Reflection e Annotations

Instituto Federal do Espirito Santo – Ifes
Campus Colatina – Programação Orientada a Objetos I
Professor: Giovany Frossard Teixeira

Reflection

- O pacote: javax.reflection consiste num conjunto de classes que permite a chamada meta programação, ou seja, é possível obter a classe de um objeto, seus métodos, atributos, etc.
  - Com reflection podemos instanciar um objeto de uma classe que só vamos conhecer em tempo de execução.
  - Também é possível invocar um método dinamicamente.
  - Por fim, permite também a captura de annotations que podem potencializar ainda mais a criação de comportamentos dinâmicos.

Alguns métodos

```
Pessoa p1 = new Pessoa("A", 1, 1.0);
System.out.println("----------------------------------");
Class<Pessoa> classe = (Class<Pessoa>) p1.getClass();
System.out.println("Nome da classe: " + classe.getName());
System.out.println("----------------------------------");
System.out.println("Atributos: ");
for (Field atributo : classe.getDeclaredFields()) {
    System.out.println(atributo.getName());
}
System.out.println("----------------------------------");
System.out.println("Métodos: ");
for (Method metodo : classe.getMethods()) {
    System.out.println(metodo.getName());
}
System.out.println("----------------------------------");
```

Annotations

- Annotations são metadados que podem ser utilizados para parametrizar comportamentos dinâmicos.
- Os annotations *@Deprecated, @Override, @Test* são anotações que servem muito mais como advertência ou para uma melhor organização do código, entretanto sua presença não influencia no comportamento da classe.
- Annotations podem ser muito mais poderosas que isso. No exemplo a seguir criaremos uma Annotation para implementar um Comparador Genérico para listas. Isso vai deixar a ordenação muito mais fácil e dinâmica para que utilizar a anotação criada.

Annotations – criando

```
import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface Ordenacao {
    int ordem();
}
```

Essa anotação vai afetar um método

Método/atributo que será usado na anotação

Classe Pessoa usando a anotação

```
public class Pessoa {
    private String nome;
    private int idade;
    private double altura;

    @Ordenacao(ordem = 1)
    public String getNome() {
        return nome;
    }

    @Ordenacao(ordem = 2)
    public int getIdade() {
        return idade;
    }

    @Ordenacao(ordem = 3)
    public double getAltura() {
        return altura;
    }
....
}
```

Primeiro critério

Segundo critério (empate no primeiro)

Terceiro critério (empate nos 2 anteriores)

Usando o ComparadorGenerico

```
   public static void main(String[] args) {
      ArrayList<Pessoa> pessoas = new
       ArrayList(10);
      Pessoa p;
      pessoas.add(new Pessoa("Pedro", 10,
5.0));
      pessoas.add(new Pessoa("Medro", 5, 2.3));
      pessoas.add(new Pessoa("Ane", 20, 3.5));
      pessoas.add(new Pessoa("Zana", 30, 4.5));
      pessoas.add(new Pessoa("Tina", 15, 7.2));
      pessoas.add(new Pessoa("Pedro", 5, 7.0));
      pessoas.add(new Pessoa("Pedro", 5, 2.0));

      pessoas.add(new Pessoa("Pedro", 7, 2.0));
      pessoas.add(new Pessoa("Medro", 5, 2.0));
```

```
Pessoa{ nome = Ane idade = 20 altura = 3.5 }
Pessoa{ nome = Medro idade = 5 altura = 2.0 }
Pessoa{ nome = Medro idade = 5 altura = 2.3 }
Pessoa{ nome = Pedro idade = 5 altura = 2.0 }
Pessoa{ nome = Pedro idade = 5 altura = 7.0 }
Pessoa{ nome = Pedro idade = 7 altura = 2.0 }
Pessoa{ nome = Pedro idade = 10 altura = 5.0 }
Pessoa{ nome = Tina idade = 15 altura = 7.2 }
Pessoa{ nome = Zana idade = 30 altura = 4.5 }
```

```
      Collections.sort(pessoas,  new
      ComparadorGenerico());
      Iterator<Pessoa> it = pessoas.iterator();
      while (it.hasNext())  {
         p = it.next();
         System.out.println(p);
      }

   }
```

toString em Pessoa:

```
   @Override
   public String toString() {
      return "Pessoa{ " + "nome = " + nome + "
      idade = " + idade  + " altura = " + altura +
      " }";
   }
```

ComparadorGenerico

```
public class ComparadorGenerico  implements Comparator{
    private Method obterMetodo(Class classe, int ordem){
        for (Method metodo : classe.getDeclaredMethods()) {
            if (metodo.isAnnotationPresent(Ordenacao.class)) {
                Ordenacao ordenacao = metodo.getAnnotation(Ordenacao.class);
                if (ordenacao.ordem() == ordem)
                    return metodo;
            }
        }
        return null;
    }
    private ArrayList<Ordenacao> obterArrayAnotacoesMetodos(Class classe){
        ArrayList<Ordenacao> criterios = new ArrayList();
        for (Method metodo : classe.getDeclaredMethods()) {
            if (metodo.isAnnotationPresent(Ordenacao.class)) {
                Ordenacao ordenacao = metodo.getAnnotation(Ordenacao.class);
                criterios.add(ordenacao);
            }
        }
        return criterios;
    }
```

ComparadorGenerico

```
@Override
public int compare(Object t1, Object t2) {
    try{
        ArrayList<Ordenacao> criterios = this.obterArrayAnotacoesMetodos(t1.getClass());
        int ordem=1;
        for (int i=0; i<criterios.size(); i++){
            for(Ordenacao ordenacao : criterios){
                if (ordenacao.ordem() == ordem){
                    int comparacao=0;
                    Method metodo = this.obterMetodo(t1.getClass(), ordem);
                    if (metodo.getReturnType().equals(String.class)){
                        String s1 = (String)metodo.invoke(t1);
                        String s2 = (String)metodo.invoke(t2);
                        comparacao = s1.compareTo(s2);
                    } else if (metodo.getReturnType().equals(int.class)) {
                        int n1 = (int)metodo.invoke(t1);
                        int n2 = (int)metodo.invoke(t2);
                        if (n1 > n2) comparacao = 1;
                        else if (n1 < n2) comparacao = -1;
                        else comparacao = 0;
                    } else if (metodo.getReturnType().equals(double.class)) {
                        double n1 = (double)metodo.invoke(t1);
                        double n2 = (double)metodo.invoke(t2);
                        if (n1 > n2) comparacao = 1;
                        else if (n1 < n2) comparacao = -1;
                        else comparacao = 0;
                    }
                    if (comparacao != 0) return comparacao;
                    // em caso de empate
                    else ordem++;
                }
            }
        }
    } catch (Exception erro){
        System.out.println(erro.getMessage());
    }
    return 0;
}
```

Caso um dos objetos seja maior acabou a comparação

Se houve empate é necessário avançar para o próximo critério (ou ordem).
1, 2, 3, 4 … n

Dúvidas?