# Java

## beans


# O que é um JavaBean e quais convenções ele segue?

Um JavaBean é uma classe Java que adota um conjunto de convenções para que possa ser manipulada facilmente por ferramentas e frameworks.

Ele precisa ter um construtor público sem argumentos, expor suas propriedades através de métodos getters e setters padronizados e, quando necessário, notificar ou restringir alterações via eventos.

É comum também implementar Serializable para persistência. Essas regras permitem que o bean seja analisado em tempo de execução por meio de introspecção.


# Para que servem PropertyDescriptor, EventSetDescriptor e MethodDescriptor?

Esses descritores são usados para representar metadados de um bean.

O PropertyDescriptor descreve propriedades, apontando seus métodos de leitura e escrita.

O EventSetDescriptor descreve eventos que o bean pode disparar e como listeners podem ser adicionados ou removidos.

Já o MethodDescriptor descreve métodos individuais. Essas estruturas tornam possível que editores visuais e bibliotecas trabalhem com beans sem conhecer detalhes internos da implementação.


# O que é uma propriedade bound em um bean?

Uma propriedade bound é aquela que notifica ouvintes quando sofre alteração. Para implementar, utiliza-se a classe de apoio PropertyChangeSupport, que dispara eventos sempre que o valor mudar. Exemplo:

```
public class Person {
    private String name;
    private final PropertyChangeSupport pcs = new
PropertyChangeSupport(this);

    public String getName() { return name; }
    public void setName(String name) {
        String old = this.name;
        this.name = name;
        pcs.firePropertyChange("name", old, name);
    }
}
```


# Como funciona uma propriedade constrained?

Uma propriedade constrained é semelhante a uma bound, mas com a diferença de que ouvintes podem vetar a mudança. Isso é feito com VetoableChangeSupport.

Ao tentar alterar o valor, dispara-se fireVetoableChange, e se algum listener lançar PropertyVetoException, a modificação não acontece. Esse padrão é útil quando a alteração de uma propriedade depende de regras de negócio que precisam ser respeitadas antes da confirmação.


# Como usar o Introspector sem trazer propriedades herdadas de Object?

Se você usar Introspector.getBeanInfo, por padrão ele mostra também métodos herdados de Object (como getClass). Para evitar isso, basta passar Object.class como limite. Assim você vê só as propriedades que realmente interessam.

```
BeanInfo info = Introspector.getBeanInfo(MyBean.class,
Object.class);
for (PropertyDescriptor pd : info.getPropertyDescriptors()) {
    System.out.println(pd.getName());
}
```


# Qual a função de um PropertyEditor?

Um PropertyEditor serve para converter valores de propriedades entre objetos e representações em texto. Isso é muito útil em ferramentas de design que trabalham com valores textuais.

A forma mais prática é estender PropertyEditorSupport e sobrescrever os métodos setAsText e getAsText. Depois, o editor pode ser registrado no PropertyEditorManager. Esse mecanismo facilita a edição de propriedades complexas em ambientes que só manipulam strings.


# Como criar e usar um PropertyEditor para converter texto em objeto?

Um **PropertyEditor** é útil quando você quer transformar texto em objeto automaticamente. Veja um exemplo simples para inteiros:

```
public class IntEditor extends PropertyEditorSupport {
    @Override public void setAsText(String text) {
setValue(Integer.parseInt(text)); }
}

PropertyEditor ed = new IntEditor();
ed.setAsText("42");
System.out.println(ed.getValue()); // imprime 42
```


# Como usar @Transient para ignorar uma propriedade?

**Às vezes você não quer que uma propriedade apareça em introspecção ou seja gravada em XML. Para isso existe a anotação @Transient.**

```
public class User {
    private String password;

    @Transient
    public String getPassword() { return password; }
}
```

**Assim, password é ignorado em persistência e introspecção, mesmo sendo uma propriedade válida no bean.**

Java

luisfabriciodellamas