

# Java Development Kit

Урок 3

Обобщенное программирование



# Обобщённое программирование





## На предыдущем уроке

- 📌 Программные интерфейсы — понятие и принцип работы;
- 📌 Ключевое слово `implements`;
- 📌 Наследование и множественное наследование интерфейсов;
- 📌 Реализация, реализация по-умолчанию;
- 📌 Частичная реализация интерфейсов, адаптеры;
- 📌 Анонимные классы.





## Что будет на уроке сегодня

- 📌 Обобщенное программирование;
- 📌 Diamond operator;
- 📌 Обобщённые методы;
- 📌 Подстановочные символы (Wildcards);
- 📌 Ограничения сверху и снизу;
- 📌 Выводы и целевые типы;
- 📌 Стирание типа;
- 📌 Загрязнение кучи.





# Понятие обобщения





## Что такое обобщение

Обобщённое программирование в Java позволяет создавать классы, интерфейсы и методы, которые могут работать с различными типами данных.

В результате, код становится более универсальным и повторно используемым.





## Простой класс Box

```
3 public class Main {
4     private static class Box {
5         private Object obj;
6         public Box(Object obj) { this.obj = obj; }
7
8
9         public Object getObj() { return obj; }
10        public void setObj(Object obj) { this.obj = obj; }
11        public void printInfo() {
12            System.out.printf("Box (%s): %s",
13                obj.getClass().getSimpleName(), obj.toString());
14        }
15    }
16
17    public static void main(String[] args) {
18        Box b1 = new Box(20);
19        Box b2 = new Box(30);
20        System.out.println(b1.getObj() + b2.getObj());
21    }
22 }
```



## Простой класс Box

```
22 ▶ public static void main(String[] args) {
23     Box b1 = new Box(20);
24     Box b2 = new Box(30);
25     System.out.println((Integer) b1.getObj() + (Integer) b2.getObj());
26
27     Box b3 = new Box("Hello, ");
28     Box b4 = new Box("World!");
29     System.out.println((String) b3.getObj() + (String) b4.getObj());
30 }
31 }
```

Run: ru.gb.jdk.three.Main x

50

Hello, World!





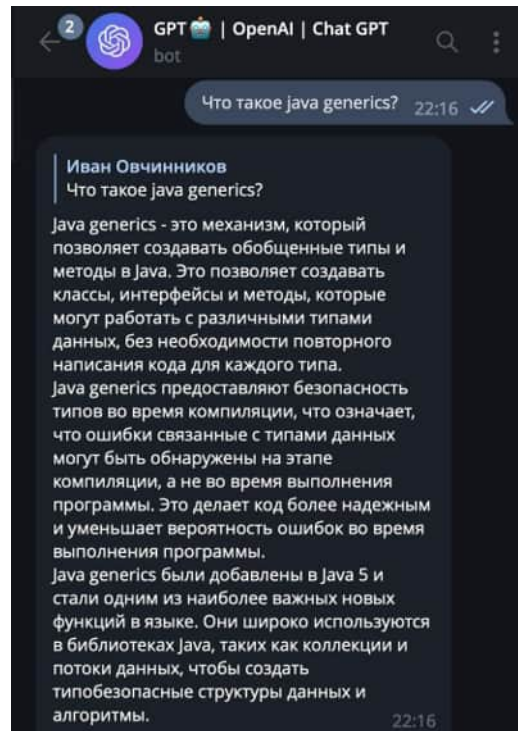
## Простой класс Box

```
Main.java
22  public static void main(String[] args) {
23      Box iBox1 = new Box(20);
24      Box iBox2 = new Box(30);
25      if (iBox1.getObj() instanceof Integer && iBox2.getObj() instanceof Integer) {
26          int sum = (Integer) iBox1.getObj() + (Integer) iBox2.getObj();
27          System.out.println("sum = " + sum);
28      } else {
29          System.out.println("The contents of the boxes differ by type");
30      }
31      iBox1.setObj("sdf"); // Java: "What can go wrong here? You can do it!"
32  }
33  }
```



## Обобщения на языке Java

- **Java generics** — это механизм языка, который позволяет создавать обобщенные типы и методы в Java.
- **Java generics** предоставляют безопасность типов во время компиляции, что означает, что ошибки связанные с типами данных могут быть обнаружены на этапе компиляции, а не во время выполнения программы.
- **Java generics** были добавлены в Java 1.5 и стали одной из наиболее важных новых функций в языке.
- **Java generics** работают только со ссылочными типами данных



## Обобщения на языке Java



## Обобщённая коробка GBox

```
4      private static class Box {...}
21     private static class GBox<T> {
22         private T value;
23
24         public GBox(T value) { this.value = value; }
27
28         public T getValue() { return value; }
31         public void setValue(T value) { this.value = value; }
34         public void showType() {
35             System.out.printf("Type is %s, with value %s\n",
36                             value.getClass().getName(), getValue());
37         }
38     }
```





## Обобщённая коробка GBox

```
40 public static void main(String[] args) {
41     GBox<String> stringBox = new GBox<>("Hello!");
42     stringBox.showType();
43     GBox<Integer> integerBox = new GBox<>(12);
44     integerBox.showType();
45     GBox<Integer> newBox = new GBox<>();
46     //...
47 }
48 }
```

Integer value

Run: ru.gb.jdk.three.Main

/Library/Java/JavaVirtualMachines/liberica-jdk-11.jdk/Con  
Type is java.lang.String, with value Hello!  
Type is java.lang.Integer, with value 12



## Обобщённая коробка GBox

```
40 public static void main(String[] args) {
41     GBox<String> stringBox = new GBox<>("Hello!");
42     stringBox.showType();
43     GBox<Integer> integerBox = new GBox<>(12);
44     integerBox.showType();
45     integerBox.setValue("World!");
46
47     // ...
56 }
57 }
```

Required type: Integer

Provided: String

Wrap using 'Integer.valueOf()' More actions...

Run: ru.gb.jdk.three.Main

```
/Library/Java/JavaVirtualMachines/liberica-jdk-11.jdk/Con
Type is java.lang.String, with value Hello!
Type is java.lang.Integer, with value 12
```



## Соглашение об именовании параметров типа

- E— элемент (Element, Entity, всеобъемлюще используемые коллекции Java)
- K— Ключ
- N— Число
- T— Тип
- V— Значение
- S, U, и т. п. — 2-й, 3-й, 4-й типы







## Ложка дёгтя

```
Main.java x
21     private static class GBox<T> {
22         private T value;
23
24         public GBox(T value) { this.value = value; }
27
28         public void methodOne() {
29             // T data = new T();
30         }
31
```





## Ложка дёгтя

```
Main.java x
21 private static class GBox<T> {
22     private T value;
23
24     public GBox(T value) { this.value = value; }
27
28     public void methodOne() {
29         T data = new T();
30     }
31
```

Type parameter 'T' cannot be instantiated directly

ru.gb.jdk.three

class T

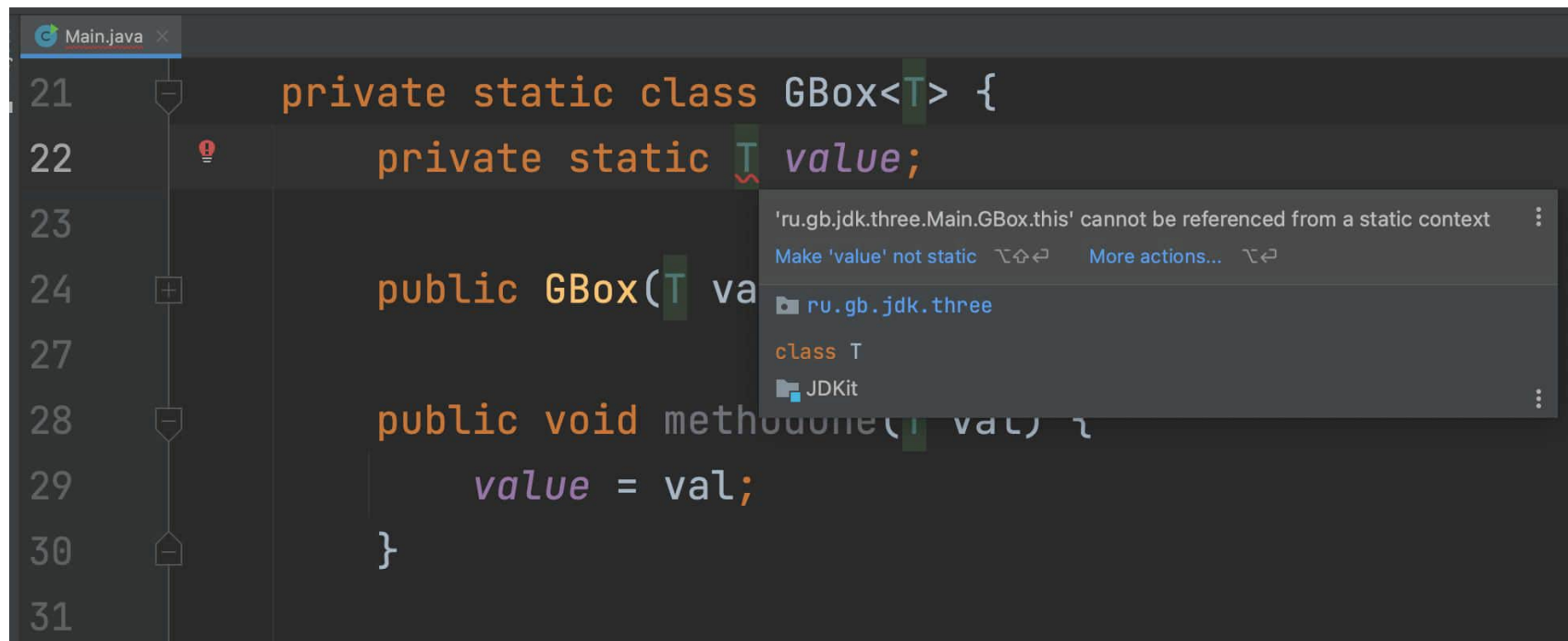


## Ложка дёгтя

```
Main.java x
21     private static class GBox<T> {
22         private T[] value;
23
24         public GBox(T[] value) { this.value = value; }
27
28         public void methodOne(T[] arr) {
29             value = arr;
30         }
31
```



## Ложка дёгтя





## Ложка дёгтя

The screenshot shows a code editor with a file named `Main.java`. The code defines a `Box` class and a generic `GBox` class that extends `RuntimeException`. The `GBox` class has a `private T value` field and methods `public GBox(T value)`, `public T getValue()`, `public void setValue(T value)`, and `public void showType()`. The `showType()` method uses `System.out.printf` to print the type of the value.


A compiler error is shown in a tooltip on the right side of the editor, indicating that a generic class cannot extend `Throwable` (which `RuntimeException` inherits from). The tooltip suggests making `GBox` not extend `RuntimeException` and provides more actions.

The tooltip content is as follows:

- Generic class may not extend 'java.lang.Throwable'
- Make 'GBox' not extend 'java.lang.RuntimeException'
- More actions...
- java.lang
  - public class RuntimeException
    - extends Exception
- RuntimeException is the superclass of those exceptions that can be thrown during the normal operation of the Java Virtual Machine.
- RuntimeException and its subclasses are *unchecked exceptions*. Unchecked exceptions do *not* need to be declared in a method or constructor's throws clause if they can be thrown by the execution of the method or constructor and propagate outside the method or constructor boundary.
- Since: 1.0
- Author: Frank Yellin
- JLS 11.2 Compile-Time Checking of Exceptions
- < 11 >



## Создание объекта и обращение к нему

```
39
40 ►  public static void main(String[] args) {
41     GBox<Integer> integerBox0;
42     GBox<Integer> integerBox1 = new GBox<Integer>(1);
43
44
45
```



## Создание объекта и обращение к нему

```
39
40 ▶ public static void main(String[] args) {
41     GBox<Integer> integerBox0;
42     GBox<Integer> integerBox1 = new GBox<Integer>(1);
43     GBox<Integer> integerBox2 = new GBox<>(1);
44     GBox<Integer> integerBox3 = new GBox<>();
45
```

Expected 1 arguments but found 0  
Remove 1st parameter from constructor

- примитивные типы;
- ссылочные типы;
- интерфейсные типы;
- параметризованные типы.

## Варианты параметризации

```
39 private static class KVBox<K, V> {
40     private K key;
41     private V value;
42
43     public KVBox(K key, V value) {
44         this.key = key;
45         this.value = value;
46     }
47
48     public V getValue() { return value; }
49     public K getKey() { return key; }
50     public void showType() {
51         System.out.printf("Type of key is %s, key = %s, " +
52             "type of value is %s, value = %s\n",
53             key.getClass().getName(), getKey(),
54             value.getClass().getName(), getValue());
55     }
56 }
57
58 }
```

```
61
62 public static void main(String[] args) {
63     KVBox<Integer, String> kvb0 = new KVBox<>(1, "Hello");
64     KVBox<String, GBox<String>> kvb1 = new KVBox<>("World", new GBox<>("Java"));
65 }
```





## Raw Types или сырые типы

```
66 GBox box = new GBox(1);
67 box.get
68
69
```

get  
getValue() Object  
getClass Class<? extends...

Press ← to insert, → to replace [Next Tip](#)





## Raw Types или сырые типы

```
66 GBox<Integer> box = new GBox<>(1);
67 box.get
68     getValue() Integer
69     getClass Class<? extends...
Press ↵ to insert, ⇠ to replace Next Tip
```



## Raw Types или сырые типы

```
67 GBox<Integer> intBox = new GBox<>(1);  
68 GBox box = intBox;  
69
```



## Raw Types или сырые типы

```
67      GBox box = new GBox(1);
68      GBox<Integer> intBox = box;
69
70
71      GBox<Integer> intBox0 = new GBox<>(1);
72      GBox box0 = intBox0;
73      box.setValue(4);
```



## Сообщения об ошибках unchecked

```
Unchecked assignment: 'ru.gb.jdk.three.Main.GBox' to 'ru.gb.jdk.three.Main.GBox<java.lang.Integer>'
Change variable 'intBox' type to 'GBox'  More actions...

Main.GBox box = new Main.GBox(1)
JDKit
```

```
Unchecked call to 'setValue(T)' as a member of raw type 'ru.gb.jdk.three.Main.GBox'
Try to generify 'Main.java'  More actions...

ru.gb.jdk.three.Main.GBox<T>
@Contract(mutates = "this")
public void setValue(
    T value
)
JDKit
```

```
Terminal: Local + -
ivan-igorevich@MacBook-Pro-Ivan JDKit % javac -Xlint:unchecked src/ru/gb/jdk/three/Main.java
src/ru/gb/jdk/three/Main.java:67: warning: [unchecked] unchecked call to GBox(T) as a member of the raw type GBox
    GBox box = new GBox(1);
                  ^
    where T is a type-variable:
      T extends Object declared in class GBox
src/ru/gb/jdk/three/Main.java:68: warning: [unchecked] unchecked conversion
    GBox<Integer> intBox = box;
                        ^
    required: GBox<Integer>
    found:    GBox
```





# Обобщённые методы





## Обобщённые методы

```
62 @ private static <T> void setIfNull(GBox<T> box, T t) {  
63     if (box.getValue() == null) {  
64         box.setValue(t);  
65     }  
66 }  
67  
68 ▶ public static void main(String[] args) {  
69     GBox<Integer> box = new GBox<>(null);  
70     setIfNull(box, 13);  
71     System.out.println(box.getValue());  
72     GBox<Integer> box0 = new GBox<>(1);  
73     setIfNull(box0, 13);  
74     System.out.println(box0.getValue());
```

## Ответьте на вопросы сообщением в чат

### Вопросы:

- **Обобщения – это способ создания общих**
  - a. классов для разных пакетов;
  - b. алгоритмов для разных типов данных;
  - c. библиотек для разных приложений.
- **Что именно означает буква в угловых скобках**
  - a. название обобщённого класса;
  - b. имя класса, с которым будет работать обобщение;
  - c. название параметра, используемого в обобщении.
- **Возможно ли передать обобщённым аргументом примитивный тип?**
  - a. да
  - b. нет
  - c. только строку





## Ограниченные параметры типа

**Bounded type parameters** позволяют ограничить типы данных, которые могут быть использованы в качестве параметров. Использование bounded type parameters в Java является хорошей практикой, которая позволяет более точно определить используемые типы данных и обеспечивает более безопасный и читаемый код.







## Ограниченные параметры типа

```
Main.java x
61 private static class BBox<V extends Number> {
62     public V getValue() {
63         return value;
64     }
65     public void setValue(V value) {
66         this.value = value;
67     }
68     private V value;
69 }
```

## Ограниченные параметры типа

```
70  
71 @private static <T extends Number> void setIfNull(BBox<T> box, T t) {  
72     if (box.getValue() == null) {  
73         box.setValue(t);  
74     }  
75 }  
76  
77 public static void main(String[] args) {  
78     BBox<Integer> integerBBox = new BBox<>();  
79     BBox<String>  
80 }
```

Type parameter 'java.lang.String' is not within its bound: should extend 'java.lang.Number'  
Fix: java.lang

```
77 public static void main(String[] args) {  
78     BBox<Integer> integerBBox = new BBox<>();  
79     BBox<String> stringBBox = new BBox<>();  
80  
81     setIfNull(integerBBox, 4);  
82     setIfNull(stringBBox, "hello");  
83  
84     ///...  
119 }  
120 }
```

	Required type	Provided
box:	BBox<T>	BBox<String>
t:	T	String


reason: no instance(s) of type variable(s) exist so that String conforms to Number

```
71 class Bird{}  
72 interface Animal{}  
73 interface Man{}  
74 class CBox<T extends Bird & Animal & Man> {  
75     // ...  
76 }
```



## Обобщения, исследования и дочерние типы

Integer является Object

```
84 ▶  public static void main(String[] args) {  
85     Object someObject;  
86     Integer someInteger = new Integer(13);  
87     someObject = someInteger;  
88     System.out.println(someObject);  
}
```

## Обобщения, исследования и дочерние типы

```
84     private static void someMethod(Number n) { /* ... */ }
85
86     public static void main(String[] args) {
87         someMethod(new Integer(10));
88         someMethod(new Double(10.1));
89     }
```





## Обобщения, исследования и дочерние типы

Наследование не работает в дженериках Java так, как оно работает в обычной Java.

```
85
86     private static void boxTest(GBox<Number> n) { /* ... */ }
87
88     public static void main(String[] args) {
89         boxTest(new GBox<Number>(10));
90         boxTest(new GBox<Integer>(1));
91         boxTest(new GBox<Float>(1.0f));
92
```



## Обобщения, исследования и дочерние типы

```
78  @  private static <T extends Number> void setIfNull(BBox<T> box, T t) {...}
83      private static void someMethod(Number n) { /* ... */ }
84      private static void boxTest(GBox n) { /* ... */ }
85  ▶  public static void main(String[] args) {
86      boxTest(new GBox<Number>(10));
87      boxTest(new GBox<Integer>(1));
88      boxTest(new GBox<Float>(1.0f));
89
90      someMethod(new Integer(10));
91      someMethod(new Double(10.1));
```



## Ограничения (маскирование) «сверху» и «снизу»

```
86  @  private static <T extends Number> boolean compare(T src, T dst) {
87      return src.equals(dst);
88  }
89
90  ▶  public static void main(String[] args) {
91      System.out.println(compare(1, 1.0f));
92      System.out.println(compare(1.0f, 1.0f));
93      System.out.println(compare(1, 1));
94
95
```

Run: ru.gb.jdk.three.Main x

/Library/Java/JavaVirtualMachines/liberica-jdk-11.jdk/Contents/Home

false

true

true



## Ограничения (маскирование) «сверху» и «снизу»

```
93  @      public static void copyTo(ArrayList src, ArrayList dst) {  
94      for (Object o : src) dst.add(o);  
95  }  
96  
97  ▶      public static void main(String[] args) {  
98      ArrayList<Integer> ial = new ArrayList<>(Arrays.asList(1, 2, 3));  
99      ArrayList<Number> nal = new ArrayList<>(Arrays.asList(1f, 2, 3.0));
```



## Ограничения (маскирование) «сверху» и «снизу»

```
93  @ public static void copyTo(ArrayList src, ArrayList dst) {
94      for (Object o : src) dst.add(o);
95  }
96
97  public static void main(String[] args) {
98      ArrayList<Integer> ial = new ArrayList<>(Arrays.asList(1, 2, 3));
99      ArrayList<Number> nal = new ArrayList<>(Arrays.asList(1f, 2, 3.0));
100      System.out.println(ial);
101      System.out.println(nal);
102      copyTo(ial, nal);
103      System.out.println(nal);
104      copyTo(nal, ial);
105      System.out.println(ial);
106
Run: ru.gb.jdk.three.Main
[1, 2, 3]
[1.0, 2, 3.0]
[1.0, 2, 3.0, 1, 2, 3]
[1, 2, 3, 1.0, 2, 3.0, 1, 2, 3]
```



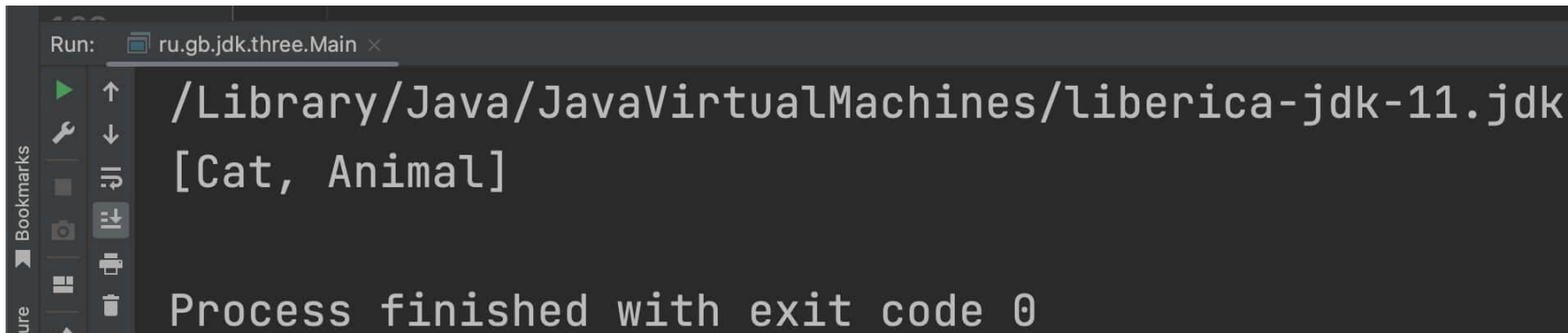


## Ограничения (маскирование) «сверху» и «снизу»

```
91  @ public static void copyTo(ArrayList src, ArrayList dst) { for (Object o : src) dst.add(o); }
94
95  private static class Animal {
96      protected String name;
97      protected Animal() { this.name = "Animal"; }
98  @Override public String toString() { return name; }
99  }
100 private static class Cat extends Animal {
101     protected Cat() { this.name = "Cat"; }
102 }
103
104 public static void main(String[] args) {
105     ArrayList<Cat> cats = new ArrayList<>(Arrays.asList(new Cat()));
106     ArrayList<Animal> animals = new ArrayList<>(Arrays.asList(new Animal()));
107     copyTo(animals, cats);
108     System.out.println(cats);
}
```



## Ограничения (маскирование) «сверху» и «снизу»



```
Run: ru.gb.jdk.three.Main x
/Library/Java/JavaVirtualMachines/liberica-jdk-11.jdk
[Cat, Animal]
Process finished with exit code 0
```



## Ограничения (маскирование) «сверху» и «снизу»

```
100     private static class Cat extends Animal {
101         protected Cat() { this.name = "Cat"; }
102         public void voice(){ System.out.println("meow"); }
103     }
104
105     public static void main(String[] args) {
106         ArrayList<Cat> cats = new ArrayList<>(Arrays.asList(new Cat()));
107         ArrayList<Animal> animals = new ArrayList<>(Arrays.asList(new Animal()));
108         copyTo(animals, cats);
109         System.out.println(cats);
110         cats.get(1).voice();
111     }
```



## Ограничения (маскирование) «сверху» и «снизу»

```
109      System.out.println(cats);
110      cats.get(1).voice();
111
112
```

Run: ru.gb.jdk.three.Main x

/Library/Java/JavaVirtualMachines/liberica-jdk-11.jdk/Contents/Home/bin/java -ja  
[Cat, Animal]  
Exception in thread "main" java.lang.ClassCastException Create breakpoint : class ru  
at ru.gb.jdk.three.Main.main(Main.java:110)

marks

Process finished with exit code 1



## Ограничения (маскирование) «сверху» и «снизу»

```
91  @      public static <T> void copyTo (  
92          ArrayList<? extends T> src, ArrayList<? super T> dst) {  
93          for (T o : src) {  
94              dst.add(o);  
95          }  
96      }
```

```
108  ►      public static void main(String[] args) {  
109          ArrayList<Cat> cats = new ArrayList<>(Arrays.asList(new Cat()));  
110          ArrayList<Animal> animals = new ArrayList<>(Arrays.asList(new Animal()));  
111          copyTo(animals, cats);  
112          System.out.println(cats);  
113          cats.get(1).voice();
```



# Выведение ТИПОВ





## Выведение типов

Выведение типов — это возможность компилятора автоматически определять аргументы типа на основе контекста.

```
98  private static class Animal {...}
103  private static class Cat extends Animal implements Serializable {...}
107  private static <T> T pick(T first, T second) { return second; }
108
109  public static void main(String[] args) {
110      Serializable se1 = pick("d", new Cat());
111      Serializable se2 = pick("d", new ArrayList<String>());
```





## Выведение типов

Выведение типов и обобщённые методы

```
public static <U> void addBox(U u, List<Box<U>> boxes) {  
    App.<Cat>addBox(new Cat("Kusya"), catsInBoxes);  
    addBox(new Cat("Kusya"), catsInBoxes);  
}
```

Выведение типов и создание экземпляра обобщённого класса, бриллиантовая операция

Выведение типа и обобщённые конструкторы обобщённых и необобщённых классов

```
public class Box<T> {  
    <U> Box(U u) {  
    }
```



# Целевые типы





## Выведение типов

Целевой тип выражения — это тип данных, который компилятор Java ожидает в зависимости от местоположения выражения.

```
108
109     public static class TBox<T> {
110         public static final TBox EMPTY_BOX = new TBox<>();
111         private T value;
112
113         public T getValue() { return value; }
114         public void setValue(T value) { this.value = value; }
115         static <T> TBox<T> emptyBox() {
116             return (TBox<T>) EMPTY_BOX;
117         }
118     }
119
120     public static void main(String[] args) {
121         TBox<String> box = TBox.emptyBox();
122     }
```



## Ответьте на вопросы сообщением в чат

### Вопросы:

- **Что из следующего является недопустимым?**

- a. `ArrayList<? extends Number> al1 = new ArrayList<Number>();`
- b. `ArrayList<? extends Number> al2 = new ArrayList<Integer>();`
- c. `ArrayList<? extends Number> al3 = new ArrayList<String>();`
- d. Всё допустимо

параметры метода `ArrayList<? extends T> src, ArrayList<? super T> dst`  
вызов метода `copyTo(cats, animals);`

- **Какой тип данных будет взят в качестве T?**

- a. `Animal`
- b. `Cat`
- c. `Object`





Подстановочный  
символ (wildcard)



## Подстановочный символ <?>

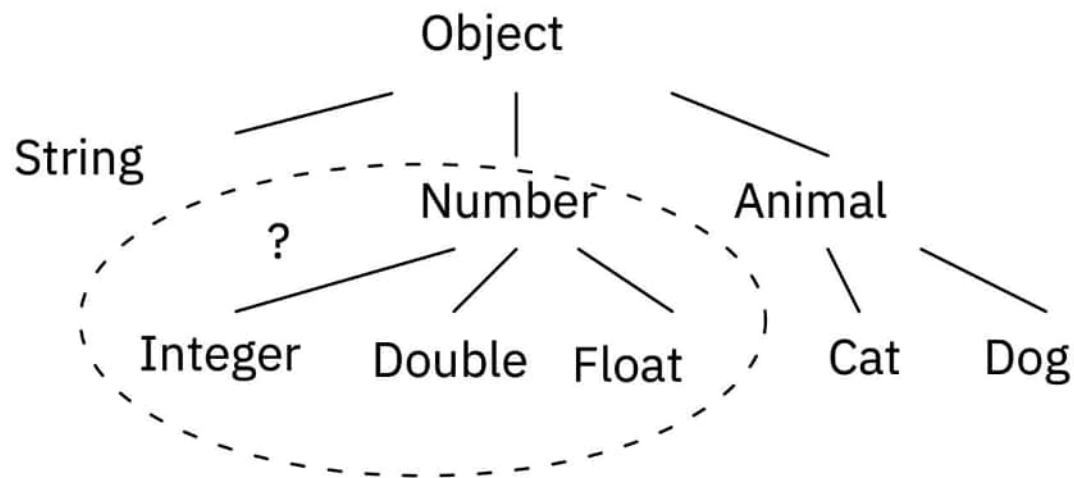
Подстановочный символ никогда не используется в качестве аргумента типа для вызова обобщённого метода, создания экземпляра обобщённого класса или супертипа

```
119 |  
120 private static <?> T method(T first, T second) { return second; }  
121 ▶ | public static void main(String[] args) {  
122     TBox<String> b = new TBox<?>();  
123 }
```





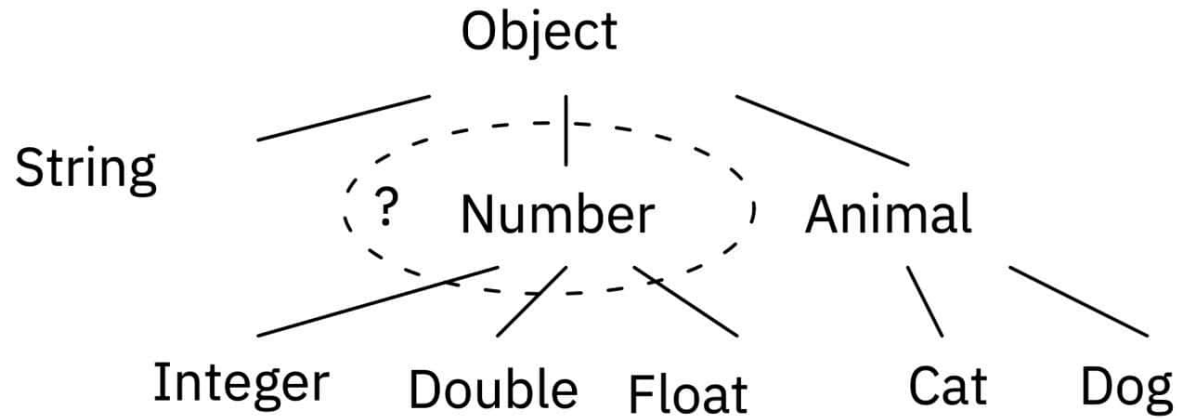
## Подстановочный символ <?> ограниченный сверху



Box <? extends Number>



## Подстановочный символ <?> ограниченный сверху



Box <Number>





## Подстановочный символ <?> ограниченный сверху

```
97
98   private static class Animal {
99       protected String name;
100       protected Animal(String name) { this.name = name; }
101       @Override public String toString() {
102           return this.getClass().getSimpleName() + " with name " + name;
103       }
104   }
105   private static class Cat extends Animal {
106       protected Cat(String name) { super(name); }
107   }
108   private static class Dog extends Animal {
109       protected Dog(String name) { super(name); }
110   }
111
```



## Подстановочный символ <?> ограниченный сверху

```
99
100 public static class TBox<T> {
101     public static final TBox EMPTY_BOX = new TBox<>();
102     private T value;
103
104     public T getValue() { return value; }
105     public void setValue(T value) { this.value = value; }
106     static <T> TBox<T> emptyBox() { return (TBox<T>) EMPTY_BOX; }
107     @Override public String toString() { return value.toString(); }
108 }
109
110
111
112
113
114 private static class Animal {...}
121 private static class Cat extends Animal {...}
124 private static class Dog extends Animal {...}
127
128 static void printInfo(TBox<? extends Animal> animalInBox){
129     System.out.println("Information about animal: " + animalInBox);
130 }
131
```



## Подстановочный символ <?> ограниченный сверху

```
132  ▶  public static void main(String[] args) {
133      TBox<Cat> catInBox = TBox.emptyBox();
134      catInBox.setValue(new Cat("Vasya"));
135      printInfo(catInBox);
136
137      TBox<Dog> dogInBox = TBox.emptyBox();
138      dogInBox.setValue(new Dog("Kusya"));
139      printInfo(dogInBox);

```

Run: ru.gb.jdk.three.Main x

```
Information about animal: Cat with name Vasya
Information about animal: Dog with name Kusya

```



## Неограниченный подстановочный символ <?>

1. У обобщённого типа используются только методы `Object`;
2. В методе используется только функциональность контейнера.

Например, часто используется тип `Class<?>`, потому что для `Class<T>` не используются методы `T`.

**Важно!** `Box<Object>` и `Box<?>` это не одно и то же





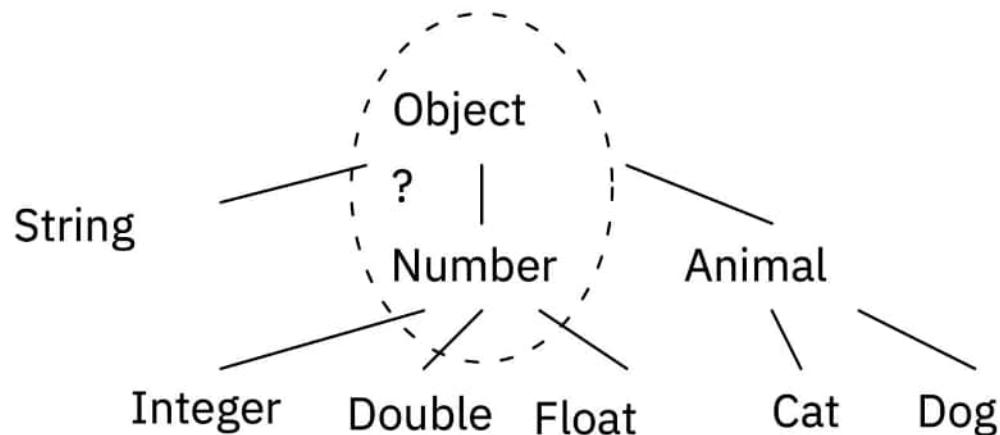
## Неограниченный подстановочный символ <?>

```
127
128 static void printInfo(TBox<?> animalInBox){
129     System.out.println("Information about animal: " + animalInBox);
130 }
131
```

```
127
128 static void printInfo(TBox<Object> animalInBox){
129     System.out.println("Information about animal: " + animalInBox);
130 }
131
132 public static void main(String[] args) {
133     TBox<Cat> catInBox = TBox.emptyBox();
134     catInBox.setValue(new Cat("Vasya"));
135     printInfo(catInBox);
136
137     TBox<Dog> dogInBox = TBox.emptyBox();
138     dogInBox.setValue(new Dog("Boby"));
139     printInfo(dogInBox);
}
```

Required type: TBox <Object>  
Provided: TBox <Cat>  
Change 1st parameter of method 'printInfo' from 'TBox<Object>' to 'TBox<Cat>'  
Main.TBox<Main.Cat> catInBox = TBox.emptyBox()  
JDK 11

## Подстановочный символ <?> ограниченный снизу



Box <? super Number>

```
127
128 static void printInfo(TBox<? extends Number & ? super Integer> animalInBox){
129     System.out.println("Information about a
130 }
131
```

'>' or ',' expected  
Identifier expected  
java.lang



## Подстановочный символ <?> ограниченный снизу

```
127
128     static void printInfo(TBox<? super Animal> animalInBox){
129         System.out.println("Information about animal: " + animalInBox);
130     }
131
132     public static void main(String[] args) {
133         TBox<Cat> catInBox = TBox.emptyBox();
134         catInBox.setValue(new Cat("Vasya"));
135         printInfo(catInBox);
136
137         TBox<Dog> dog
```

Required type: TBox <? super Animal>

Provided: TBox <Cat>



## Подстановочный символ <?> и дочерние типы

```
117  private static class Animal {...}
124  private static class Cat extends Animal {...}
127  private static class Dog extends Animal {...}
130
131  public static void main(String[] args) {
132      Cat cat = new Cat("Vasya");
133      Animal animal = cat;
134
```

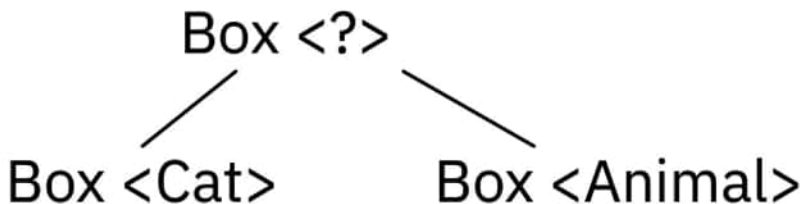




## Подстановочный символ <?> и дочерние типы

```
117 private static class Animal {...}
124 private static class Cat extends Animal {...}
127 private static class Dog extends Animal {...}
130
131 public static void main(String[] args) {
132     Cat cat = new Cat("Yasya");
133     Animal animal = cat;
134
135     TBox<Cat> catInBox = new TBox<>();
136     TBox<Animal> animalInBox = catInBox; // Incompatible types
137
138     //
139 }
```

Required type: TBox <Animal>  
Provided: TBox <Cat>  
Change variable 'animalInBox' type to 'TBox<Cat>' More actions...







## Подстановочный символ <?> и дочерние типы

```
116
117   private static class Animal {...}
124   private static class Cat extends Animal {...}
127   private static class Dog extends Animal {...}
130
131   public static void main(String[] args) {
132       Cat cat = new Cat("Vasya");
133       Animal animal = cat;
134
135       TBox<? extends Cat> catInBox = new TBox<>();
136       TBox<? extends Animal> animalInBox = catInBox; // OK
137
```



## Захват подстановочного символа и вспомогательные методы

```
130
131 @   static void testError(TBox<?> box){
132     box.setValue(box.getValue());
133 }
134
135
```

Required type: capture of ?  
Provided: capture of ?  
Change 1st parameter of method 'setValue' from 'T' to 'Object'  More actions... 





## Захват подстановочного символа и вспомогательные методы

```
130
131 @  static void testError(TBox<?> box){
132     box.setValue(box.getValue());
133 }
134
135
```

Required type: capture of ?  
Provided: capture of ?  
Change 1st parameter of method 'setValue' from 'T' to 'Object' More actions...

```
130
131 @  private static <T> void testErrorHelper(TBox<T> box){
132     box.setValue(box.getValue());
133 }
134
135  static void testError(TBox<?> box){
136     testErrorHelper(box);
137 }
138
```



## Руководство по использованию подстановочного символа

Входная переменная. Предоставляет данные для кода. Для метода `copy(src, dst)` параметр `src` предоставляет данные для копирования, поэтому он считается входной переменной.

Выходная переменная. Содержит данные для использования в другом месте. В примере `copy(src, dst)` параметр `dst` принимает данные и будет считаться выходной переменной.

- Входная переменная определяется с ограничением сверху
- Выходная переменная определяется с ограничением снизу
- Если ко входной переменной можно обращаться только как к `Object` – неограниченный подстановочный символ.
- Если переменная должна использоваться как входная и как выходная одновременно, НЕ использовать подстановочный символ.
- Не использовать подстановочные символы в возвращаемых типах

## Ответьте на вопросы сообщением в чат

### Вопросы:

- **Ограниченный снизу подстановочный символ позволяет передать в качестве аргумента типа**
  - a. только родителей типа;
  - b. только наследников типа;
  - c. сам тип и его родителей;
  - d. сам тип и его наследников.
- **Конструкция `<? super Object>`**
  - a. не скомпилируется;
  - b. не имеет смысла;
  - c. не позволит ничего передать в аргумент типа;
  - d. не является чем-то необычным.





# Стирание типа (Type Erasure) и загрязнение кучи





## Стирание типа (Type Erasure)

- заменяет все параметры типа их границами или Object.
- вставляет приведение типов.
- генерирует связующие методы.
- обеспечивает отсутствие новых классов для параметризованных типов.







## Стирание типа в обобщённых методах

```
138 @ private static <T> void setIfNull(TBox<T> box, T t) {
139     if (box.getValue() == null) {
140         box.setValue(t);
141     }
142 }
143
144
145 @ private static void setIfNull(TBox<Object> box, Object t) {
146     if (box.getValue() == null) {
147         box.setValue(t);
148     }
149 }
150
```

IDE tooltip message:

'setIfNull(TBox<T>, T)' clashes with 'setIfNull(TBox<Object>, Object)'; both methods have same erasure  
Fix method 'setIfNull' parameters with bounded wildcard: [More actions...](#)

ru.gb.jdk.three.Main

```
private static <T> void setIfNull(
    @NotNull TBox<T> box,
    T t
)
```

JDKKit



## Стирание типа в обобщённых методах

- Материализуемые типы (reifiable types) — это типы, информация о которых полностью доступна во время выполнения
- Нематериализуемые типы (Non-reifiable types) — это типы, информация о которых удаляется во время компиляции стиранием типов





## Загрязнение кучи (Heap pollution)

Загрязнение кучи (heap pollution) возникает, когда переменная параметризованного типа ссылается на объект, который не является параметризованным типом.

Такая ситуация возникает, если программа выполнила операцию, которая генерирует unchecked warning.

Предупреждение unchecked warning генерируется, если правильность операции с параметризованным типом не может быть проверена.



# Ограничения обобщений



## Ограничения обобщений

Нельзя создавать экземпляры обобщённых типов с примитивными типами в качестве аргументов типа.



```
155  
156     TBox<int> box = new TBox<>();  
157
```





## Ограничения обобщений

Нельзя создавать экземпляры параметров типа

```
149
150  @  static <T> void add(Box<T> box) {
151      T t = new T();
152      // ...
153      box.setValue(t);
154       }
155
```



## Ограничения обобщений

Нельзя объявлять статические поля с типом параметра типа.

```
149 |
150 | public class SBox<T> {
151 |     private static T value;
152 |
153 |     // ...
154 | }
155 |
156 |
```

Static declarations in inner classes are not supported at language level '11'

[Upgrade JDK to 16+](#) [More actions...](#)

ru.gb.jdk.three.Main.SBox<T>

private static T value

JDKit



## Ограничения обобщений

Нельзя использовать приведения типа или instanceof с параметризованными типами

```
163  
164     TBox<Integer> box1 = new TBox<>();  
165     TBox<Number> box2 = (TBox<Number>) box1;  
166
```





## Ограничения обобщений

Невозможно создавать массивы параметризованных типов.

```
159
160     Object[] stringLists = new TBox<String>[];
161     // compilation error, but let's say it's possible
162     stringLists[0] = new TBox<String>(); // OK
163     stringLists[1] = new TBox<Integer>(); // here should be
164                                           // an ArrayStoreException exception,
165                                           // but the runtime cannot notice it.
166
```



## Ограничения обобщений

Нельзя создавать, ловить (catch) или бросать (throw) объекты параметризованных типов

```
157 // Extends Throwable non-direct
158 class MathException<T> extends Exception { /* ... */ } // compilation error
159
160 // Extends Throwable directly
161 class QueueFullException<T> extends Throwable { /* ... */ } // compilation error
162
163 @
164 public static <T extends Exception, J> void execute(TBox<J> box) {
165     try {
166         J j = box.getValue();
167         // ...
168     } catch (T e) { // compilation error
169     }
170 }
171
172 class Parser<E> extends Exception {
173     public void parse(File file) throws T { // OK
174         // ...
175     }
176 }
```



## Ограничения обобщений

Нельзя перегружать метод так, чтобы формальные параметры типа стирались в один и тот же сырой тип.

```
138 @ private static <T> void setIfNull(TBox<T> box, T t) {
139     if (box.getValue() == null) {
140         box.setValue(t);
141     }
142 }
143
144
145 @ private static void setIfNull(TBox<Object> box, Object t) {
146     if (box.getValue() == null) {
147         box.setValue(t);
148     }
149 }
150
```

'setIfNull(TBox<T>, T)' clashes with 'setIfNull(TBox<Object>, Object)'; both methods have same erasure  
Fix method 'setIfNull' parameters with bounded wildcard: [More actions...](#)

ru.gb.jdk.three.Main

```
private static <T> void setIfNull(
    @NotNull Main.TBox<T> box,
    T t
)
```

JDKit



## На этом уроке

- 📌 Обобщённое программирование
- 📌 Diamond operator
- 📌 Обобщённые методы
- 📌 Подстановочные символы (Wildcards)
- 📌 Ограничения сверху и снизу
- 📌 Выведение типов и целевые типы
- 📌 Стирание типа
- 📌 Загрязнение кучи





## Практическое задание

Написать метод, который меняет два элемента массива местами.(массив может быть любого ссылочного типа);

### **Большая задача:**

a. Есть классы Fruit -> Apple, Orange; (больше не надо)

b. Класс Box в который можно складывать фрукты, коробки условно сортируются по типу фрукта, поэтому в одну коробку нельзя сложить и яблоки, и апельсины; Для хранения фруктов внутри коробки можете использовать ArrayList;

c. Сделать метод `getWeight()` который высчитывает вес коробки, зная количество фруктов и вес одного фрукта (вес яблока — 1.0f, апельсина — 1.5f, не важно в каких единицах);

d. Внутри класса коробки сделать метод `compare`, который позволяет сравнить текущую коробку с той, которую подадут в `compare` в качестве параметра, `true` — если их веса равны, `false` в противном случае(коробки с яблоками мы можем сравнивать с коробками с апельсинами);

e. Написать метод, который позволяет пересыпать фрукты из текущей коробки в другую коробку(помним про сортировку фруктов, нельзя яблоки высыпать в коробку с апельсинами), соответственно в текущей коробке фруктов не остается, а в другую перекидываются объекты, которые были в этой коробке.



Важно не количество знаний, а качество их. Можно  
знать очень многое, не зная самого нужного.

Л.Н.Толстой

