Обобщенное программирование

1. Обобщенное программирование означает написание кода, который может быть неоднократно использован с объектами самых разных типов.
2. Обобщения предлагают лучшее решение: параметры типа. Класс ArrayList теперь принимает параметр типа, обозначающий тип элементов коллеции, как показано ниже. Благодаря этому код получается более удобочитаемым. Теперь становится сразу понятно, что этот конкретный списочный массив содержит объекты типа String.

ArrayList<String> files = new ArrayList<String> () ;

также можно писать так,

ArrayList<String> files = new ArrayList<> () ;

1. Приведем пример типичного затруднения, которое пришлось преодолевать разработчикам стандартной библиотеки классов Java. Класс ArrayList содержит метод addAll (), предназначенный для добавления элементов из другой коллекции. Так, у программиста может возникнуть потребность добавить все элементы из коллекции типа ArrayList в коллекцию типа ArrayList. Но обратное, разумеется, недопустимо. Как же разрешить один вызов и запретить другой? Создатели Java нашли искусный выход из этого затруднительного положения, внедрив понятие подстановочного типа. Подстановочные типы довольно абстрактны, но они позволяют разработчику библиотеки сделать методы как можно более гибкими.

В связи с этим можно сформулировать следующее эмпирическое правило: от применения параметров типа выигрывает только тот код, в котором традиционно присутствует много операций приведения от самых общих типов, как, например, класс Object или интерфейс Comparablе.

4)Какой-то пиздец со стиранием типов

5) Какой-то пиздец со стиранием типов

6) Какой-то пиздец со стиранием типов

7) Примитивный тип нельзя подставить вместо типа параметра. Это означает, что не бывает объекта типа Pair<double>, а только объект типа Pair<Double>. Причину, конечно, следует искать в стирании типов. После такого стирания в классе Pair отсутствуют поля типа Object, и поэтому их нельзя использовать для хранения значений типа double.

8)Переменные типа нельзя использовать в выражениях вроде new Т (...), new Т [... ] или Т.class. Например, следующий конструктор Pair<T> недопустим:

public Pair() { first = new Т(); second = new T(); } // ОШИБКА!

Стирание типов может изменить обобщенный тип Т на Object, а вызывать конструктор new Object (), конечно, не стоит. Начиная с версии Java SE 8, можно прибегнуть к наилучшему обходному приему, предоставив в вызывающем коде ссылку на конструктор, как показано в следующем примере:

Pair< String> р = Pair.makePair(String: :new) ;

Метод makePair () получает ссылку на функциональный интерфейс для вызова функции без аргументов и возврата результата типа Т следующим образом:

public static <T> Pair<T> makePair(Supplier constr) { return new Pair<>(constr.get(), constr.get()); }

9) На переменные типа нельзя ссылаться в статических полях или методах. Например, следующая замечательная идея на сработает:

public class Singleton<T> {

private static T singlelnstance; // ОШИБКА!

public static T getSinglelnstance() // ОШИБКА!

10) Генерировать или перехватывать объекты обобщенного класса в виде исключений не допускается. На самом деле обобщенный класс не может расширять класс Throwable. Например, приведенное ниже определение обобщенного класса не будет скомпилировано.

public class Problem<T> extends Throwable {/\*...\*/}// ОШИБКА!

// Класс Throwable расширить нельзя!

Кроме того, переменную типа нельзя использовать в блоке catch. Например, следующий метод не будет скомпилирован:

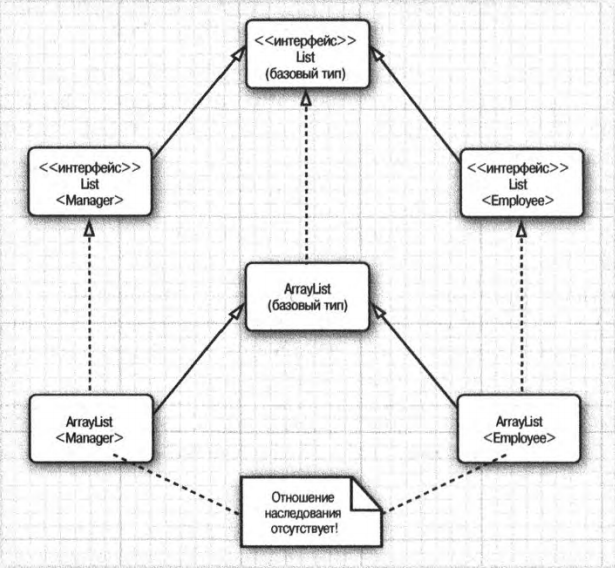
public static <T extends Throwable> void doWork(Class t) {

try

{ выполнить нужные действия }

catch (Т е) // ОШИБКА! Перехватывать переменную типа нельзя

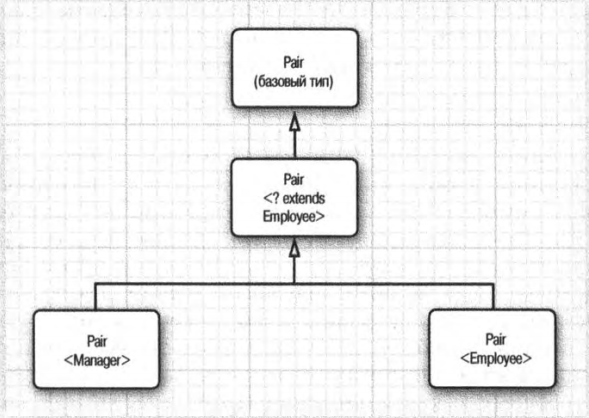
{ Logger.global.info (...) }

11)

12) Как было показано в предыдущем разделе, передать объект типа Pair<Manager> этому методу нельзя, что не совсем удобно. Но из этого положения имеется простой выход — использовать подстановочный тип следующим образом:

public static void printBuddies(Pair<? Extends Employee> p)

Тип Pair<Manager> является подтипом Pair<? Extends Employee>



13)

