**https://vertex-academy.com/tutorials/ru/interfejsy-comparable-comparator-java/**

**Что такое Comparable и Comparator в Java**

В своей работе программисты часто сталкиваются с тем, что надо что-то сортировать - например, покупая товары в интернет-магазине Вы можете сортировать их по **цене**, **популярности** и т.д.

Но для того, чтобы что-то отсортировать, нам нужно сравнивать объекты по каким-то правилам. Тут, казалось бы, все просто - мы можем сортировать числа, да и в сортировке по алфавиту нет ничего сложного. Да, с такими данными все легко. Но как нам сравнить два объекта класса **Car**? По цене, пробегу, лошадиным силам или дате выпуска? А может по количеству владельцев?



Если у нас есть два объекта класса **Cat** - как сравнить их? По кличке? По породе? По возрасту?



Как видите, не всегда очевидно как именно можно сравнить два объекта. Но не беда - **мы сами можем прописать эти правила**. Именно для этого мы можем реализовать интерфейсы **Comparable** и **Comparator**.

* Кроме того, некоторые встроенные возможности в **Java** можно использовать только, если Ваш класс реализует **Comparable** или **Comparator**.

**Интерфейс Comparable**

С английского "**Comparable**" переводится как "сравнимый". Имплементируя этот интерфейс мы как бы говорим "Эй, теперь объекты этого класса можно сравнивать между собой! И я знаю, как это сделать!" А до этого было нельзя

Так как выглядит интерфейс **Comparable**? Очень просто - в нем находится всего один метод:



|  |  |
| --- | --- |
| 1  2  3 | public interface Comparable<T> {      public int compareTo(T o);  } |

Как видите, если мы реализуем этот интерфейс нам придется определить только один метод - **compareTo(T o)**. С английского "**compareTo**" переводится как "сравнить с". Именно этот метод буде использоваться во всяких сортировках.

Вы могли заметить, что метод **compareTo(T o)**возвращает **int**. Он возвращает:

* ноль, если два объекта равны;
* число >0, если первый объект (на котором вызывается метод) больше, чем второй (который передается в качестве параметра);
* число <0, если первый объект меньше второго.

Давайте посмотрим на примере. Представим, что мы хотим сравнить два дома.

Давайте у нас будет класс **House**:



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | public class House {      int area;      int price;      String city;      boolean hasFurniture;  } |

Как Вы можете видеть, у нас есть четыре параметра - размер дома, цена, город, в котором дом находится, и **boolean** "**hasFurniture**" ("продается ли дом с мебелью"). Мы НЕ сделали эти переменные приватными чтобы не отягощать код и не писать гетеры и сеттеры на каждый параметр - но Вы для практики можете это сделать  Давайте просто добавим конструктор и метод "вывести все параметры":



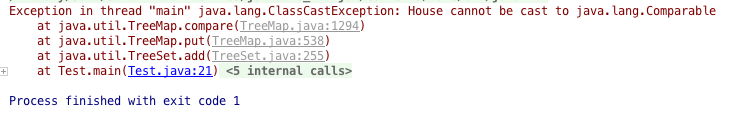
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | public class House {      int area;      int price;      String city;      boolean hasFurniture;        public House(int area, int price, String city, boolean hasFurniture)      {          this.area = area;          this.price = price;          this.city = city;          this.hasFurniture = hasFurniture;      }        @Override      public String toString() {          final StringBuffer sb = new StringBuffer("House{");              sb.append("area=").append(area);              sb.append(", price=").append(price);              sb.append(", city='").append(city).append('\'');              sb.append(", hasFurniture=").append(hasFurniture);              sb.append('}');          return sb.toString();      }  } |

Отлично! Но пока мы не можем сравнить два объекта типа **House**. Например, если мы попробуем создать **TreeSet** из объектов типа **House** (а **TreeSet**всегда сортирует свои элементы. О TreeSet можете почитать в [нашей статье о коллекциях](https://vertex-academy.com/tutorials/ru/set-v-java-hashset/)) и добавить туда элемент:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | public class Test {        public static void main(String[] args) {           TreeSet<House> myHouseArrayList = new TreeSet<House>();            House firstHouse = new House(100, 120000, "Tokyo", true);            myHouseArrayList.add(firstHouse);      }  } |

получим ошибку:



Как мы уже говорили, **TreeSet**сортирует свои элементы - но в данном случае сортировать у него не получится, поскольку он не знает, по какому критерию нужно сортировать

Теперь, давайте имплементируем  **Comparable**:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | public class House implements Comparable<House>{      int area;      int price;      String city;      boolean hasFurniture;        public House(int area, int price, String city, boolean hasFurniture)      {          this.area = area;          this.price = price;          this.city = city;          this.hasFurniture = hasFurniture;      }        @Override      public String toString() {          final StringBuffer sb = new StringBuffer("House{");              sb.append("area=").append(area);              sb.append(", price=").append(price);              sb.append(", city='").append(city).append('\'');              sb.append(", hasFurniture=").append(hasFurniture);              sb.append('}');          return sb.toString();      }        public int compareTo(House anotherHouse)      {          if (this.area == anotherHouse.area) {              return 0;          } else if (this.area < anotherHouse.area) {              return -1;          } else {              return 1;          }      }  } |

Как видите, мы решили сортировать дома по площади.

Теперь давайте создадим три объекта **House** и положим их в **TreeSet**:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | public class Test {        public static void main(String[] args) {           TreeSet<House> myHouseArrayList = new TreeSet<House>();            House firstHouse = new House(100, 120000, "Tokyo", true);          House secondHouse = new House(40, 70000, "Oxford", true);          House thirdHouse = new House(70, 180000, "Paris", false);            myHouseArrayList.add(firstHouse);          myHouseArrayList.add(secondHouse);          myHouseArrayList.add(thirdHouse);            for (House h: myHouseArrayList) {              System.out.println(h);          }      }  } |

На экране получим:



|  |  |
| --- | --- |
| 1  2  3  4  5 | Area: 40, price: 70000, city: Oxford, hasFurniture: true  Area: 70, price: 180000, city: Paris, hasFurniture: false  Area: 100, price: 120000, city: Tokyo, hasFurniture: true    Process finished with exit code 0 |

Как видите, наши дома стоят не в порядке добавления (Токио, Оксфорд, Париж), а отсортированы по площади (Оксфорд, Париж, Токио). Ну, и ошибок, естественно, тоже нет.

Кстати, метод **compareTo(T o)**, который требует реализовать интерфейс **Comparable**, часто называют "естественным сравнением" ("*natural comparison method*") - т.е. методом по умолчанию. Основные типы (например, Integer, String, Float) уже имеют свои методы **compareTo(T o)**.

Тем не менее, если Вам нужен "нестандартный" вид сортировки - следует использовать **Comparator**.

**Интерфейс Comparator**

Итак, нестандартная сортировка. Допустим, мы все согласны что логичнее всего сравнивать дома по площади. Ну а если их нужно отсортировать, например, по цене?

Для этой цели мы можем создать отдельный класс, который реализует интерфейс **Comparator**.

Например, у нас уже есть класс **House**. Давайте создадим отдельный класс, которые будут выполнять функцию сравнения - **PriceComparator**:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | public class PriceComparator implements Comparator<House> {        public int compare(House h1, House h2) {          if (h1.price == h2.price) {              return 0;          }          if (h1.price > h2.price) {              return 1;          }          else {              return -1;          }      }  } |

Обратите внимение: мы указываем тип объектов, которые хотим сравнивать (**House**) в скобках после слова "**Comparator**".

Теперь давайте возьмем **main** из предыдущего примера, только поместим наши объекты не в **TreeSet**, а в **ArrayList**:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | public class Test {        public static void main(String[] args) {           ArrayList<House> myHouseArrayList = new ArrayList<House>();            House firstHouse = new House(100, 120000, "Tokyo", true);          House secondHouse = new House(40, 70000, "Oxford", true);          House thirdHouse = new House(70, 180000, "Paris", false);            myHouseArrayList.add(firstHouse);          myHouseArrayList.add(secondHouse);          myHouseArrayList.add(thirdHouse);            for (House h: myHouseArrayList) {              System.out.println(h);          }      }  } |

Если запустить этот код, то мы увидим, что все наши элементы лежат в порядке добавление - т.е. они не отсортированы.

Теперь давайте создадим объект класса **PriceComparator**, а потом вызовем у нашего ArrayList метод sort(), который принимает на вход как раз объект класса, реализующего интерфейс Comparator, в нашем  **PriceComparator**-а отсортируем наш **ArrayList**:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | public class Test {        public static void main(String[] args) {           ArrayList<House> myHouseArrayList = new ArrayList<House>();            House firstHouse = new House(100, 120000, "Tokyo", true);          House secondHouse = new House(40, 70000, "Oxford", true);          House thirdHouse = new House(70, 180000, "Paris", false);            myHouseArrayList.add(firstHouse);          myHouseArrayList.add(secondHouse);          myHouseArrayList.add(thirdHouse);            for (House h: myHouseArrayList) {              System.out.println(h);          }            PriceComparator myPriceComparator = new PriceComparator();          myHouseArrayList.sort(myPriceComparator);            System.out.println("Sorted: ");          for (House h: myHouseArrayList) {              System.out.println(h);          }      }  } |

В консоли получим:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | Area: 100, price: 120000, city: Tokyo, hasFurniture: true  Area: 40, price: 70000, city: Oxford, hasFurniture: true  Area: 70, price: 180000, city: Paris, hasFurniture: false  Sorted:  Area: 40, price: 70000, city: Oxford, hasFurniture: true  Area: 100, price: 120000, city: Tokyo, hasFurniture: true  Area: 70, price: 180000, city: Paris, hasFurniture: false    Process finished with exit code 0 |

Ура! Наши дома отсортированы по цене. Теперь Вы знаете как использовать **Comparable** и **Comparator.**