



# **Интерфейсы Comparable и Comparator**

## Интерфейсы

- ❑ Интерфейс **Comparable** используется для упорядочивания объектов классов:
  - Он находится в пакете **java.lang** и содержит метод:  
**public int *compareTo*(Object obj)**
  - Сравнивает текущий объект с указанным объектом для определения естественного порядка следования:
    - ✓ Возвращает отрицательное число, ноль или положительное число, если текущий объект меньше, равен или больше, чем указанный объект
  - Используется в методе *Arrays.sort(...)* для сортировки элементов массива по возрастанию согласно установленному порядку.



Объекты пользовательских классов должны реализовать интерфейс **Comparable** для использования метода *Arrays.sort(...)* !

Иначе будет брошено исключение типа **ClassCastException**

## Реализация принципа ООП: наследование



### *Свойства метода*

Для любых ненулевых значений ссылок  $x$ ,  $y$  и  $z$ :

- ❑ *Антикоммутативность*:  $x.compareTo(y)$  имеет противоположный знак с  $y.compareTo(x)$ ;
- ❑ *Симметрия\_исключений*:  $x.compareTo(y)$  бросает те же исключения, что и  $y.compareTo(x)$ ;
- ❑ *переносимость*: если  $x.compareTo(y) > 0$  и  $y.compareTo(z) > 0$ , то  $x.compareTo(z) > 0$  (и то же самое для меньше), а также если  $x.compareTo(y) == 0$ , тогда  $x.compareTo(z)$  имеет тот же знак, что и  $y.compareTo(z)$ ;
- ❑ *согласованность с равными* (настоятельно рекомендуется, но не обязательно):  $x.compareTo(y) == 0$ , если и только если  $x.equals(y)$ .

## Интерфейсы

Пример 2:

```
public class Student implements Comparable {  
    private String firstName;  
    private int group;  
    public Student(String firstName, int group) {  
        this.firstName = firstName;  
        this.group = group;  
    }  
    public String toString() {  
        return "name = " + firstName + ", group = " + group;  
    }  
    public int compareTo(Object o) {  
        if (this.group > ( (Student )o).group) return 1;  
        if (this.group < ( (Student )o).group) return -1;  
        return 0;  
    }  
}
```



Реализация compareTo()

## Интерфейсы

Продолжение примера 2,

```
public class Demo3 {  
    public static void main(String[] arg) {  
        Student[] myStuds = { new Student("Ivan", 302),  
                               new Student("Alex", 105), new Student("Peter", 102),  
                               new Student("Dasha", 504), new Student("Igor", 304)};  
        Arrays.sort(myStuds);  
        for(Student student : myStuds) {  
            System.out.println(student);  
        }  
    }  
}
```

### Console output:

```
name = Peter, group = 102  
name = Alex, group = 105  
name = Ivan, group = 302  
name = Igor, group = 304  
name = Dasha, group = 504
```

## Интерфейсы



- ❑ Для обеспечения многократной различной последовательности сортировки (т.е. сортировки по любым членам данных или не в порядке возрастания) используется интерфейс **Comparator**.
  - Находится в пакете **java.util** и содержит метод:  
**public int *compare*(Object obj1, Object obj2)**
  - Сравнивает объект **obj1** с объектом **obj2** для определения порядка следования:
    - ✓ Возвращает отрицательное число, ноль или положительное целое число, если объект **obj1** меньше, равен или больше, чем объект **obj2**
  - Передается в метод **Arrays.sort(...)** как параметр.



**Определение порядка следования объектов отделяется от описания типа объектов!**

## Интерфейсы

Пример 3:

```
public class Student {  
    private String firstName;  
    private int group;  
    public Student(String firstName, int group) {  
        this.firstName = firstName;  
        this.group = group;  
    }  
    //...  
    public String getFirstName() {  
        return firstName;  
    }  
    public int getGroup() {  
        return group;  
    }  
}
```

## Интерфейсы

Например, сортировка по убыванию значения группы

```
class StudentGroupComparator implements Comparator {  
    @Override  
    public int compare(Object o1, Object o2) {  
        Student stud1 = (Student)o1;  
        Student stud2 = (Student)o2;  
        if (stud1.getGroup() > stud2.getGroup()) return -1;  
        if (stud1.getGroup() < stud2.getGroup()) return 1;  
        return 0;  
    }  
}
```



## Интерфейсы

*Продолжение примера 3,*

```
public class Demo4 {  
    public static void main(String[] arg) {  
        Student[] myStuds = { new Student("Ivan", 302),  
                               new Student("Alex", 105), new Student("Peter", 102),  
                               new Student("Dasha", 504), new Student("Igor", 304)};  
        Arrays.sort(myStuds, new StudentGroupComparator());  
        for(Student student : myStuds) {  
            System.out.println(student);  
        }  
    }  
}
```

Передача компаратора  
для установки порядка  
следования

### Console output:

```
name = Dasha, group = 504  
name = Igor, group = 304  
name = Ivan, group = 302  
name = Alex, group = 105  
name = Peter, group = 102
```

## Интерфейсы

*Например, сортировка по имени студента в алфавитном порядке*

```
class StudentNameComparator
    implements Comparator {
    @Override
    public int compare(Object o1, Object o2) {
        String name1 = ( (Student)o1).getFirstName();
        String name2 = ( (Student)o2).getFirstName();
        return name1.compareTo(name2);
    }
}
```

## Интерфейсы

Продолжение примера 3,

```
public class Demo5 {  
    public static void main(String[] arg) {  
        Student[] myStuds = { new Student("Ivan", 302),  
                               new Student("Alex", 105), new Student("Peter", 102),  
                               new Student("Dasha", 504), new Student("Igor", 304)};  
        Arrays.sort(myStuds, new StudentNameComparator());  
        for(Student student : myStuds) {  
            System.out.println(student);  
        }  
    }  
}
```

### Console output:

```
name = Alex, group = 105  
name = Dasha, group = 504  
name = Igor, group = 304  
name = Ivan, group = 302  
name = Peter, group = 102
```



# **Клонирование** **объектов**

## Клонирование объектов

**Клонирование объекта** - это способ создать точную копию объекта.

Для клонирования объекта в Java можно использовать четыре способа:

- ❑ Переопределить метод *clone()* класса **Object** и реализовать интерфейс **Cloneable**;
- ❑ Использовать конструктор копирования;
- ❑ Использовать механизм сериализации;
- ❑ Использовать статический фабричный метод, возвращающий экземпляр своего класса.



Наиболее удобным и гибким способом клонирования является механизм сериализации, а также безопасным – фабричный метод.

## Клонирование объектов

### МЕТОД `clone()`

- Имеет следующую сигнатуру:

**protected** Object *clone()* **throws** CloneNotSupportedException {...}

- Реализует требование (для любого объекта *x*):
  - ✓ выражение ***x.clone() != x*** будет **true**;
  - ✓ выражение ***x.clone().getClass() == x.getClass()*** будет **true**;
  - ✓ типично, что ***x.clone().equals(x)*** будет **true**.



- ✓ По умолчанию Java клонирует "**копия поля**", т.е.:
  - ✓ Если поля примитивных типов, то будет независимая новая копия;
  - ✓ Если есть ссылочные поля, то будет разделение объектов (не независимая копия).

## Клонирование объектов

### Пример 7:

```
public class Student {  
    private Date yearSet;  
    private int group;  
    public Student(Date year, int group) {  
        this.yearSet = year;  
        this.group = group;  
    }  
    public String toString() {  
        return "year = " + yearSet + ", group = " + group;  
    }  
    public Object clone() throws CloneNotSupportedException {  
        return super.clone();  
    }  
}
```

## Клонирование объектов

Продолжение примера 8,

```
public class Main {  
    public static void main(String[] arg) throws Exception {  
        Date dd = new Date(System.currentTimeMillis());  
        Student myStud1 = new Student(dd, 201);  
        System.out.println(myStud1);  
        Student myStud2 = (Student) myStud1.clone();  
        System.out.println(myStud2);  
    }  
}
```

### Console output:

year = Tue Oct 11 12:09:42 EEST 2016, group = 201

Exception in thread "main"

java.lang.CloneNotSupportedException ...



Класс должен реализовать интерфейс **Cloneable** !



## Клонирование объектов

Редактирование примера 9,

```
public class Student implements Cloneable {  
    private Date yearSet;  
    private int group;  
    public Student(Date year, int group) {  
        this.yearSet = year;  
        this.group = group;  
    }  
    public String toString() {  
        return "year = " + yearSet + ", group = " + group;  
    }  
    public Object clone() throws CloneNotSupportedException {  
        return super.clone();  
    }  
}
```

### Console output:

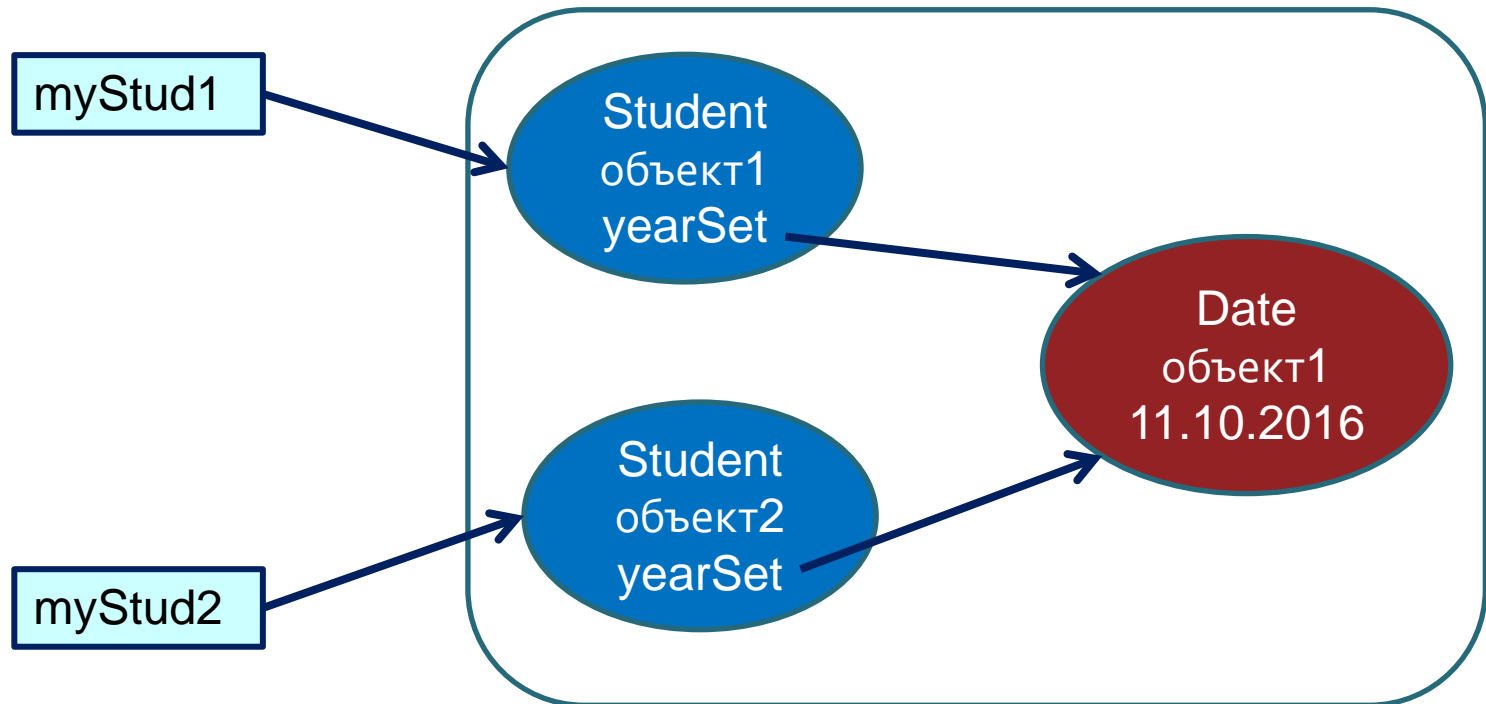
```
year = Tue Oct 11 12:19:48 EEST 2016, group = 201  
year = Tue Oct 11 12:19:48 EEST 2016, group = 201
```

## Клонирование объектов

- ❑ Java поддерживает два типа клонирования:
  - Поверхностное;
  - Глубокое.



Метод `clone()` класса **Object** использует поверхностное клонирование.



## Клонирование объектов



- ❑ Поверхностные копии просты и дешевы, и как правило, могут быть реализованы путем простого копирования битов.
- ❑ *Глубокое копирование* - означает разыменовывание поля: вместо копии ссылки на объект, создается новая копия объекта ссылочного типа.
  - В результате ссылочные объекты в **B** независимы от таких же ссылок в **A**;
  - Глубокие копии дороже из-за необходимости создавать дополнительные объекты, и могут быть более сложными из-за ссылок.

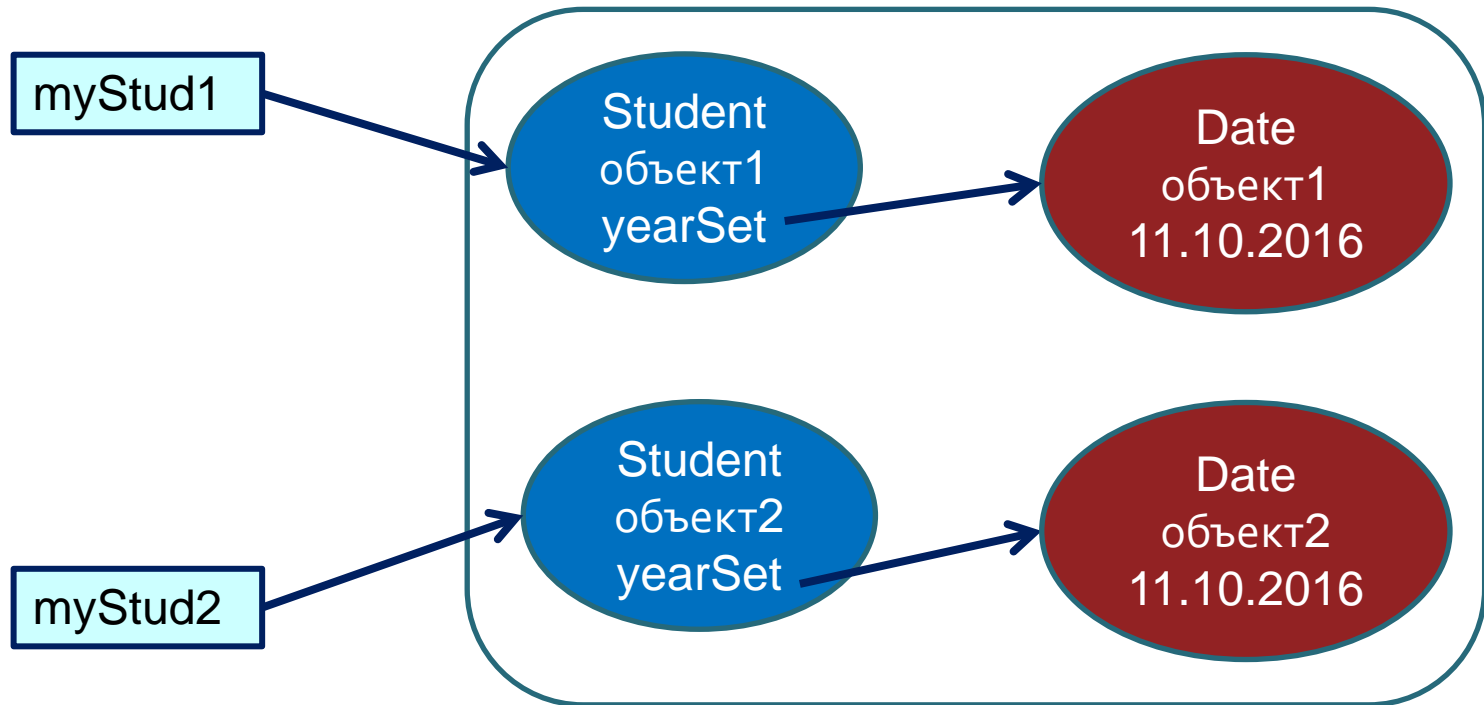
## Клонирование объектов

Пример 10:

```
public class Student implements Cloneable {  
    private Date yearSet;  
    private int group;  
    public Student(Date year, int group) {  
        this.yearSet = year;  
        this.group = group;  
    }  
    public String toString() {  
        return "year = " + yearSet + ", group = " + group;  
    }  
    public Object clone() throws CloneNotSupportedException {  
        Student stud = (Student)super.clone();  
        stud.yearSet = (Date)this.yearSet.clone();  
        return stud;  
    }  
}
```

## Клонирование объектов

### Глубокое клонирование



## Клонирование объектов

Пример 11, использование конструктора копирования:

```
public class MyObject {  
    private int field1 = 100;  
    private String field2 = "Hello";  
    public int getField1() { return field1; }  
    public String getField2() { return field2; }  
    public MyObject() { }  
    public MyObject(MyObject other) {  
        this.field1 = other.getField1();  
        this.field2 = new String(other.getField2());  
    }  
}
```

Конструктор  
копирования  
для  
ссылочного  
поля

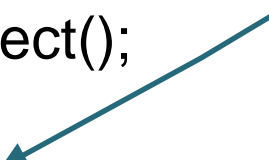


О реализации инициализации полей полностью должен позаботиться разработчик класса.

## Клонирование объектов

Продолжение примера 11:

```
public class Main {  
    public static void main(String[] args) {  
        MyObject obj1 = new MyObject();  
        //...  
        MyObject obj2 = new MyObject(obj1);  
    }  
}
```



Вызов  
конструктора  
копирования