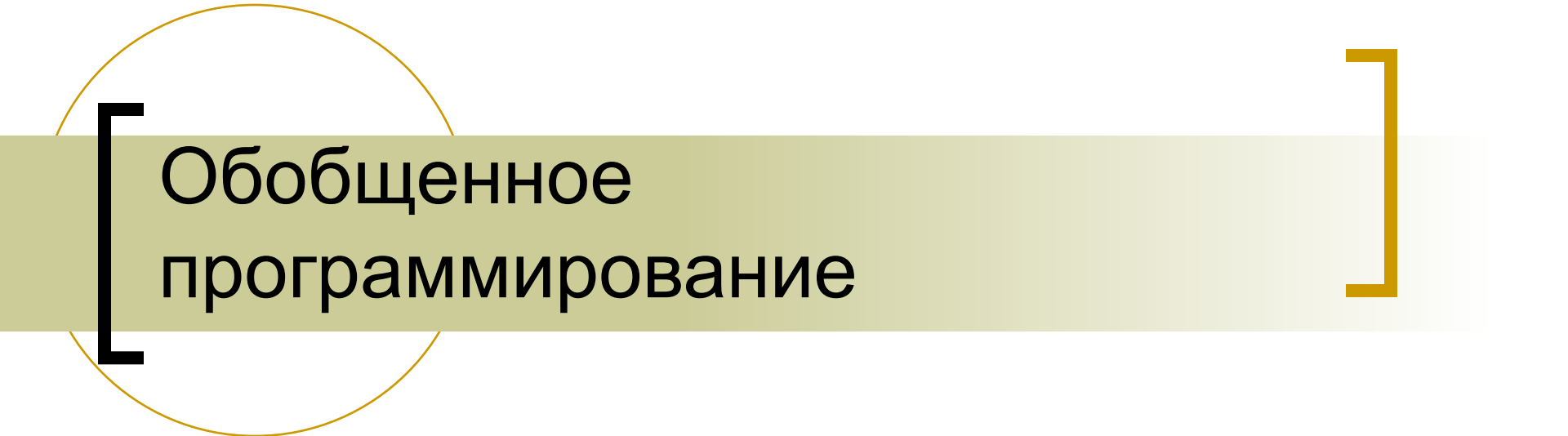




# Язык программирования JAVA

Дополнительные возможности Java 5  
Обобщенное программирование  
Аннотации



Обобщенное  
программирование

# Обобщенное программирование

- Механизм Generics позволяет абстрагироваться от типов данных
- Типичный пример – работа с коллекциями

```
List myIntList = new LinkedList(); // 1
myIntList.add(new Integer(0)); // 2
Integer x = (Integer) myIntList.iterator().next(); // 3
```

# Обобщенное программирование

- В Java 5 вы можете параметризовать классы

```
List<Integer> myIntList = new LinkedList(); // 1
myIntList.add(new Integer(0)); // 2
Integer x = myIntList.iterator().next(); // 3
```

- В интерфейс List передается параметр – ТИП помещаемых данных

# Определение простого Generic класса

```
public interface List <E>{  
    void add(E x);  
    Iterator<E> iterator();  
}  
  
public interface Iterator<E>{  
    E next();  
    boolean hasNext();  
}
```

- Отличие от обычного класса состоит в параметре в угловых скобках

# Наследование и Generic классы

- Правильен ли этот код?

```
List<String> ls = new ArrayList<String>(); // 1  
List<Object> lo = ls; // 2
```

- А этот?

```
List<String> ls = new ArrayList<String>(); // 1  
List<Object> lo = ls; // 2
```

- При использовании Generic классов с параметрами, входящими в иерархию, необходимо соблюдать осторожность.

# [ Маски ]

- При задании типов-параметров можно указывать маски типов, допустимых для класса Generic

```
void printCollection(Collection<?> c) {  
    for (Object e : c) {  
        System.out.println(e);  
    }  
}
```

```
public void drawAll(List<Shape> shapes) {  
    ...  
}
```

```
public void drawAll(List<? extends Shape> shapes) {  
    ...  
}
```

# Generic методы

- Можно добавлять параметры только к методам класса

```
static <T> void fromArrayToCollection(T[] a, Collection<T> c) {  
    for (T o : a) {  
        c.add(o);  
    }  
}
```

```
static <T> void fromArrayToCollection(T[] a, Collection<T> c) {  
    for (T o : a) {  
        c.add(o);  
    }  
}
```

```
Collection<Object> co = new ArrayList<Object>();  
String[] sa = new String[100];  
Collection<String> cs = new ArrayList<String>();  
fromArrayToCollection(sa, cs); // T будет типа String  
fromArrayToCollection(sa, co); // T будет типа Object
```



# Варианты использования Generics



```
interface Transformer<IT, OT>{
    public OT transformData(IT data);
    public addInputFilter (InputFiter<IT> filter);
    public addOutputFilter (OutputFilter<OT> filter);
}

interface InputFilter<T>{
    T doFilter(T data)
}

interface OutputFilter<T>{
    T doFilter(T data)
}
```



Аннотации

# Что такое аннотации

- Аннотация – дополнительная информация о методе или классе
- Аннотации предназначены для обработки сторонними утилитами (компилятор, IDE, среда выполнения)
- С помощью аннотаций можно генерировать дополнительные файлы (документация, маппинг, дескрипторы развертывания)

# Зачем использовать аннотации?

- «Декларативный» стиль программирования
- Нет необходимости поддерживать сторонние файлы в актуальном состоянии – все хранится в исходном коде

# Создание аннотаций

- Похоже на объявление интерфейса
  - Необходимо перед названием поставить знак “@”
  - Каждое объявление метода определяет атрибут аннотации
  - Объявления методов не должны иметь параметров
  - Возвращаемые типы методов должны быть примитивами, String, Class, enum, аннотацией или массивом
  - Возможно объявлять значения по умолчанию

# [ Создание аннотаций ]

```
@Retention(value = RetentionPolicy.RUNTIME)
@Target(value = ElementType.METHOD)
public @interface PropertySettings {
    String description() default "";
    int order() default 0;
    String displayName() default "";
    boolean required() default false;
    String defaultValue() default "";
}
```

# Использование аннотаций

- После объявления аннотации она может быть использована на уровне описания
  - Класса
  - Метода
  - Свойства
- Аннотация предшествует другим модификаторам

# [ Использование аннотаций ]

```
@PropertySettings(order = 2, displayName = "Private Key Password")
public void setPrivateKeyPassword(String password) {
    adapter.setPrivateKeyPassword(password);
}
```



# [ Типы аннотаций ]

- Маркер
- Аннотация с единственным значением
- Обычная аннотация (был рассмотрен выше)

# [ Маркер-аннотация ]

- Аннотация без элементов
- Объявление

```
public @interface DatabaseDatasource {  
}
```

- Использование

```
@DatabaseDatasource  
public void setDbName(String name) {  
    ...  
}
```

# Аннотация с единственным значением

- Аннотация, в которой может содержаться только одно значение
  - Элемент должен называться «value»
- Объявление

```
public @interface DatabaseDatasource {  
    String value()  
}
```

- Использование

```
@DatabaseDatasource ("Oracle")  
public void setDbName(String name) {  
    ...  
}
```

# Мета-аннотации

- @Retention
  - Как долго аннотация остается в коде
    - SOURCE
    - CLASS (по умолчанию)
    - RUNTIME
- @Target
  - Ограничение использования
    - TYPE
    - FIELD
    - METHOD
    - ...

# Получение значений аннотаций

## ■ Маркер

```
boolean isDataSource = MyClass.class.isAnnotationPresent  
(DatabaseDatasource.class)
```

## ■ С единственным значением

```
String dataSourceName =  
MyClass.class.getAnnotation(DatabaseDatasource.class).value()
```

## ■ Обычная аннотация

```
PropertySettings propSettings =  
writeMethod.getAnnotation(PropertySettings.class);  
prp.setOrder(propSettings.order());  
prp.setDisplayName(propSettings.displayName());
```



Вопросы?