# Тема

«Родовые типы Java (Generics)»

# План занятия

## Введение

Анализ кода коллекций

List list = new ArrayList();

list.add(5);

list.add(“some text”);

Данный участок кода не безопасен и приводит к ClassCastException при попытке считать значение с неожидаемым типом данных.

Выход – зафиксировать типы, с которыми может работать конкретный экземпляр коллекции.

Параметр типа – абстракция на уровне типов.

Практический пример:

1. Реализовать тип Box для хранения Integer.
2. Реализовать тип Box для хранения String.
3. Выделить отличия в коде и вывести их в отдельный параметр

## Generics не коварианты

Пример ковариантности массивов:

Number -> Integer

Number[] -> Integer[]

Но List<Number> !-> List<Integer>

Если бы родовые типы были ковариантны, то возможно было бы следующее:

List< Integer > list = new ArrayList< Integer >;

List<Number> numList = list;

numList.add(3.5f);

В лист целых чисел добавлено дробное значение.

## WildCards

Из-за нековариантности невозможно работать с методами, которые бы принимали на вход любые коллекции:

void print(Collection<Object> o){

…}

Данный метод будет принимать только Collection<Object>. Для решения таких проблем введен групповой символ “?”. Его используют, когда тип параметризации не важен. При этом нет никакой информации о параметризованном типе, для каждого аргумента генерируется свой тип-заглушка.

## Границы wildcards

PECS (provider - extends, consumer - super)

Пример иерархии классов:

class Animal { void feed() {} }

class Pet extends Animal { void call() {} }

class Cat extends Pet{ void mew() {} }

class Dog extends Pet{ void bark() {} }

Необходимо реализовать метод по копированию коллекций:

public static <T> void copy(List<? super T> dest, List<? extends T> src) (стандартный метод в Collections)

Источник должен быть ограничен сверху – это гарантирует приводимость к T

Копирование коллекций

List<Cat> src = getCats();

List<Pet> dest = new ArrayList<Pet>();

Collections.copy(dest, src);

for(Pet p: dest) p.call();

Пример корректен. Следующий пример покажет некорректность кода-

List<Cat> src = getCats();

List<Dog> dest = new ArrayList<Dog>();

Collections.copy(dest, src);

for(Dog dog: dest) dog.bark();

Таким образом мы должны установить нижнюю границу для коллекции-назначения, ниже которой опускаться в предположении о типах нельзя.

## Механизм затирания

Вся информация о параметризованных типах доступна только на уровне компиляции, в runtime все приводится к обобщенному типу – параметризованному Object. Следствие – невозможно создать экземпляр параметризованного типа.

# Материалы для подготовки:

Спецификация языка Java

http://www.ibm.com/developerworks/ru/java/library/j-jtp01255/index.html

https://docs.oracle.com/javase/tutorial/extra/generics/