INF01120 – Técnicas de Construção de Programas

Aplicação e Estudo de Padrões de Implementação

Hélio Carlos Brauner Filho – 180182
Jefferson Rodrigo Stoffel - 180685
João Luiz Grave Gross – 180171

Prof. Marcelo Soares Pimenta

Algumas Notas Importantes

- Para motivar o estudo de padrões de implementação, são apresentados alguns pensamentos importantes:
  - Programadores deveriam se focar em realmente desenvolver soluções para resolver o problema, ao invés de se preocupar com detalhes da implementação;
  - Um programa, enquanto 'vivo', é muito mais lido do que escrito, e, portanto, é melhor escrever um programa claro de ler do que mais rápido de escrever;
  - Um software de qualidade deve permitir fácil manutenção e extensibilidade;
  - Padrões são um grande auxílio para o programador que tem os três pensamentos acima em mente.

Os Padrões

- Dos padrões apresentados pelo texto de Kent Beck, foram escolhidos os seguintes:
  - Classe: Simple Superclass Name e Value Object;
  - Estado: Lazy initialization;
  - Comportamento: Guard Clause;
  - Método: Creation.
- Serão apresentadas algumas informações sobre cada um e, depois, será apresentada uma implementação de um destes padrões, com sua justificativa.

Simple Superclass Name

- Classes são conceitos centrais de design. Por isso, é importante que seus nomes sejam bem escolhidos.
- Uma vez que as classes estiverem bem nomeadas, o nome das operações surge quase que automaticamente.
- A dificuldade em nomear classes está na tensão entre brevidade e expressividade. Os nomes precisam ser curtos e enérgicos. No entanto, às vezes, nomes precisos requerem várias palavras.
- Encontrar bons nomes pode levar tempo.

Value Object

- O conceito principal é “tudo se cria, nada se transforma”;
- Nenhum estado é alterado: apenas novos são criados;
- Interfaces funcionais têm menor preocupação com seqüências de operações;
- Objetos funcionam como valores imutáveis após sua criação, o que garante que os dados não serão alterados até a criação de um novo objeto;
- Operações em objetos sempre retornam novos objetos;
- Contra este padrão, se fala da performance, por sobrecarregar a memória; é também dito difícil de projetar.

Lazy Initialization

- 'Filosofia do Baiano': só se mexer quando algo precisa ser feito. Nesse caso, só inicializar a variável no momento em que ela for utilizada no código;
- Isso é justificado pelo alto custo de algumas inicializações. Se a variável não for utilizada, há na verdade desperdício computacional;
- Por reduzir o desperdício, é ideal para sistemas com baixo poder de processamento ou para aplicações muito pesadas;
- Visa, portanto, melhorar a performance.

Guard Clause

- É uma forma local e simples de expressar situações excepcionais com conseqüências de escopo local;
- Remove 'else's e 'if's encadeados do código. Isto facilita a leitura do código, evitando empilhar mentalmente conjuntos destas cláusulas;
- Mantém simples a estrutura de controle.

Creation

- A ideia é modificar a forma de criação dos programas;
- Os primeiros programas eram massas indecifráveis de código e dados. O fluxo de execução não seguia um padrão lógico e os dados podiam ser acessados de qualquer lugar. Eram rápidos e eficientes;
- O problema surgiu quando foi notado que os programas eram tão modificados quanto executados. A mesma permissividade que os tornava eficientes também os fazia muito difíceis de serem modificados;
- Novos modelos de computação surgiram para evitar o chamado "efeito dominó" no código;

Creation

- Uma das primeiras estratégias para fazer programas facilmente modificáveis foi dividir um computador rodando um grande programa em vários computadores menores rodando programas pequenos. Estes pequenos computadores podem ser representados como objetos;
- Essa subdivisão não afeta em nada a forma da máquina de realizar suas operações. Seu único propósito é facilitar a vida dos humanos e reduzir as chances de falhas.

Implementação: Guard Clause

- Dos cinco padrões que foram escolhidos, optamos pela implementação do padrão de comportamento Guard Clause;
- A escolha ocorreu pela facilidade de aplicar a idéia, e pela rápida percepção de um dos integrantes do grupo de como utilizar esse padrão em um dos códigos que escreveu;
- O trecho do código em questão é responsável por realizar login de usuários em um site, e foi escrito em PHP. Serão apresentadas aqui tanto a versão original, sem guard clause, como a versão com guard clause.

Código sem Guard Clause

```
<?php

//Sem guard clause
if(!empty($username) && !empty($password)) {
     $forumMembersInfo = $forum->getMembersInfo($username);

     if( !empty($forumMembersInfo['username']) &&
          !empty($forumMembersInfo['members_pass_hash']) &&
          !empty($forumMembersInfo['members_pass_salt']) ) {

          $hash = md5( md5( $forumMembersInfo['members_pass_salt'] ) . md5( $password ) );

          if( strcmp($hash, $forumMembersInfo['members_pass_hash']) == 0 ) {
               session_start("seguranca");
               $_SESSION['loggedUser'] = $username;
               echo "<script>document.location='index.php'</script>";
               exit;
          } else {
               echo "<script>alert('".utf8_decode("Usuário")." ou senha incorretos. Tente
novamente.');document.location='login.php'</script>";
               exit;
          }
     } else {
          echo "<script>alert('".utf8_encode("Usuário")." ou senha incorretos. Tente
novamente.');document.location='login.php'</script>";
          exit;
     }
} else {
     echo "<script>alert('Preencha os dois campos.');document.location='login.php'</script>";
     exit;
}
```

Código com Guard Clause

```
//Com Guard Clause
if(empty($username) || empty($password)) {
     echo "<script>alert('Preencha os dois campos.');document.location='login.php'</script>";
     return;
}
$forumMembersInfo = $forum->getMembersInfo($username);

if( empty($forumMembersInfo['username']) ||
     empty($forumMembersInfo['members_pass_hash']) ||
     empty($forumMembersInfo['members_pass_salt']) ) {
     echo "<script>alert('".utf8_encode("Usuário")." ou senha incorretos. Tente
novamente.');document.location='login.php'</script>";
     return;
}

$hash = md5( md5( $forumMembersInfo['members_pass_salt'] ) . md5( $password ) );

if( strcmp($hash, $forumMembersInfo['members_pass_hash']) != 0 ) {
     echo "<script>alert('".utf8_decode("Usuário")." ou senha incorretos. Tente
novamente.');document.location='login.php'</script>";
     exit;
}

session_start("seguranca");
$_SESSION['loggedUser'] = $username;
echo "<script>document.location='index.php'</script>";

?>
```

Considerações finais

- Comparando os dois trechos de código apresentados, foi possível perceber como o código se tornou mais claro;
- Embora tenha sido apresentada a implementação de apenas um padrão, podemos imaginar os benefícios dos outros padrões para a criação de código mais claro e mais fácil de editar;
- A leitura mais aprofundada do texto utilizado e de outros materiais da área contribui largamente para o aprimoramento dos programadores como nós, e, por conseguinte, na formação de melhores profissionais.

Bibliografia

- Implementation Patterns – Beck, Kent – Capítulos 5 a 9 – Conforme disponibilizado em http://moodle.inf.ufrgs.br/mod/resource/view.php?id=31665 - Acessado entre os dias 15 e 24 de setembro