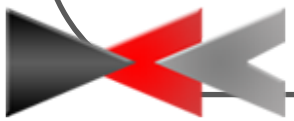
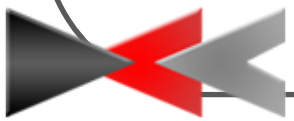
Mais sobre persistência...

Mapeamento objeto/relacional



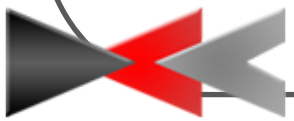
ORM

- Persistência automatizada e transparente de objetos de um aplicativo (Java) para as tabelas em um banco de dados relacionais
  - Utilização de metadados que descrevem o mapeamento entre os objetos e o banco de dados
- Frameworks
  - Hibernate
  - OJB
  - Torque
  - Castor
  - Cayenne etc.



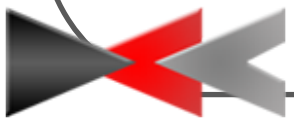
Paradigmas Distintos

- Produtividade:
  - Elimina (ou ao menos diminui) a necessidade de escrever código SQL
- Manutenção:
  - Diminui a quantidade de código associada ao uso de banco de dados
- Desempenho:
  - Construção do framework pode otimizar aspectos desconhecidos para muitos programadores
- Independência:
  - Geração automática de código SQL específico do SGBD



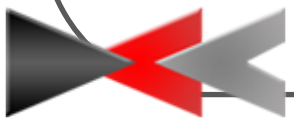
ORM

- Banco de Dados Relacional
  - Sucesso no armazenamento de dados
- Banco de Dados Orientado a Objetos
  - Facilidade na especificação do conteúdo
  - Não alcançou o sucesso das bases relacionais
- Solução:
  - Mapeamento Objeto Relacional



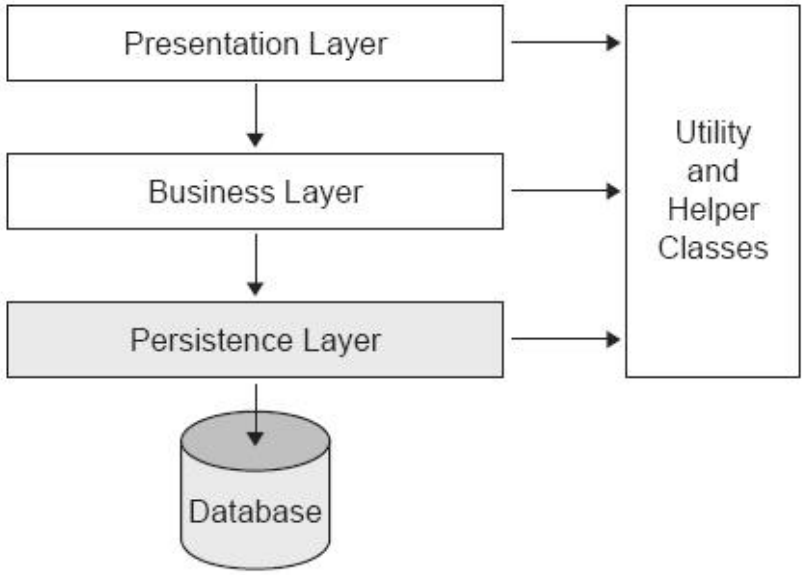
JPA

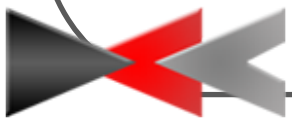
- JPA (*Java Persistence API*)
  - especificação padrão para o gerenciamento de persistência e mapeamento objeto relacional (ORM)
  - surgiu na plataforma Java EE 5.0 (Java *Enterprise Edition*) na especificação do EJB 3.0 (*Enterprise Java Beans* 3.0).
  - substitui os *Entity Beans* (que foram descontinuados) para simplificar o desenvolvimento de aplicações que utilizem persistência de dados.
- Completa especificação para realizar mapeamento objeto relacional
  - Utiliza anotações da linguagem Java (JSE 5.0 ou superior).
  - Oferece suporte a uma linguagem de consulta, semelhante à SQL, permitindo consultas estáticas ou dinâmicas.



Camada de Persistência

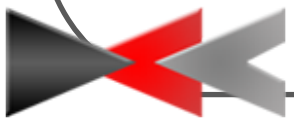
- JPA
  - Framework da camada de persistência
  - Maior produtividade através do mapeamento objeto relacional





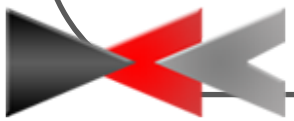
JPA

- Mapeamento:
  - Entity Beans passam a ser apenas Entity (Entidade)
  - Não é apenas uma questão de nomenclatura, pois as entidades são simplificadas
  - POJOS: Plain Old Java Objects
  - Esses objetos são mapeandos para a base relacional através de metadados



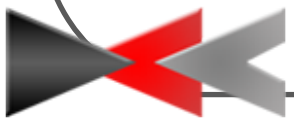
POJO x Java Beans

- POJO (Plain Old Java Object)
  - Não deriva de nenhuma interface ou classe
  - Contém atributos privados
  - Um construtor padrão sem argumentos
  - Métodos publicos get/set para acessar os seus atributos.
- Criada com a intenção de defender designs mais simples em contraposição aos diferentes frameworks
  - Em oposição especial ao Enterprise Java Beans anterior
  - Parte do EJB 3.0



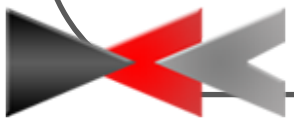
Java Beans

- Objetos de Interface reutilizáveis
  - Componentes que possam ser manipulados visualmente por IDEs (Eclipse, Netbeans etc) para a construção de aplicações
- Hoje em dia utilizado usualmente para descrever objetos simples com get/set
  - Não implicam na criação de interfaces



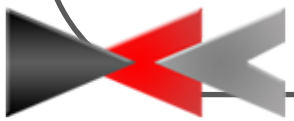
Java Beans

- Convenções
  - Construtor padrão sem argumentos (instanciação mais simples e por várias aplicações)
  - Atributos privados e métodos de acessos públicos get/set para acessar os atributos
  - Deve ser serializada, implementando a interface Serializable do pacote java.io.



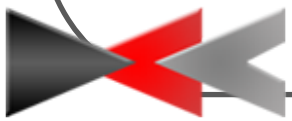
Java Beans

```
public class PessoaBean implements java.io.Serializable{
  private String  nome;
  private boolean homem;

  public PessoaBean() { }

  public String getNome() { return this.nome; }
  public void setNome(String name) { this.nome = nome; }

  public boolean isHomem() { return this.homem; }
  public void    setHomem(boolean homem) {
                    this.homem = homem; }
}
```



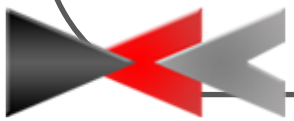
Java Beans

```
public class JavaBeans {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        PessoaBean pessoa = new PessoaBean();
            pessoa.setNome("Joao");
            pessoa.setHomem(true);
            System.out.print(pessoa.getNome());
            System.out.println(pessoa.isHomem() ?
                    " [homem]" : " [mulher]");
    }

}
```



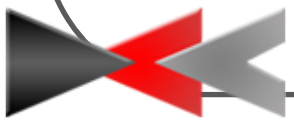
JPA

- Entidade
  - Corresponde a um objeto que pode ser gravado na base de dados a partir de um mecanismo de persistência.
- POJO
  - Contém uma chave primária (atributo identificador)
  - Segue as convenções básicas JavaBeans
- JPA
  - Define o mapeamento das entidades para o banco de dados



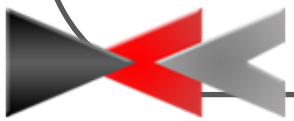
Java Annotations

- Anotações
  - forma especial de declaração de metadados que podem ser adicionadas ao código-fonte
  - Provêem informações sobre o comportamento de um programa
  - Aplicáveis à classes, métodos, atributos e outros elementos de um programa
  - Não tem efeito sobre a execução do código
  - São reflexivas, isto é, podem ser consultadas durante a execução pela JVM (byte-codes)



Java Annotations

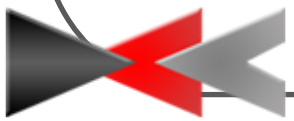
- Aplicações
  - Instruções para o compilador – detecção de erros e supressão de warnings
  - Instruções para outros geradores – usados para ampliação do código, documentação, construção de arquivos xml etc.
  - Instruções em tempo de execução (runtime) – criação de informações que podem ser consultadas durante a execução



Java Annotations

- Exemplo

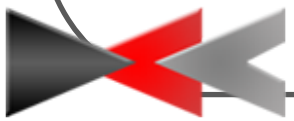
```
public class Funcionario {
    protected double salario;

    public double getSalarioTotal(double bonus) {
        return this.salario + bonus;
    }


    class Auxiliar extends Funcionario {
       protected double extra;

       public double getSalarioTotal(float bonus) {
        return this.salario + this.extra + bonus;
       }
    }
}
```



Java Annotations

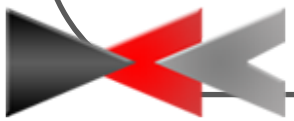
- Exemplo

```
public class Funcionario {
    protected double salario;

    public double getSalarioTotal(double bonus) {
        return this.salario + bonus;
    }


    class Auxiliar extends Funcionario {
       protected double extra;
       @Override
       public double getSalarioTotal(float bonus) {
        return this.salario + this.extra + bonus;
       }
    }
}
```



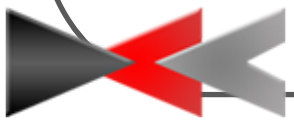
Java Annotations

- Símbolo arroba (@) seguido pelo nome
- Três categorias principais:
  - Anotações Marcadoras: não possuem membros
    - @Test, @Override etc.
  - Anotações de valor único: possuem um único membro, chamado valor, passado entre parênteses
    - @SuppressWarnings(“unchecked”)
  - Anotações completas: possuem múltiplos membros. Sintaxe nome=valor
    - @SuppressWarnings(value=“unchecked”)



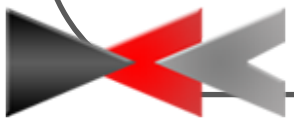
Java Annotations

- Podem ser definidas pelo desenvolvedor
- Java possui um amplo conjunto de pré-definidas
  - java.lang – não precisam ser importadas
  - @Override – usada apenas com métodos, serve para indicar que o método anotado está sobrescrevendo um método da superclasse
  - @Deprecated – usada para indicar que um método não deveria mais ser utilizado. Aplicado na assinatura do método
  - @SuppressWarnings – permite desligar os altertas de uma parte do código da aplicação – classe, método ou inicialização de variável. Existem dezenas de possíveis valores.



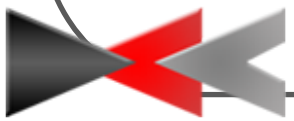
Java Annotations

- Java possui um amplo conjunto de pré-definidas
  - @Documented – indica que os tipos de anotação utilizados serão incluídos na documentação Javadoc
  - @Inherited – indica a necessitade de herança das anotações entre as classes (anotações não são, por padrão, herdadas).
  - ...



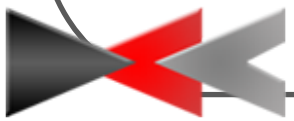
Java Annotations

- Java possui um amplo conjunto de pré-definidas
  - @Documented – indica que os tipos de anotação utilizados serão incluídos na documentação Javadoc
  - @Inherited – indica a necessitade de herança das anotações entre as classes (anotações não são, por padrão, herdadas).
  - @Retention: indica onde e quanto tempo a anotação será considerada
  - ...



Java Annotations

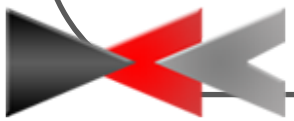
- Anotações padrões do Java não atenderão a todas as necessidades
  - Podem ser construídas outras anotações (usuário, frameworks etc)
- Anotações
  - java.lang.annotation.Annotation
  - @interface



TODO

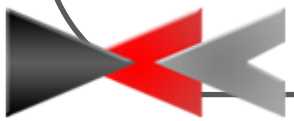
- Exemplo

```
public @interface TODO {
    String value();
}
```



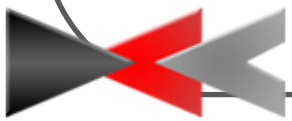
Funcionario

```java
public class Funcionario {
    protected double salario;

    @SuppressWarnings("unchecked")
    public double getSalarioTotal(double bonus) {
        return this.salario + bonus;
    }

    @SuppressWarnings(value = "unchecked")
    class Auxiliar extends Funcionario {
       protected double extra;

       @TODO("O salário total do funcionário = salário + bonus")
       @Override
       public double getSalarioTotal(double bonus) {
        return this.salario + this.extra + bonus;
       }
    }
}
```



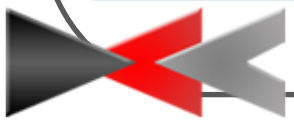
TODO

```
public @interface TODO {

    public enum Severity
            { CRITICAL, IMPORTANT, TRIVIAL, DOCUMENTATION };

    Severity severity( ) default Severity.IMPORTANT;

    String item( );

    String assignedTo( );

}
```



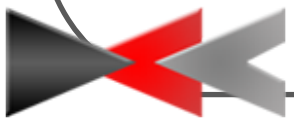
Funcionario

```
public class Funcionario {
    protected double salario;

    @SuppressWarnings("unchecked")
    public double getSalarioTotal(double bonus) {
        return this.salario + bonus;
    }

    @SuppressWarnings(value = "unchecked")
    class Auxiliar extends Funcionario {
       protected double extra;

       @TODO(severity=TODO.Severity.TRIVIAL,
               item="O salário total do funcionário = salário + bonus",
               assignedTo="Carlos Araújo" )
       @Override
       public double getSalarioTotal(double bonus) {
        return this.salario + this.extra + bonus;
       }
    }
}
```



Reflexão

- Anotações não são apenas para programadores ou para verificação de código
  - Depuradores, frameworks e outros empregados em tempo de execução.
- Pacote java.lang.reflect
- Utilizando essa API é possível
  - Determinar a classe de um objeto
  - Descobrir constantes e declarações de métodos
  - Criar uma instância de uma classe
  - Obter e especificar o valor de um campo de objeto
  - Invocar um método de um objeto



TesteAnotacao

```
public class TesteAnotacao {
    @Deprecated public static int value = 1;

    public static void main(String[] args) throws Exception {
        Field field = TesteAnotacao.class.getField ("value");
        if (field.isAnnotationPresent (Deprecated.class)) {
            System.out.println ("Campo anotado com Deprecated");
        } else {
            System.out.println ("Campo não anotado com Deprecated");
        }
    }
}
```