Java Pro 3.1

TEL-RAN
by Starta Institute

Обобщённые типы данных

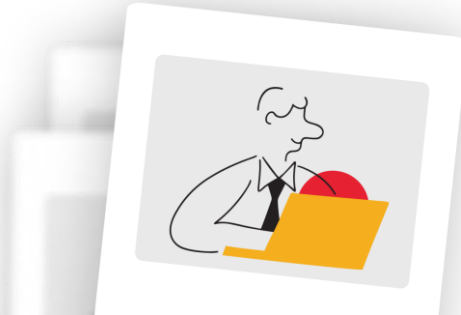
ПРЕПОДАВАТЕЛЬ

TEL-RAN
by Starta Institute

Юрий Костяной

Java/Kotlin backend-разработчик

- 3+ года опыта в коммерческой разработке
- 2+ года опыта в преподавании
- Проекты по интеграции сторонних платформ, CRM
- Проблемно-ориентированный подход в преподавании



ВАЖНО:

- Камера должна быть включена на протяжении всего занятия.
- Если у Вас возник вопрос в процессе занятия, пожалуйста, поднимите руку и дождитесь, пока преподаватель закончит мысль и спросит Вас, также можно задать вопрос в чате или когда преподаватель скажет, что начался блок вопросов.
- Организационные вопросы по обучению решаются с кураторами, а не на тематических занятиях.
- Вести себя уважительно и этично по отношению к остальным участникам занятия.
- Во время занятия будут интерактивные задания, будьте готовы включить камеру или демонстрацию экрана по просьбе преподавателя.

TEL-RAN
by Starta Institute

ПЛАН ЗАНЯТИЯ

1. Основной блок
2. Вопросы по основному блоку
3. Домашняя работа

TEL-RAN
by Starta Institute

TEL-RAN
by Starta Institute

1 ПОВТОРЕНИЕ ИЗУЧЕННОГО

Повторение

TEL-RAN
by Starta Institute

7. Что вы знаете о коллекциях типа Queue?
8. Что вы знаете о коллекциях типа Map, в чем их принципиальное отличие?
9. Назовите основные реализации List, Set, Map.
18. Что будет, если в Map положить два значения с одинаковым ключом?
26. Почему Map не наследуется от Collection?

Ответы https://javastudy.ru/interview/collections/

Повторение

TEL-RAN
by Starta Institute

Что будет выведено в консоль?

A. 95
B. 88
C. Ошибка компиляции
D. Ошибка выполнения

```
import java.util.HashMap;
import java.util.Map;

public class Main {
    public static void main(String[] args) {
        Map<String, Integer> studentGrades = new HashMap<>();

        studentGrades.put("Alice", 95);
        studentGrades.put("Bob", 87);
        studentGrades.put("Charlie", 92);

        studentGrades.put("Alice", 88);

        System.out.println(studentGrades.get("Alice"));
    }
}
```

Повторение

TEL-RAN
by Starta Institute

Что будет выведено в консоль?

A. 95

B. 88

C. Ошибка компиляции

D. Ошибка выполнения

При добавлении значения по ключу, который уже существует в мапе, значение в мапе будет обновлено.

```java
import java.util.HashMap;
import java.util.Map;

public class Main {
    public static void main(String[] args) {
        Map<String, Integer> studentGrades = new HashMap<>();

        studentGrades.put("Alice", 95);
        studentGrades.put("Bob", 87);
        studentGrades.put("Charlie", 92);

        studentGrades.put("Alice", 88);

        System.out.println(studentGrades.get("Alice"));
    }
}
```

Повторение

TEL-RAN
by Starta Institute

Что будет выведено в консоль?

A. 95
B. 88
C. Ошибка компиляции
D. Ошибка выполнения

```
import java.util.HashMap;
import java.util.Map;

public class Main {
    public static void main(String[] args) {
        Map<String, Integer> studentGrades = new HashMap<>();

        studentGrades.put("Alice", 95);
        studentGrades.put("Bob", 87);
        studentGrades.put("Charlie", 92);

        studentGrades.putIfAbsent("Alice", 88);

        System.out.println(studentGrades.get("Alice"));
    }
}
```

Повторение

TEL-RAN
by Starta Institute

Что будет выведено в консоль?

A. 95
B. 88
C. Ошибка компиляции
D. Ошибка выполнения

Метод *putIfAbsent()* добавляет значение только в том случае, если переданный ключ отсутствует в мапе.

```
import java.util.HashMap;
import java.util.Map;

public class Main {
    public static void main(String[] args) {
        Map<String, Integer> studentGrades = new HashMap<>();

        studentGrades.put("Alice", 95);
        studentGrades.put("Bob", 87);
        studentGrades.put("Charlie", 92);

        studentGrades.putIfAbsent("Alice", 88);

        System.out.println(studentGrades.get("Alice"));
    }
}
```

Повторение

TEL-RAN
by Starta Institute

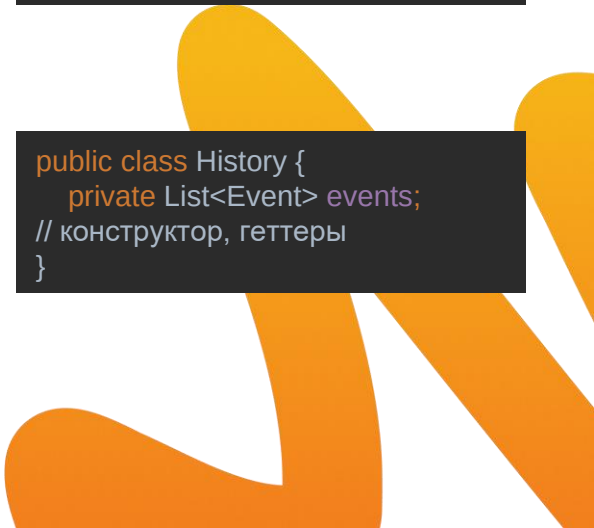
Необходимо каждому автомобилю назначить историю (выпуск заводом, сделки продажи, дорожно-транспортные происшествия и т.д.).

Какую коллекцию необходимо использовать? Что-то потребуется доработать в классах Car и History?

```
public class Car {
    private String brand;
    private String model;
    private int engineNumber;
    private BodyType body;

// конструктор, геттеры
}
```

```
public class History {
    private List<Event> events;
// конструктор, геттеры
}
```



Повторение

TEL-RAN
by Starta Institute

Необходимо каждому автомобилю назначить историю (выпуск заводом, сделки продажи, дорожно-транспортные происшествия и т.д.).

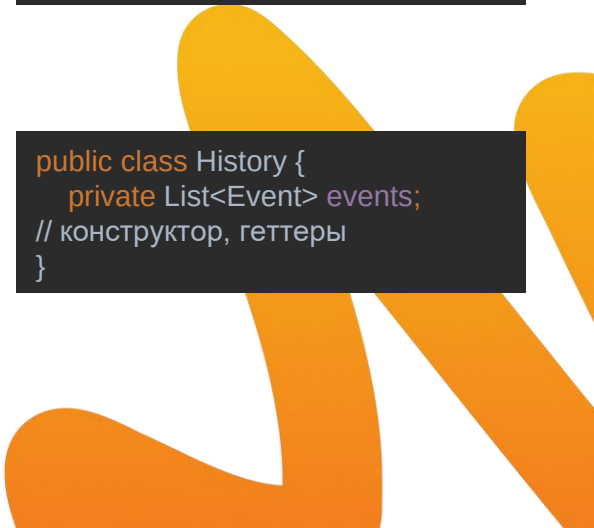
Какую коллекцию необходимо использовать? Что-то потребуется доработать в классах Car и History?

```
public class Car {
    private String brand;
    private String model;
    private int engineNumber;
    private BodyType body;

// конструктор, геттеры
}
```

```
public class History {
    private List<Event> events;
// конструктор, геттеры
}
```

Для назначения соответствия потребуется одна из реализаций Map. HashMap требует, чтоб класс Car и, желательно, класс History переопределили методы equals() и HashCode(). Если же выбрать реализацию TreeMap, то нужно имплементировать интерфейс Comparable в классе Car, либо создать компаратор, который нужно передать в конструктор TreeMap.



Повторение

Отгадайте скрытые названия коллекций

Повторение

Отгадайте скрытые названия коллекций

TEL-RAN
by Starta Institute

2 ОСНОВНОЙ БЛОК

Введение

TEL-RAN
by Starta Institute

- Ничего общего
- В границах дозволенного
- Брекс-пекс
- Сыроват

Проблема

Изучая коллекции, мы видели, что коллекции могут хранить любой тип данных.

Достаточно указать его в скобках <>.

Предположим, что нам нужно написать универсальный класс-обёртку, экземпляр которого мог бы содержать в себе один объект любого типа.

Прибегая к принципу полиморфизма, мы можем создать в таком классе поле с типом *Object* и хранить в таком поле всё, что угодно.

GenericExamples.zip

Проблема

TEL-RAN
by Starta Institute

Как только мы положили в один экземпляр такой обёртки *String*, а в другой, например, *Integer*, компилятор «забывает» изначальный тип содержимого – для него теперь это просто *Object*. Это приводит к тому, что при доставании объекта из обёртки нужно писать много кода, проверяющего тип содержимого обёртки. При том, что код всех проверок очень похож.

Вот бы такой механизм, который позволил написать одинаковую логику для разных типов данных и без лишних проверок.

Ничего общего

Делегируем компилятору

TEL-RAN
by Starta Institute

Для того, чтобы упростить код, убрать лишние проверки и перенести исключения, связанные с типами данных, из *runtime* на этап компиляции, придумали механизм **обобщённых типов** (*generics*).

Теперь тип данных может быть переменным (**параметризированным**).

При создании параметризированного объекта (например, экземпляра класса-обёртки) мы явно указываем компилятору тип содержимого, после чего контроль за типами становится его задачей, а не задачей разработчика.

Ничего общего

Обобщённые типы

TEL-RAN
by Starta Institute

Generics – это обобщённые типы данных, которые могут хранить любой тип данных внутри или работать с разными типами данных, при этом на стадии компиляции *Generics* имеют параметр, который указывает, какой тип хранится внутри обобщённого. Это позволяет компилятору контролировать преобразование типов.

Ничего общего

Применение generics

TEL-RAN
by Starta Institute

Где мы уже встречали:

1 Интерфейсы коллекций – Collection<T> и Map<K, V>, их наследники, а также многие их реализации.

2 Вспомогательные интерейсы Iterator<T>, Comparable<T>, Comparator<T>.

Где ещё встретятся:

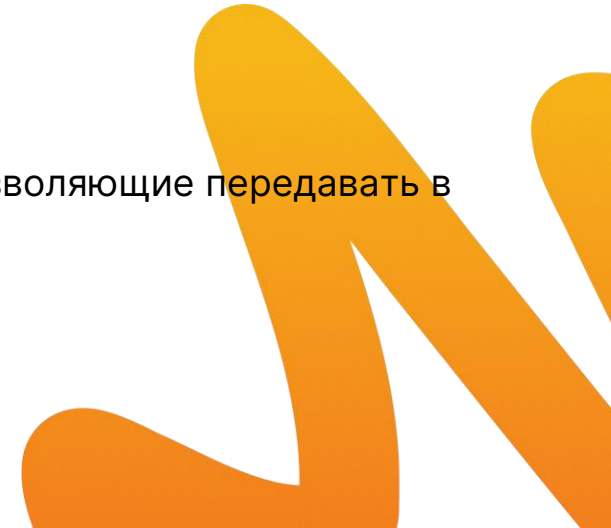
3 Optional<T> - null-safe обёртка над объектом

4 Stream<T> - конвейерная обработка коллекций

5 Функциональные интерфейсы – специальные интерфейсы, позволяющие передавать в методы не объекты, а ссылки на другие методы

6 В многопоточности – интерфейс Callable<T>

7 Многие утилитарные методы в реальных проектах.



Ничего общего

Применение generics

TEL-RAN
by Starta Institute

Пример кода из реального проекта

```
/**
 * Метод для выполнения вставки объекта, если он не {@code null}
 *
 * @param value  объект для вставки
 * @param setter операция проставления
 * @param <T>    тип объекта
 */
public static <T> void setIfNotNull(T value, Consumer<T> setter) {
    if (value != null) { setter.accept(value); }
}

/**
 * Выполнение операции над коллекцией, если она не пустая
 *
 * @param value  коллекция над которой необходимо выполнить операцию
 * @param getter ф-ия которую необходимо выполнить
 * @param <C>    тип наследника коллекции
 * @param <R>    тип результата операции
 * @return результат операции или {@code null}
 */
public static <T, C extends Collection<T>, R> R getIfNotEmpty(C value, Function<C, R> getter) {
    return isEmpty(value) ? null : getter.apply(value);
}
```

Ничего общего

Виды обобщённых типов

TEL-RAN
by Starta Institute

1 На уровне классов

2 На уровне метода

3 На уровне интерфейса

Ничего общего

Виды обобщённых типов

TEL-RAN
by Starta Institute

1 На уровне классов

2 На уровне метода

3 На уровне интерфейса

```
public static class GenericWrapper<T> {
    private final T value;

    public GenericWrapper(T value) {
        this.value = value;
    }

    public T getValue() {
        return value;
    }
}
```

Разница между обычными и общими классами заключается только в параметрах типов в определениях классов. Любой класс (конкретный , абстрактный или final) может быть параметризован с использованием обобщений.

Ничего общего

Виды обобщённых типов

TEL-RAN by Starta Institute

1 На уровне классов

2 На уровне метода

3 На уровне интерфейса

```java
/**
 * Метод, который удаляет из мапы элементы с null-значениями
 * @param map целевая мапа для удаления null-элементов
 * @param <K> тип ключей мапы
 * @param <V> тип значений мапы
 */
public <K, V> void removeNullValues(Map<K, V> map) {
    Iterator<Map.Entry<K, V>> iterator = map.entrySet().iterator();
    while (iterator.hasNext()) {
        if (iterator.next() == null) {
            iterator.remove();
        }
    }
}
```

Методы могут параметризировать передаваемые в них аргументы и возвращаемое значение. Метод может вводить свои обобщения и использовать при необходимости обобщения своего класса или интерфейса. Все методы (абстрактные, статические, конкретные, final, конструкторы) могут использовать обобщения.

Ничего общего

Виды обобщённых типов

TEL-RAN
by Starta Institute

1 На уровне классов

2 На уровне метода

```
public interface GenericInterface<T, R> {
    void performAction(final T action);

    R performAction( final T action, T secondAction);
}
```

3 На уровне интерфейса

Всякий раз, когда какой-либо класс хочет реализовать интерфейс, он имеет возможность указать точные типы вместо обобщённых, например, String вместо T. При этом R можно оставить обобщённым. Можно заменить сразу оба параметра.

Ничего общего

Минусы обобщённых типов

TEL-RAN by Starta Institute

1 Примитивные типы не допускаются к использованию в обобщениях – нужно использовать соответствующий класс-обёртку.

2 Стирание типов. Обобщённые типы существуют только во время компиляции: полученный байт-код JVM удаляет все конкретные типы (и заменяется на класс *Object*). Следующий код вызовет ошибку:

```
void sort(Collection<String> strings) {
    // Some implementation over strings heres
}

void sort(Collection<Number> numbers) {
    // Some implementation over numbers here
}
```

Ничего общего

Минусы обобщённых типов

TEL-RAN by Starta Institute

3 Невозможно получить конкретный тип параметра даже на этапе выполнения. Код ниже не скомпилируется.

```
public <T> void action(final T action) {
    if (action instanceof T) {
        // Do something here
    }
}

public <T> void action(final T action) {
    if (T.class.isAssignableFrom(Number.class)) {
        // Do something here
    }
}
```

4 Невозможно создать экземпляры массива, используя параметры типа generics. Код справа не скомпилируется:

```
public <T> void performAction(final T action) {
    T[] actions = new T[0];
}
```

Кто крайний?

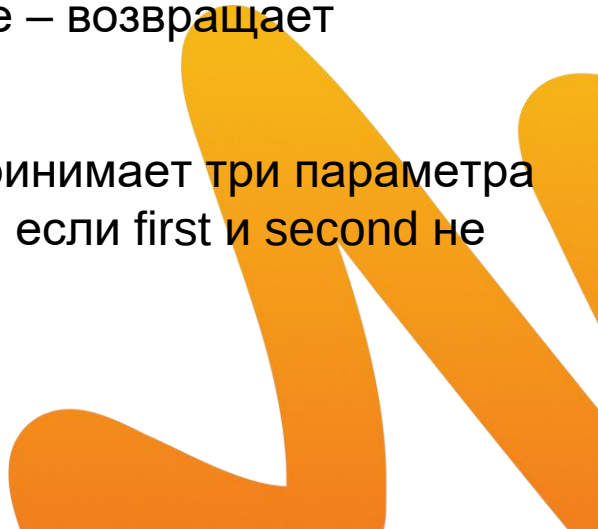
Задание

TEL-RAN
by Starta Institute

1 Создайте параметризированный класс-обёртку Pair с двумя полями разных типов - first и second.

2 Создайте параметризированный метод, который принимает два параметра одного типа – значение и значение по умолчанию. Метод возвращает значение, если оно не равно null. В противном случае – возвращает значение по умолчанию.

3 Создайте параметризированный метод, который принимает три параметра разных типов – first, second и third и возвращает third, если first и second не равны null.



Проблема

TEL-RAN
by Starta Institute

Если класс Employee – это наследник класса Person, то List<Employee> - это то же, что List<Person>?

Можно ли применять принцип полиморфизма к параметризированному типу?

В границах дозволенного
Generics и полиморфизм

TEL-RAN
by Starta Institute

Мы можем использовать любой подкласс в методе, который работает с предком.

Например, *Number* – предок *Integer* и *Double* (**ковариантные** типы):

```
public void someMethod(Number n) { /* ... */ }

someMethod(new Integer(10));   // OK
someMethod(new Double(10.1));   // OK
```

Обобщённые типы позволяют использовать такой же тип полиморфизма. Например:

```
Box<Number> box = new Box<>();
box.add(new Integer(10));   // OK
box.add(new Double(10.1));  // OK
```

Но если взглянуть на метод:

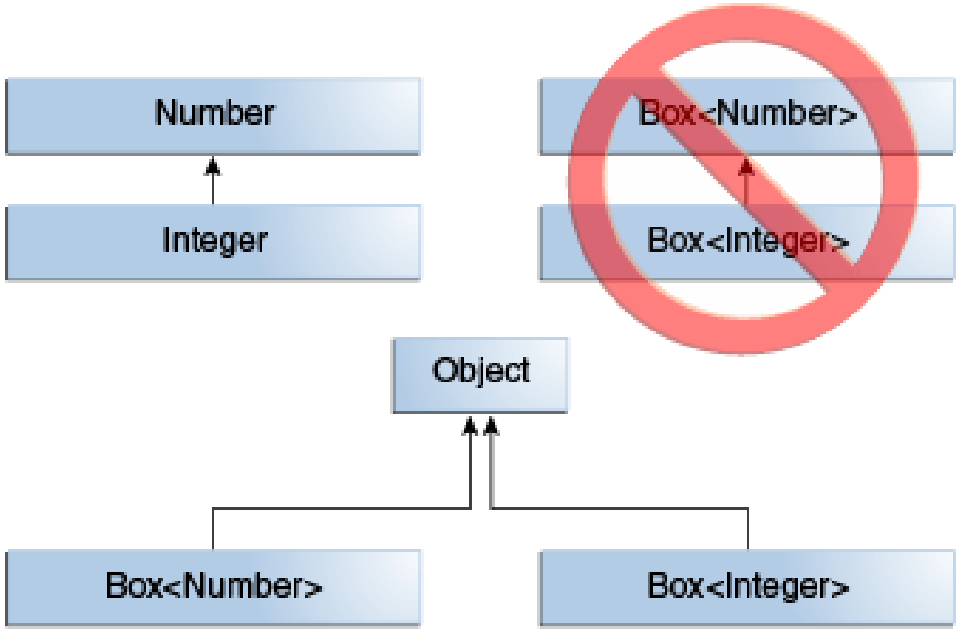
```
public void boxTest(Box<Number> n) { /* ... */ }
```

то передать в него *Box<Integer>* или *Box<Double>* не получится, т.к. каждый *Box<>* с конкретным типом содержимого по своей сути является новым подтипом *Box<>* и не связан иерархией своего содержимого (**инвариантен**).

В границах дозволенного

Ковариантность и инвариантность

Пояснение на схеме:

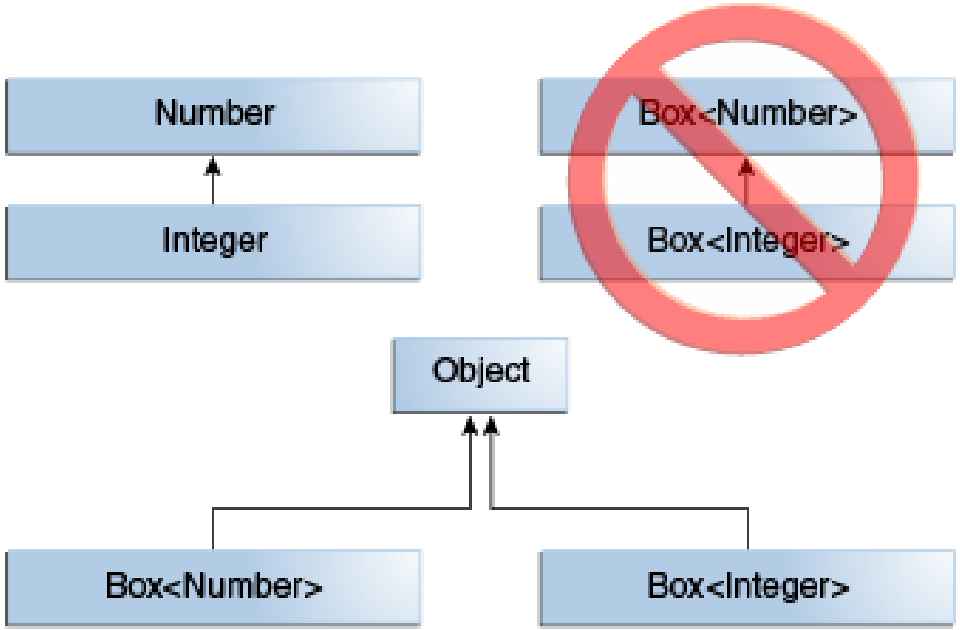


В границах дозволенного

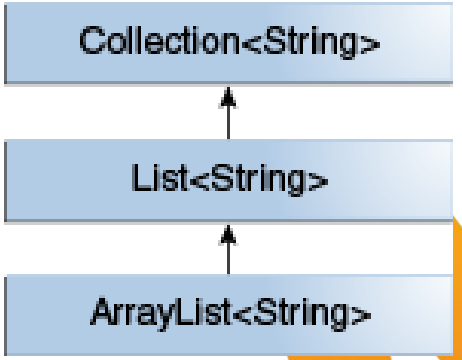
Ковариантность и инвариантность

TEL-RAN by Starta Institute

Пояснение на схеме:



Однако нам уже известно, что коллекции – яркий пример дженериков – можно преобразовывать друг в друга:



Для того, чтобы увязать типизированные классы как подтипы используют специальные ограничения.

В границах дозволенного

Ограничения обобщённых типов

TEL-RAN by Starta Institute

Обобщённые типы позволяют задавать границы типов, которые могут подставляться для параметризации.

Ключевое слово *extends* ограничивает параметр типа наследниками какого-либо класса или реализациями одного или нескольких интерфейсов. Например:

```
public <T extends InputStream> void read(final T stream) {
    // Some implementation here
}
```

Параметр типа *T* в объявлении метода *read* должен быть наследником класса *InputStream*. Аналогично с интерфейсами.

В границах дозволенного

Ограничения обобщённых типов

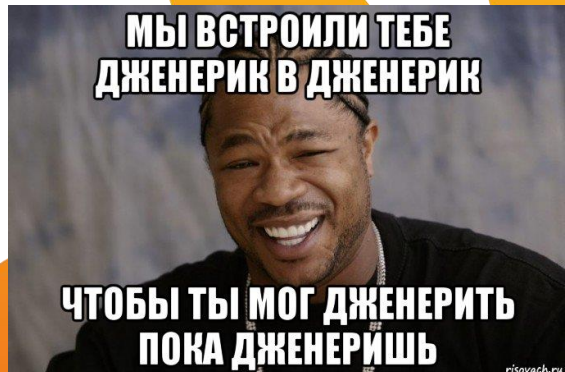
TEL-RAN by Starta Institute

Границы могут быть объединены с помощью оператора *&* . Может быть указано несколько интерфейсов, но разрешен только один класс.

```
public <T extends InputStream & Serializable> void storeToRead(final T stream) {
    // Some implementation here
}
public <T extends Serializable & Externalizable & Cloneable> void persist(
        final T object) {
    // Some implementation here
}
```

Можно использовать другой параметр типа в качестве привязки для ключевого слова *extends:*

```
public <T, J extends T> void action(final T initial, final J next) {
    // Some implementation here
}
```



В границах дозволенного

Стирание обобщённых типов с границами

TEL-RAN
by Starta Institute

Если у типа задана верхняя граница (предок), то при стирании типов вместо параметра будет указан тип верхней границы, а не Object. Если верхняя граница сама является типизированной, то её параметр будет заменён по тому же принципу.

```
public class Test<T extends Comparable<T>> {

    private T data;
    private Test<T> next;

    public Test(T d, Test<T> n) {
        this.data = d;
        this.next = n;
    }

    public T getData() { return this.data; }
}
```

The Java compiler replaces the bounded type parameter T with the first bound interface, Comparable

```
public class Test {

    private Comparable data;
    private Test next;

    public Node(Comparable d, Test n) {
        this.data = d;
        this.next = n;
    }

    public Comparable getData() { return data; }
}
```

В границах дозволенного

Ограничения по Wildcard

TEL-RAN by Starta Institute

Если параметр типа не представляет интереса в классе, интерфейсе или методе, то он может быть заменен символом *?*

```
public void store( final Collection< ? extends Serializable> objects ) {
    // Some implementation here
}
```

Т.е. нужно, чтобы используемый в коллекции тип данных реализовывал интерфейс *Serializable* . Подстановочный знак можно использовать без границ:

```
public void store(final Collection< ? > objects ) {
    // Some implementation here
}
```

Символ *?* называют **wildcard** по аналогии с картой джокера в покере – так называют карту, которая может сыграть вместо любой другой.

В границах дозволенного

Ограничения по Wildcard

TEL-RAN
by Starta Institute

Подстановочные знаки нельзя писать везде вместо параметров. Чаще всего они используются совместно для ограничения параметров:

```
List<? extends Garbage> example2 = new ArrayList<Paper>();
```

Для ограничения обобщённого типа снизу используется ключевое слово *super*.

```
List<? super Garbage> example3 = new ArrayList<Garbage>();
```

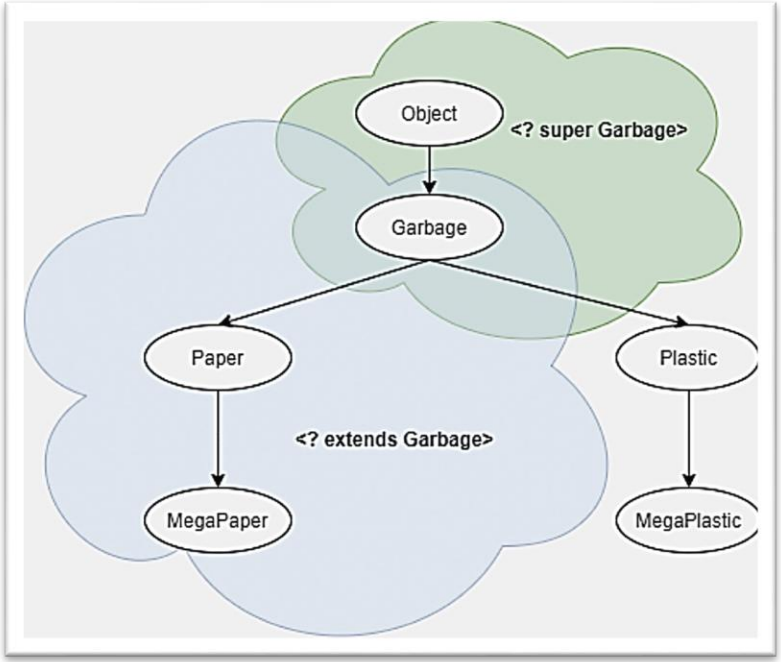
Здесь *<? super Garbage>* означает, что вместо «*?*» можно подставить *Garbage* или любой его класс-предок (не только непосредственного предка, а любого!). Все ссылочные классы неявно наследуют класс Object, так что в правой части ещё может быть *ArrayList<Object>*.

В границах дозволенного

Ограничения по Wildcard

TEL-RAN
by Starta Institute

Используя верхние и нижние границы типов (с *extends* и *super*) вместе с подстановочными знаками типов, обобщенные элементы обеспечивают способ точной настройки требований к параметрам типа.

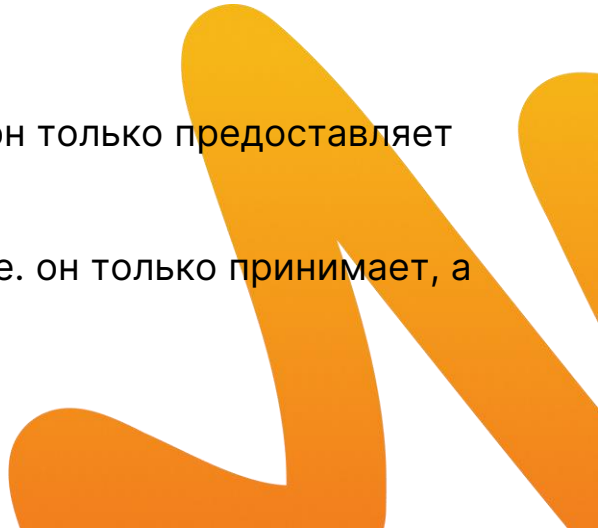


Брекс-пекс
PECS

TEL-RAN
by Starta Institute

Особенность *wildcard* с верхней и нижней границей дает дополнительные фичи, связанные с безопасным использованием типов. Из одного типа переменных можно только *читать*, в другой – только *вписывать* (исключением является возможность записать *null* для *extends* и прочитать *Object* для *super*). Чтобы было легче запомнить, когда какой *wildcard* использовать, существует принцип **PECS** – *Producer Extends Consumer Super*.

- Если мы объявили *wildcard* с *extends*, то это *producer*, т.е. он только предоставляет элемент из контейнера, а сам ничего не принимает.
- Если же мы объявили *wildcard* с *super* – то это *consumer*, т.е. он только принимает, а предоставить ничего не может.



Брекс-пекс
PECS

TEL-RAN
by Starta Institute

Рассмотрим использование Wildcard и принципа PECS на примере метода *copy()* в классе *java.util.Collections*.

```
public static <T > void copy (List < ? super T > dest, List < ? extends T > src) {
        …
    }
```

Метод осуществляет копирование элементов из исходного списка *src* в список *dest. src* — объявлен с wildcard *? extends* и является продюсером, а *dest* — объявлен с wildcard *? super* и является потребителем. Учитывая ковариантность и контравариантность wildcard, можно скопировать элементы из списка ints в список nums:

```
List<Number> nums = Arrays.asList(4.1F, 0.2F);
List<Integer> ints = Arrays.asList(1,2);
Collections.copy(nums, ints);
```

Если же мы по ошибке перепутаем параметры метода *copy()* и попытаемся выполнить копирование из списка nums в список ints, то компилятор не позволит нам это сделать.

Задание

TEL-RAN
by Starta Institute

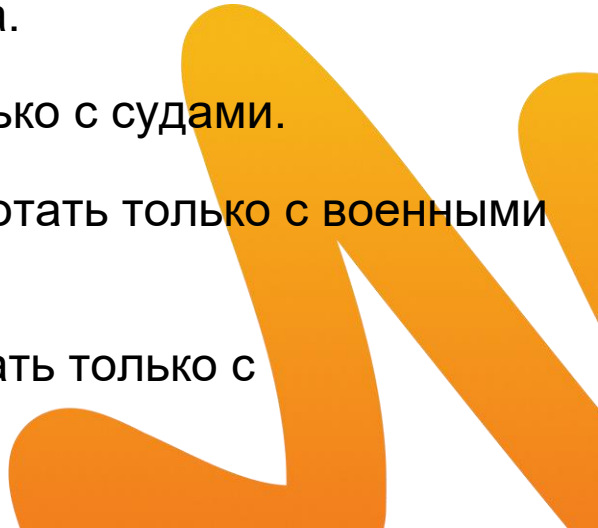
1.1 Создайте иерархию классов: транспорт -> судно -> пассажирский лайнер/прогулочная яхта/военный корабль. Пассажирский лайнер и прогулочная яхта должны имплементировать интерфейс Entertainment, содержащий метод entertain.

1.2 Создайте класс-обёртку для хранения транспорта.

1.3 Создайте метод sail, который будет работать только с судами.

1.4 Создайте метод useTransport, который будет работать только с военными кораблями и их любым предком.

1.5 Создайте метод getVoyage, который будет работать только с пассажирскими лайнерами и яхтами.



Сыроват

Generics без <>

TEL-RAN
by Starta Institute

Можно ли не указывать тип параметра generic?

Да, можно. В таком случае это будет тип, параметризированный Object. Например, для списка:

```
List list = new ArrayList();
```

В таком случае IDE и анализаторы кода будут «ругаться» на использование сырого типа (**raw type**). Т.е. программист использовал параметризированный тип, но вместо того, чтобы указать тип параметра использовал Object, т.е. проигнорировал возможность применения преимуществ generic’ов.

Это плохая практика, старайтесь всегда параметризировать используемые generic.

Мне только спросить

Задание

TEL-RAN
by Starta Institute

Создайте список строк (ArrayList) без указания параметра.

Попробуйте взять из него элемент с индексом 0 и присвоить его переменной типа String. Выведите переменную в консоль. Запустите программу.

Сделайте элемент с индексом 0 в списке числом. Запустите программу. Объясните результат.

TEL-RAN
by Starta Institute

4 Домашнее задание

Домашнее задание

TEL-RAN
by Starta Institute

1 Напишите параметризированный метод для обмена позициями двух разных элементов в массиве. Метод принимает параметризированный массив и индексы элементов, которые нужно обменять.

2 Напишите параметризированный метод для нахождения максимального элемента в диапазоне [begin, end) среди элементов List. Какую границу должен иметь параметр такого метода?

3 Создайте иерархию учебных предметов (Subject -> Natural sciences/exact sciences/humanities -> Biology, Chemistry / Physics, Mathematics / Literature, Philology). Напишите класс Student, параметризируемый предметом. Создайте классы студентов по каждому из направлений (Natural sciences/exact sciences/humanities), укажите в наследуемом параметре конкретный тип направления.

ЗАКЛЮЧЕНИЕ

TEL-RAN
by Starta Institute