Generics

Начиная с JDK 1.5, в Java появляются новые возможности для программирования. Одним из таких нововведений являются Generics. Generics являются аналогией с конструкцией "Шаблонов"(template) в С++, но имеет свои нюансы. Generics позволяют абстрагировать множество типов. Наиболее распространенными примерами являются Коллекции.

Вот типичное использование такого рода (без Generics):

1. List myIntList = new LinkedList();

2. myIntList.add(new Integer(0));

3. Integer x = **(Integer)** myIntList.iterator().next();

Как правило, программист знает, какие данные должны быть в List'e. Тем не менее, стоит обратить особое внимание на Приведение типа (**"Cast"**) в строчке 3. Компилятор может лишь гарантировать, что метод next() вернёт Object, но чтобы обеспечить присвоение переменной типа Integer правильным и безопасным, требуется Cast. Cast не только создает беспорядки, но дает возможность появление ошибки "Runtime Error" из-за невнимательности программиста.

И появляется такой вопрос: "Как с этим бороться? " В частности: "Как же зарезервировать List для определенного типа данных?"

Как раз такую проблему решают Generics.

1. List**<Integer>** myIntList = new LinkedList**<Integer>**();

2. myIntList.add(new Integer(0));

3. Integer x = myIntList.iterator().next();

Обратите внимание на объявления типа для переменной myIntList. Он указывает на то, что это не просто произвольный List, а List<Integer>. Мы говорим, что List является generic-интерфейсом, который принимает параметр типа - в этом случае, Integer. Кроме того, необходимо обратить внимание на то, что теперь Cast выполняется в строчке 3 автоматически.

Некоторые могут задуматься, что беспорядок в коде увеличился, но это не так. Вместо приведения к Integer в строчке 3, у нас теперь есть Integer в качестве параметра в строчке 1. Здесь существенное отличие. Теперь компилятор может проверить этот тип на корректность во время компиляции.

**И когда мы говорим**, что myIntList объявлен как List<Integer>, это будет ***справедливо*** во всем коде и компилятор это гарантирует.

* На заметку:

Эффект от Generics особенно проявляется в крупных проектах: он улучшает читаемость и надежность кода в целом.

[[править](https://neerc.ifmo.ru/wiki/index.php?title=Generics&action=edit&section=2)]Свойства

* Строгая типизация
* Единая реализация
* Отсутствие информации о типе

[[править](https://neerc.ifmo.ru/wiki/index.php?title=Generics&action=edit&section=3)]Пример реализации Generic-класса

public interface List<E> {

E get(int i);

set(int i, E e);

add(E e);

Iterator<E> iterator();

…

}

Для того чтобы использовать класс как Generics, мы должны прописать после имени класса **<...>**, куда можно подставить любое имя, wildcard и т.д.

После того как было объявлено имя generic-типа его можно использовать как обычный тип внутри метода. И когда в коде будет объявлен, к примеру, List<Integer>, то Е станет Integer для переменной list (как показано ниже).

Теперь рассмотрим чем старая реализация кода отличается от новой:

List<E> ─ список элементов E

**Раньше** :

List list = new List();

list.add(new Integer(1));

Integer i = (Integer) list.get(0);

**Теперь** :

List<Integer> list = new List<Integer>();

list.add(new Integer(1));

Integer i = list.get(0);

Как видите, больше не нужно приводить Integer, так как метод get() возвращает ссылку на объект конкретного типа (в данном случае – Integer).

[[править](https://neerc.ifmo.ru/wiki/index.php?title=Generics&action=edit&section=4)]Несовместимость generic-типов

Это одна из самых важных вещей, которую вы должны узнать о Generics

Как говориться: "В бочке мёда есть ложка дегтя". Для того чтобы сохранить целостности и независимости друг от друга Коллекции, у Generics существует так называемая "Несовместимость generic-типов".

* Суть такова:

Пусть у нас есть тип Foo, который является подтипом Bar, и еще G - наследник Коллекции.

То G<Foo> **не является** наследником G<Bar>.

* **Пример**:

List<Integer> li = new ArrayList<Integer>();

List<Object> lo = li;

Иначе — ошибки

lo.add(“hello”);

// **ClassCastException**: String -> int

Integer li = lo.get(0);

[[править](https://neerc.ifmo.ru/wiki/index.php?title=Generics&action=edit&section=5)]Проблемы реализации Generics

* Решение 1 - **Wildcard**

Пусть мы захотели написать метод, который берет Collection<Object> и выводит на экран. И мы захотели вызвать dump для Integer.

* **Проблема**

void dump(Collection<Object> c) {

for (Iterator<Object> i = c.iterator(); i.hasNext(); ) {

Object o = i.next();

System.out.println(o);

}

}

List<Object> l; dump(l);

List<Integer> l; dump(l); // **Ошибка**

В этом примере List<Integer> не может использовать метод dump, так как он не является подтипом List<Object>.

Проблема в том что эта реализация кода не эффективна, так как Collection<Object> не является полностью родительской коллекцией всех остальных коллекции, грубо говоря Collection<Object> имеет ограничения.

Для решения этой проблемы используется Wildcard ("?"). Он не имеет ограничения в использовании(то есть имеет соответствие с любым типом) и в этом его плюсы. И теперь, мы можем назвать это с любым типом коллекции.

* **Решение**

void dump(Collection<?> c) {

for (Iterator<?> i = c.iterator(); i.hasNext(); ) {

Object o = i.next();

System.out.println(o);

}

}

* Решение 2 – **Bounded Wildcard**

Пусть мы захотели написать метод, который рисует List<Shape>. И у Shape есть наследник Circle. И мы хотим вызвать draw для Circle.

* **Проблема**

void draw(List<Shape> c) {

for (Iterator<Shape> i = c.iterator(); i.hasNext(); ) {

Shape s = i.next();

s.draw();

}

}

List<Shape> l; draw(l);

List<Circle> l; draw(l); // **Ошибка**

Проблема в том, что у нас не получиться из-за несовместимости типов. Предложенное решение используется, если метод который нужно реализовать использовал бы определенный тип и его подтипов. Так называемое "Ограничение сверху". Для этого нужно вместо <Shape> прописать <? extends Shape>.

* **Решение**

void draw(List<? extends Shape> c) {

for (Iterator<? extends Shape> i = c.iterator();

i.hasNext(); ) {

Shape s = i.next();

s.draw();

}

}

* Решение 3 – **Generic-Метод**

Пусть вы захотели сделать метод, который берет массив Object и переносить их в коллекцию.

* **Проблема**

void addAll(Object[] a, Collection<?> c) {

for (int i = 0; i < a.length; i++) {

c.add(a[i]);

}

}

addAll(new String[10], new ArrayList<String>());

addAll(new Object[10], new ArrayList<Object>());

addAll(new Object[10], new ArrayList<String>()); // **Ошибка**

addAll(new String[10], new ArrayList<Object>()); // **Ошибка**

Напомним, что вы не можете просто засунуть Object в коллекции неизвестного типа. Способ решения этой проблемы является использование "Generic-Метод" Для этого перед методом нужно объявить <T> и использовать его.

* **Решение**

<T> void addAll(T[] a, Collection<T> c) {

for (int i = 0; i < a.length; i++) {

c.add(a[i]);

}

}

Но все равно после выполнение останется ошибка в третьей строчке :

addAll(new Object[10], new ArrayList<String>()); // **Ошибка**

* Решение 4 – **Bounded type argument**

Реализуем метод копирование из одной коллекции в другую

* **Проблема**

<M> void addAll(Collection<M> c, Collection<M> c2) {

for (Iterator<M> i = c.iterator(); i.hasNext(); ) {

M o = i.next();

c2.add(o);

}

}

addAll(new AL<Integer>(), new AL<Integer>());

addAll(new AL<Integer>(), new AL<Object>()); //Ошибка

Проблема в том что две Коллекции могут быть разных типов (несовместимость generic-типов). Для таких случаев было придуман Bounded type argument. Он нужен если метод ,который мы пишем использовал бы определенный тип данных. Для этого нужно ввести <N extends M> (N принимает только значения M). Также можно корректно писать <T extends A & B & C>. (Принимает значения нескольких переменных)

* **Решение**

<M, N extends M> void addAll(Collection<N> c, Collection<M> c2) {

for (Iterator<N> i = c.iterator(); i.hasNext(); ) {

N o = i.next();

c2.add(o);

}

}

* Решение 5 – **Lower bounded wcard**

Реализуем метод нахождение максимума в коллекции.

* **Проблема**

<T extends Comparable<T>>

T max(Collection<T> c) {

…

}

List<Integer> il; Integer I = max(il);

class Test implements Comparable<Object> {…}

List<Test> tl; Test t = max(tl); // Ошибка

* <T extends Comparable<T>> обозначает что Т обязан реализовывать интерфейс Comparable<T>.

Ошибка возникает из за того что Test реализует интерфейс Comparable<Object>. Решение этой проблемы - Lower bounded wcard("Ограничение снизу"). Суть в том что мы будет реализовывать метод не только для Т, но и для его Супер-типов(Родительских типов). Например: Если мы напишем

List<T super Integer> list;

Мы можем заполнить его List<Integer>, List<Number> или List<Object>.

* **Решение**

<T extends Comparable<? super T>>

T max(Collection<T> c) {

…

}

* Решение 6 – **Wildcard Capture**

Реализуем метод Swap в List<?>

* **Проблема**

void swap(List<?> list, int i, int j) {

list.set(i, list.get(j)); // Ошибка

}

Проблема в том, что метод List.set() не может работать с List<?>, так как ему не известно какой он List. Для решение этой проблемы используют "Wildcard Capture" (или "Capture helpers"). Суть заключается в том, чтобы обмануть компилятор. Напишем еще один метод с параметризованной переменной и будем его использовать внутри нашего метода.

* **Решение**

void swap(List<?> list, int i, int j) {

swapImpl(list, i, j);

}

<T> void swapImpl(List<T> list, int i, int j) {

T temp = list.get(i);

list.set(i, list.get(j));

list.set(j, temp);

}

[[править](https://neerc.ifmo.ru/wiki/index.php?title=Generics&action=edit&section=6)]Ограничения Generic

Также нужно запомнить простые правила для работы с Generics.

* Невозможно создать массив параметра типа

Collection<T> c;

T[] ta;

new T[10]; // Ошибка !!

* Невозможно создать массив Generic-классов

new ArrayList<List<Integer>>();

List<?>[] la = new List<?>[10]; // Ошибка !!

[[править](https://neerc.ifmo.ru/wiki/index.php?title=Generics&action=edit&section=7)]Преобразование типов

В Generics также можно манипулировать с информацией, хранящийся в переменных.

* Уничтожение информации о типе

List l = new ArrayList<String>();

* Добавление информации о типе

List<String> l = (List<String>) new ArrayList();

List<String> l1 = new ArrayList();

[[править](https://neerc.ifmo.ru/wiki/index.php?title=Generics&action=edit&section=8)]Примеры кода

* Первый пример:

List<String> ls;

List<Integer> li;

ls.getClass() == li.getClass() // True

ls instanceof List // True

ls instanceof List<String> // Запрещено

* Второй пример:

Нахождение максимума в Коллекции Integer.

* Без Generics:

Collection c;

Iterator i = c.iterator();

Integer max = **(Integer)** i.next();

while(i.hasNext()) {

Integer next = **(Integer)** i.next();

if (next.compareTo(max) > 0) {

max = next;

}

}

* С помощью Generics

Collection**<Integer>** c;

Iterator**<Integer>** i = c.iterator();

Integer max = i.next();

while(i.hasNext()) {

Integer next = i.next();

if (next.compareTo(max) > 0) {

max = next;

}

}