Название темы: **Компаратор и хеш-функция**

Описание темы: При работе с коллекциями часто приходится сравнивать и сортировать данные, которые в них хранятся. Т.к. Java объектно-ориентированные язык, то в коллекциях очень часто хранятся данные в виде объектов, и перед программистом периодически возникает задача сравнения объектов по какому-либо принципу. В этом видео мы разберем основные методы сравнивания объектов между собой и рассмотрим способы сортировки данных.

Тезисы: Компаратор. Метод equalsTo, hashCode

Текст занятия:

Для того чтобы отсортировать контейнер, нужно уметь сравнивать элементы между собой. Для примитивных числовых типов все естественно: числа можно сравнивать «как в математике», а у объектов нужно указать способ, ввести порядок, упорядочить их. Например, как сортировать список сотрудников? Возможно, по фамилиям, а, возможно, по зарплате. При этом какого сотрудника поставить в начало списка с наибольшей или, наоборот, с наименьшей зарплатой?

Для упорядочивания объектов в языке Java есть два способа: «научить» сами объекты класса сравниваться между собой (реализовать в классе интерфейс `Comparable` и создать еще один класс, реализующий интерфейс `Comparator` — «сравниватель»), такие классы называют *компараторами*.

Рассмотрим подробно второй вариант.

Интерфейс `Comparator` обязывает определить метод сравнения:

```
compare(Object obj1, Object obj2)
```

Метод `compare(Object obj1, object obj2)` должен возвращать:

- **отрицательное число, если `obj1` считается меньше, чем `obj2`;**
- **0, если они считаются равными;**
- **положительное число, если `obj1` считается больше, чем `obj2`.**
-

У классов `Arrays` и `Collections` есть соответствующие методы `sort`(), которые принимают объект компаратора последним параметром.

Параметром метода сортировки с использованием компаратора обязательно должен быть наследник класса `Object`, а не примитивный тип.

Рассмотрим использование компаратора:

Отсортировать массив по возрастанию значений младшей цифры элементов массива при помощи функции `sort()` класса `Arrays`.

Для такой сортировки создадим компаратор. Будем использовать объекты класса `Integer`.

```
class LastDigitComp implements Comparator<Integer> {
        @Override
        public int compare(Integer obj1, Integer obj2) {
            // получаем последние цифры чисел
            int m1 = obj1 % 10;
```

```
                int m2 = obj2 % 10;
                // и сравниваем их
                if (m1 < m2)
                    return -1;
                else if (m1 > m2)
                    return 1;
                else
                    return 0;
            }
        }
```

Теперь создадим массив и заполним его случайными числами:

```
final int MAXRANGE = 100;
Random rnd  = new Random();
Integer[] a = new Integer[10];
for (int i = 0; i < a.length; ++i) {
     a[i] = rnd.nextInt(MAXRANGE);
}
```

Отсортируем массив (и выведем его до и после сортировки):

```
System.out.println("Random: " + Arrays.toString(a));
Arrays.sort(a, new LastDigitComp());
System.out.println(Arrays.toString(a));
```

На экране появится:

Random: [56, 36, 92, 38, 76, 34, 52, 62, 86, 93]

Sorted:[92, 52, 62, 93, 34, 56, 36, 76, 86, 38]

Изменяя условия в компараторе, мы можем менять порядок сортировки массива (например, поменяв местами команды `return -1` и `return 1`, мы изменим сортировку по возрастанию на сортировку по убыванию значений младшей цифры элементов массива).

hashCode() и equals()

Методы `hashCode()` и `equals()` — базовые методы языка Java. На их основе работают коллекции. Оба эти метода объявлены в классе `java.lang.Object`.

Дочерние классы могут, а в некоторых случаях даже должны переопределять их. Про эти методы я уже как-то писал, но нужно упомянуть об этом ещё раз.

Метод `hashCode()`

Объявление в классе `Object`:

```
public int hashCode();
```

Метод `hashCode()` возвращает число, которое является хеш-кодом объекта. Реализация по умолчанию в классе `Object` обычно возвращает адрес объекта, но это в спецификации Java это не закреплено, так что некоторые реализации Java-машин вполне могут возвращать что-нибудь другое.

Метод `equals()`

Объявление в классе Object:

```
public boolean equals(Object obj)
```

Возвращает `true`, если `obj` равен этому объекту и `false` в противном случае.

Реализация по умолчанию в классе `Object` просто сравнивает ссылки на объекты, то есть возвращает `true` в том случае, если обе ссылки указывают на один и тот же объект.

Соглашение между реализациями hashCode() и equals()

В большинстве случаев реализации `hashCode()` и `equals()`, которую ваши классы наследуют от `Object` вам не подойдёт, так как вам нужно, чтобы при вызове `equals()` сравнивались не ссылки на объекты, а значения полей объектов.

Именно поэтому вам нужно будет переопределять `equals()` для тех классов, которые будут использоваться в качестве ключей коллекций `Map` и `Set`. При этом нужно иметь в виду, что при переопределении `equals()` нужно всегда переопределять `hashCode()` так, чтобы сохранялись следующие соглашения:

Если `x.equals(y)` возвращает `true`, то `hashCode`() у обоих экземпляров объектов должны возвращать одинаковые значения.

Но если x`.hashCode() == y.hashCode()`, то вовсе не обязательно, чтобы `x.equals(y)` возвращало `true`, оно может возвращать как `true`, так и `false`.

Как писать `hashCode()` и `equals()`:

В большинстве IDE уже есть готовые генераторы `hashCode()` и `equals()`, где вам нужно будет только указать поля, которые необходимо учитывать при генерации кода этих методов.