Unidade 2 – Problema 1

QUESTIONAMENTOS

1) O código está fracamente acoplado? Ou, ao contrário, uma classe está muito dependente de outras com linhas de código com referências explícitas/diretas entre elas (*hard coded*)?

2) O código está altamente coeso? Ou seja, as linhas de código em cada método atendem a um único propósito ou estão fazendo várias coisas?

   As linhas de código de uma classe atendem ao objetivo da classe ou a classe está com muitos objetivos?

   A classe está com muitos comandos de seleção para identificar situações diferentes de execução (if / switch)? Isto demonstra que não está usando o poder da Orientação a Objetos (generalização e polimorfismo) para aumentar a coesão do código.

3) A estrutura da solução está simples para que mais classes sejam padronizadas pela interface FormatoAudio?

   A empresa esqueceu de repassar mais duas classes: MP3DJ e AACPlayer. É fácil incorporá-las na solução?

   Para incorporar estas duas classes na solução, você terá que modificar muito código já escrito? Ou seja, vai ter que modificar as classes que já estavam prontas (sinal de fraca coesão)?

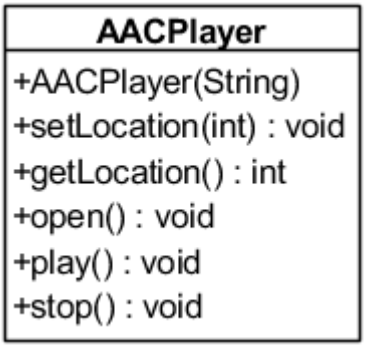
4) A forma de uso simplificada está realmente simples de ser utilizada? Ou seja, está separada e o desenvolvedor que vai utilizá-la consegue se concentrar apenas nos dois métodos solicitados?

5) Há várias classes para serem usadas/instanciadas. Há uma forma mais limpa, mais alto nível para indicar qual classe deve ser usada? Principalmente na forma simplificada, há um modo de definir qual formato de áudio será usado?

Para aprimorar sua solução, a equipe fará uso de Padrões de Projeto: Adapter, Façade e FactoryMethod. Estudem e apliquem estes padrões. Ao utilizá-los, realize o mapeamento entre as classes definidas pelo Padrão (participantes) e as classes de seu projeto. Coloque isto num arquivo texto no GitHub com nome "Mapeamento.txt".

Exemplo: no padrão Adapter dois papeis desempenhados são Target (Alvo) e Adaptee (Adaptado). Quem realiza estes papeis em seu projeto?

Classes “esquecidas” pela empresa:



O construtor é utilizado para definir o nome do arquivo que será utilizado pelo objeto de reprodução de arquivos AAC.

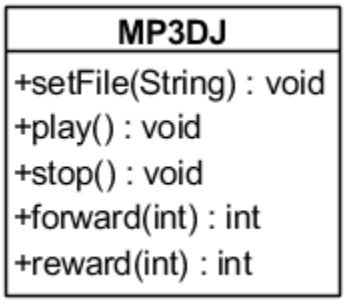
O método open() é utilizado para abrir o arquivo indicado no construtor. Porém, não define qual é o ponto inicial de reprodução.

O método setLocation() é utilizado para indicar a posição do arquivo (segundos) onde deve iniciar a reprodução. Para começar a partir do início deve-se fornecer como argumento o valor “0”.

O método getLocation() retorna em que posição (segundos) se encontra a reprodução do arquivo.

O método play() reproduz o arquivo aberto com o método open(). O arquivo de áudio começa a ser reproduzido a partir da posição indicada pelo método setLocation.

O método stop() para a reprodução do arquivo. Caso a próxima mensagem seja play(), reinicia a execução do ponto onde parou. Caso seja stop(), volta ao início do arquivo (setLocation(0)).



O método setFile() recebe o nome do arquivo a ser aberto e o abre, deixando-o pronto para execução.

O método play() reproduz o arquivo aberto, a partir do ponto em que se encontra.

O método stop() para a reprodução do arquivo. Pode ser retomado do ponto em que parou com play().

O método forward(int) recebe como parâmetro um valor em segundos em que deve ocorrer um salto para frente na posição do arquivo. Seu retorno é a nova posição, também em segundos.

O método reward(int) recebe como parâmetro um valor em segundos em que deve ocorrer um salto para trás na posição do arquivo. Seu retorno é a nova posição, também em segundos.