Теория и технология программирования

Основы программирования на языке Java

Настраиваемые типы
(generic types)

Настраиваемые типы (generics)

- Настраиваемые типы
  - Позволяют создавать классы и интерфейсы, работающие с разными типами данных, обеспечивая при этом безопасность типов
    - С J2SE 5.0 все коллекции переписаны с использованием настраиваемых типов
  - Несколько напоминают шаблоны C++, но есть существенные отличия
  - Настраиваемый тип = параметризованный тип

Demo: Generic1Sample

Настраиваемые типы (generics)

- Определение простого настраиваемого класса

```
class Gen<T> {
    T obj;
    Gen( T obj ) { this.obj = obj; }
    T getObject() { return obj; }
    void showType() {
        System.out.println( obj.getClass().getName() );
    }
}
```

Demo: Generic1Sample

Настраиваемые типы (generics)

- Создание объекта настраиваемого класса

```
class GenDemo {
    public static void test() {
        Gen<Integer> intObj = new Gen<Integer>( 100 );
        int intVal = intObj.getObject();
        System.out.println( "intObj contains " + intVal );
        intObj.showType();
        //...
    }
```

Demo: Generic1Sample

Настраиваемые типы (generics)

- Создание объекта настраиваемого класса

```
class GenDemo {
    public static void test() {
        //...
        Gen<String> strObj = new Gen<String>( "abc" );
        String strVal = strObj.getObject();
        System.out.println( "strObj contains " + strVal );
        strObj.showType();
        //...
    }
```

Demo: Generic1Sample

Настраиваемые типы (generics)

- Использование метасимвольного аргумента (wildcard type parameter)
  - Позволяет работать с объектами настраиваемых типов, инстанцированными разными типами
  - Сначала рассмотрим простой случай

Demo: Generic1Sample

Настраиваемые типы (generics)

- Использование метасимвольного аргумента (wildcard type parameter)

```
class GenDemo {
    public static void test() {
        Gen<Integer> intObj = new Gen<Integer>( 100 );
        info( intObj );
        Gen<String> strObj = new Gen<String>("abc" );
        info( strObj );
    }
    public static void info( Gen<?> gen ) {
        gen.showType();
    }
```

Demo: Generic1Sample

Настраиваемые типы (generics)

- Безопасность типов
  - > Обеспечивается на стадии компиляции

```
class GenDemo {
    public static void test() {
        Gen<Integer> intObj = new Gen<Integer>( 100 );
        Gen<String> strObj = new Gen<String>("abc" );
        // Не компилируется!
        intObj = strObj;
    }
```

Demo: Generic1Sample

Настраиваемые типы (generics)

- А если используется обычный тип?
  - > Единственно возможный вариант до J2SE 5.0

```
class NonGen {
    Object obj;
    NonGen( Object obj ) { this.obj = obj; }
    Object getObject() { return obj; }
    void showType() {
        System.out.println( obj.getClass().getName() );
    }
}
```

Demo: Generic1Sample

Настраиваемые типы (generics)

- А если используется обычный тип?
  - > При работе с объектами требуется явное преобразование типа

```
class NonGenDemo {
    public static void test() {
        NonGen intObj = new NonGen( 100 );
        int intVal = (Integer) intObj.getObject();
        System.out.println( "intObj contains " + intVal );
        intObj.showType();
```

Demo: Generic1Sample

Настраиваемые типы (generics)

- А если используется обычный тип?
  - > Ошибки несоответствия типов выявляются только на стадии выполнения

Runtime error

```
class NonGenDemo {
    public static void test() {
        NonGen intObj = new NonGen( 100 );
        NonGen strObj = new NonGen( "abc" );
        // Компилируется, хотя концептуально не корректно
        intObj = strObj;
        int val = (Integer) intObj.getObject();
```

Настраиваемые типы (generics)

- Промежуточные выводы
  - Настраиваемый тип позволяет избавиться от преобразований типов, обеспечивая безопасность типов
  - Ошибки несоответствия типов выявляются на стадии компиляции
  - Если не стадии компиляции нет ошибок, то это дает гарантию того, что данный код не выбрасывает ClassCastException

Demo: Generic2Sample

Настраиваемые типы (generics)

- Ограниченный настраиваемый тип
  - Во многих случаях это замечательно, что параметр настраиваемого типа может заменяться любым классом
  - Но иногда полезно ограничить множество типов, которые можно передавать параметру настриваемого типа
    - класс BadStats – пример с ошибкой
    - класс Stats – исправленный вариант

Demo: Generic2Sample

Настраиваемые типы (generics)

- Ограниченный настраиваемый тип

```
class BadStats<T> {
    T[] values;
    BadStats( T[] values ) { this.values = values; }
    double average() {
        double sum = 0.0;
        for( T x: values ) {
            // Не компилируется: имеет смысл не для любого типа
            sum += x.doubleValue();
        }
        return sum / values.length;
    }
}
```

Demo: Generic2Sample

Настраиваемые типы (generics)

- Ограниченный настраиваемый тип

```
class Stats<T extends Number> {
    T[] values;
    Stats( T[] values ) { this.values = values; }
    double average() {
        double sum = 0.0;
        for( T x: values ) {
            // Теперь верно, т.к. множество типов ограничено
            sum += x.doubleValue();
        }
        return sum / values.length;
    }
}
```

Demo: Generic2Sample

Настраиваемые типы (generics)

- Ограниченный настраиваемый тип
  - > Корректность контролируется на стадии компиляции

```
{
    String strValues[] = { “abc”, “def”, “abc” };
    // Не компилируется:
    Stats<String> = new Stats<String> ( strValues );
}
```

Demo: Generic2Sample

Настраиваемые типы (generics)

- Ограниченный настраиваемый тип
  - Еще один случай применения метасимвольной маски
  - Предположим, мы хотим иметь возможность сравнивать средние значения для разных объектов Stats

Demo: Generic2Sample

Настраиваемые типы (generics)

- Ограниченный настраиваемый тип
  - > Этот пример не будет работать

```
class Stats<T extends Number> {
    //...
    boolean sameAvg( Stats<T> arg ) {
        if( average() == arg.average() ) return true;
            return false;
        }
}
```

Demo: Generic2Sample

Настраиваемые типы (generics)

- Ограниченный настраиваемый тип
  - Исправленный вариант использует метасимвольный аргумент
    - Фактически, в данном случае ? extends Number

```
class Stats<T extends Number> {
    //...
    boolean sameAvg( Stats<?> arg ) {
        if( average() == arg.average() ) return true;
            return false;
        }
}
```

Demo: Generic2Sample

Настраиваемые типы (generics)

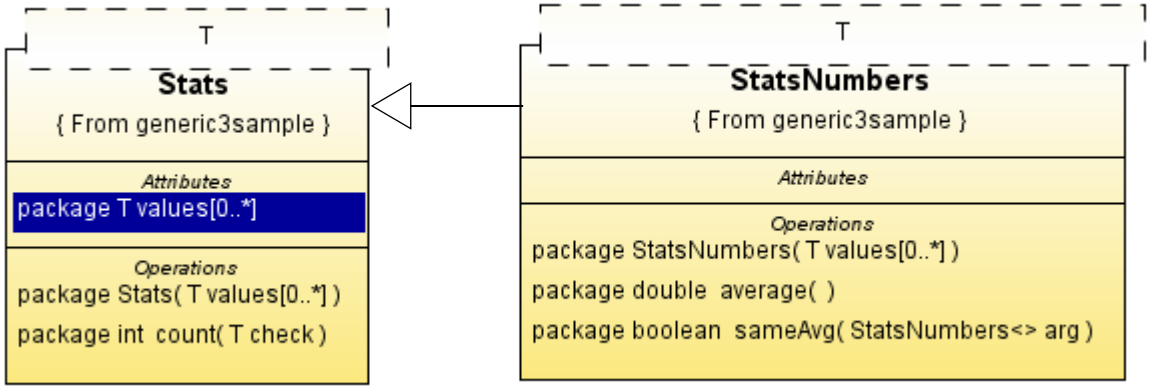
- Ограниченный настраиваемый тип
  - Обращение к методу sameAvg()

```
{
    Integer intVals[] = { 1, 2, 3, 4, 5 };
    Stats<Integer> intStats = new Stats<Integer>( intVals );
    Double dblVals[] = { 0.8, 1.9, 3.0, 4.1, 5.2 };
    Stats<Double> dblStats = new Stats<Double>( dblVals );
    if( intStats.sameAvg(dblStats)) {
        System.out.println( "Average values are the same");
    }
}
```

Demo: Generic3Sample

Настраиваемые типы (generics)

- Настраиваемые классы и иерархии классов
  - > Модифицируем предыдущее решение
    - > Есть алгоритмы, которые могут работать как с числовыми объектами, так и с прочими
    - > Есть алгоритмы, которые имеют смысл именно для числовых объектов



Demo: Generic3Sample

Настраиваемые типы (generics)

- Настраиваемые классы и иерархии классов
  - > Базовый настраиваемый класс

```
class Stats<T> {
    T[] values;
    Stats( T[] values ) { this.values = values; }
    int count( T check ) {
        int counter = 0;
        for( int i=0; i<values.length; i++ ) {
            if( check.equals( values[i] )) counter++;
        }
        return counter;
    }
}
```

Demo: Generic3Sample

Настраиваемые типы (generics)

- Настраиваемые классы и иерархии классов

```
class StatsNumbers<T extends Number> extends Stats<T> {
    StatsNumbers( T[] values ) { super( values );
    }
    double average() {
        double sum = 0.0;
        for( T x: values ) {
            sum += x.doubleValue();
        }
        return sum / values.length;
    }
    //...
}
```

Настраиваемые типы (generics)

- Настраиваемые классы и иерархии классов
  - Не следует путать иерархию самих настраиваемый типов и иерархию типов-параметров
    - Нижеследующий код НЕ КОМПИЛИРУЕТСЯ!

```
{
    // Некорректно!
    StatsNumber<Number> st = new StatsNumber<Integer>(...);
}
```

Demo: Generic4Sample

Настраиваемые типы (generics)

- Ограниченные метасимвольные аргументы
  - ? означает «неизвестный тип», а не «любой тип». ? ≠ Object !!!
  - Иногда требуется ограничить множество допустимых типов, которые могут заместить метасимвольный аргумент
    - ? extends ClassName
    - ? super ClassName

Demo: Generic5Sample

Настраиваемые типы (generics)

- Настраиваемые интерфейсы

```
interface MinMax<T extends Comparable<T>> {
    T min();
    T max();
}
```

> Обратите внимание, что интерфейс Comparable также является настраиваемым

> В реализации min() и max() потребуется использовать метод compareTo() интерфейса Comparable

Настраиваемые типы (generics)

- Промежуточные выводы
  - Настраиваемые классы могут образовывать иерархии
    - В том числе, с ограниченными типами
  - Настраиваемыми могут быть отдельные методы обычного (ненастраиваемого) класса
    - В том числе - конструкторы
  - Метасимвольные аргументы могут быть ограниченными
  - Интерфейсы тоже могут быть настраиваемыми

Demo: Generic6Sample

Настраиваемые типы (generics)

- Несформированные типы (raw types)
  - > Обеспечивают возможность совместимости с ранее написанным кодом, не поддерживающим настраиваемые типы

```
class Gen<T> {
    T obj;
    Gen( T obj ) { this.obj = obj; }
    T getObject() {
        return obj;
    }
}
```

Demo: Generic6Sample

Настраиваемые типы (generics)

- Несформированные типы (raw types)

```
// Создаем объект настраиваемого типа,
// опуская параметр типа
Gen intObj = new Gen( new Integer(100) );
// Необходимо приведение типа
int intVal = (Integer) intObj.getObject();
System.out.println( "intVal is " + intVal );
Gen strObj = new Gen( "Raw type sample" );
// Безопасность на этапе компиляции не гарантируется
intObj = strObj;
// Следующая строка вызывает ошибку времени выполнения
int intVal2 = (Integer) intObj.getObject();
```

Demo: Generic7Sample

Настраиваемые типы (generics)

- Настраиваемые типы и коллекции
  - Все классы и интерфейсы Collection Framework – теперь настраиваемые

```
// Старый стиль
ArrayList list = new ArrayList();
    // (Предполагается, что список
    // хранит объекты Object)

// Новый стиль
ArrayList<String> = new ArrayList<String>();
```

Demo: Generic8Sample

Настраиваемые типы (generics)

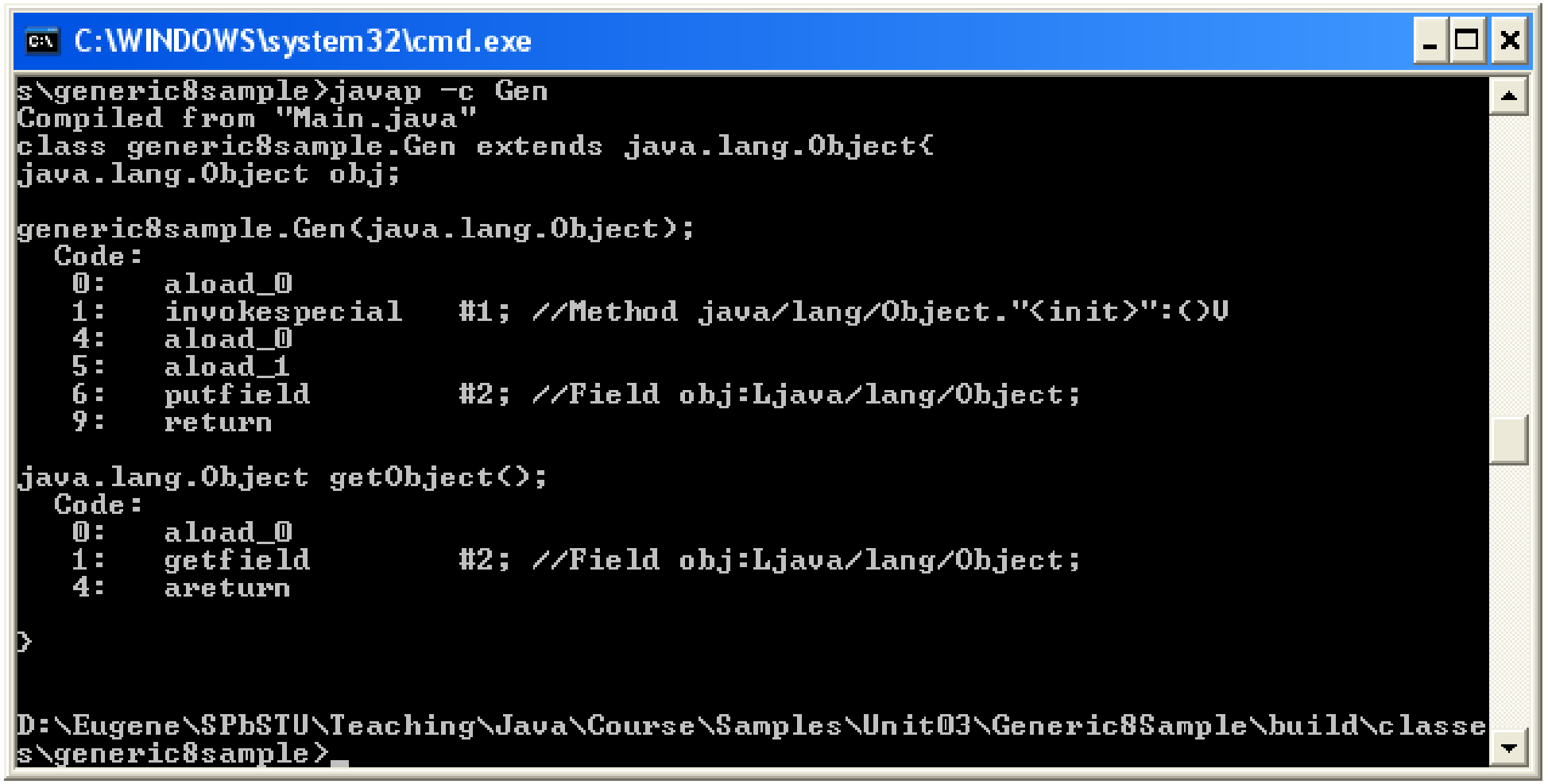
- Стирание
  - Модель реализации настраиваемых типов в Java предполагает стирание: при компиляции вся информация о настраиваемых типах стирается
    - Дизассемблирование Java-кода

`T`	`Object`
`T extends SomeClass`	`SomeClass`
`Gen<Integer> obj = new Gen<Integer>( 10 );` `int x = obj.getObject();`	`Gen obj = new Gen( 10 );` `int x = (Integer) obj.getObject();`

Demo: Generic8Sample

Настраиваемые типы (generics)

- Стирание: результат



Настраиваемые типы (generics)

- Ограничения
  - > Нельзя создавать объекты, используя тип параметра

```
class Bad1<T> {
    T obj;
    Bad1() {
        obj = new T(); // Недопустимо!
    }
}
```

Настраиваемые типы (generics)

- Ограничения
  - > Нельзя создать массив объектов, тип которых задан с помощью параметра типа

```
class Bad2<T> {
    T vals[]; // OK!
    Bad2( T[] nums ) {
        vals = new T[10]; // Недопустимо!
        vals = nums; // OK!
    }
}
```

Настраиваемые типы (generics)

- Ограничения
  - > Статические члены класса не могут использовать параметр T

```
class Bad3<T> {
    static T obj; // Недопустимо!
    static T getObj() { // Недопустимо!
        return obj;
    }
}
```

Настраиваемые типы (generics)

- Ограничения
  - > Настраиваемый тип не может расширять класс Throwable
    - > Классы исключений не могут быть настраиваемыми

```
// Недопустимо!
class BadException<T> extends Exception {
    //...
}
```

Упражнения для самостоятельной работы

- Переработайте решение задачи проверки упорядоченности массива таким образом, чтобы обеспечить возможность работы с числовыми типами
- Переработайте решение задачи проверки упорядоченности массива таким образом, чтобы обеспечить возможность работы с любыми типами, реализующими интерфейс Comparable

Q&A