Павел Степанов

Кафедра компьютерной математики и программирования СПб ГУАП

Язык программирования Java

Тема 6 Коллекции

- generics
- Collections framework
- Проект Lambda

6.1 Параметризованный класс

- Позволяют типу быть параметром класса или метода
  - Не нужно приводить типы
- Похожи на параметризованные классы из C++, но только похожи
- Соответствие типов параметров проверяется только на этапе компиляции
  - В частности, вместо <T> подставляется Object
- Основное применение –коллекции

6.2 Пример параметризованного класса

```
package sample_6_2;

/**
 *
 * @author Kaoru
 */
public class ParamClass<K, V> {

    V getK(K k) {
        return null;
    }

    public static void main(String ... args){
        Integer i= new ParamClass<String, Integer>().getK((String)null);
    }
}
```

6.3 Инстанциация

- Обычная
  - List<Integer> l = new LinkedList<Integer>();
- Diamond оператор <>
  - List<Integer> l = new LinkedList<>();
  - JDK 7++
- Ray type
  - List l = new LinkedList();
  - Для совместимости со старым кодом
  - Результат – warning
    - @SuppressWarnings("unchecked")
- Нельзя
  - List<Ineger> l = new LinkedList();

6.4 Как реализовано

```
public static void main(java.lang.String...);
  descriptor: ([Ljava/lang/String;)V
  flags: ACC_PUBLIC, ACC_STATIC, ACC_VARARGS
  Code:
    stack=2, locals=2, args_size=1
       0: new           #2                  // class sample_6_2/ParamClass
       3: dup
       4: invokespecial #3                  // Method "<init>":()V
       7: aconst_null
       8: checkcast     #4                  // class java/lang/String
      11: invokevirtual #5                  // Method getK:(Ljava/lang/Object;)Ljava/lang/Object;
      14: checkcast     #6                  // class java/lang/Integer
      17: astore_1
      18: return
    LineNumberTable:
      line 19: 0
      line 20: 18
    LocalVariableTable:
      Start  Length  Slot  Name   Signature
          0      19     0  args   [Ljava/lang/String;
         18       1     1     i   Ljava/lang/Integer;
```

6.5 Соглашения по именам

- E - Element (элемент коллекции)
- K – Key (ключ коллекции)
- N – Number (число)
- T – Type (тип)
- V – Value (значение)
- S,U,V – 2й, 3й и 4й типы

6.6 То же для методов

- Тип параметра определяется тем, что подано на вход и выход при вызове метода

6.7 Пример метода

```
public class Sample_6_6 {

    public static <K,V> V getV(K k){
        return null;
    }

    public static void main(String[] args) {
        String s = getV(5);
    }

}
```

6.8 Как реализовано

```
public static void main(java.lang.String[]);
  descriptor: ([Ljava/lang/String;)V
  flags: ACC_PUBLIC, ACC_STATIC
  Code:
    stack=1, locals=2, args_size=1
       0: iconst_5
       1: invokestatic  #2                  // Method java/lang/Integer.valueOf:(I)Ljava/lang/Integer;
       4: invokestatic  #3                  // Method getV:(Ljava/lang/Object;)Ljava/lang/Object;
       7: checkcast     #4                  // class java/lang/String
      10: astore_1
      11: return
    LineNumberTable:
      line 20: 0
      line 21: 11
    LocalVariableTable:
      Start  Length  Slot  Name   Signature
          0      12     0  args   [Ljava/lang/String;
         11       1     1     s   Ljava/lang/String;
```

6.9 extends

```
public class Sample_6_9 {

    static interface I1{};
    static interface I2{};

    static class A implements I1, I2{};
    static class B<U extends I1 & I2>{};


    public static void main(String[] args) {
        B<A> b = new B<>();
    }

}
```

6.10 Как реализовано

- Реально компилируется тип первого из интерфейсов
  - То есть <U extends A & B> при компиляции заменяется на A

6.11 Наследование и совместимость

- Можно сделать так:
  - Number n = new Integer(10);
  - List list = new LinkedList();
  - List<Integer> list = new LinkedList<Integer>();
- Нельзя сделать так:
  - List<Number> list = new LinkedList<Integer>();
    - Все дело в том, что мы не можем в будущем гарантировать, что обращения к list будут содержать в себе именно Integer, а не какой-нибудь другой супертип Number. Поэтому компилятор такое запрещает
    - Вообще логика разработчиков довольно спорна, но как сделали – так и есть

6.12 wildcard

- Используются при передаче параметров, если параметр – дженерик
- Могут ограничивать параметр-класс по супертипу и по субтипу

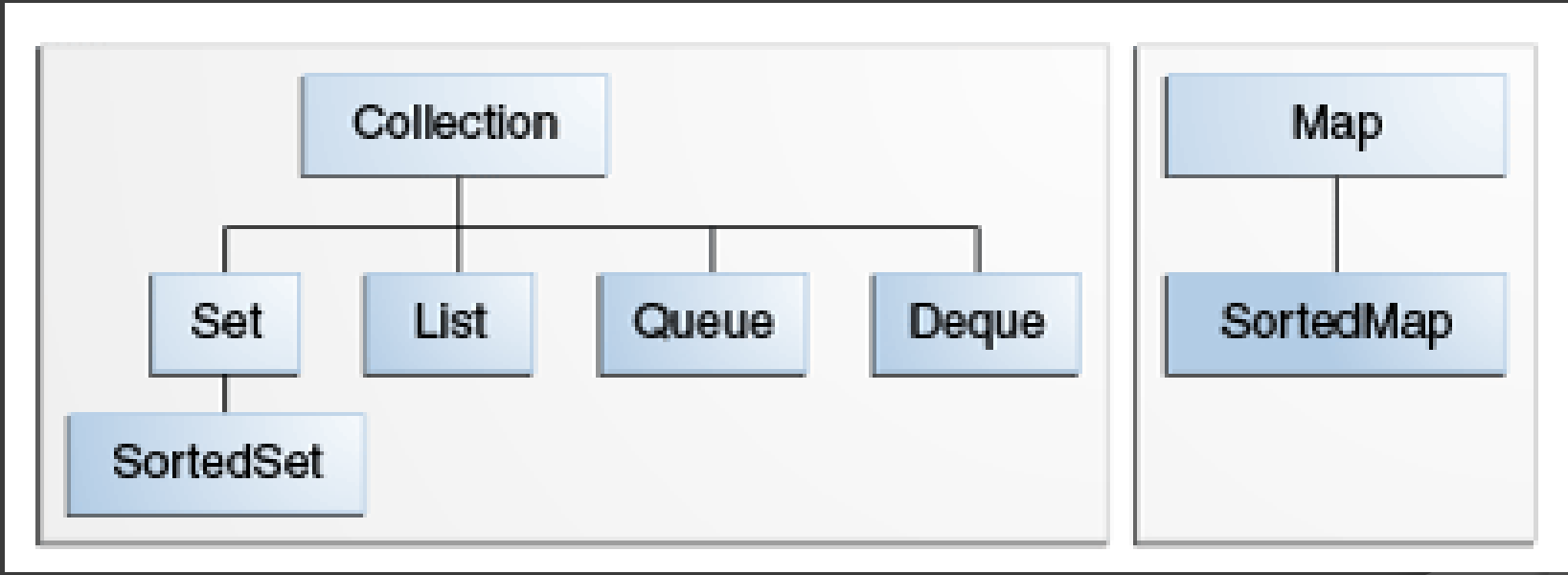
6.13 Пример

```
public void method1 (java.util.List<? extends Number> list){}
public void method2 (java.util.List<? super String> list){}
public void method3 (java.util.List<?> list){}
```

6.14 Ограничения

- Нельзя параметризовывать примитивными типами
- Нельзя создавать экземпляры типов параметров
- Нельзя объявлять статические поля с параметризованными типами
- Нельзя использовать instance of с параметризованными классами
- Нельзя создавать массивы и исключения параметризованных типов
- Нельзя перегружать методы по параметризованным типам

6.15 Коллекции



6.16 состав фреймворка

- Интерфейсы
  - Collection
  - List
  - Set
  - Map
- Классы
- Алгоритмы

6.17 java.util.Collection

- Интерфейс Iterable
  - Iterator iterator()
- Методы add, addAll
- Методы contains, containsAll
- Методы remove, removeAll
- Метод retain
- Методы size(), isEmpty(), clear(), toArray()

6.18 Интерфейс Iterator

- Метод hasNext()
- Метод next()
- Метод remove()
- Spliterator (JDK 8+)

6.19 Интерфейс Set

- Множество
- Не может содержать null
- Основные реализации
  - TreeSet
  - HashSet
  - LinkedHashSet
  - Специальные
    - EnumSet
    - CopyOnWriteArraySet
- Не добавляет новых методов по сравнению с Collection

6.19 Интерфейс List

- Список
- Может содержать null
- Основные реализации
  - ArrayList
  - LinkedList
  - Специальные
    - CopyOnWriteArrayList

6.20 Интерфейс Deque

- Стек и очередь одновременно
- Может содержать null
- Основные реализации
  - LinkedList
  - ArrayDeque

6.21 Интерфейс Map

- Отображение ключей в значения
- Основные реализации
  - HashMap
  - TreeMap
  - LinkedHashMap
  - Специальные
    - WeakHashMap
    - EnumMap
    - IdentityHashMap
    - ConcurrentMap

6.22 Collections

- Метод sort
  - Comparable
  - Comparator
- Метод binarySearch
- Метод shuffle
- Метод reverse
- Метод fill
- Метод swap
- Метод addAll
- Метод disjoint
- Метод frequency

6.23 Arrays

- Метод binarySearch
- Метод copyOf
- Метод copyOfRange
- Метод equals
- Метод fill
- Метод deepHashCode

6.24 Лямбды (JDK 8+)

- Фактически это ссылка на анонимную функцию
- Основное применение – функциональные параллельные операции над коллекциями
  - Вспоминаем mapcar
- При реализации широко используется JSR292 Dynamic Invocation
- Являются логической переработкой анонимных внутренних классов

6.25 Функциональный интерфейс

- Интерфейс, в котором есть только один абстрактный метод с любым именем
- Лямбда-выражение имеет тип этого интерфейса

6.26 Пример

```
public class Sample_6_26 {

    interface UnaryOperation {

        int methodWithAnyName(int a);
    }

    interface BinaryOperation {

        int methodWithAnyName(int a, int b);
    }

    static int myMethod(int x, int y){
        return x=y;
    };


    public static void main(String[] args) {
        UnaryOperation invert = (arg) -> 0 - arg;
        UnaryOperation sign = (arg) -> (arg > 0) ? 1 : -1;
        BinaryOperation add = (arg1, arg2) -> arg1 + arg2;
        BinaryOperation multiply = (arg1, arg2) -> arg1 * arg2;
        BinaryOperation custom = Sample_6_26::myMethod;
    }

}
```

6.27 Лямбда выражение

- Состоит из трех частей
  - Аргументы в скобках
  - Знак ->
  - Тело
  - Примеры – см. предыдущий слайд
  - Другой вариант определения имя_класса::имя_метода
    - Подходящий метод из перегруженных подбирается по совпадению параметров

6.28 Что дальше делать с лямбдой?

- Первый вариант – просто применить
  - Можно передать в функцию как функциональный аргумент
- Второй вариант – использовать стандартные возможности коллекций

6.29 Простой пример применения

```
public class Sample_6_29 {

    interface BinaryOperation {
        int methodWithAnyName(int a, int b);
    }

    static void printResult(BinaryOperation op, int arg1, int arg2){
        System.out.println(op.methodWithAnyName(arg2, arg2));
    }

    static int myMethod(int x, int y){
        return x=y;
    };

    public static void main(String[] args) {
        BinaryOperation add = (arg1, arg2) -> arg1 + arg2;
        BinaryOperation multiply = (arg1, arg2) -> arg1 * arg2;
        BinaryOperation custom = Sample_6_29::myMethod;
        printResult(add, 5,7);
        printResult(multiply, 5,7);
    }


}
```

6.30 Использование в коллекциях

- Метод stream() и класс Stream
- Метод filter
- Метод map
- Метод forEach
- Метод collect
- И другие полезные методы

6.31 пример

```
roster
     .stream()
     .filter(e -> e.getGender() == Person.Sex.MALE)
     .forEach(e -> System.out.println(e.getName());
```

6.32 Q & A