ATIVIDADE FORMATIVA 3

Período de realização: Decorre de 2 a 17 de maio de 2022

Tema: Mundo dos retângulos

Objetivos

- Reutilização de código
- Melhoria e adaptação de código

Descrição da E-atividade

Escreva um programa para manipular um conjunto de retângulos num plano vertical, e sem que haja interseções entre eles. O programa deve aceitar comandos para: criar retângulos com posição e dimensões indicadas; deslocar retângulos horizontalmente, para a esquerda e para a direita. O plano que serve de cenário a este "mundo" baseia-se num sistema de coordenadas inteiras (x=abcissa, y=ordenada), em que x,y >=1, x <= 80 e y <= 25, e e nele funciona a gravidade, isto é, todos os retângulos criados devem "cair" até ao chão (linha y=1), ou até encontrarem o topo de outro retângulo existente.Os comandos a serem identificados pelo programa são:

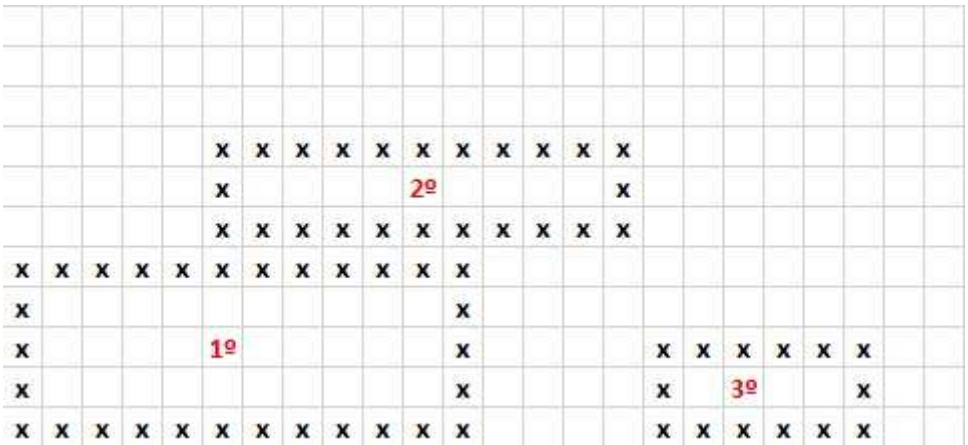
- create x,y+l,h - cria um retângulo em que *(x,y)* são as coordenadas do canto inferior esquerdo e *(l,h)* o comprimento e altura, respetivamente.
- moveright x,y+p - desloca o retângulo situado nas coordenadas *(x,y)* para a direita p posições
- moveleftx,y+p - desloca o retângulo que contém o ponto *(x,y)* para a esquerda p posições

Qualquer destas operações, sempre que for possível (se não for, o programa deve emitir mensagem de erro adequada), implica ajustar todos os retângulos existentes tendo em conta o efeito da gravidade. O resultado dos comandos deverá poder ser visualizado graficamente no ecrã, usando o símbolo x para compor as linhas horizontais e verticais.

Exemplo: A seguinte sequência de comandos:

```
create 1,3+12,5
create 9,6+11,3
create 18,10+6,3
moveleft 12,7+3
```

deverá produzir o resultado:



(os números a vermelho não são para mostrar no ecrã, servem apenas para ilustrar a ordem da respetiva criação no exemplo dado). Na organização do seu programa, sugere-se que considere os seguintes módulos:

- análise do texto dos comandos
- controlo das posições dos retângulos
- visualização do resultado

== METODOLOGIA DE TRABALHO ==

Defina uma estrutura de dados para representar os retângulos e planeie a organização do programa em módulos. Defina a interface de cada módulo, desenvolva o código e teste o seu programa. **Nota:** Não é necessário efetuar a animação dos retângulos a deslocarem-se, mostre apenas a posição final.

Última alteração: Segunda, 2 Maio 2022, 09:29

Dicas de Legibilidade de Código-fonte

Dino Esposito

Você já ouviu falar do Concurso International Obfuscated C Code? Em resumo, é um concurso aberto que seleciona um vencedor de alguns dos programas C que resolve um problema - qualquer problema - com um código C extremamente obscuro e ofuscado. Você pode encontrar o código-fonte dos programas vencedores nos anos passados em ioccc.org/years.html.

O Concurso Obfuscated C Code é uma forma leve de demonstrar a importância do estilo e da legibilidade na programação. Essa coluna resumirá algumas das práticas mais importantes que você vai querer seguir, a fim de ter um código que é fácil de ler e compreender - tanto para seu próprio bem como o de seus colegas.

Legibilidade como um atributo

No desenvolvimento de software, a manutenção é o atributo que se refere à facilidade com que você pode modificar o código existente para atingir metas, tais como corrigir um bug, serviço de limpeza, implementação de um novo recurso ou apenas a refatoração de alguns padrões. A facilidade de manutenção é um dos atributos fundamentais do software, de acordo com o artigo ISO/IEC 9126. Para obter mais informações sobre os atributos de software, consulte o artigo em bit.ly/VCpe9q.

A facilidade de manutenção do código resulta de uma variedade de fatores, um deles é a legibilidade. O código que é difícil de ler também é difícil de compreender. Os desenvolvedores que colocam suas mãos nos códigos não conhecem claramente e compreendem que são suscetíveis a deixar o código ainda pior.

Infelizmente, a legibilidade é um assunto extremamente subjetivo. Desenvolver uma ferramenta automática para verificar e informar sobre o nível de legibilidade do código é virtualmente impossível. No entanto, mesmo se a medição da legibilidade automática fosse possível, qualquer uma dessas ferramentas provavelmente seria considerada altamente pouco confiável e não seria confiável para ninguém. No final, a legibilidade é um atributo manual que os desenvolvedores individuais devem verificar durante todo o seu código. A capacidade de gravar código que seja fácil de ler deve ser parte da responsabilidade cultural dos desenvolvedores individuais, ampliando seu conjunto de habilidades.

De modo geral, a legibilidade é um atributo do código que você pode e deve aprender para adotar logo no começo de sua carreira de programação e desenvolver e melhorar com o tempo. Como estilo e bom design, a legibilidade não deve ser reservada aos especialistas. Mais importante, não deve ser adiada para quando você só tiver tempo suficiente para ela.

Uma abordagem pragmática para a legibilidade

Produzir código legível é uma questão de respeito com os outros desenvolvedores. Como um usuário StackOverflow postou uma vez: “você deve sempre fazer o código como se a pessoa que acaba mantendo o seu código fosse um psicopata violento que conhece onde você mora”. Você também deve considerar que o desenvolvedor que acaba mantendo o seu código, um dia, pode realmente ser você.

Ao ler o código de outras pessoas, existe uma porção de coisas que podem deixá-lo louco. Um aspecto que faz com que seja difícil ler código são as estruturas de dados e os algoritmos sem metas claras. Outro

aspecto são as estratégias que não estão claras no código, que são difíceis de determinar e não são bem anotadas através dos comentários. Veja um exemplo:

XML

```
// Find the smallest number in a list of integers
private int mininList(params int[] numbers)
{
  var min = Int32.MaxValue;
  for (var i = 0; i < numbers.length; i++) {
    int number = numbers[i];
    if (number < min)
      min = number;
  }
  return min;
}
```

Mesmo que eu tenha utilizado um exemplo em C# para clareza, eu tenho que admitir que esta não é uma parte do código que qualquer desenvolvedor em C# jamais consideraria criar. O motivo é que com o C# e o Microsoft .NET Framework, você pode atingir muitos dos mesmos resultados usando o LINQ. Eu encontrei um código semelhante em um projeto, exceto que ele foi escrito em Java. Em algum momento, eu foi contratado para levar a base de código para o .NET Framework e C#.

Você pode também se deparar com código como aquele em um projeto .NET real. Você pode aplicar algumas considerações de legibilidade sem perder sua intenção. A primeira coisa que não funciona nessa função é o nome. A convenção é questionável. O nome perde um verbo e usa uma lógica de maiúsculas e minúsculas mista. Algo como GetMinFromList provavelmente seria um nome melhor. No entanto, o ponto mais discutível do código é o qualificador particular usado no nome.

Qualquer leitor casual desse código pode ver que a função serve como um utilitário. Em outras palavras, ele representa uma parte do código potencialmente reutilizável que você pode chamar de vários locais dentro do resto da base de código. Portanto, marcá-lo como particular nem sempre faz sentido. No entanto, os desenvolvedores sabem do poder da regra YAGNI - You Ain't Gonna Need It (Você não vai precisar dele) - e, razoavelmente, tendem a não expor o código que não seja estritamente necessário.

O autor desse código poderia ter previsto a função como potencialmente reutilizável, mas não quando foi gravado. É por isso que a função foi gravada para ser facilmente transformada em uma função auxiliar reutilizável, mas foi marcada como particular para ser visível somente dentro da classe host. Essa estratégia de codificação pode ser difícil de descobrir imediatamente para leitores externos. No entanto, é apenas uma decisão que requer algumas linhas de comentários para explicar sua motivação. Se você não adicionar comentários adequados, você não estará sendo um bom cidadão no mundo da codificação. Os leitores basicamente veem sentido nele, mas desperdiçam alguns minutos e, pior ainda, deixam os leitores um pouco hostis com o autor.

Regras Práticas de Legibilidade

A legibilidade do código é um dos assuntos em que a importância é amplamente reconhecida, mas não necessariamente formalizada. Ao mesmo tempo, sem alguma formalização, a legibilidade do código é quase um conceito vazio. Em geral, você pode se aproximar da legibilidade com a regra dos três C's: uma função combinada de comentários, consistência e clareza.

As ferramentas IDE são muito mais inteligentes hoje do que há alguns anos. Os desenvolvedores não têm a necessidade estrita de gravar arquivos de ajuda e integrar sua documentação personalizada nos projetos

do Visual Studio. As dicas de ferramentas são criadas em todos os locais e automaticamente a partir dos comentários. Os IDEs modernos tornam muito mais fácil definir os comentários institucionais, tudo que você tem que pensar é no texto e o IDE fará o resto. Um comentário institucional é o comentário clássico que você adiciona os métodos e classes para fornecer uma descrição concisa dos objetivos. Você deve redigir esses comentários seguindo os padrões da plataforma ou da linguagem. Você também deve considerar esses comentários obrigatórios para qualquer parte do código público que você criar.

Proibir comentários óbvios é outra etapa fundamental no caminho para melhorar a legibilidade. Comentários óbvios apenas adicionam ruído e não informações relevantes. Por definição, um comentário é um texto explicativo para qualquer decisão que você tomar no código, que não é imediatamente óbvio. Um comentário deve apenas ser uma observação criteriosa sobre um aspecto particular do código.

O segundo “C” é a consistência. Cada equipe precisa usar as mesmas diretrizes para redigir o código. É ainda melhor se essas diretrizes forem usadas em uma base para toda a empresa. Quando se trata de diretrizes, muitos param no momento de definir quais devem ser as diretrizes e param de tentar entender o que é certo e o que é errado. Ouso dizer que o certo e o errado é um ponto secundário comparado à importância de sempre fazer a mesma coisa, da mesma forma em todo o código.

Suponha, por um momento, que você está gravando uma biblioteca que faz a manipulação de cadeia de caracteres. Em vários lugares dentro desta biblioteca, é provável que você precise verificar se uma cadeia de caracteres contém uma determinada sub-cadeia de caracteres. Como você faria isso? No .NET Framework, assim como no SDK do Java, você tem pelo menos duas formas de atingir o mesmo resultado. Você pode usar o método Contains ou IndexOf. Esses dois métodos, porém, servem finalidades diferentes.

O método Contains retorna uma resposta booliana e apenas informa se uma sub-cadeia de caracteres está contida dentro de determinada cadeia de caracteres. O método IndexOf retorna o índice com base em 0, onde a cadeia de caracteres pesquisada está localizada. Se não houver sub-cadeia de caracteres, o IndexOf retorna um -1. Do ponto de vista puramente funcional, portanto, você pode usar Contains e IndexOf para alcançar os mesmos objetivos.

No entanto, eles dão uma mensagem diferente para qualquer um que ler o código e obriga o leitor a fazer uma segunda passagem no código para ver se existe uma razão especial para usar IndexOf em vez de Contains. Uma única leitura de segunda passagem sobre uma linha de código não é um problema, claro. Mas quando acontece em toda a base de código de milhares de linhas do código, isso tem um impacto no tempo e, posteriormente, nos custos. Esse é o custo direto de não ter um código altamente legível.

Um sentido de consistência do código deve ser parte de sua responsabilidade inata. Como um desenvolvedor, você deve ter como objetivo gravar o código limpo pela primeira vez sem esperar ter tempo suficiente para limpá-lo depois. Como líder da equipe, você deve impor diretrizes consistentes do código através de políticas de check-in. O ideal é que você não deve permitir o check-in de qualquer código que não passe por um teste de consistência.

A última versão do ReSharper pode ser uma ajuda considerável ao tornar essa ideia concreta. Você pode usar essas ferramentas de linha de comando gratuitas - um conjunto independente de ferramentas - para integrar formas de análise de qualidade de código direto em sua integração contínua (CI) ou sistema de controle de versão. Essas ferramentas de linhas de comando podem realizar inspeções de código offline. Esse é o mesmo conjunto de inspeções de código que você pode realizar direto dentro do Visual Studio com o ReSharper instalado para capturar duplicatas em seu código. Dependendo dos recursos de sua personalização CI, você pode precisar encapsular as ferramentas da linha de comando em um componente ad hoc. Para obter mais informações sobre oReSharper, verifique bit.ly/1avsZ2R.

O terceiro e último “C” da legibilidade do código é clareza. Seu código é claro se o estilo dele for de uma forma que leia bem e facilmente. Isso inclui agrupamento e aninhamento adequados. Em geral, as instruções IF adicionam vários ruídos ao código. Algumas vezes você não consegue evitar instruções condicionais - um pilar das linguagens de programação - mas tentar limitar o número de instruções IF mantém o aninhamento sob controle e torna o código mais fácil de ler. Você também pode usar uma estrutura IF… ELSE … IF … ELSE ao invés de instruções IF aninhadas.

Algumas tarefas podem exigir algumas linhas do código e pode ser difícil ou apenas inadequado fazer uma refatoração de “Extrair Método”. Nesse caso, é bom manter essas linhas em blocos separados por linhas em branco. Isso não muda a substância do código, mas o mantém mais fácil de ler. Finalmente, se estiver procurando por inspiração sobre como dar estilo ao seu código-fonte, dê uma olhada em alguns projetos de código aberto.

Quando mais curto melhor

Linhas mais longas torna a leitura difícil para os olhos humanos. É por isso que jornais e revistas imprimem seu texto em colunas. Quando se trata de seu código, você deve fazer o mesmo e limitar tanto o comprimento horizontal das linhas e a rolagem vertical dos métodos. Qual é o comprimento ideal de um corpo do método? Geralmente, 30 linhas deve ser um nível máximo que dispara um alarme e sugere que você considere a refatoração. Finalmente, uma boa organização das pastas do projeto e a combinação entre as pastas e os namespaces geralmente reflete uma boa organização dos elementos do código individual.

Quando você acionar um tema de legibilidade e código de limpeza, geralmente está exposto a uma objeção comum - o código de limpeza da gravação é difícil e leva muito tempo. Você deve tentar neutralizar esse ponto de vista e existem duas formas para fazer isso. Uma é usar uma ferramenta de assistente do código, que ajuda a gravar o código de limpeza ao sugerir refatorações, inspecionando seu código para os padrões ruins e verificando para código morto ou duplicado. Se você pode tornar todas essas características acessíveis, realmente não tem mais desculpas para não gravar um código mais limpo e legível. As ferramentas de assistente de código são oferecidas pelos maiores fornecedores. Escolha qualquer um, mas escolha um.

A outra é enfatizar o auto-aperfeiçoamento e a atitude de seus desenvolvedores individuais para gravar códigos mais limpos como uma questão de prática geral. Os desenvolvedores experientes fazem isso muito bem. A capacidade de saber, aproximadamente, o quão longe você está da versão final do seu código, e então, realmente limpá-lo é uma característica chave que separa os desenvolvedores experientes dos menos experientes.

Dino Esposito *é o co-autor de “Microsoft .NET: Architecting Applications for the Enterprise” (Microsoft Press, 2014) e “Programming ASP.NET MVC 5” (Microsoft Press, 2014). Um evangelista técnico para as plataformas .NET Framework e Android na JetBrains e palestrante frequente em eventos do setor em todo mundo, Esposito compartilha sua visão do software em software2cents.wordpress.com e no Twitter em twitter.com/despos.*

Agradecemos ao seguinte especialista técnico da Microsoft pela revisão deste artigo: James McCaffrey

https://docs.microsoft.com/pt-br/archive/msdn-magazine/2014/october/cutting-edge-source-code-readability-tips