INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

RELATÓRIO CAP-392-3: PADRÕES DE PROJETO

PROFESSOR: DR. EDUARDO MARTINS GUERRA

ATIVIDADE #1

PRINCÍPIOS BÁSICOS DA ORIENTAÇÃO A OBJETOS

EVENTOS

LUIZ WAGNER TAVARES NASCIMENTO

SÃO JOSÉ DOS CAMPOS

2016

1. INTRODUÇÃO

O presente relatório apresenta a solução para a Atividade nº1 – Princípios Básicos da Orientação a Objetos, realizada pelo Grupo 2 e as mudanças efetivadas na estrutura e hierarquia durante a implementação da solução. A Figura 1 apresenta o modelo da solução apresentada em sala de aula.

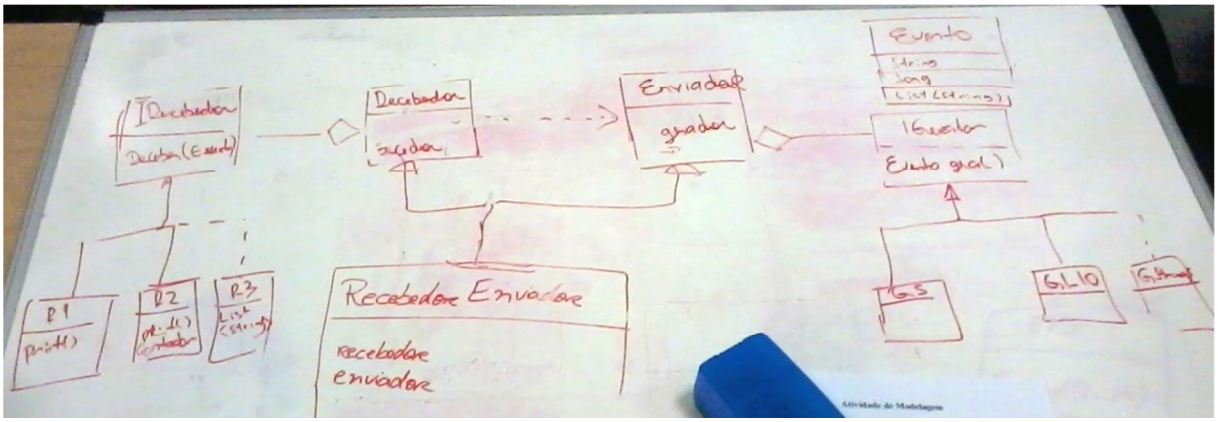

Figura 1: Modelo de Classes apresentado em sala de aula

2. IMPLEMENTAÇÃO

Durante a implementação da solução em linguagem Java, algumas considerações e mudanças foram realizadas em relação ao modelo apresentado em sala:

- A classe *Evento* inicialmente foi criada da mesma com os mesmos atributos iniciais (*string e long*), bem como a Interface *GeradorEventos* e classes *Enviador*, *GeradorEventos5Segundos,* etc. foram mantidos com os mesmos nomes do modelo apresentado em sala.
- Criada uma classe chamada GeradorEventosAbstrato que implementa a Interface GeradorEventos e serve como superclasse para a hierarquia, concentrando todo comportamento comum entre as classes, como os métodos setStringDoEvento(), *gerarEvento*(), *iniciarGeracaoEventos*().
    - Posteriormente foi implementado o código para geração de eventos em paralelo (Threads) e nome foi refatorado para *GeradorEventosThread;*
- Não foi utilizada a classe *Recebedor* apresentada no modelo. Apenas através da hierarquia da inferface *ReceptorEventos*, a classe *Enviador* possui uma lista de receptores para notificá-los após a geração de cada evento.
- A hierarquia que interface *ReceptorEventos* não utiliza estratégia de impressão do conteúdo do evento, apenas delega a impressão para o próprio evento, que sabe como "se imprimir"
    - Cada forma de impressão de evento foi transferida para uma hierarquia de evento (*Evento e EventoMomentoGeracao,* transferindo o atributo long para este evento específico);
    - Cada *GeradorEventos* então cria o seu tipo de *Evento* específico, através da sobrescrita do método *gerarEvento*().
- Foi criado uma propriedade *enviador* na classe *GeradorEventosThread*, para manter a referência utilizada dentro do método *run*() e notificar os receptores através do método *notificarReceptores*(evento).
- Ao implementar o Requisito Extra 1, percebi que haveria duplicidade de código para a geração de evento de forma aleatória. Foi criada então a classe *PeriodicidadeGeradorEventos* e inserida na

classe *GeradorEventosThread*, encapsulando assim a lógica de criação de um número fixo ou um número aleatório.

- Na implementação do Requisito Extra 2, como não utilizei a classe *Recebedor* do modelo, também não criei a classe *RecebedorEnviador* que possuiria referência para as duas classes do modelo apresentado (*Recebedor* e *Enviador*).
    - Conforme o requisito, essa seria mais uma classe que, ao receber um evento, geraria outro evento. Dessa forma, foi implementada a classe *ReceptorEnviador*, que implementa a inferface *ReceptorEventos* e ainda possui a referência para um novo *Enviador*. Assim, o método *receberEvento(Evento evento)* foi sobrescrito, implementando a lógica de armazenar em sua lista a string recebida do evento e ainda utilizar seu *enviador* para gerar um novo evento do tipo *EventoComListaStrings*, criado como subclasse de *Evento* para somente encapsular a lista de string enviada.
- Foi adicionado ao *enviador* do *receptorEnviador* um *ReceptorInformacoes* que imprime a string do evento (que no caso do *EventoComListaStrings* apenas utiliza o *toString*() da classe *ArrayList*) e um novo *ReceptorContador*, apenas para efeito de teste da implementação e visualização da contagem dos novos eventos gerados.

2.1. MODELO IMPLEMENTADO

A Figura 2 apresenta o modelo de classes da implementação com as modificações realizadas.

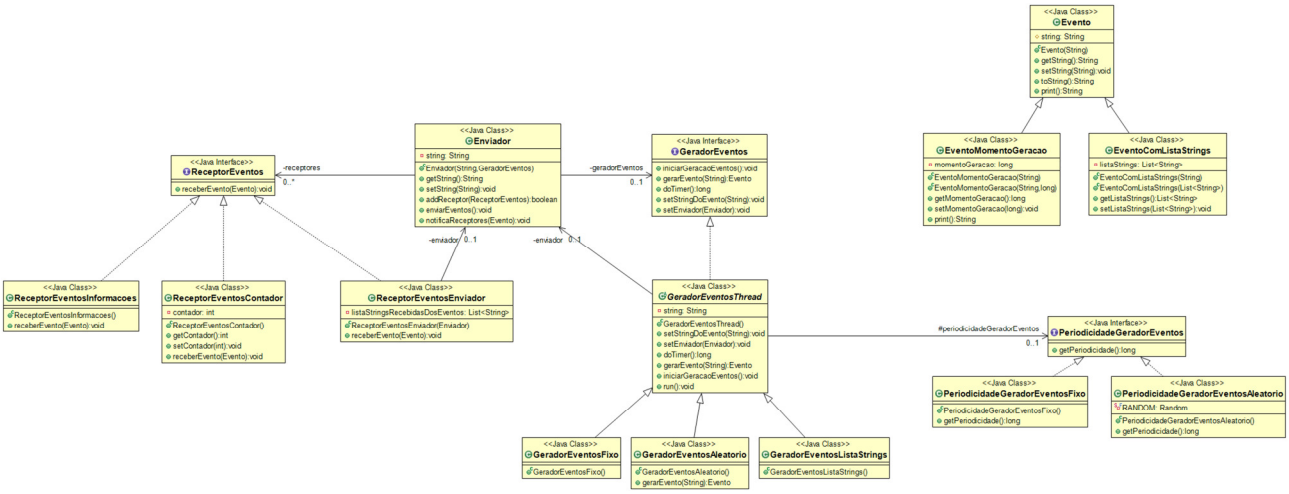

Figura 2: Modelo de Classes com modificações após implementação

..\modelo\Eventos-modificações.png

3. CONCLUSÃO

A atividade proposta proporcionou uma melhor solidificação da teoria apresentada na vídeo-aula. Aparentemente a solução parecia trivial, porém após a implementação percebeu-se a real necessidade de se separar os comportamentos que precisaram mudar com a evolução dos requisitos.

Algumas alterações estavam bem claras no momento da geração do modelo, como o encapsulamento da forma de impressão dentro do próprio *Evento* e sua hierarquia para não "inchar" o evento básico com propriedades que nem sempre são utilizadas. Outros detalhes pareciam resolvidos no modelo, porém só ficaram claros no momento da implementação, como no caso do Requisito Extra 1, que foi necessária a separação do comportamento de Periodicidade para não haver duplicidade de código.