**Класс Collections**

Наиболее распространенные операции, которые совершаются при использовании коллекций, реализованы в специальном статическом классе **Collections**.

“Коллекции” — общее название для нескольких структур данных в Java. Данные можно хранить многими разными способами. Мы пока изучили только [класс ArrayList](https://javarush.ru/quests/lectures/questsyntax.level07.lecture05), где данные хранятся в массиве. С остальными коллекциями мы познакомимся позднее. Сейчас достаточно понимать что класс Collections рассчитан на работу не только с ArrayList, но и с другими видами коллекций (отсюда, собственно, и его название). Итак, какие же задачи при работе с ArrayList позволяет решить класс Collections? Первое и самое очевидное — сортировка. В лекции о массивах мы рассматривали пример с числами, а теперь рассмотрим пример со строками. Для сортировки содержимого коллекций в классе Collections реализован метод **sort()**:

**public** **class** Main {

**public** **static** **void** main(java.lang.String[] args) {

String mercury = **new** String("Меркурий");

String venus = **new** String("Венера");

String earth = **new** String("Земля");

String mars = **new** String("Марс");

String jupiter = **new** String("Юпитер");

String saturn = **new** String("Сатурн");

String uranus = **new** String("Уран");

String neptune = **new** String("Нептун");

ArrayList<String> solarSystem = **new** ArrayList<>(Arrays.asList(mercury, venus, earth, mars, jupiter, saturn, uranus, neptune));

Collections.sort(solarSystem);

System.out.println(solarSystem);

}

}

**Вывод: *[Венера, Земля, Марс, Меркурий, Нептун, Сатурн, Уран, Юпитер]*** Строки были отсортированы в алфавитном порядке! Почему именно в алфавитном? В классе String запрограммировано, как именно строки сравниваются между собой (как раз — по алфавиту). Для классов, которые ты будешь создавать сам, можешь реализовать свой механизм сравнения, но об этом мы поговорим в других лекциях. Кроме того, класс Collections позволяет найти минимальный и максимальный элемент в ArrayList. Это делается с помощью методов min() и max():

**public** **static** **void** main(java.lang.String[] args) {

ArrayList<Integer> numbers = **new** ArrayList<>(Arrays.asList(1,2,3,4,5,6,7));

System.out.println(Collections.max(numbers));

System.out.println(Collections.min(numbers));

}

**Вывод: *7 1*** Это, конечно, гораздо удобнее, чем вручную писать код для прохождения по всем элементам и поискам наибольшего/наименьшего элемента :) Еще один крайне полезный метод — **reverse()**. Если бы нам нужно было “перевернуть” список, чтобы элементы шли в обратном порядке — как бы мы это делали? Вероятно, написать такой алгоритм самому было бы не так просто :) К счастью, метод **reverse()** уже это умеет. Например, нам не нравится как метод **sort()** отсортировал наши планеты в алфавитном порядке, и мы хотим изменить порядок на обратный — от Я до А:

**public** **class** Main {

**public** **static** **void** main(java.lang.String[] args) {

String mercury = **new** String("Меркурий");

String venus = **new** String("Венера");

String earth = **new** String("Земля");

String mars = **new** String("Марс");

String jupiter = **new** String("Юпитер");

String saturn = **new** String("Сатурн");

String uranus = **new** String("Уран");

String neptune = **new** String("Нептун");

ArrayList<String> solarSystem = **new** ArrayList<>(Arrays.asList(mercury, venus, earth, mars, jupiter, saturn, uranus, neptune));

Collections.sort(solarSystem);

Collections.reverse(solarSystem);

System.out.println(solarSystem);

}

}

**Вывод: *[Юпитер, Уран, Сатурн, Нептун, Меркурий, Марс, Земля, Венера]*** Кстати, мы тут часто говорим о сортировке, порядке элементов и т.д. *А что, если задача у нас будет прямо противоположная?* Например, мы пытаемся реализовать механизм работы лотереи. Мы добавили в барабан 100 чисел, которые по одному должны появляться на экране. Кто из участников первым зачеркнет все числа на своем билете — побеждает. Реализовать такой механизм очень легко с помощью метода **shuffle()**:

**public** **class** Main {

**public** **static** **void** main(java.lang.String[] args) {

ArrayList<Integer> lottery = **new** ArrayList<>(100);

**for** (**int** i = 1; i <= 100; i++) {

lottery.add(i);//добавляем в барабан числа от 1 до 100

}

Collections.shuffle(lottery);//перемешиваем

System.out.println("Внимание! Из барабана появляются первые 10 чисел!");

**for** (**int** i = 0; i < 10; i++) {

System.out.println(lottery.get(i));

}

}

}

Вывод: ***Внимание! Из барабана появляются первые 10 чисел! 32 61 4 81 25 8 66 35 42 71*** Вот так просто! Задача решена, а наш кусочек игры написан :) Теперь представим другую ситуацию. Ранее мы создали список solarSystem с перечисленными в нем планетами. И он нас, вроде как, всем устраивает, если бы не одно но: из него можно удалять элементы и добавлять новые! Это явно не то поведение, которое мы ждем: Солнечная система в нашей программе должна быть в неизменном состоянии. В классе Collections есть очень интересный метод — **unmodifiableList()**. Он создает из переданного списка его неизменяемый вариант. В него нельзя будет ни добавить, ни удалить элемент. В случае со списком планет Солнечной системы это именно то, что нам нужно!

**public** **class** Main {

**public** **static** **void** main(java.lang.String[] args) {

String mercury = **new** String("Меркурий");

String venus = **new** String("Венера");

String earth = **new** String("Земля");

String mars = **new** String("Марс");

String jupiter = **new** String("Юпитер");

String saturn = **new** String("Сатурн");

String uranus = **new** String("Уран");

String neptune = **new** String("Нептун");

List<String> solarSystem = Collections.unmodifiableList(**new** ArrayList<>(Arrays.asList(mercury, venus, earth, mars,

jupiter, saturn, uranus, neptune)));

solarSystem.add("Плутон");//попробуем добавить новый элемент

}

}

***Exception in thread "main" java.lang.UnsupportedOperationException at java.util.Collections$UnmodifiableCollection.add(Collections.java:1075) at Main.main(Main.java:21)*** Ошибка в solarSystem теперь нельзя ничего добавлять! Единственное, на что в данном случае нужно обратить внимание — тип переменной должен быть List<>, а не ArrayList<> (данный метод возвращает объект именно такого типа, общего для всех видов списков). Еще одна рядовая ситуация, которая может произойти во время работы — программист добавил элементы в неправильном порядке. Если такое произошло, и Меркурий и Нептун неожиданно поменялись местами — исправить эту оплошность нам поможет метод **swap()**:

**public** **class** Main {

**public** **static** **void** main(java.lang.String[] args) {

String mercury = **new** String("Меркурий");

String venus = **new** String("Венера");

String earth = **new** String("Земля");

String mars = **new** String("Марс");

String jupiter = **new** String("Юпитер");

String saturn = **new** String("Сатурн");

String uranus = **new** String("Уран");

String neptune = **new** String("Нептун");

ArrayList<String> solarSystem = **new** ArrayList<>(Arrays.asList(neptune, venus, earth, mars

, jupiter, saturn, uranus, mercury));// неправильный порядок планет

System.out.println(solarSystem);

Collections.swap(solarSystem, solarSystem.indexOf(mercury), solarSystem.indexOf(neptune));

System.out.println(solarSystem);

}

}

В метод swap() мы передали наш список, а также индексы двух элементов, которые нужно поменять местами. Обрати внимание: метод работает именно с индексами, а не со ссылками. Поэтому здесь нам понадобился метод **ArrayList.indexOf()**. **Вывод: *[Нептун, Венера, Земля, Марс, Юпитер, Сатурн, Уран, Меркурий] [Меркурий, Венера, Земля, Марс, Юпитер, Сатурн, Уран, Нептун]*** Напоследок познакомимся с очень интересным методом — **disjoint()**. Он проверяет, есть ли у двух коллекций пересечения, то есть **хоть один одинаковый элемент**. Если нет — возвращает true, если есть — false.

**public** **class** Main {

**public** **static** **void** main(java.lang.String[] args) {

String mercury = **new** String("Меркурий");

String venus = **new** String("Венера");

String earth = **new** String("Земля");

String mars = **new** String("Марс");

String jupiter = **new** String("Юпитер");

String saturn = **new** String("Сатурн");

String uranus = **new** String("Уран");

String neptune = **new** String("Нептун");

ArrayList<String> solarSystemPart1 = **new** ArrayList<>(Arrays.asList(mercury, venus, earth, mars));

ArrayList<String> solarSystemPart2 = **new** ArrayList<>(Arrays.asList(jupiter, saturn, uranus, neptune));

System.out.println(Collections.disjoint(solarSystemPart1, solarSystemPart2));

}

}

Как видишь, в наших двух списках элементы полностью разные, поэтому программа выводит true. Вот такой интересный и очень полезный класс. Как и Arrays, он выполняет вместо нас много рутинной, черновой работы, позволяя сосредоточиться на других вещах.