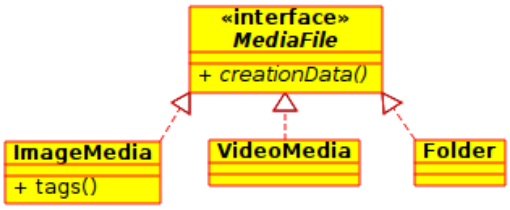
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA

IFBA – Instituto Federal de Educação, Ciência e Tecnologia
Análise e Desenvolvimento de Sistemas
INF011 – Padrões de Projeto
Prof.: Sandro Santos Andrade

Aluno: _____

3ª Avaliação Individual – 2013.1

Questão 1) (2,0) Você está implementando um *Media Center* que deve realizar o armazenamento de fotos e arquivos de vídeo. O *software* deve permitir a extração de uma série de estatísticas sobre os dados armazenados, como por exemplo o número de arquivos criados em determinado mês ou o número de arquivos marcados com uma determinada *tag* (exemplo na figura abaixo à esquerda). Os arquivos são armazenados hierarquicamente em pastas (*folders*). O diagrama abaixo à direita indica as classes que compõem a estrutura. A sua tarefa é implementar os algoritmos de: i) contagem de imagens com uma determinada *tag* e ii) contagem de arquivos criados em um determinado mês, através da utilização do padrão de projeto *Visitor*. Apresente o diagrama de classes da sua solução e a classe completa de um dos dois *Visitors*.



Questão 2) (2,0) Ainda em relação ao sistema da questão anterior: considerando que as operações que podem ser realizadas em um arquivo de imagem sejam: i) adicionar *tag* e remover *tag*, indique como você implementaria ações de *undo* e *redo* destas operações. Apresente o diagrama de classes da sua solução e o código-fonte completo que suporta tais ações em uma das operações.

Questão 3) (2,0) Cite dois exemplos inéditos e reais do uso do padrão *Observer* nos modos *push* e *pull*. Para cada exemplo explique o que justificou o uso do modo *pull* ou *push* e apresente a API do *Observer* abstrato. Discuta as soluções.

Questão 4) (2,0) Apresente um exemplo inédito que ilustre como o padrão *Iterator* separa os *concerns* de varredura e representação interna da *collection*. Indique como a representação interna da *collection* pode ser modificada sem ter impacto nas operações de varredura.

Questão 5) (2,0) Você está desenvolvendo um *framework* flexível para autenticação onde diferentes métodos de verificação de credenciais são utilizados. O sistema convencional de *login* e senha é apenas uma das possibilidades. O *framework* deve facilmente suportar a utilização de outros mecanismos, tais como autenticação biométrica ou por dispositivos de *tokens*. Apresente o diagrama de classes da sua solução de modo a suportar os requisitos acima. Qual padrão de projeto foi utilizado ?

Don't Panic !

DESTAQUE E LEVE PARA CASA PARA SE DIVERTIR :) **Não perca tempo lendo durante a prova**

Resign Patterns Ailments of Unsuitable Project-Disoriented Software – by Michael Duell - mitework@yercompany.com

1 CREMATIONAL PATTERNS

1.1 Abject Poverty: The Abject Poverty Pattern is evident in software that is so difficult to test and maintain that doing so results in massive budget overruns.

1.2 Blinder: The Blinder Pattern is an expedient solution to a problem without regard for future changes in requirements. It is unclear as to whether the Blinder is named for the blinders worn by the software designer during the coding phase, or the desire to gouge his eyes out during the maintenance phase.

1.3 Fallacy Method: The Fallacy method is evident in handling corner cases. The logic looks correct, but if anyone actually bothers to test it, or if a corner case occurs, the Fallacy of the logic will become known.

1.4 ProtoTry: The ProtoTry Pattern is a quick and dirty attempt to develop a working model of software. The original intent is to rewrite the ProtoTry, using lessons learned, but schedules never permit. The ProtoTry is also known as legacy code.

1.5 Simpleton: The Simpleton Pattern is an extremely complex pattern used for the most trivial of tasks. The Simpleton is an accurate indicator of the skill level of its creator.

2 DESTRUCTURAL PATTERNS

2.1 Adopter: The Adopter Pattern provides a home for orphaned functions. The result is a large family of functions that don't look anything alike, whose only relation to one another is through the Adopter.

2.2 Brig: The Brig Pattern is a container class for bad software. Also known as module.

2.3 Compromise: The Compromise Pattern is used to balance the forces of schedule vs. quality. The result is software of inferior quality that is still late.

2.4 Detonator: The Detonator is extremely common, but often undetected. A common example is the calculations based on a 2 digit year field. This bomb is out there, and waiting to explode!

2.5 Fromage: The Fromage Pattern is often full of holes. Fromage consists of cheesy little software tricks that make portability impossible. The older this pattern gets, the riper it smells.

2.6 Flypaper: The Flypaper Pattern is written by one designer and maintained by another. The designer maintaining the Flypaper Pattern finds herself stuck, and will likely perish before getting loose.

2.7 ePoxy: The ePoxy Pattern is evident in tightly coupled software modules. As coupling between modules increases, there appears to be an epoxy bond between them.

3 MISBEHAVIORAL PATTERNS

3.1 Chain of Possibilities: The Chain of Possibilities Pattern is evident in big, poorly documented modules. Nobody is sure of the full extent of its functionality, but the possibilities seem endless. Also known as Non-Deterministic.

3.2 Commando: The Commando Pattern is used to get in and out quick, and get the job done. This pattern can break any encapsulation to accomplish its mission. It takes no prisoners.

3.3 Intersperser: The Intersperser Pattern scatters pieces of functionality throughout a system, making a function impossible to test, modify, or understand.

3.4 Instigator: The Instigator Pattern is seemingly benign, but wreaks havoc on other parts of the software system.

3.5 Momentum: The Momentum Pattern grows exponentially, increasing size, memory requirements, complexity, and processing time.

3.6 Medicator: The Medicator Pattern is a real time hog that makes the rest of the system appear to be medicated with strong sedatives.

3.7 Absolver: The Absolver Pattern is evident in problem ridden code developed by former employees. So many historical problems have been traced to this software that current employees can absolve their software of blame by claiming that the absolver is responsible for any problem reported. Also known as It's-not-in-my-code.

3.8 Stake: The Stake Pattern is evident in problem ridden software written by designers who have since chosen the management ladder. Although fraught with problems, the manager's stake in this software is too high to allow anyone to rewrite it, as it represents the pinnacle of the manager's technical achievement.

3.9 Eulogy: The Eulogy Pattern is eventually used on all projects employing the other 22 Resign Patterns. Also known as Post Mortem.

3.10 Tempest Method: The Tempest Method is used in the last few days before software delivery. The Tempest Method is characterized by lack of comments, and introduction of several Detonator Patterns.

3.11 Visitor From Hell: The Visitor From Hell Pattern is coincident with the absence of run time bounds checking on arrays. Inevitably, at least one control loop per system will have a Visitor From Hell Pattern that will overwrite critical data.