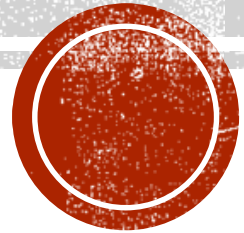
ОБЈЕКТНО ОРЈЕНТИСАНО ПРОГРАМИРАЊЕ

ПРОГРАМСКИ ЈЕЗИК ЈАВА — 2



Анотације

АНОТАЦИЈЕ (ЗАБЕЛЕШКЕ)

Анотације (још се називају и забелешке) обезбеђују податке о програму које саме по себи нису део програма.

- Анотације су специјална врста коментара
  - Исто као коментари, анотације не мењају нити утичу на семантику програма, тј. на понашање програма током његовог извршавања.
- Анотације су мета-описи
  - За разлику од коментара, анотацијама могу приступити и њих могу користити други развијени програмски алати или чак и сам програм који развијамо.

АНОТАЦИЈЕ (2)

Анотације се користе у различите сврхе:

- Обезбеђивање додатних информација преводиоцу – анотације се могу користити од стране преводиоца ради детектовања грешака или ускраћивања упозорења.
- Процесирање у времену превођења и у времену испоручивања – софтверски алати могу процесирати анотације ради генерисања кода, XML датотека, итд.
- Процесирање у времену извршавања — неке анотације су доступне за испитивање у времену извршавања.

Анотације се могу применити на декларације: декларације класа, поља, метода и других елемената програма.

Када се примењују на декларације, конвенција је да се свака анотација пише у новој линији.

ФОРМАТ АНОТАЦИЈЕ

У свом најростијем облику, анотација има следећи формат:

```
@Entity
```

Знак `@` указује преводиоцу да се ради о анотацији.
У примеру који следи, име анотације је `Override`:

```
@Override
  void mySuperMethod() { ... }
```

Анотација може да садржи елементе, било именоване или неименоване, и да поставе вредности за те елементе:

```
@Author(
     name = "Benjamin Franklin",
     date = "3/27/2003"
)
class MyClass() { ... }
```

или

```
@SuppressWarnings(value = "unchecked")
  void myMethod() { ... }
```

ФОРМАТ АНОТАЦИЈЕ (2)

Ако постоји само један именовани елемент, тада се име елемента може уклонити приликом доделе вредности, као у:

```
@SuppressWarnings("unchecked")
void myMethod() { ... }
```

Ако анотација нема елемената (маркерска), тада се и заграде могу уклонити, што је и био случај у примеру са `@Override` анотацијом.

Такође је могуће поставити више анотација на исту декларацију:

```
@Author(name = "Jane Doe")
@EBook
class MyClass { ... }
```

Тип анотације може бити новонаправљени (енг. custom) тип, или један од типова из пакета `java.lang` или `java.lang.annotation` у оквиру **Java SE API.** У претходним примерима, анотације `Override` и `SuppressWarnings` су предефинисани типови Јава анотације, а `Author` и `EBook` су новонаправљени типови анотације.

ПРЕДЕФИНИСАНИ ТИПОВИ АНОТАЦИЈЕ

Предефинисани типови анотације већ постоје и користе се у језику Јава.
Ови типови анотације су дефинисани у пакету `java.lang`.

То су:

- `Deprecated`
- `Override`
- `SuppressWarnings`
- `SafeVarargs`

Поред тога, од верзије 8 постоји још један тип анотације, назван `FunctionalInterface`.

АНОТАЦИЈА `Deprecated`

Анотација `@Deprecated` указује да означени елеменат више није потребан (застарео је) и да се надаље неће користити.

Кад год програм користи метод, класу или поље које је анотирано `Deprecated` анотацијом, Јава преводилац генерише упозорење.

Када елеменат постане застарео, то такође треба и документовати коришћењем the Javadoc тага `@deprecated`.

И таг и анотација почињу симболом `@`, иза кога код Javadoc елемента следи мало слово `d` а код анотације велико слово `D`.

Пример.

```
// Javadoc komentar
 /**
  * @deprecated
  * objasnjenje zasto je metod nepotreban i kako se preporucuje da se radi
  */
 @Deprecated
 static void deprecatedMethod() { }
```

АНОТАЦИЈА `Override`

Анотација `@Override` информише преводилац да анотирани елемент треба да превазиђе елеменат који је декларисан у надкласи.

Пример.

```
// mark method as a superclass method
// that has been overridden
@Override
int overriddenMethod() { }
```

Иако се у Јава програмирању не захтева да се приликом превазилажења метода користи ова анотација, њено коришење помаже у превенцији грешака.

Ако метод маркиран са анотацијом `Override` некоректно превазиђе метод надкласе, тада преводилац генерише грешку.

АНОТАЦИЈА `SuppressWarnings`

Анотација `@SuppressWarnings` налаже преводиоцу да не приказује конкретни тип упозорења, које би у супротном било приказано.

Пример. У коду који следи се користи застарели метод `deprecatedMethod`, за који преводилац обично генерише упозорење. У овом случају, међутим, анотација метода `useDeprecatedMethod` је блокирала генерисање једног типа упозорења.

```
    // koristi zastareli metod i nalaze prevodiocu da ne generise upozorenje
   @SuppressWarnings("deprecation")
    void useDeprecatedMethod() {
       objectOne.deprecatedMethod();
    }
```

Спецификација језика Јава разликује две категорије упозорења преводиоца: застарелост (енг. deprication) и непровереност (енг. unchecked). Непроверена упозорења се могу појавити кад се ради са старим кодом, кодом који је писан пре развоја генеричких класа и метода.

Пример. Да би се блокирало генерисање обе категорије упозорења, користи се следећа синтакса:

```
@SuppressWarnings({"unchecked", "deprecation"})
```

АНОТАЦИЈА `SafeVarargs`

Анотација `@SafeVarargs` , када се примени на метод или на конструктор, обезбеђује да се у телу метода/конструктора не извршавају операције које могу бити несигурне за параметре променљивог типа овог метода/конструктора, тј. за `varargs` параметре.

Ако се користи овај тип анотације, тада се код преводиоца блокира и генерисање упозорења непроверивости који би се односили на коришћење `varargs` параметара.

АНОТАЦИЈА `FunctionalInterface`

Анотација `@FunctionalInterface` , која уведена у верзији Јава 8, указује да декларација типа треба да буде функционални интерфејс, као што је дефинисано у спецификацији језика Јава. Прецизније, интерфејс са овом анотацијом садржи један апстрактни метод.

Ова анотација ће бити детаљније објашњена приликом проучавања ламбда израза у Јави.

КРЕИРАЊЕ НОВОГ ТИПА АНОТАЦИЈЕ

Креирање новог типа анотације је слично креирању интерфејса, при чему декларацији типа анотације претходи знак `@`.

Анотација не сме садржавати кључну реч `extends`. Међутим, анотације имплицитно наслеђују интерфејс `Annotation`.

Тело анотације се састоји од декларације метода (без тела метода). Методи унутар тела анотације се понашају као поља.

Пример. Тип анотације `Description`, којом се нпр. описује програмска датотека, може да има следећи облик:

```
@Retention( RetentionPolicy.RUNTIME )
@interface Description
{
   String author();
   String date();
}
```

КРЕИРАЊЕ НОВОГ ТИПА АНОТАЦИЈЕ (2)

Када анотација датог типа придружује декларацији, потребно је обезбедити одговарајуће вредности за чланове типа анотације.

Пример. Када је креиран тип анотације `Description`, тада се њиме могу анотирати класе и методе (нпр. `Test` и `testMethod`):

```
@Description( author = "Vlado", date = "22/10/2011,23/10/2011" )
public class Test {
   @Description( author = "Vlado", date = "22/10/2011" )
   public static void testMethod(){
      System.out.println( "Welcome to Java" );
      System.out.println( "This is an example of Annotations" );
   }
   public static void main( String args[] ){
      testMethod();
      showAnnotations();
   }
}
```

КРЕИРАЊЕ НОВОГ ТИПА АНОТАЦИЈЕ (3)

Напомена. У претходном примеру није приказан метод `showAnnotations`, који коришћењем рефлексије приказује на стандардном излазу анотације које су придружене класи и методама.

Метод `showAnnotations` ће бити приказан и детаљно анализиран нешто касније, у делу презентације који се односи на коришћење рефлексије ради испитивања анотација.

МЕТА-АНОТАЦИЈЕ

У претходном примеру се, приликом креирања новог типа анотације **Description**, користила анотација `Retention`.

Ова анотација `Retention` је предефинисана (већ постоји), служи за анотирање анотација, па се због тога назива **мета-анотација**.

Мета-анотације су дефинисане у пакету `java.lang.annotation`. То су:

- `Retention`
- `Documented`
- `Target`
- `Inherited`

Поред ових, у Јава 8 се појављује још један тип мета-анотације, назван `Repeatable`.

МЕТА-АНОТАЦИЈА `Retention`

Мета-анотација `@Retention` одређује како се памти анотација:

- `RetentionPolicy.SOURCE`

анотација се памти само на нивоу изворног кода и бива игнорисана од стране преводиоца.

- `RetentionPolicy.CLASS`

анотација се памти од стране преводиоца током превођења, али је игнорише JVM.

- `RetentionPolicy.RUNTIME`

анотација се памтио од стране JVM, па се може користити у окружењу извршавања.

МЕТА-АНОТАЦИЈА Documented

Мета-анотација @Documented указује да се анотација која је маркирана са Documented треба документовати коришћењем Javadoc алата (подразумевано је да се анотације не укључују у Javadoc).

За више информација о овој анотацији треба консултовати документацију о Javadoc алатима.

МЕТА-АНОТАЦИЈА `Target`

Мета-анотација `@Target` означава ограничење типа Јава елемента на који се може примењивати тако маркирана анотација.

Могући циљ маркиране анотације, тј. вредност мета-анотације је нека од следећих осам вредности:

- `ElementType.ANNOTATION_TYPE`

примењује се на тип анотације.

- `ElementType.CONSTRUCTOR`

примењује се на конструктор .

- `ElementType.FIELD`

примењује се на поље или на особину.

- `ElementType.LOCAL_VARIABLE`

примењује се на локалну променљиву.

- `ElementType.METHOD`

примењује се на метод.

- `ElementType.PACKAGE`

примењује се на декларацију пакета.

- `ElementType.PARAMETER`

примењује се на параметар метода.

- `ElementType.TYPE`

примењује се на ма који елеменат класе.

МЕТА-АНОТАЦИЈА `Inherited`

Мета-анотација `@Inherited` указује да тип анотације може бити наслеђен из надкласе, што није подразумевано понашање.

Када корисник испитује тип анотације, а класа нема анотацију датог типа, тада ће се надкласа испитивати за дати тип анотације.

Наравно, ова мета-анотација се односи само на анотирања кода класа.

МЕТА-АНОТАЦИЈА `Repeatable`

Мета-анотација **@Repeatable** указује да означена анотација може бити примењена више пута на исту декларацију.

Пре постојања ове анотације, морао је да се користи низ како би се имитирало постојање вишеструке анотације једног елемента.

ИСПИТИВАЊЕ АНОТАЦИЈА

Reflection API садржи методе за испитивање анотација.

Старији начин за испитивање анотација су методи `getAnnotation(Class<T>)` и `getAnnotation(Class<T>)`, који су дефинисани у класама `Class` и `Method` респективно.

Ови методи враћају једну анотацију, исто као и новији метод `getAnnotationByType(Class<T>)` класе `AnnotatedElement`, под претпоставком да постоји анотација захтеваног типа.

У Јава су, почев од верзије 8, укључени и додтани методи који пролазе кроз скуп анотација и једним позивом врањз све анотације датог типа. Такав је метод `getAnnotations(Class<T>)` класе `AnnotatedElement`.

ИСПИТИВАЊЕ АНОТАЦИЈА (2)

Пример. Метод који следи приказује анотације:

```
public static void showAnnotations()
{
   Test test = new Test();
   try{
      Class c = test.getClass();
      Description annotation1 = (Description) c.getAnnotation( Description.class );
      System.out.println( "Name of the class: " + c.getName() );
      System.out.println( "Author of the class: " + annotation1.author() );
      System.out.println( "Date of Writing the class: " + annotation1.date() );
      Method m = c.getMethod( "testMethod" );
      Description annotation2 = m.getAnnotation( Description.class );
      System.out.println( "Name of the method: " + m.getName() );
      System.out.println( "Author of the method: " + annotation2.author() );
      System.out.println( "Date of Writing the method: " + annotation2.date() );
   } catch (NoSuchMethodException ex){
       System.out.println( "Invalid Method..." + ex.getMessage() );
   }
}
```