Checklist - Projeto Jogo de Adivinhação de Número

Encapsulamento:

- **Definição:** Encapsulamento é uma forma de restringir o acesso a alguns componentes internos dos objetos, não permitindo a disponibilidade, de forma direta, das informações privativas de uma classe, tornando-as disponíveis somente através de métodos.

Lista utilizada para verificar o encapsulamento deste projeto:

1 - Classes: O projeto encontra-se dividido em 3 classes, o que resulta em uma maior organização, melhor distribuição de código e mais facilidade caso seja necessário realizar alguma alteração.

2 - Atributos não Acessíveis Diretamente por Outras Classes: Protege os atributos do acesso direto de outras classes.

- **Classe "Jogo"** - Atendido, todos os atributos estão classificados como "private";
- **Classe "ExecutaJogo"** - Passa e recebe valores que são manipulados em outra classe, dessa forma não possui atributos próprios;
- **Classe "Funcoes"** - Não possui atributos próprios, faz uso dos atributos da classe "Pai";

3 - Quantidade de Métodos: Cada classe possui métodos para realizar as operações necessárias ou para proporcionar o acesso correto aos seus atributos:

- **Classe "Jogo"** - Possuir os métodos "Getters" e "Setters" padrão Java, dessa forma cada atributo é acessado através de seu próprio método. Ao todo esta classe possui 6 métodos.
- **Classe "ExecutaJogo"** - Esta é a classe responsável por executar o código em si, portanto possui somente o método "Main", porém esta classe faz uso de métodos de outra classe;
- **Classe "Funcoes"** - Nesta classe encontram-se todas as funções operacionais do projeto. Esta classe é "herdeira" da classe "Jogo", dessa forma faz uso dos métodos presentes na classe Pai, bem como possui diversas operações, todas claramente divididas em métodos, proporcionando organização e padronização das funções. Esta classe possui 5 métodos próprios;

4 - Quantidade de atributos: Neste projeto não houve a necessidade da criação de muitos atributos, tendo em vista se tratar de um projeto pequeno. Considerando e utilizando a ideia da herança, os atributos estão concentrados na classe "Jogo" (superclasse). Ao todo 3 atributos foram utilizados.

5 - Tamanho dos métodos: Todos os métodos possuem tamanho pequeno, visto que o projeto encontra-se dividido em vários deles. De forma geral, este contexto atende ao que propõe o Encapsulamento e a Orientação a Objeto, assim cada método faz o que realmente deve ser executado, realizando **apenas o necessário** para seu funcionamento.

Modo Resumido - Encapsulamento:

1 - Divisão por Classes - ***Atendido***;
2 - Atributos não Acessíveis Diretamente por Outras Classes - ***Atendido***;
3 - Quantidade de Métodos - ***Atendido***;
4 - Quantidade de atributos - ***Atendido***;
5 - Tamanho dos métodos - ***Atendido***;

Herança:

- **Definição:** A Herança tem por objetivo facilitar o reaproveitamento de código do projeto. Desse modo é possível criar classes que derivem de uma superclasse (Pai) onde as classes derivadas herdam seus atributos e métodos.

**Uso de Herança neste projeto:*

A fim de promover o uso do conceito de Herança da programação Orientada a Objeto neste projeto, criou-se uma classe Pai contendo atributos usados durante o jogo e os métodos "Getters" e "Setters" para cada atributo respectivamente. Assim, a classe derivada (subclasse) faz uso da Herança e consequentemente dos atributos da classe "Pai" através de cada método. Destaca-se que trabalhando desta maneira é possível disponibilizar os métodos da classe "Jogo" à outras classes caso seja necessário, bem como realizar a criação de novos métodos de uso global que poderão ser implementados no projeto.

1 - Aplicação de Herança: A classe "Funcoes" é herdeira da classe "Jogo", assim sendo está classe consegue fazer uso dos atributos e métodos da classe "Pai".

Modo Resumido - Herança:

1 - Uso do conceito de Herança (superclasse e subclasse) - ***Atendido***.

Polimorfismo:

- **Definição:** Polimorfismo em definição livre, várias formas, permite invocar métodos que tem a mesma identificação, mas componentes distintos, adequando-se a necessidade em cada situação. Um método pode ter um padrão global escrito em uma Superclasse, mas dependendo da particularidade da Subclasse alguns objetos podem ser sobrescritos. Há também a possibilidade de criação de métodos com mesmo nome, porém com assinaturas diferentes, dessa forma há um "reuso" de métodos em classes distintas porém que fazem uso de atributos e métodos similares.

Uso de Polimorfismo neste projeto:

1 - A classe Pai (Jogo) possui um método padrão para colocar valores nas variáveis. Porém para aplicar os dados ao jogo, este método é invocado na classe filha (Funcoes) sendo subscrito pelos parâmetros corretos em 2 situações, antes do inicio do jogo, onde mostra para o usuário uma mensagem de boa vindas e após o fim do jogo onde informa ao usuário que o jogo acabou e que as variáveis foram redefinidas.

1 - Uso de Polimorfismo: O método limpaVariaveis() está presente por padrão na classe Jogo, porém a classe Funcoes faz uso do mesmo método subscrevendo-o.

Modo Resumido - Polimorfismo:

1 - Uso do método presente na classe Pai, sendo este subscrito conforme necessidade da classe que o invoca, a classe filha. - ***Atendido*.**

Resumo Geral:

De forma geral, observa-se que este pequeno projeto faz uso de todos os elementos propostos no exercício, tendo como finalidade demostrar os benefícios da programação Orientada a Objeto. Desse modo, denota-se que o trabalho atende a todos os requisitos sugeridos. O projeto, apesar de simples, ajudou a tornar mais claro o entendimento sobre como é a Orientação a Objeto e como ela realmente funciona, causado uma boa compreensão do tema elencado na disciplina.