Введение в обобщения Java

Java Generics

Лекция #19

Пустовалова О.Г.
доцент. каф. мат.мод.
ИММиКН ЮФУ

Содержание

Обобщенные классы

Введение в обобщения Java

Обобщённое программирование — это такой подход к описанию данных и алгоритмов, который позволяет их использовать с различными типами данных без изменения их описания.

В Java, начиная с версии J2SE 5.0, добавлены средства обобщённого программирования, синтаксически основанные на C++.

<контейнеры типа T> — подмножество обобщённого программирования

Введение в обобщения Java. Пример

```
class BoxPrinter {
    private Object val;

    public BoxPrinter(Object arg) { val = arg; }
    public String toString() { return "{" + val + "}";}
    public Object getValue() { return val; }
}

class Main {
    public static void main(String[] args) {

        BoxPrinter value1 = new BoxPrinter(new Integer(10));
        System.out.println(value1);

        Integer intValue1 = (Integer) value1.getValue();
        BoxPrinter value2 = new BoxPrinter("Hello world");
        System.out.println(value2);

        // Здесь программист допустил ошибку, присваивая
        // переменной типа Integer значение типа String.
        Integer intValue2 = (Integer) value2.getValue();
    }}
```

Применение обобщённого программирования позволяют обнаруживать ошибки на этапе компиляции, а не на этапе выполнения.

Ошибка во время выполнения

Введение в обобщения Java. Пример

```
class BoxPrinter<T> {
    private T val;

    public BoxPrinter(T arg) { val = arg; }

    public String toString() { return "{" + val + "}"; }

    public T getValue() { return val; }
}

class Main {
    public static void main(String[] args) {

        BoxPrinter<Integer> value1 = new BoxPrinter<Integer>(new Integer(10));
        System.out.println(value1);
        Integer intValue1 = value1.getValue();

        BoxPrinter<String> value2 = new BoxPrinter<String>("Hello world");
        System.out.println(value2);

        // Здесь повторяется ошибка предыдущего фрагмента кода
        Integer intValue2 = value2.getValue();
    }
}
```

Ошибка во время компиляции

Введение в обобщения Java. Объяснение

```
class BoxPrinter<T>
```

После имени класса в угловых скобках "<" и ">" указано имя типа "T", которое может использоваться внутри класса. Фактически T – это тип, который должен быть определён позже (при создании объекта класса).

```
private T val;
```

объявляется переменная дженерик-типа (generic type), её тип будет указан позже, при создании объекта класса BoxPrinter.

Введение в обобщения Java. Объяснение

```
BoxPrinter <Integer>  value1
```

Здесь указывается, что T имеет тип Integer. То есть, для объекта value1 все поля T-типа его класса BoxPrinter становятся полями типа Integer.

```
public BoxPrinter(T arg) { val = arg; }
```

аргумент для конструктора BoxPrinter имеет тип T.

```
new BoxPrinter<Integer>(new Integer(10))
```

при вызове конструктора в new, указывается, что T имеет тип Integer

Обобщения (Generics). Обобщенные классы

Обобщения или **generics** (обобщенные типы и методы) позволяют нам уйти от жесткого определения используемых типов.

```
class Account<T>{

    private T id;
    private int sum;

    public Account(T id, int sum) {
        this.id = id;
        this.sum = sum;
    }
    public T getId() {return id;}

    public int getSum() {return sum;}

    public void setSum(int sum) {this.sum = sum;}
}
```

С помощью буквы **T** в определении класса **class Account<T>** мы указываем, что данный тип **T** будет использоваться этим классом.

Параметр **T** в угловых скобках называется **универсальным параметром**, так как вместо него можно подставить любой тип.

Обобщения (Generics). Обобщенные классы

При определении переменной данного класса и создании объекта после имени класса в угловых скобках нужно указать, **какой именно тип будет использоваться вместо универсального параметра**.

```
public class Main{

    public static void main(String[] args) {

        Account<String> acc1 = new Account<String>("2345", 5000);
        String acc1Id = acc1.getId();
        System.out.println(acc1Id);

        Account<Integer> acc2 = new Account<Integer>(2345, 5000);
        Integer acc2Id = acc2.getId();
        System.out.println(acc2Id);
    }
}
```

```
2345
2345
```

Обобщения (Generics). Обобщенные классы

Обобщенные типы работают **только с объектами**, но не работают с примитивными типами.

```
Account<integer> acc2 = new Account<Integer>( id: 2345,  sum: 5000);
Integer acc2Id = acc2.getId();
System.out.println(acc2Id);
```

Ошибка!

```
Account<Integer> acc2 = new Account<Integer>( id: 2345,  sum: 5000);
Integer acc2Id = acc2.getId();
System.out.println(acc2Id);
```

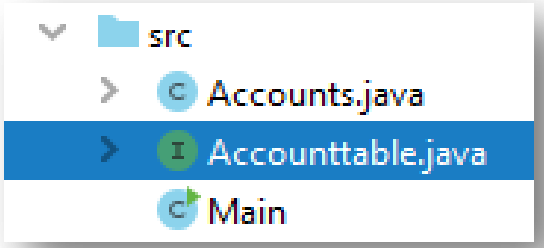
Обобщенные интерфейсы

Обобщения (Generics). Обобщенные интерфейсы

Интерфейсы, как и классы, также могут быть обобщенными.

Обобщенный интерфейс Accountable:

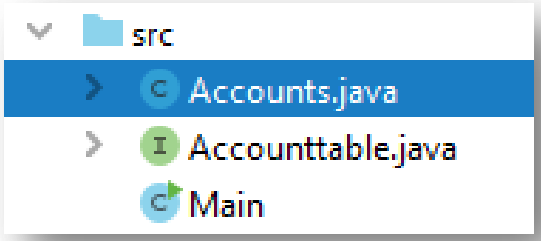
```
interface Accountable<T>{
    T getId();
    int getSum();
    void setSum(int sum);
}
```



Обобщения (Generics). Обобщенные интерфейсы

При реализации для универсального параметра интерфейса задается конкретный тип, как например, в данном случае это тип String.
Тогда класс, реализующий интерфейс, **жестко привязан к этому типу**.

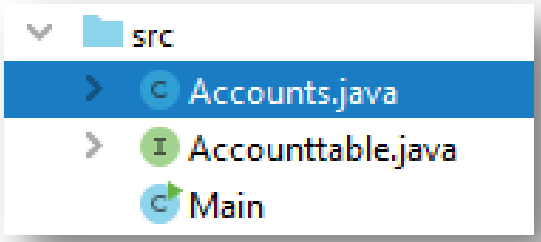
```
class Account implements Accountable<String>{

    private String id;
    private int sum;

    Account(String id, int sum){
        this.id = id;
        this.sum = sum;
    }

    public String getId() { return id; }
    public int getSum() { return sum; }
    public void setSum(int sum) { this.sum = sum; }
}
```



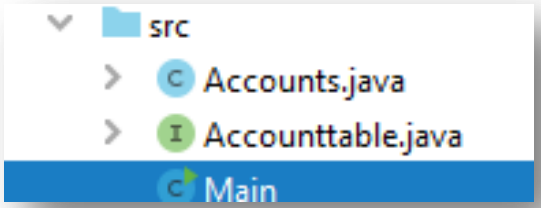
Обобщения (Generics). Обобщенные интерфейсы

Можно действовать так – определить обобщенный класс, который также использует тот же универсальный параметр.

```
class Account <T> implements Accountable<T>{

    private T id;
    private int sum;

    Account(T id, int sum){
        this.id = id;
        this.sum = sum;
    }

    public T getId() { return id; }
    public int getSum() { return sum; }
    public void setSum(int sum) { this.sum = sum; }
}
```



Обобщения (Generics). Обобщенные интерфейсы




```
public class Main{

    public static void main(String[] args) {

        Accountable<String> acc1 = new Account("1235rwr", 5000);
        Account acc2 = new Account("2373", 4300);

        System.out.println(acc1.getId());
        System.out.println(acc2.getId());
    }
}
```

```
1235rwr
2373
```

Обобщенные методы

Обобщения (Generics). Обобщенные методы

```
public class Main {
    public static void main(String[] args) {

        Printer printer = new Printer();
        String[] people = {"Tom", "Alice", "Sam", "Bob", "Helen"};
        Integer[] numbers = {23, 4, 5, 2, 13, 456, 4};

         printer.<String>print(people);
         printer.<Integer>print(numbers);
    }
}

class Printer{

    public <T> void print(T[] items){
        for(T item: items){
            System.out.println(item);
        }
    }}
```

Особенностью обобщенного метода является использование универсального параметра в объявлении метода после всех модификаторов и перед типом возвращаемого значения.

Использование нескольких обобщенных параметров

Обобщения (Generics). Несколько параметров

```
public class Main{

    public static void main(String[] args) {

        Account<String, Double> acc1 = new Account<String, Double>("354", 5000.87);
        String id = acc1.getId();
        Double sum = acc1.getSum();
        System.out.printf("Id: %s  Sum: %f \n", id, sum);
    }
}
class Account<T, S>{

    private T id;
    private S sum;

    Account(T id, S sum){
        this.id = id;
        this.sum = sum;
    }

    public T getId() { return id; }
    public S getSum() { return sum; }
    public void setSum(S sum) { this.sum = sum; }
}
```

В данном случае тип String будет передаваться на место параметра T, а тип Double - на место параметра S.

Обобщенные конструкторы

Обобщения (Generics). Обобщенные конструкторы

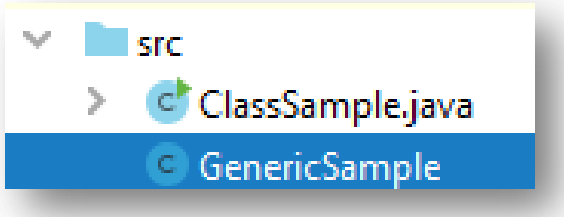
```
public class Main{
    public static void main(String[] args) {

        Account acc1 = new Account("cid2373", 5000);
        Account acc2 = new Account(53757, 4000);
        System.out.println(acc1.getId());
        System.out.println(acc2.getId());
    }
}

class Account{

    private String id;
    private int sum;

     <T>Account(T id, int sum){
         this.id = id.toString();
         this.sum = sum;
     }
    public String getId() { return id; }
    public int getSum() { return sum; }
    public void setSum(int sum) { this.sum = sum; }
}
```

Конструкторы как и методы также могут быть обобщенными. В этом случае перед конструктором также указываются в угловых скобках универсальные параметры.

Примеры

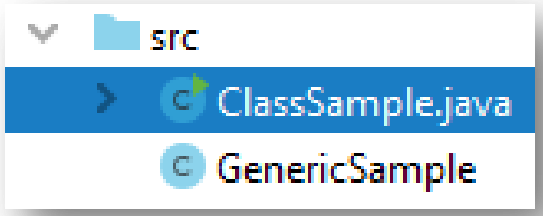
Пример использования generic-класса. Часть 1

```
class GenericSample<T>
{
    private T value;

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

    public String toString() {
        return "{" + value + "}";
    }

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



Пример использования generic-класса. Часть 2

```
class TestSample
{
    public static void main(String[] args)
    {
        GenericSample<Integer> value1;
        value1 = new GenericSample<Integer>(new Integer(10));
        System.out.println(value1);

        // Ошибки нет
        Integer intValue1 = value1.getValue();

        GenericSample<String> value2;
        value2 = new GenericSample<String>("Hello world");
        System.out.println(value2);

        // Здесь возникает ошибка несоответствия типа
        // Integer intValue2 = value2.getValue();
    }
}
```



{10}

{Hello world}

Спасибо за внимание!