```
<T extends Object &
   Comparable<? super T>>

    // compile error
```

Неочевидные <Generic>’и

СБЕРБАНК ТЕХНОЛОГИИ

Александр Маторин

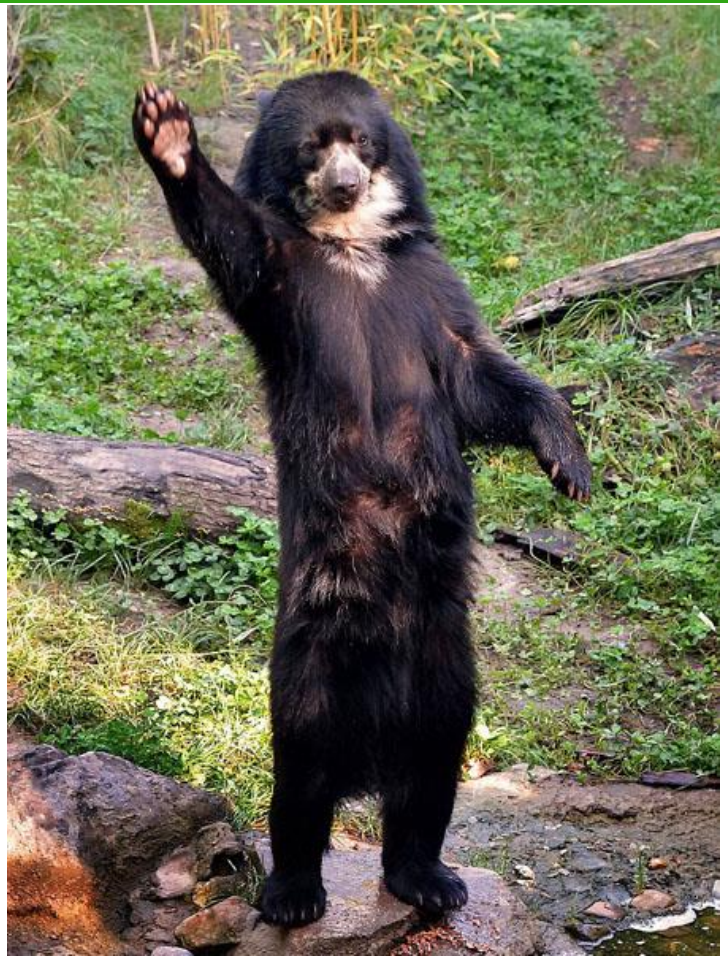
Об авторе

Александр Маторин | Сбербанк-Технологии

- Работаю в отделе рисков на финансовых рынках.
- Преподаю на кафедре СберТеха в МФТИ курсы по Java и по распределенным системам.
- Участвую в организации внутренних конференций для разработчиков в банке.

Кто это ?

Очковый медведь



ВНЕЗАПНО терять мех стали все медведицы в зоопарке в Лейпциге.
(Долорес, Бьянка и Лолита)

А что это ?

```
interface Stream<T> extends BaseStream<T, Stream<T>> {

 <R> Stream<R> flatMap(Function<? super T,
                       ? extends Stream<? extends R>> mapper);

}
```

Вот еще

```
Integer i = 2016;
Class<Integer> c = i.getClass();
```

Как так ??

```
Integer i = 2016;
Class<Integer> c = i.getClass(); // compile error
```

Как так ??

```
Integer i = 2016;
Class<Integer> c = i.getClass(); // compile error
```

Обсудим

- Странные дженерики в JDK
- Правильное использование Wildcards
- Во что все это компилируется (Type Erasure, Bridge methods)
- Изменения в Java8

- **“Философия” Дженериков**
- **Простые правила для написания гибкого API**

Стирание Дженериков

Что тут не так ?

```
public class Person implements Comparable<Person> {
    private String name;

    @Override
    public int compareTo(Person o) {
        return name.compareTo(o.name);
    }

    public int compareTo(Object o) {
        return toString()
            .compareTo(o.toString());
    }
}
```

Compile error: both methods have same erasure

```
public class Person implements Comparable<Person> {
    private String name;

    @Override
    public int compareTo(Person o) {
        return name.compareTo(o.name);
    }

    public int compareTo(Object o) {
        return toString()
            .compareTo(o.toString());
    }
}
```

Compile error: both methods have same erasure

```
public class Person implements Comparable<Person> {
    private String name;

    @Override
    public int compareTo(Person o) {
        return name.compareTo(o.name);
    }

    public int compareTo(Object o) {
        return toString()
            .compareTo(o.toString());
    }
}
```

Type Erasure

- Процедура стирания информации о дженериках на уровне компиляции.
- Вставка кастования
- Генерация бридж методов

До Type Erasure

```
class Holder<T> {
    private T value;

    public Holder(T t) {
        this.value = t;
    }
    public T get() {
        return value;
    }
}
```

До Type Erasure

```
class Holder<T> {
    private T value;

    public Holder(T t) {
        this.value = t;
    }
    public T get() {
        return value;
    }
```

```
Holder<Integer> holder = new Holder<>(10);
Integer integer = holder.get();
```

После Type Erasure

```java
class Holder<T> {
    private T value;

    public Holder(T t) {
        this.value = t;
    }
    public T get() {
        return value;
    }
```

```java
class Holder {
    private Object value;

    public Holder(Object t) {
        this.value = t;
    }
    public Object get() {
        return value;
    }
```

```java
Holder<Integer> holder = new Holder<>(10);
Integer integer = holder.get();
```

После Type Erasure

```java
class Holder<T> {
    private T value;

    public Holder(T t) {
        this.value = t;
    }
    public T get() {
        return value;
    }
```

```java
class Holder {
    private Object value;

    public Holder(Object t) {
        this.value = t;
    }
    public Object get() {
        return value;
    }
```

```java
Holder<Integer> holder = new Holder<>(10);
Integer integer = holder.get();

Holder holder = new Holder(10);
Integer integer = (Integer) holder.get();
```

Type Erasure

```
class Holder<T extends Comparable<? extends Number>> {
    public T get() {
        return value;
    }
}
```

```
class Holder {
    public Comparable get() {
        return value;
    }
}
```

Bridge методы

Bridge methods

```
public class Person<T> {
    public int compareTo(T o) {
        return 0;
    }
}

Что будет на консоли ?
Stream.of(Person.class.getDeclaredMethods())
            .forEach(System.out::println);
```

Bridge methods

```
public class Person<T> {
    public int compareTo(T o) {
        return 0;
    }
}
```

```
Stream.of(Person.class.getDeclaredMethods())
            .forEach(System.out::println);
```

public int Person.compareTo(java.lang.Object)

После erasure

```
public class Person {
    public int compareTo(Object o) {
        return 0;
    }
}
```

```
Stream.of(Person.class.getDeclaredMethods())
              .forEach(System.out::println);
```

public int Person.compareTo(java.lang.Object)

Добавим интерфейс

```
public class Person implements Comparable<Person> {
    @Override
    public int compareTo(Person o) {
        return 0;
    }

//Что будет на консоли ?
Stream.of(Person.class.getDeclaredMethods())
            .forEach(System.out::println);
```

Добавим интерфейс

```
public class Person implements Comparable<Person> {
    @Override
    public int compareTo(Person o) {
        return 0;
    }
```

```
Stream.of(Person.class.getDeclaredMethods())
            .forEach(System.out::println);
```

public int Person.compareTo(Person)
public int Person.compareTo(java.lang.Object) oO???

Попробуем достать ВМ через reflection

```
public class Person implements Comparable<Person> {
  @Override
  public int compareTo(Person o) {
     return 0;
  }

 public static void main(String[] args) throws Exception {
    Method m1 = Person.class.getMethod("compareTo", Person.class);
    Method m2 = Person.class.getMethod("compareTo", Object.class);

    System.out.println(m1.isSynthetic());  //false
    System.out.println(m2.isSynthetic());  //true
 }
}
```

Bridge method. Зачем ?

```
public interface Comparable<T> {
      int compareTo(T t);
}
```

Bridge method. Зачем ?

```
public interface Comparable {
      int compareTo(Object o);
}
```

Bridge method. Зачем ?

```
public interface Comparable {
      int compareTo(Object o);
}
public class Person implements Comparable<Person> {
    @Override
    public int compareTo(Person o) {
        return 0;
    }
}
```

Bridge method. Зачем ?

```
public interface Comparable {
      int compareTo(Object o);
}

public class Person implements Comparable<Person> {
     @Override
     public int compareTo(Person o) {
          return 0;
     }

     @RealOverride
     public int compareTo(Object o) {
          return compareTo((Person)o)
     }
}
```

Дженерики, доступные в рантайме

```
public class Runtime<T extends Number>
                          implements Callable<Double> {
    private final List<Integer> integers = emptyList();

    public List<T> numbers() {return emptyList();}

    public List<String> strings() {return emptyList();}

    @Override
    public Double call() {return 0d;}
}
```


Внимание, вопрос...

Что тут не так ?

```
class GenericException<T> extends Exception {
    private final T details;

    public GenericException(T details) {
        this.details = details;
    }

    public T getDetails() {
        return details;
    }
}
```

Что тут не так ?

```
//compile error oO
class GenericException<T> extends Exception {
    private final T details;

    public GenericException(T details) {
        this.details = details;
    }

    public T getDetails() {
        return details;
    }
}
```

Нельзя параметризовывать

- Классы, имеющие в предках Throwable
- Анонимные классы
- Enums

Нельзя параметризовывать

- Классы, имеющие в предках Throwable
- Анонимные классы
- Enums

```
try {
    run();
} catch (GenericException<String> e) {
    …
} catch (GenericException<Integer> e) {
    …
}
```

Heap pollution

Heap pollution

```
public void run(List<String>... lists) {
}
```

Heap pollution

```
//warning: possible heap pollution
public void run(List<String>... lists) {
}
```

Heap pollution

```
//warning: possible heap pollution
public void run(List<String>... lists) {
}
```

Heap pollution occurs when a variable of a parameterized type refers to an object that is not of that parameterized type(Oracle docs)

List<String> = List<Integer>

Ковариантность

```
//Скомпилируется ?

Number[] numbers      = new Integer[10];
List<Number> numbers = new ArrayList<Integer>();
```

Дженерики инвариантны

```
Number[] numbers      = new Integer[10]; // ok
List<Number> numbers = new ArrayList<Integer>();//error
```

Дженерики инвариантны

```
Number[] numbers = new Integer[10];
numbers[0] = 9.7d;
```

Дженерики инвариантны

```
Number[] numbers = new Integer[10];
numbers[0] = 9.7d; // ArrayStoreException
```

Дженерики инвариантны

```
Number[] numbers = new Integer[10];
numbers[0] = 9.7d; // ArrayStoreException

List<Number> numbers = new ArrayList<Integer>();
```

Дженерики инвариантны

```
Number[] numbers = new Integer[10];
numbers[0] = 9.7d; // ArrayStoreException

List<Number> numbers = new ArrayList<Integer>();

List<Integer> integers = new ArrayList<Integer>();
List list =  integers; //warning
```

Дженерики инвариантны

```
Number[] numbers = new Integer[10];
numbers[0] = 9.7d; // ArrayStoreException

List<Number> numbers = new ArrayList<Integer>();

List<Integer> integers = new ArrayList<Integer>();
List list =  integers; //warning
List<Number> numbers = list; //warning
```

Дженерики инвариантны

```
Number[] numbers = new Integer[10];
numbers[0] = 9.7d; // ArrayStoreException

List<Number> numbers = new ArrayList<Integer>();

List<Integer> integers = new ArrayList<Integer>();
List list =  integers; //warning
List<Number> numbers = list; //warning
numbers.add(9.7d);
```

Дженерики инвариантны

```
Number[] numbers = new Integer[10];
numbers[0] = 9.7d; // ArrayStoreException

List<Number> numbers = new ArrayList<Integer>();

List<Integer> integers = new ArrayList<Integer>();
List list =  integers; //warning
List<Number> numbers = list; //warning
numbers.add(9.7d); // ну ок
```

Дженерики инвариантны

```
Number[] numbers = new Integer[10];
numbers[0] = 9.7d; // ArrayStoreException

List<Number> numbers = new ArrayList<Integer>();

List<Integer> integers = new ArrayList<Integer>();
List list =  integers; //warning
List<Number> numbers = list; //warning
numbers.add(9.7d);

Integer i = integers.get(0); //ClassCastException
Integer i = (Integer) integers.get(0);
```

Generic array creation

```
//Скомпилируется ?

List<Number>[] lists = new ArrayList<Number>[10];
List<?>[]      lists = new ArrayList<?>[10];
```

Generic array creation

```
//Скомпилируется ?

List<Number>[] lists = new ArrayList<Number>[10]; //error
List<?>[]      lists = new ArrayList<?>[10];      //ok
```

Возможен heap pollution

```
List<Number>[] lists = new ArrayList<Number>[10];
```

Возможен heap pollution

```
List<Number>[] lists = new ArrayList<Number>[10];

Object[] objects = lists;
```

Возможен heap pollution

```
List<Number>[] lists = new ArrayList<Number>[10];

Object[] objects = lists;
objects[0] = new ArrayList<String>();
```

Возможен heap pollution

```
List<Number>[] lists = new ArrayList<Number>[10];

Object[] objects = lists;
objects[0] = new ArrayList<String>();

lists[0].add(1L); // ☹
```

Но массив дженериков можно создать через VarArgs

```
//warning
public void run(List<String>... lists) {
    Object[] objectArray = lists;
    objectArray[0] = Arrays.asList(42);
    String s = lists[0].get(0); // ClassCastException
}
```

Generic array creation

```
List<?>[] lists = new ArrayList<?>[10]; // почему ok ?
```

Wildcards <?>

Вопросы на стенде СБТ

- **Что можно положить сюда ? .add()**

```
List<? extends Number> numbers = new ArrayList<>()
```

- **А сюда ?**

```
List<? super Number> numbers = new ArrayList<Object>();
```

Статистика:
Правильных ответов

Wildcards

- **Что можно положить сюда ? .add()**

```
List<? extends Number> numbers = new ArrayList<>()
//Number, Integer, Double, Long..
```

Wildcards

- **Что можно положить сюда ? .add()**

```
List<? extends Number> numbers = new ArrayList<>()
//Number, Integer, Double, Long..
```

- **А сюда ?**

```
List<? super Number> numbers = new ArrayList<>();
//Object, Number
```

Wildcards

- **Что можно положить сюда ? .add()**

```
List<? extends Number> numbers = new ArrayList<>()
//Number, Integer, Double, Long..
```

- **А сюда ?**

```
List<? super Number> numbers = new ArrayList<>();
//Object, Number
```

- **А сюда ?**

```
List<?> list = new ArrayList<>();
//Да что угодно
```

Wildcards

- **Что можно положить сюда ? .add()**

```
List<? extends Number> numbers = new ArrayList<>()
//Number, Integer, Double, Long..
```

- **А сюда ?**

```
List<? super Number> numbers = new ArrayList<>();
//Object, Number
```

- **А сюда ?**

```
List<?> list = new ArrayList<>();
//Да что угодно
```

Что сюда можно положить ?

```
List<? extends Number> numbers = new ArrayList<>()
```

казалось бы..

- Number x
- Integer x
- Double x
- ….

Что сюда можно положить ?

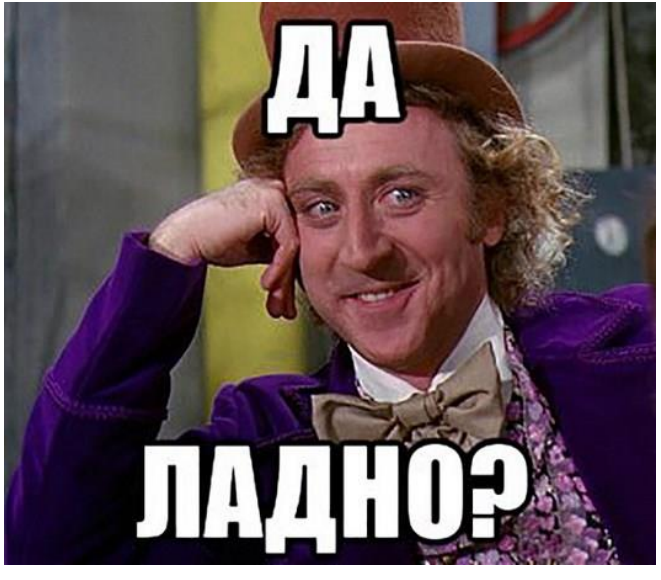
```
List<? extends Number> numbers = new ArrayList<>()
```

казалось бы..

- Number x
- Integer x
- Double x
- ….

На самом деле

- null



Что сюда можно положить

```
// что видит компилятор
List<? extends Number> numbers = ????
```

Что сюда можно положить

```
// что видит компилятор
List<? extends Number> numbers = ????

// что можно присвоить
numbers = new ArrayList<Number>();
numbers = new ArrayList<Integer>();
numbers = new ArrayList<Long>();
…
```

Что сюда можно положить

```
// что видит компилятор
List<? extends Number> numbers = ????

// что можно присвоить
numbers = new ArrayList<Number>();
numbers = new ArrayList<Integer>();
numbers = new ArrayList<Long>();
…


public void process(List<? extends Number> numbers) {
 numbers.add(234L);
}
```

Что сюда можно положить

```
// что видит компилятор
List<? extends Number> numbers = ????

// что можно присвоить
numbers = new ArrayList<Number>();
numbers = new ArrayList<Integer>();
numbers = new ArrayList<Long>();
…


public void process(List<? extends Number> numbers) {
 numbers.add(234L); // валидно только для List<Number> и
}                                                  <Long>
```

Что сюда можно положить

```
// что видит компилятор
List<? extends Number> numbers = ????

// что можно присвоить
numbers = new ArrayList<Number>();
numbers = new ArrayList<Integer>();
numbers = new ArrayList<Long>();
…
// Компилятор не знает, чем на самом деле параметризован
List, поэтому безопасно можно добавить только null
public void process(List<? extends Number> numbers) {
 numbers.add(234L); // валидно только для List<Number> и
                                                  <Long>
}
```

Зачем нужен ?

List<? **extends** Number> похож на Number[]

с разрешением только на чтение

```
Number[] numbers = new Integer[10];
List<? extends Number> numbers = new ArrayList<Integer>();
```

Что сюда можно положить ?

```
List<? super Number> numbers = new ArrayList<>()
```

казалось бы..

- Object
- Number

Что сюда можно положить ?

```
List<? super Number> numbers = new ArrayList<>()
```

казалось бы..

- Object x
- Number +

На самом деле

- все что ? extends Number
- Number +
- Integer +
- Double +
- null +

Что сюда можно положить

```
// что видит компилятор
List<? super Number> numbers = ????
```

Что сюда можно положить

```
// что видит компилятор
List<? super Number> numbers = ????

// что можно присвоить
numbers = new ArrayList<Object>();
numbers = new ArrayList<Number>();
```

Что сюда можно положить

```
// что видит компилятор
List<? super Number> numbers = ????

// что можно присвоить
numbers = new ArrayList<Object>();
numbers = new ArrayList<Number>();
```

```
public void process(List<? super Number> numbers) {
    numbers.add(234L);
    numbers.add(100D);
    numbers.add(new Object());
}
```

Что сюда можно положить

```
// что видит компилятор
List<? super Number> numbers = ????

// что можно присвоить
numbers = new ArrayList<Object>();
numbers = new ArrayList<Number>();

// Компилятор знает, что List параметризован максимум
Number'ом поэтому можно безопасно положить любой Number
public void process(List<? super Number> numbers) {
    numbers.add(234L);
    numbers.add(100D);
    numbers.add(new Object());
}
```

Standard JDK methods

Сигнатура Collections.max

```
//Ожидание
public static<T> T  max(Collection<T> coll)
```

Сигнатура Collections.max

```
//Ожидание
public static<T> T  max(Collection<T> coll)
```

```
//Реальность
public static <T extends Object & Comparable<? super T>>
                    T max(Collection<? extends T> coll)
```

Во что сотрется Collections.max?

```
public static <T extends Object & Comparable<? super T>>
                    T max(Collection<? extends T> coll)
```

Collections.max erasure

```
public static <T extends Object & Comparable<? super T>>
                    T max(Collection<? extends T> coll)

public static Object max(Collection coll)
```

Collections.max erasure

```
public static <T extends Object & Comparable<? super T>>
                    T max(Collection<? extends T> coll)

public static Object max(Collection coll)
```

```
public static <T extends Comparable<? super T>>
                    T max(Collection<? extends T> coll)

public static Comparable max(Collection coll)
```

Попробуем повторить

```
public class Binary {
    public Object get() {
        return "object";
    }
}

public class BinaryMain {
    public static void main(String[] args) {
        Object o = new Binary().get();
    }
}
```

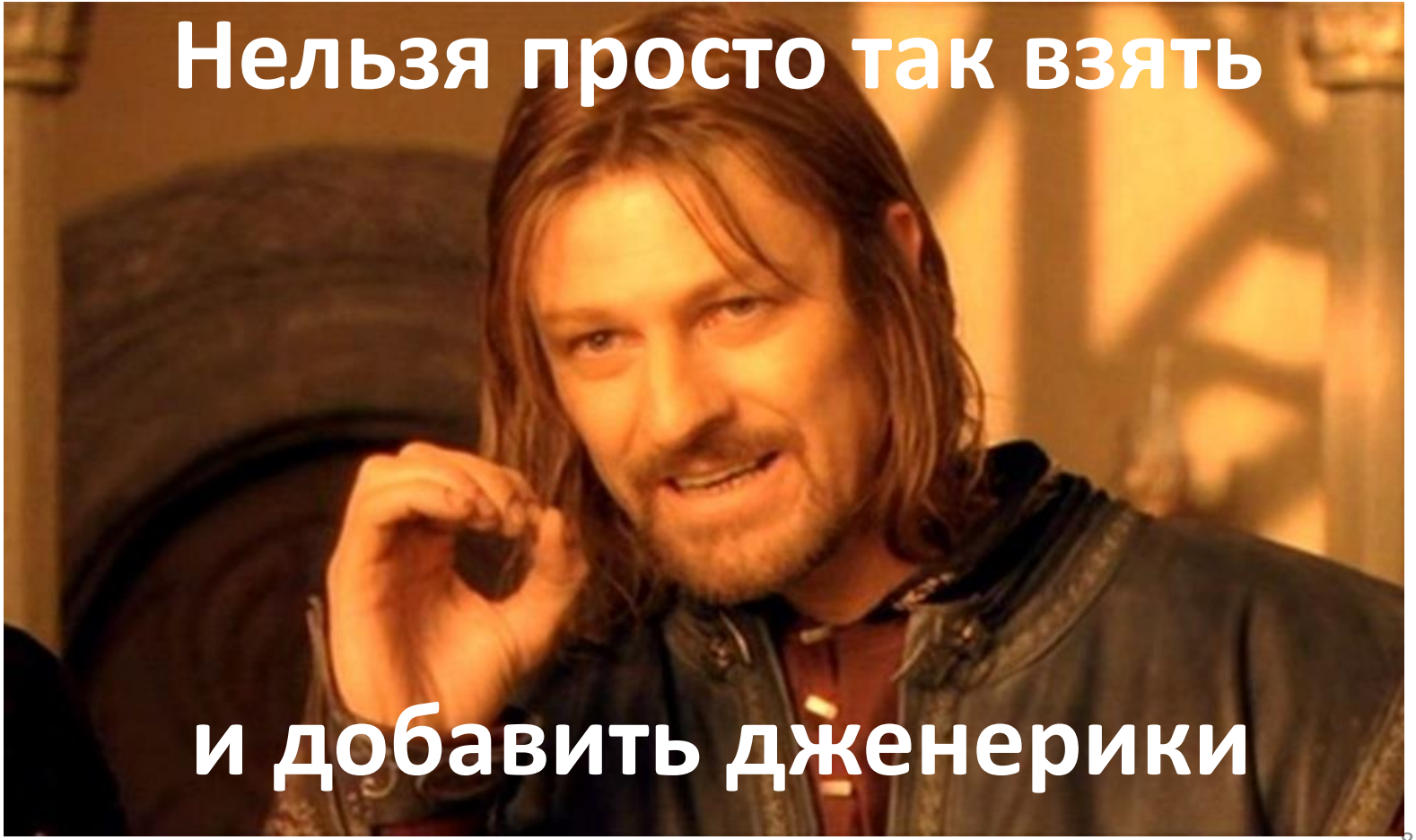
Попробуем повторить

```
public class Binary {
    public String get() {
        return "object";
    }
}

public class BinaryMain {
    public static void main(String[] args) {
        Object o = new Binary().get();
    }
}
```

Попробуем повторить

```
public class Binary {
    public String get() {
        return "object";
    }
}

public class BinaryMain {
    public static void main(String[] args) {
        Object o = new Binary().get();
    }
}
```

Exception in thread "main" java.lang.NoSuchMethodError: Binary.get()Ljava/lang/Object; at BinaryMain.main(BinaryMain.java:5)


Нельзя просто так взять
и добавить дженерики

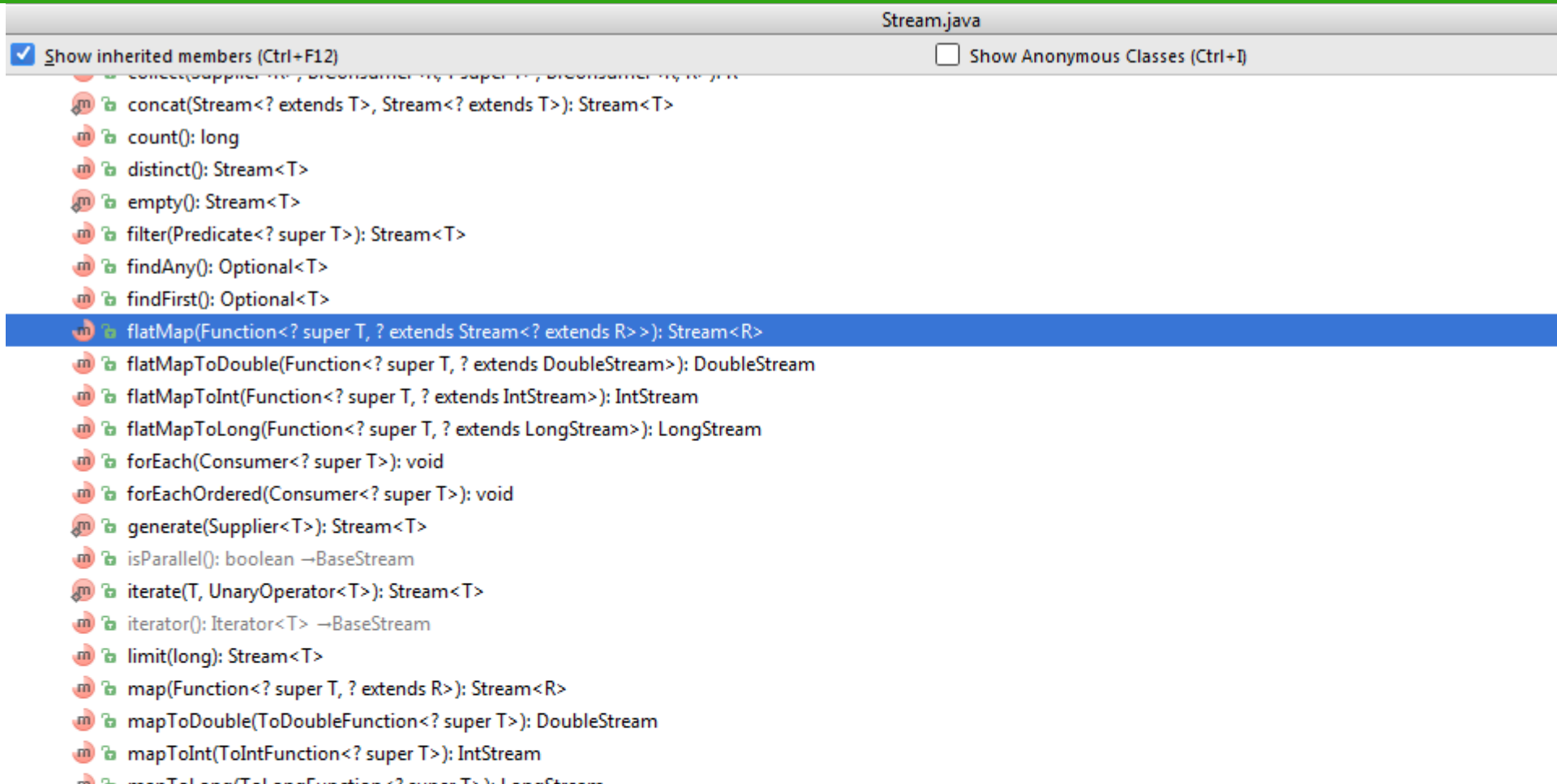
Collections.max

```
public static <T extends Object & Comparable<? super T>>
                    T max(Collection<? extends T> coll)
```

Можно так ?

```
public static <T extends Object & Comparable<T>>
                    T max(Collection<T> coll)
```

Почему так много вопросиков ?



PECS (Producer-extends, consumer-super)

If a parameterized type represents a T producer,
use <? extends T>;

if it represents a T consumer, use <? super T>.

Joshua Bloch

PECS (Producer-extends, consumer-super)

If a parameterized type represents a T producer,
use <? extends T>;

if it represents a T consumer, use <? super T>.

Joshua Bloch

```
public static <T> T max(Collection<? extends T> coll,
                        Comparator<? super T> comp)
```

PECS (Producer-extends, consumer-super)

If a parameterized type represents a T producer,
use <? extends T>;

if it represents a T consumer, use <? super T>.

Joshua Bloch

```
public static <T> T max(Collection<? extends T> coll,
                        Comparator<? super T> comp)
```

```
Collections.max(List<Integer>, Comparator<Number>);
Collections.max(List<String>, Comparator<Object>);
```

Рекурсивные Дженерики

Enum<E **extends** Enum<E>>

Рекурсивные Дженерики

```
BaseStream<T, S extends BaseStream<T, S>> {
    S sequential();
    S parallel()
}

Stream<T> extends BaseStream<T, Stream<T>>
```

Упростим

```
BaseStream<T> {
    BaseStream<T> sequential();
    BaseStream<T> parallel()
}

Stream<T> extends BaseStream<T>
```

Упростим

```
BaseStream<T> {
    BaseStream<T> sequential();
    BaseStream<T> parallel()
}

Stream<T> extends BaseStream<T>


stream.filter(Objects::nonNull)
      .parallel()
      .map(Integer::parseInt)
```

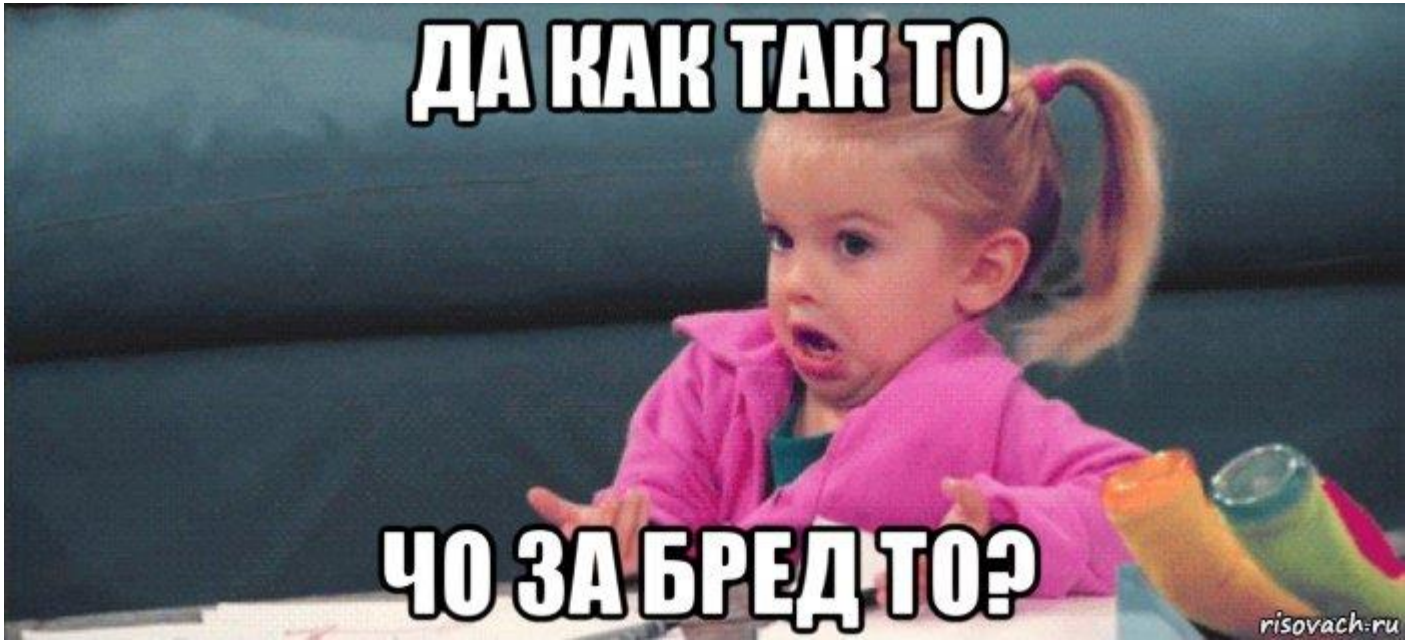
Рекурсивные Дженерики

```
BaseStream<T, S extends BaseStream<T, S>> {
    S sequential();
    S parallel()
}
```

```
Stream<T> extends BaseStream<T, Stream<T>>
```

```
stream.filter(Objects::nonNull)
      .parallel()
      .map(Integer::parseInt)
```

Еще непонятные



Object.getClass()

```
Class<Integer> clazz = Integer.class;

Integer integer = 2016;
Class<Integer> clazz = integer.getClass();//compile error
```

Ковариантность возвращаемого значения

```
class Parent {
    public Number run(String s){…}
}
```

```
class Child extends Parent {
    @Override
    public Integer run(String s){…}
}
```

Object.getClass()

```
public class Object {

      public final native Class<?> getClass();
..
}
```

Если бы возвращался Class<T>

```
class Object {
    Class<?> getClass();
}
```

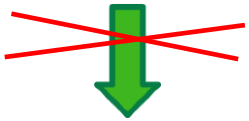
```
class Number {
    Class<Number> getClass();
}
```

```
class Integer {
    Class<Integer> getClass();
}
```

Если бы возвращался Class<T>

```
class Object {
    Class<?> getClass();
}
```

```
class Number {
    Class<Number> getClass();
}
```



```
class Integer {
    Class<Integer> getClass();
}
```

Ковариантность возвращаемого значения

```
class Object {
    Class<?> getClass();
}
```

```
class Number {
    Class<Number> getClass();
}
```

```
Number n = new Integer(1);
Class<Number> clazz = n.getClass();
```

```
class Integer {
    Class<Integer> getClass();
}
```

Ковариантность возвращаемого значения

```
class Object {
    Class<?> getClass();
}
```

```
class Number {
    Class<? extends Number> getClass();
}
```

```
Number n = new Integer(1);
Class<? extends Number> c = n.getClass();
```

```
class Integer {
    Class<? extends Integer> getClass();
}
```

Что тут не так ?

```
public class Helper<T> {
    public List<Integer> numbers() {
        return Arrays.asList(1, 2);
    }

    public static void main(String[] args) {
        Helper helper = new Helper<>();
        for (Integer number : helper.numbers()) {
            ...
        }
    }
}
```

Да как так ???

```
public class Helper<T> {
    public List<Integer> numbers() {
        return Arrays.asList(1, 2);
    }

    public static void main(String[] args) {
        Helper helper = new Helper<>();
        for (Integer number : helper.numbers()) {//error
            ...
        }
    }
}
```

Raw удаляет ВСЮ информацию о дженериках. JLS 4.8

```
public class Helper<T> {
    public List<Integer> numbers() {
        return Arrays.asList(1, 2);
    }

    public static void main(String[] args) {
        Helper helper = new Helper<>();
        for (Integer number : helper.numbers()) {//error
            ...
        }
    }
}
```

Raw удаляет ВСЮ информацию о дженериках. JLS 4.8

```
public class Helper<T> {
    public List<Integer> numbers() {
        return Arrays.asList(1, 2);
    }

    public static void main(String[] args) {
        Helper helper = new Helper<>();
        List list = helper.numbers();
        for (Integer number : list) {//error
            ...
        }
    }
}
```

Всегда параметризируйте код!

```
public class Helper<T> {
    public List<Integer> numbers() {
        return Arrays.asList(1, 2);
    }

    public static void main(String[] args) {
        Helper<?> helper = new Helper<>();
        for (Integer number : helper.numbers()) {
            ...
        }
    }
}
```

Java 8 Type inference enhancement

```
//Java 1.8.25:Compile error
//Java 1.8.73:Compile ok

public void run() {
    reduce(
      Stream.of("1", "2", "3")
          .collect(toMap(identity(),t -> t.length()))
    );
}

private <T> T reduce(Map<T, ?> map) {
  …
}
```

Java 8 Type inference enhancement

```
//Java 1.8.25:Compile error
//Java 1.8.73:Compile ok

public void run() {
    reduce(
      Stream.of("1", "2", "3")
          .collect(toMap(identity(),t -> t.length()))
    );
}
```

Error: Cannot infer type-variable(s) T,R,A,capture#1 of ?,T,K,U,T
(argument mismatch; R cannot be converted to java.util.Map<java.lang.String,?>)

Fixes for Java 1.8.25

```
private void run2() {
    Map<String, Integer> map = Stream.of("1", "2", "3")
            .collect(toMap(identity(), t -> t.length()));
    reduce(map);
}
```

Fixes for Java 1.8.25

```
private void run2() {
    Map<String, Integer> map = Stream.of("1", "2", "3")
            .collect(toMap(identity(), t -> t.length()));
    reduce(map);
}

private void run3() {
    reduce(Stream.of("1", "2", "3")
            .collect(toMap(identity(), String::length)));
}
```

Fixes for Java 1.8.25

```
private void run2() {
    Map<String, Integer> map = Stream.of("1", "2", "3")
            .collect(toMap(identity(), t -> t.length()));
    reduce(map);
}

private void run3() {
    reduce(Stream.of("1", "2", "3")
            .collect(toMap(identity(), String::length)));
}

private void run4() {
    reduce(Stream.of("1", "2", "3")
            .collect(toMap(t -> t, t -> t.length())));
}
```

Совсем не понятно

```
public static void main(String[] args) {
    String s = newList(); // почему компилируется????????
}

private static <T extends List<Integer>> T newList() {
    return (T) new ArrayList<Integer>();
}
```

Мы обсудили

- Во что компилируются
- Heap pollution
- Отличия от массивов
- Почему Дженерики инвариантны
- Wildcards
- PECS
- Raw
- Java 8 changes

Спасибо!

Буду рад ответить на вопросы!

Александр Маторин | aamatorin.sbt@sberbank.ru

Java 7

```
static <T> void setFirst(T[] ar, T s) {
    ar[0] = s;
}

public static void main(String[] args) {
     setFirst(new String[10], new Integer(1));
}
```

Java 7

```
static <T> void setFirst(T[] ar, T s) {
    ar[0] = s; //ArrayStoreException
}

public static void main(String[] args) {
     setFirst(new String[10], new Integer(1));
}
```

Java 7

```
static <T> void setFirst(T[] ar, T s) {
    ar[0] = s;
}

public static void main(String[] args) {
     setFirst(new String[10], new Integer(1));
}
```

```
Общий тип T =
Object & Serializable & Comparable<? extends Object &
                                   Serializable & Comparable<?>>
```

Java 7

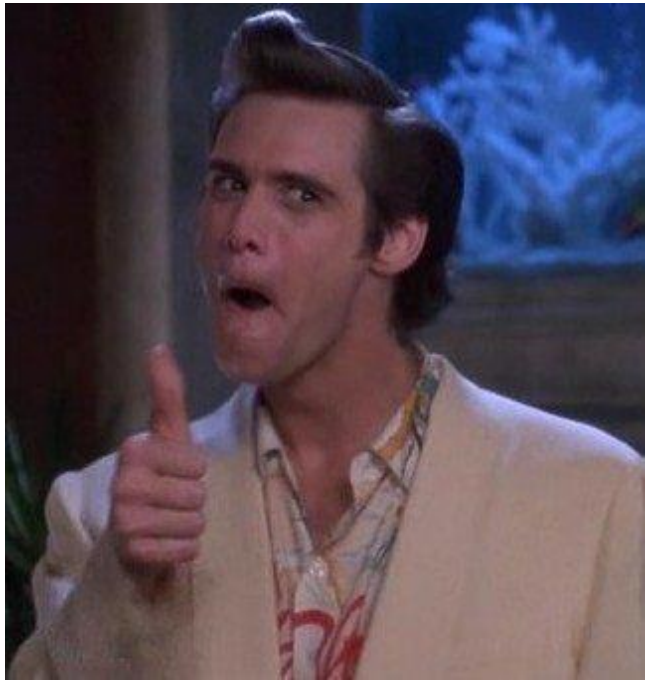
```
static <T, S extends T> void setFirst(T[] ar, S s) {
    ar[0] = s;
}

public static void main(String[] args) {
     setFirst(new String[10], new Integer(1));//compile err
}
```

Java 8: Type inference enhancement

```
static <T, S extends T> void setFirst(T[] ar, S s) {
    ar[0] = s;
}

public static void main(String[] args) {
     setFirst(new String[10], new Integer(1));//compile ok
}
```

Java 8: Type inference enhancement

```
static <T, S extends T> void setFirst(T[] ar, S s) {
    ar[0] = s; //ArrayStoreException
}

public static void main(String[] args)
    setFirst(new String[10], new Intege
}
```



Кидаем Unchecked Exception. Java7

```
public static void main(String[] args) {
    ThisClass.<RuntimeException>throwIt(new IOException());
}

private static <E extends Exception> void throwIt(Exception e) throws E {
    throw (E) e;
}
```

Кидаем Unchecked Exception. Java8

```
public static void main(String[] args) {
    ThisClass.throwIt(new IOException());
}

private static <E extends Exception> void throwIt(Exception e) throws E {
    throw (E) e;
}
```

Collections.max

```
public static <T extends Object & Comparable<? super T>>
                    T max(Collection<? extends T> coll)
```

Можно так ?

```
public static <T extends Object & Comparable<T>>
                    T max(Collection<? extends T> coll)
```

<T extends Comparable<T>> T max(Collection<? extends T> c)

```
public class Parent implements Comparable<Parent> {…}
public class Child extends Parent {…}


List<Child> children = new ArrayList<>();
Collections.max(children);  // скомпилируется ?
```

<T extends Comparable<T>> T max(Collection<? extends T> c)

```
public class Parent implements Comparable<Parent> {…}
public class Child extends Parent {…}
```

```
List<Child> children = new ArrayList<>();
Collections.max(children);  // скомпилируется ?
```

T – это Child.
Надо: T **extends** Comparable<T>
Но: Child **implements** Comparable<Parent>

<T extends Comparable<T>> T max(Collection<? extends T> c)

```
public class Parent implements Comparable<Parent> {…}
public class Child extends Parent {…}


List<Child> children = new ArrayList<>();
Collections.max(children);  //Java7 compile error
```

T – это Child.
Надо: T **extends** Comparable<T>
Но: Child **implements** Comparable<Parent>

Java 8: Type inference enhancement

Java8 пытается подставить под **T** тип, удовлетворяющий всем требованиям

Java 8: Type inference enhancement

Java8 пытается подставить под **T** тип, удовлетворяющий всем требованиям

```
<T extends Comparable<T>> T max(Collection<? extends T> c)
```

```
Collections.max(children);
```

Java 8: Type inference enhancement

Java8 пытается подставить под **T** тип, удовлетворяющий всем требованиям

```
<T extends Comparable<T>> T max(Collection<? extends T> c)
```

```
Collections.max(children);
```

```
Под T подходит Parent:
Parent implements Comparable<Parent>
Collection<? extends Parent> c = Collection<Child>
```

Java 8: Type inference enhancement

Java8 пытается подставить под **T** тип, удовлетворяющий всем требованиям

```
<T extends Comparable<T>> T max(Collection<? extends T> c)
```

Но тогда присвоить можно только в Parent

```
Parent max = Collections.max(children);
```

Под T подходит Parent:

```
Parent implements Comparable<Parent>
Collection<? extends Parent> c = Collection<Child>
```