02

Gerando conteúdo eficientemente

Nossa classe `Principal` foi apenas o começo. Partiremos de um projeto com a finalidade de registrar os gastos mensais de cada funcionário e, portanto, iniciaremos por uma classe simples de modelo que representa um `Gasto` em nosso sistema.

Crie uma classe sem ajuda do mouse usando ctrl + N e começando a digitar Class na janela - duas letras devem ser suficientes. Selecione a opção Class com as setinhas direcionais e dê enter.

Escreva o nome da classe `Gasto` e, então, com ajuda do shift + tab navegue para o campo Package e o preencha com `br.com.caelum.empresa.modelo` . Desse campo mesmo, aperte enter para confirmar a criação dessa classe.

Então, nela, crie os atributos necessários, sem se preocupar com a visibilidade deles ainda. Não se esqueça de usar o ctrl+espaço para que os tipos sejam importados automaticamente.Você terá algo como:

```
public class Gasto {
    double valor;
    String tipo;
    Funcionario funcionario;
    Calendar data;
}
```

Quando aprendemos orientação a objetos, um dos conceitos mais importantes é o encapsulamento, ou seja, não expor os dados internos da classe. A forma mais direta de fazer isso é deixando os atributos privados. Para fazer isso poderíamos ir linha por linha adicionando a palavra chave `private` , mas esse é um trabalho repetitivo. Como fazer para evitar esse trabalho?

Uma das maneiras é usar um recurso chamado "Seleção em bloco" onde, em vez de selecionar linha a linha, selecionamos por coluna. Portanto, podemos habilitar o modo de seleção em bloco para adicionar o `private` em todas as linhas de uma vez só. Faça alt + shift + A e selecione (shift e seta para baixo, sem o mouse) o início das linhas dos atributos, então digite `private` e ele copiará tal palavra em cada linha selecionada.

Repare que se o cursor do mouse estiver sobre a área do editor, quando o modo de seleção em bloco for selecionado o cursor do mouse se transforma na cruz de seleção de área.

Lembre-se de salvar as modificações a todo momento (ctrl + S), e de desabilitar o modo de seleção em bloco apertando alt + shift + A novamente após finalizar o trabalho.

```
public class Gasto {
    private double valor;
    private String tipo;
    private Funcionario funcionario;
    private Calendar data;

}
```

Nesse momento, o Eclipse nos avisa com warnings que esses atributos nunca são lidos - e, se pensarmos bem, também nunca são preenchidos. Se quisermos mostrar os dados de um objeto dessa classe ou preencher esses dados usando um formulário em, por exemplo, uma página da web precisamos permitir esse acesso de alguma maneira. Seguindo a convenção do java, o comum é usar [getters e setters (http://blog.caelum.com.br/nao-aprender-oo-getters-e-setters/)](http://blog.caelum.com.br/nao-aprender-oo-getters-e-setters/) para expor esse acesso e, talvez, um construtor para já preencher os valores obrigatórios.

Já que gerar getters, setters e construtores faz parte do dia a dia de qualquer desenvolvedor Java, nada mais justo do que gerá-los automaticamente e garantir que estamos usando os nomes e os tipos corretos. Para nossa sorte, basta conhecer mais um comando para efetuar essas ações (e muitas outras). O ctrl + 3 é o atalho mais versátil do Eclipse: ele chama qualquer menu por uma parte de seu nome ou suas iniciais.

Existe um item do menu para gerar os getters e setters chamado generate getters and setters. Para acessá-lo rapidamente basta fazer ctrl + 3 e começar a digitar qualquer parte do nome desse menu ou as suas iniciais. Por exemplo, você pode fazer ctrl + 3 getter (ou as iniciais ggas) e ele trará o menu correto.

Alternativamente você pode utilizar Alt + Shift + S e depois R para abrir o gerador de get/set, Alt + Shift + S e depois S para gerar o toString com os atributos da classe, Alt + Shift + S e depois E para gerar o equals e o hashcode.

Ao acessar o menu, vai aparecer uma tela onde você pode selecionar quais serão os getters e setters desejados. Podemos usar as setas para cima e para baixo, e a barra de espaço para marcar os checkboxes. Se apertarmos + em um dos atributos podemos selecionar só o getter ou só o setter. Com o tab conseguimos navegar pelos botões e outras configurações dessa tela. Ao apertar enter os getters e setters selecionados serão criados automaticamente.

Mas ainda há diversas situações em que, para que a classe funcione, ela precisa que alguns de seus atributos estejam preenchidos. A essas informações necessárias damos o nome de dependências (http://blog.caelum.com.br/singletons-e-static-perigo-a-vista/) e, para forçar que elas estejam presentes desde a criação do objeto, indicamos que na criação do objeto dessa classe, vamos precisar de tais informações. Isto é, faremos um construtor que recebe nossas dependências.

O construtor também pode ser criado através de ctrl + 3 constructor ou com as iniciais gcuf, já que o nome do menu é generate constructor using fields. Uma tela bem parecida com a dos getters/setters vai aparecer para que você selecione os atributos que devem ser recebidos no construtor. Ao apertar enter o construtor solicitado será gerado automaticamente.

O conceito da injeção de dependências (http://www.martinfowler.com/articles/injection.html) é bastante importante e vale a pena conhecê-lo mais já que ele é cada vez mais usado nos frameworks que utilizamos como uma forma de reduzir o acoplamento de um sistema. Veja mais sobre isso no curso FJ-21 (https://www.caelum.com.br/curso/fj21)

Agora que já temos nossa classes com getters, setters e construtor queremos que a impressão de um Gasto mostre a qual funcionário ele se refere e o valor.

Como qualquer outra ação comum, há um menu para sobrescrever métodos. O override/implement methods mostra a lista de métodos que podem ser sobrescritos na classe e o que ele faz é colocar a assinatura completa do método para que nós apenas o implementemos.

Perceba que, em uma breve distração, poderíamos passar um parâmetro para esse método, fazendo uma sobrecarga acidental em vez da sobrescrita. Mas como é o `sysout` que chama o método `toString`, o comportamento seria exatamente o que não queríamos.

Isto é, se agora precisarmos imprimir a data só em algumas situações, o natural seria fazer a sobrecarga acidental:

```
public String toString(boolean mostraData) {
    String mensagem = funcionario + "\n\tGasto: " + valor;
    if(mostraData) {
        mensagem += "\n\tData: " + formatador.format(data.getTime()));
    }
    return mensagem;
}
```

Mas quando chamarmos o `sysout` ainda teremos a saída no Console com a implementação original:

```
br.com.caelum.controle.modelo.Gasto@c9ba38
br.com.caelum.controle.modelo.Gasto@1e0be38
br.com.caelum.controle.modelo.Gasto@1e859c0
```

A vantagem principal em sempre usar o atalho para sobrescrever um método é que, dessa forma, não corremos o risco de fazer uma sobrecarga em vez de uma sobrescrita - ou mesmo algo mais estranho como colocar o tipo errado de retorno e perder tempo procurando a razão do erro de compilação.

Apesar da existência de um menu, para sobrescrever um método, nossa preferência é por começar a digitar o nome do método e pedir que o autocomplete escreva a assinatura correta para nós. Então, se na classe `Gasto` fôssemos sobrescrever o `toString`, faríamos, nessa classe: `toS`, escolheríamos a primeira opção (Override method in 'Object') e o código gerado seria algo como:

```
@Override
public String toString() {
    // TODO Auto-generated method stub
    return super.toString();
}
```

Então, basta apagar as linhas não interessantes para nós e substituir pela sua implementação.

Para testar o funcionamento do `toString` para alguns `Gasto` s e usando os conhecimentos deste capítulo e do anterior, você consegue, apenas através de atalhos, chegar ao resultado:

```
public static void main(String[] args) {
    Calendar hoje = Calendar.getInstance();
    GregorianCalendar nascimento = new GregorianCalendar(1989, 3, 14);
    Funcionario funcionario = new Funcionario("Vinícius", 9, nascimento);

    Gasto gasto = new Gasto(40.0, "taxi", funcionario, hoje);

    System.out.println(gasto);
}
```

Mas a idéia era imprimir diversos gastos para ver como o `toString` se comporta. Precisaremos, então copiar e colar diversas vezes, tanto a linha que cria os gastos quanto a do `sysout` .

O velho ctrl + C e ctrl + V resolvem o problema, mas ainda seria preciso selecionar a linha para copiar. Há um atalho para replicar linhas que não exige que ela seja selecionada e copiada antes: o ctrl + alt + baixo copia a linha atual para a de baixo. Ele é ideal para as tarefas repetitivas, frequentes em testes.

```
public static void main(String[] args) {
    Calendar hoje = Calendar.getInstance();
    GregorianCalendar nascimento = new GregorianCalendar(1989, 3, 14);
    Funcionario funcionario = new Funcionario("Vinícius", 9, nascimento);

    Gasto gasto = new Gasto(40.0, "taxi", funcionario, hoje);
    Gasto gasto = new Gasto(40.0, "taxi", funcionario, hoje);
    Gasto gasto = new Gasto(40.0, "taxi", funcionario, hoje);

    System.out.println(gasto);
    System.out.println(gasto);
    System.out.println(gasto);
}
```

Dois erros de compilação aparecerão por conta das variáveis com nomes iguais. Com o ctrl + . ("control ponto"), o Eclipse leva o cursor até o próximo erro. Então, basta corrigí-lo: no nosso caso, renomear as variáveis. Não se esqueça de alterar os `sysout` s também!

Para saber mais: Atalhos equivalentes no Eclipse versão Mac OS: Seleção em bloco: alt + command + A Replicar linha para cima: alt + command + cima Replicar linha para baixo: alt + command + baixo Criar construtor: command + 3 + gcuf ou command + 3 + constructor Criar getters and setters: command + 3 + ggas ou command + 3 + getter