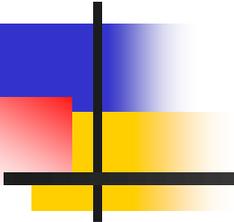
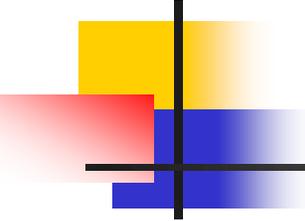


Generics

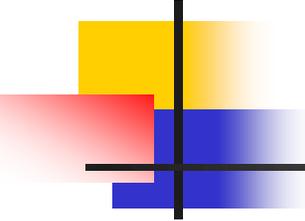


Параметризация типов



Полезные ссылки

- <http://www.quizful.net/post/java-generics-tutorial>
- <http://www.ibm.com/developerworks/ru/library/j-jtp04298/>



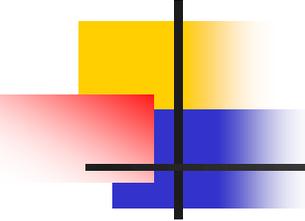
Как было Java 1

```
List myIntList = new LinkedList();  
myIntList.add(new Integer(0));  
myIntList.add(new Double(3.14));  
...  
Integer x = (Integer) myIntList.iterator().next();
```

Возможные проблемы

– Runtime error

– Или выяснять тип объекта



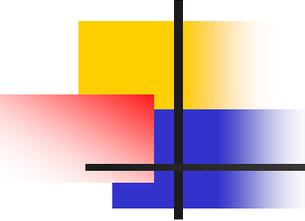
Что дает *generic*?

```
List<Integer> myIntList =  
    new LinkedList<Integer>();  
myIntList.add(new Integer(0));  
. . .  
Integer x = myIntList.iterator().next();
```

В случае неправильного использования

```
myIntList.add(new Double(3.14));
```

- Ошибка компиляции

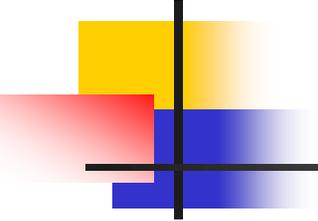


Общий вид объявления параметризованного класса

```
class имя-класса <список-параметров-типа>    { // ...  
}
```

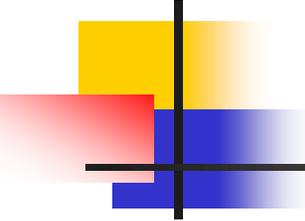
Объявление ссылки на объект параметризованного класса

```
class имя-класса<список-аргументов-типа>  
    имя-переменной =  
        new имя-класса <список-аргументов-типа>    ( . . . ) ;
```



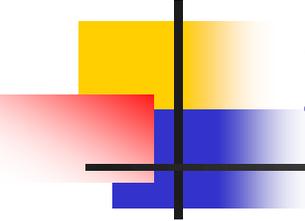
Пример описания

```
class Gen<T> {  
    T ob;  
    Gen(T o) { ob = o; }  
    T getob() { return ob; }  
    void showType() {  
        System.out.println("Type of T is " +  
            ob.getClass().getName());  
    }  
}
```



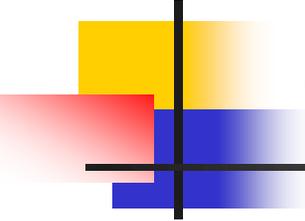
Пример использования

```
class GenDemo {
    public static void main(String args[]) {
        Gen<Integer> iOb;
        iOb = new Gen<Integer>(88);
        iOb.showType(); //!!!!!!!!!!!!
        int v = iOb.getob();
        System.out.println("value: " + v);
        Gen<String> strOb =
            new Gen<String>("Generics Test");
        strOb.showType(); //!!!!!!!!!!!!
        String str = strOb.getob();
        System.out.println("value: " + str);
    }
}
```



Несколько параметров

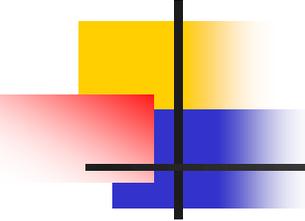
```
class Pair <T,W> { //несколько параметров
    T  ob1;
    W  ob2;
    Pair(T o1, W o2) {
        ob1 = o1;
        ob2 = o2;
    }
    . . .
}
```



Generic methods

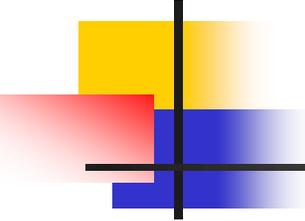
```
class Utilities {  
    public static <T> void fill(List<T> list, T val) {  
        for (int i = 0; i < list.size(); i++)  
            list.set(i, val);  
    }  
}
```

Класс при этом может быть не Generic



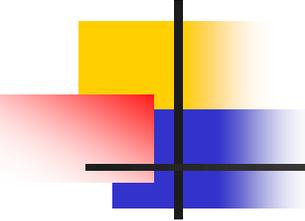
Параметризация интерфейса

```
public interface List<E> {  
    void add(E x);  
    Iterator<E> iterator();  
}  
  
public interface Iterator<E> {  
    E next();  
    boolean hasNext();  
}
```

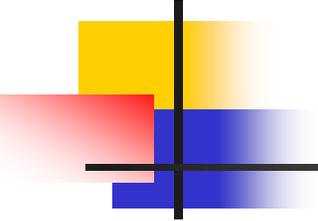


Стирание типа

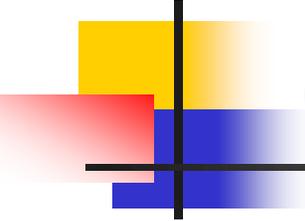
- Внутри параметризованного кода вся конкретная информация о типе утрачивается
- Становится невозможным выполнение некоторых операций в параметризованном типе



```
public class Erased <T>{
    final int SIZE=100;
    public static void f (Object arg){
        if (arg instanceof T) {} // ERROR!!!
    }
    public static void ff(){
        T var = new T(); // ERROR!!! например у Integer нет
        T[] arr = new T[SIZE]; // ERROR!!!
        T[] arr = (T[ ])new Object[SIZE]; //WARNING, но допустимо
    }
}
```

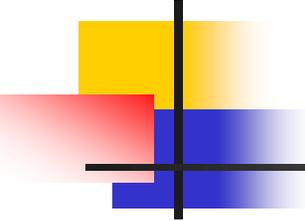


```
public class Stack <T> {  
    private int size, top;  
    private T [] st;  
    public Stack(){  
        this(10);  
    }  
    public Stack(int n){  
        size = n; top=0;  
        T[] st = (T[])new Object[SIZE];  
    }  
    public Stack(T[] el){ // ЛУЧШЕ ИСПОЛЬЗОВАТЬ ТАКОЙ КОНСТРУКТОР  
        size = el.length; top=0; st= el; }  
}
```



Ограниченные типы (bounded types)

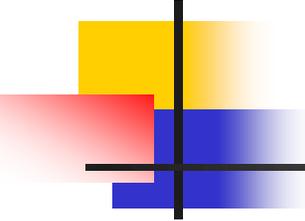
```
class Stats<T> {  
    T[] nums;  
    Stats(T[] o) {  
        nums = o;  
    }  
}
```



Проблема

```
double average() {  
    double sum = 0.0;  
    for(int i=0; i < nums.length; i++)  
        sum += nums[i].doubleValue();  
    return sum / nums.length;  
}  
}
```

Из-за стирания типа массив `nums` состоит из `Object` и не имеет метода `doubleValue()`

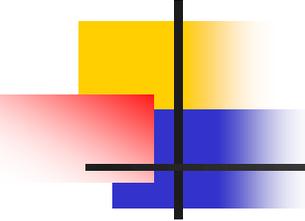


Ограниченные типы

```
// аргумент типа для
// T должен быть Number, или производный
// от Number класс

class Stats<T extends Number> {
    T[] nums;

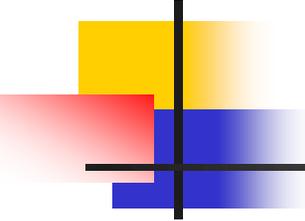
    // массив типа Number или его подкласса
```



Пример использования

```
Integer inums[] = {1, 2, 3, 4, 5 };
Double dnums[]={1.1, 2.2, 3.3, 4.4, 5.5};

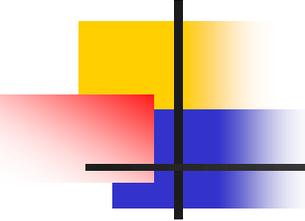
Stats<Integer> intob = new Stats<Integer>(inums);
Stats<Double> dob = new Stats<Double>(dnums);
double v = intob.average();
System.out.println("intob average is "+ v);
double w = dob.average();
System.out.println("dob average is "+ w);
```



Следующая проблема

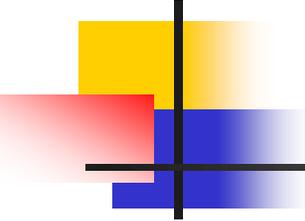
```
double v = intob.average();  
System.out.println("intob average is "+ v);  
double w = dob.average();  
System.out.println("dob average is "+ w);
```

Как сделать метод, который определяет, равны ли средние арифметические двух объектов класса `Stats<T extends Number>` ?



Метасимвольные аргументы (wildcard argument)

```
boolean sameAvg(Stats<?> ob) {  
    if (Math.abs(average()  
                - ob.average()) < 1.0e-10)  
        return true;  
    return false;  
}  
//метасимвол ? «смягчает» контроль,  
// позволяя сравнить intob.sameAvg(dob)
```



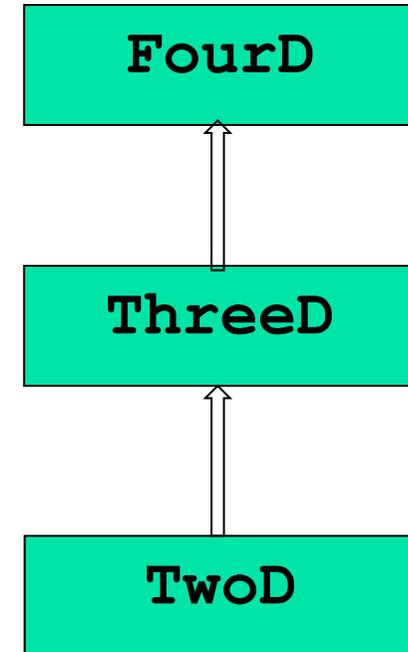
Метасимвольные аргументы (пример)

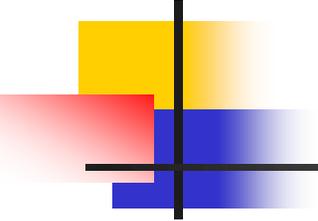
```
class WildCardUse {
    static void printList(List<?> list) {
        for(Object l:list)
            System.out.println "[" + l + " ]");
    }
    static void main(String []args) {
        List<Integer> list = new ArrayList<>();
        list.add(10);list.add(100);
        printList(list);
        List<String> strList = new ArrayList<>();
        strList.add("1a"); strList.add("ball-100");
        printList(strList);
    }
}
```

Ограниченные метасимвольные аргументы

Рассмотрим иерархию наследования классов

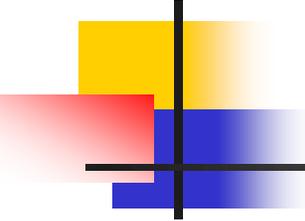
```
class TwoD {  
    int x, y;  
    TwoD(int a, int b) {  
        x = a;  
        y = b;  
    }  
}
```





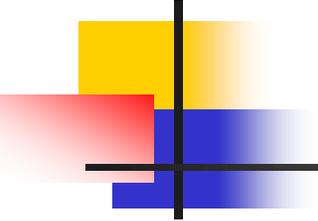
```
class ThreeD extends TwoD {  
    int z;  
    ThreeD(int a, int b, int c) {  
        super(a, b);  
        z = c;  
    }  
}
```

```
class FourD extends ThreeD {  
    int t;  
    FourD(int a, int b, int c, int d) {  
        super(a, b, c);  
        t = d;  
    }  
}
```



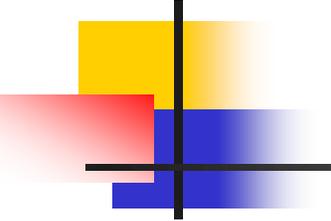
Массив координат

```
class Coords<T extends TwoD> {  
    T[] coords;  
    Coords(T[] o) { coords = o; }  
}
```



Можно использовать для массива любых координат

```
static void showXY(Coords<?> c)    {  
System.out.println("X Y Coordinates;");  
for(int i=0; i < c.coords.length; i++)  
    System.out.println(c.coords[i].x + "    "  
        +c.coords[i].y);  
    System.out.println();  
}
```



Можно использовать для массива координат не меньше,
чем ThreeD

```
static void showXYZ(Coords<? extends ThreeD> c)    {  
    System.out.println("X Y Z Coordinates:");  
    for(int i=0; i < c.coords.length; i++)  
        System.out.println(c.coords[i].x + "    " +  
                            c.coords[i].y + "    " +  
                            c.coords[i].z);  
    System.out.println();  
}
```

Метасимволы супертипов (ограничение на тип сверху)

`<? super T>`

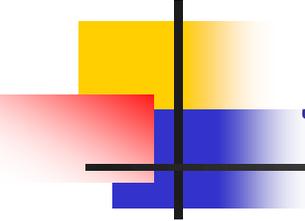
для любого предка `T`, вплоть до `Object`

Когда используем (пример – алгоритм `copy`)

Принцип PECS (*provider - extends, consumer - super*)

```
public static <T>
```

```
    void copy(List<? super T> dest,  
             List<? extends T> src)
```



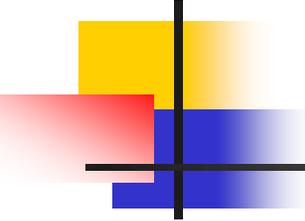
Java 7 (Diamond syntax)

```
List <Point> p = new ArrayList<Point>();
```

```
List <Point> p1 = new ArrayList <TwoD>();//!!!! ОШИБКА
```

```
List <Point> p2 = new ArrayList<>();
```

```
Pair<Integer, String> pair = new Pair<>(6, " Apr");
```



Зачем нужно знать о Generics

- Конечно, если разрабатываем свою библиотеку классов, требующую параметризации
- Для понимания организации библиотек JDK, особенно, библиотеки коллекций `java.util.*` (вспомним, с какого примера начиналось)