Annotations

O objetivo da mudança das antigas versões do JDK para o JDK5 está centrada no aumento da facilidade do desenvolvimento. Em outras palavras, os novos recursos transferem a responsabilidade pela construção de códigos completares (Xdoclet, XML, etc) do programador para o compilador. Se o código fonte é livre de código de apoio, ele se torna de fácil manutenção. O código fonte resultante também é menos propenso a erros (Bugs). Um destes novos recursos no JDK5 é chamado de Anotações (Annotations). Anotações são como meta-tags que você pode adicionar ao seu código e aplica-los em declaração de pacotes (packages), declaração de tipos, construtores, métodos, campos, parâmetros e variáveis. Desta forma, você terá mecanismos para indicar se seus métodos são dependentes de outros, se eles estão incompletos, se suas classes tem referências a outras classes, e assim por diante.

O desenvolvimento baseado em anotações é uma das últimas tendências no desenvolvimento Java. “Ele nos permite trocar a construção de códigos repetitivos pela utilização de ferramentas que geram estes a partir das anotações no código fonte. Isto permite um estilo de programação declarativa onde o programador diz o que deveria ser feito e as ferramentas geram o código para fazê-lo”. Falando de outra forma, anotação é o mecanismo para associar uma meta-tag com elementos do programa e permitir ao compilador ou a VM Java extrair comportamento destes elementos anotados e gere código interdependente quando necessário.

As Bases

Existem duas coisas que você necessita considerar em anotações. Um é a própria anotação; a outra é o tipo Annotation. Uma Anotação é uma meta-tag que você utiliza em seu código para dar alguma vida a ele. O Tipo Annotation é utilizado para definir uma anotação. Você usará ele quando for criar sua própria anotação.

Uma definição de tipo de anotação usa um @ (arroba) seguido pela palavra “interface” mais o nome da anotação. Por outro lado uma anotação leva o @ seguido pelo seu tipo. Esta é a forma simplificada de anotação. Adicionalmente, você pode colocar algum dado dentro de parênteses após o nome da anotação. Segue um exemplo:

Definindo uma Anotação (Tipo Anotação)

```
public @interface MinhaAnotacao {
  String fazAlgo();
}
```

Usando anotação em seu código

```
@MinhaAnotacao(fazAlgo=”O que fazer”)
public void meuMetodo() {
  ...
}
```

Tipos de Anotação

Existem três tipos de anotações:

- Marcador O tipo de anotação marcador não tem elementos, exceto o próprio nome

Exemplo:

```
public @interface MinhaAnotacao {
}
```

Uso:

```
@MinhaAnotacao
public void meuMetodo() {
  ...
}
```

- Único elemento O tipo único elemento ou único valor oferece apenas um único valor. Isto pode ser representado com o par dado=valor ou simplesmente o valor.

Exemplo:

```
public @interface MinhaAnotacao {
  String fazAlgo();
}
```

Uso:

```
@MinhaAnotacao("O que fazer");
public void meuMetodo() {
  ...
}
```

- Multivalorado Anotações deste tipo têm muitos membros.

Exemplo:

```
public @interface MinhaAnotacao {
  String fazAlgo();
  int contador;
  String data();
}
```

Uso:

```
@MinhaAnotacao(fazAlgo="O que fazer", contador=1, data="06-12-
                  2006")
public void meuMetodo() {
  ...
}
```

Passos para a definição de um tipo de anotação

Aqui temos os passos necessários na definição de um tipo de anotação:

1. Uma declaração de anotação deve iniciar com @, seguido da palavra reservada "interface" seguido do nome da anotação.
2. A declaração de métodos não deve conter nenhum parâmetro.

3. A declaração de métodos não deve lançar nenhuma exceção (throws).
4. Os tipos retornados pelos métodos deve ser um dos seguintes tipos:
   - tipos primitivos (byte, short, int, char, long, float, double e boolean)
   - String
   - Class
   - enum
   - um array dos tipos acima

Anotações

Existem dois tipos de anotações disponíveis no JDK5:

- **Anotações Simples**: Estes são os tipos básicos disponibilizados no JDK5, que você pode utilizar para anotar seu código, você não pode utiliza-los para a criação de anotações.
- **Meta anotações**: Estas são utilizadas para anotar quando da declaração de uma anotação.

Anotações Simples

Existem somente três tipos de anotações simples que são oferecidas pelo JDK5:

- Override
- Deprecated
- Suppresswarnings

É importante notar que o JDK5 atualmente não tem muitas anotações; ao invés, ele permite que a linguagem Java tenha a habilidade de suportar este recurso. É deixado aos programadores a incumbência de criar outros tipos de anotações.

A anotação Override

Uma anotação override indica que o método anotado necessita ser sobreposto (reimplementado) pois é herdado da superclasse. Se um método com esta anotação não utilizar a assinatura exata para o método definido pela superclasse, o compilador gerará um erro.

Exemplo:

```
public class Test_Override {
  @Override
  public String toString() {
    return super.toString() + “Testanto a anotação: 'Override'”;
  }
}
```

O que acontece se a assinatura do método estiver errada? Por exemplo, se você trocar o nome do método de “toSring” para “tostring” e compilar o código, você verá algo conforme segue:

Exemplo:

```
Test_Override.java:2: method does not override a method from its
                  superclass
  @Override
   ^
1 error
```

A anotação Deprecated

Quando um método marcado com esta anotação for utilizado em um programa, o compilador deve avisar você sobre isto.

Exemplo

Primeiro, criaremos uma classe com um método que será marcado com a anotação:

```
public class Test_Deprecated {
   @Deprecated
   public void doSomething() {
      System.out.println("Testando a anotatção: 'Deprecated'");
   }
}
```

Agora, tente chamar o método em outra classe:

```
public class TestAnnotations {
   public static void main(String arg[]) throws Exception {
      new TestAnnotations();
   }

   public TestAnnotations() {
     Test_Deprecated t2=new Test_Deprecated();
     t2.doSomething();
   }
}
```

O método neste exemplo está declarado como “deprecated”. Assim, este método não deveria ser utilizado quando esta classe é instanciada por outra classe. Se você compilar Test_Deprecated.java, nenhuma mensagem será gerada pelo compilador. Mas, se você tentar compilar TesteAnnotations.java onde é utilizado um método marcado, você verá algo conforme segue:

```
Note: TestAnnotations.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
```

Se compilarmos com a opção -Xlint:deprecation teremos:

```
TestAnnotations.java:8: warning: [deprecation] doSomething() in
                  Test_Deprecated has been deprecated
     t2.doSomething();
       ^
```

```
1 warning
```

A anotação Supresswarning

Esta anotação indica ao compilador que as mensagens de 'warning' não deverão ser emitidas para o elemento anotado bem como para todos os seus sub-elementos. Como exemplo, se você utilizar esta anotação no programa TestAnnotations do exemplo anterior, nenhuma mensagem será gerada pelo compilador.

Exemplo:

```
public class TestAnnotations {
   public static void main(String arg[]) throws Exception {
      new TestAnnotations();
   }

   @SuppressWarnings({"deprecation"})
   public TestAnnotations() {
     Test_Deprecated t2=new Test_Deprecated();
     t2.doSomething();
   }
}
```

Neste exemplo, foi suprimida a mensagem de atenção para o método 'doSomething'.

Note: É uma boa idéia utilizar esta anotação nos elementos mais externos onde este é mais efetivo. No entanto, se você quer suprimir a mensagem de atenção em um método particular, você pode anotar o método ao invés da classe.

Meta-Anotações

Meta-anotações, que são conhecidas como anotações de anotações, contém quatro tipos. Que são:

- Target
- Retention
- Documented
- Inherited

A anotação Target

Esta anotação indica o elemento alvo para a classe a qual o tipo de anotação será aplicada. Ele contém os seguintes valores:

- @Target(ElementType.TYPE) – pode ser aplicado a qualquer elemento de uma classe
- @Target(ElementType.FIELD) – pode ser aplicado a um atributo de classe
- @Target(ElementType.PARAMETER) – pode ser aplicado a um parâmetro de um método
- @Target(ElementType.CONTRUCTOR) – pode ser aplicado aos construtores

da classe

- @Target(ElementType.LOCAL_VARIABLE) – pode ser aplicado a variáveis locais de métodos numa classe
- @Target(ElementType.ANNOTATION_TYPE) – indica que o tipo declarado é um tipo de anotação

Exemplo

Primeiro, defina uma anotação chamada de Test_Target com o @Target metadata, conforme segue:

```
import java.lang.annotation.*;


@Target(ElementType.METHOD)
public @interface Test_Target {
   public String doTestTarget();
}
```

Em seguida, crie a classe que usará a anotação Test_Target:

```
public class TestAnnotations {
   public static void main(String arg[]) throws Exception {
      new TestAnnotations().doTestTarget();
   }

   @Test_Target(doTestTarget="Alo Mundo!")
   public void doTestTarget() {
     System.out.println("Teste da anotação Target");
   }
}
```

O @Target(ElementType.METHOD) indica que este tipo de anotação pode ser utilizado somente para anotar métodos. Se o código acima for compilado, não será mostrada nenhuma mensagem do compilador. Agora, se você declarar uma variável String e aplicar esta anotação, o que acontecerá?

```
public class TestAnnotations {
   @Test_Target(doTestTarget="Alo Mundo!")
   private String str;

   public static void main(String arg[]) throws Exception {
      new TestAnnotations().doTestTarget();
   }

   public void doTestTarget() {
     System.out.println("Teste da anotação Target");
```

```
    }
}
```

A única mudança que você pode notar acima é que a declaração da anotação foi deslocada para ficar acima da declaração do atributo 'private String str', o que não está correto. Por que a anotação Test_Target foi definida para ser aplicada somente em declarações de métodos, se você tentar compilar esta classe, serão exibidas as mensagens conforme segue:

```
TestAnnotations.java:2: annotation type not applicable to this
                  kind of declaration
   @Test_Target(doTestTarget="Alo Mundo!")
                              ^
1 error
```

A anotação Retention

A anotação retention indica onde e por quanto tempo as anotações de um determinado tipo serão retidas. Existem três valores possíveis:

- RetentionPolicy.SOURCE – Anotações deste tipo serão retidas somente no fonte da classe e será ignorada pelo compilador.
- RetentionPolicy.CLASS – Anotações deste tipo serão retidas até o tempo de compilação, mas serão ignoradas quando em execução na VM.
- RetentionPolicy.RUNTIME – Anotações deste tipo serão retidas pela VM assim elas poderão ser lidas (somente) em tempo de execução.

Exemplo:

```
@Retention(RetentionPolicy.RUNTIME)
public @interface Test_Retention {
  String doTestRetention();
}
```

Neste exemplo, a anotação @Retention(RetentionPolicy.RUNTIME) indica que a anotação Test_Retention será retida pela VM assim esta pode ser acessada em momento de execução pelos mecanismos de “reflexão” do Java.

A anotação Documented

A anotação documented indica que uma anotação com este tipo deverá ser documentada pela ferramenta javadoc. Por default, anotações não são incluídas no processamento do javadoc. Mas se for utilizado @Documented, esta será processada pelas ferramentas javadoc e as informações do tipo da anotação também serão incluídas no documento gerado.

Exemplo:

```
@Documented
public @interface Test_Documented {
   String doTestDocument();
}
```

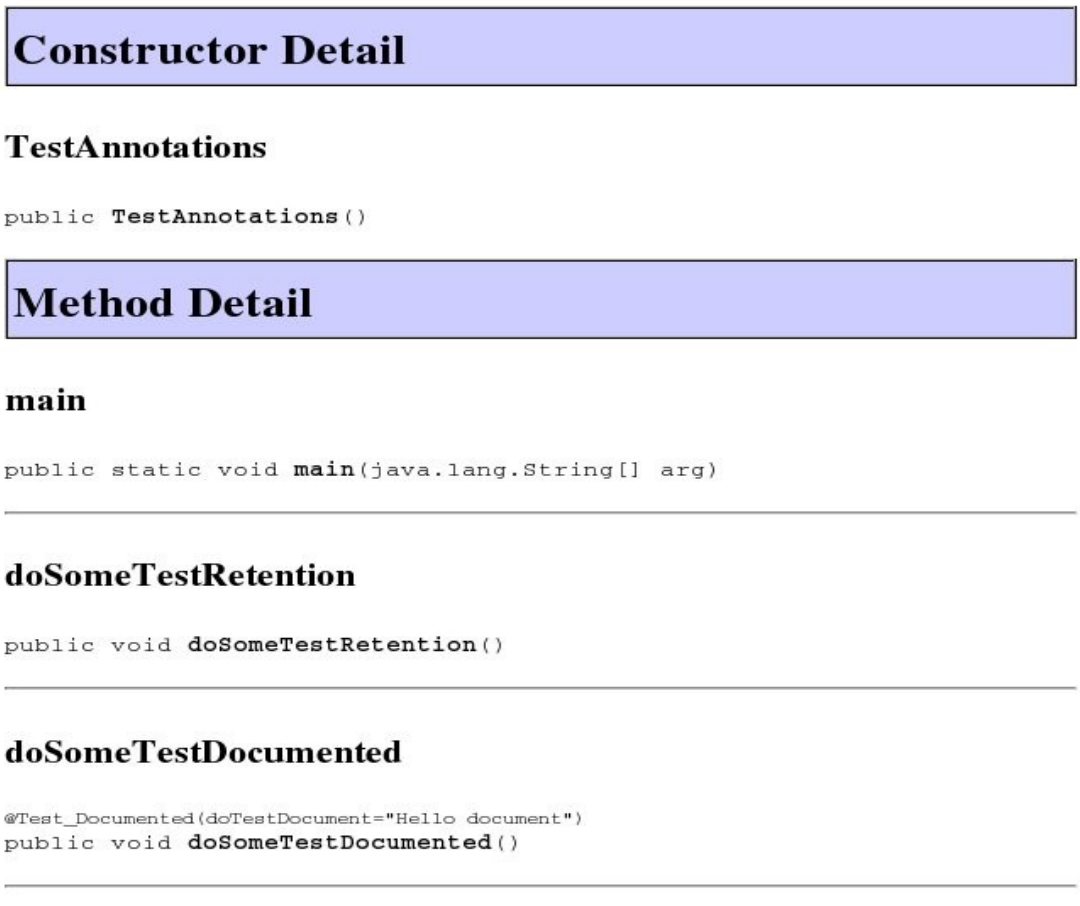
Agora que criamos uma anotação documentada vamos implementar uma classe utilizando esta anotação:

```
public class TestAnnotations {
   public static void main(String arg[]) {
      new TestAnnotations().doSomeTestRetention();
      new TestAnnotations().doSomeTestDocumented();
   }

   @Test_Retention (doTestRetention="Hello retention test")
   public void doSomeTestRetention() {
      System.out.printf("Testing annotation 'Retention'");
   }

   @Test_Documented(doTestDocument="Hello document")
   public void doSomeTestDocumented() {
      System.out.printf("Testing annotation 'Documented'");
   }
}
```

Abaixo temos o resultado gerado pela ferramenta javadoc:



Constructor Detail

TestAnnotations

```
public TestAnnotations()
```

Method Detail

main

```
public static void main(java.lang.String[] arg)
```

---

doSomeTestRetention

```
public void doSomeTestRetention()
```

---

doSomeTestDocumented

```
@Test_Documented(doTestDocument="Hello document")
public void doSomeTestDocumented()
```

Como você pode ver acima, temos um pedaço da tela apresentada por um navegador web, nele notamos que não há informações para o método doSomeTestRetention(). Mas, existe a descrição para o método doSomeTestDocumented(). Isto porque temos uma anotação @Documented anexada a Test_Documented, e não para Test_Retention.

A anotação Inherited

Uma classe anotada com um tipo de anotação marcada como inherited herda automaticamente todas as propriedades desta.

Exemplo

Primeiro, defina sua anotação:

```
import java.lang.annotation.*;


@Inherited
public @interface MyAnnotation {
  boolean isInherited() default true;
  String doSomething() default "Do what?";
}
```

Depois, anote uma classe com sua anotação:

```
@MyAnnotation
public class MyObject {
}
```

Ao definir uma anotação "MyAnnotation" como @Inherited e utilizando esta anotação na classe "MyObject", faz com que todas as sub-classes de "MyObject" herdem automaticamente a anotação "MyAnnotation".