержит значение(в данном случае номер билета) и ссылку на следующий элемент. Также заводим переменную header с типом переменных Node для хранения начала списка. Далее идет реализация методов : добавление элемента ; просмотра всех элементов в списке ;удаления элемента; проверки на существование элемента в списке; проверки на то, можно ли из элементов множества составить непрерывную цепочку подряд идущих элементов; вывода количества билетов, необходимых для полноты коллекции. Первый метод добавления элемента называется add(int number). В параметр передается значение (номер билета). Перед добавлением выполняется метод check(int number), который проверяет наличие добавляемого элемента в списке. Рассмотрим этот метод подробней. Здесь заводиться новая переменная check с типом переменных boolean, который в начальный момент времени равен false. Далее мы создаем еще одну переменную типа Node , который в начальный момент времени равен header, то есть началу списка. Потом мы проверяем пустой наш список или содержит в себе что-то. Если наш список не пустой, идем дальше и заводим цикл, который будет выполнятся до тех пор, пока мы не дойдем до конца списка или не найдем этот элемент. Если мы находим наш элемент в списке то меняем check на true и выходим из цикла. Если же мы не находим наш элемент или список пустой, check останется false. В условии задачи сказано, что в списке не должны храниться одинаковые элементы, поэтому добавляется только новые элементы. Итак после проверки, если выполняется условие, мы добавляем этот элемент в наш список. Это происходит следующим образом: создается новый узел, потом в ячейку, где хранится номер, записывается значение(номер билета), далее идет проверка: если начало списка ссылается на Null(список пустой),то ячейка, где хранится ссылка на следующий элемент, будет ссылаться на Null, потому в нашем списке только один элемент, далее мы указываем начало списка, он будет ссылаться на наш новый узел, в противном случае, если наш список уже содержит что-то, мы меняем ссылки, ссылка на следующий элемент в нашем новом узле будет ссылаться на начало списка, а начало переместиться на этот новый узел. Последние наши действия означают, что мы каждый раз при добавлении нового элемента сдвигаем наш список. Тот элемент, который пришел последним становится на первое место. Если же при добавлении нового элемента мы найдем его в нашем списке, то мы просто не выполняем никаких действий, добавления не происходит. Далее я прописал стандартный метод remove(), который возвращает нам первый элемент списка(получается элемент, добавленный последним) и удаляет его из нашего списка. Начало списка переместиться на второй элемент(идущий после начального), а ссылку на следующий элемент первого узла приравниваем к Null. Следующий метод show(), который выводит нам элементы списка. В этом методе заводится переменная типа Node, которая поможет перемещаться по списку. В начальный момент времени он равен header, то есть началу списка. Заводиться цикл, который будет выполняться до тех пор, пока мы не достигнем последнего элемента, внутри цикла выводим значение (номер билета).

Далее я реализовал метод удаления элемента по значению, я назвал его delete(int number).Аналогично, как в методе вывода, заводиться переменная temp типа Node. Проверяется наличие в списке элементов, если список пуст ничего не делаем, в противном случае мы опять выполняем метод check(number), если элемент, который мы хотим удалить есть в списке, производим удаление. Удаление производится следующим образом: проверяется условие: если элемент, который мы хотим удалить, находится в начале списка, то начало списка(header) сдвигается на следующий элемент, а ссылка на следующий элемент, удаляемого узла, приравнивается к Null. В противном же случае заводим цикл, который передвигается от начала списка до конца и ищет удаляемый элемент. Причем метка temp будет находится не на самом удаляемом элементе, а перед ним. Здесь возможно два случая: когда после удаляемого элемента есть еще элементы и удаляемый элемент находится в конце. В первом случае заводим переменную current типа Node, который будет хранить удаляемый узел. Узел перед удаляемым элементом соединяем с узлом после удаляемого элемента, потом сам удаляемый узел приравниваем к Null. Сразу же после выполнения условия выходим из цикла с помощью break. Во втором случае ссылка на следующий элемент узла перед удаляемым элементом приравнивается к Null. Также выходим из цикла с помощью break. Далее идет метод comletenessCheck(), который возвращает количество элементов(билетиков) в списке и выводит сколько билетиков необходимо для полноты коллекции. Этот метод работает по такому же принципу как метод show() , просто в новой переменной n считается количество элементов в списке. Для реализации метода checkChain(), который проверяет возможность построить из элементов множества непрерывную цепочку подряд идущих элементов, понадобилось создать еще один метод min() , который находит минимальный элемент в списке. Метод min() заводит переменную min, которая изначально равна первому элемента списка , производится обход списка, каждый элемент списка сравнивается с min и если он меньше, то становится минимальным. С этим понятно, теперь вернемся к методу checkChain(). Здесь заводятся две булевских переменные check, который изначально равен true и checkS, равный false. Заводится переменная number, в которую записывается минимальный элемент, далее переменная amt, в которую записывается количество элементов без первого. Далее заводится цикл, который будет работать до тех пор, пока не будут просмотрены все элементы, начиная со второго (только один раз, то есть сколько элементов без первого , столько раз работает цикл)или пока не нарушиться порядок. Внутри этого цикла задается еще один цикл, который будет пробегать по списку до конца или пока не найдет элемент, больше предыдущего на один. Если найдется элемент, который больше предыдущего(то есть чем number ) на один, то значение number меняется на этот найденный элемент и внешний цикл продолжает работать, а если же такого элемента нет, то цикл останавливается и получается, что из элементов данного множества нельзя составить непрерывную цепочку подряд идущих элементов. Далее создаем класс "Main", в котором создаем объект этого класса и вызываем нужные нам методы.