

Programação e Desenvolvimento de Software 2

Refatoração

Prof. Julio Cesar S. Reis
julio.reis@dcc.ufmg.br

Projeto de Software

- Projeto (design de Software)
 - Requisitos do usuário -> Software
 - Estrutura do software (módulos, classes, ...)
 - O próprio código é o design!
- Mas o desenvolvimento de um software é algo dinâmico...

Evolução / Manutenção de Software

- Um software precisa evoluir
- Com a evolução
 - O código vai sendo atualizado
 - Decisões passadas vem perdendo efeito
 - Elementos inúteis sem benefícios diretos
 - Difícil fazer alterações e manter o design inicial
- O que fazer?

Refatoração

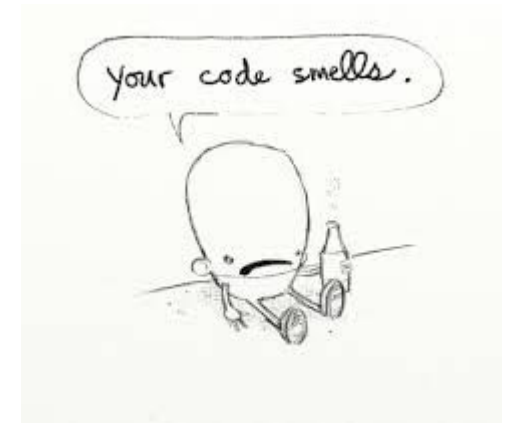
- Processo de reescrever códigos do sistema para melhorar sua estrutura de maneira geral
- Objetivo
 - Melhorar o código
 - Sem mudar as funcionalidade
 - Sem inserir bugs

Refatoração

- Modificação [pequena] no sistema que não altera o comportamento funcional, mas que melhora qualidades não funcionais
 - Flexibilidade, clareza, robustez, ...
- Alteração no design de uma aplicação
 - Atividade que estava implícita
 - Preceito básico de eXtreme Programming (XP)

Vantagens

- Melhorar aspectos como
 - Modularização
 - Reuso
 - Complexibilidade
 - Manutenabilidade
- Como?
 - Atacar os “bad smells” no código
 - Por “sorte”, temos uma série destes



Refatoração

- Design Smells/Bad Smells
 - Características (odores) que são perceptíveis em softwares (códigos) de má qualidade (podres)
 - Rigidez, Fragilidade, Imobilidade, Viscosidade, Complexidade, Repetição, Opacidade
- Propor as refatorações adequadas a partir da identificação de um desses problemas

Refatoração

Exemplos

- Mudança em nomes de variáveis e métodos
- Redução de código duplicado
 - É mais fácil fazer um “Copy and Paste”
- Generalizar/flexibilizar métodos
- Membros não encapsulados (públicos)
- Mudanças arquiteturais
 - Módulos, Classes, Interfaces, ...

Refatoração

- Não é uma reestruturação arbitrária
 - Código ainda deve funcionar (não inserir bugs)
 - Testes tentam garantir isso
 - Mudanças pequenas/pontuais (não reescrever tudo)
 - A semântica deve ser preservada
- Resultado
 - Alta coesão / Baixo acoplamento
 - Reusabilidade, legibilidade, testabilidade

Refatoração

■ Coesão

- Grau de dependência entre os elementos internos de um mesmo módulo
- Funções, responsabilidades (mesmo objetivo)

■ Acoplamento

- Grau de interdependência entre módulos
- Alteração de um demanda alteração no módulo

Refatoração

■ Esses são exemplos de refatoração?

- Adicionar novas funcionalidades
- Melhorias no desempenho
- Correção de erros existentes
- Detecção de falhas de segurança

Refatoração

■ Esses são exemplos de refatoração?

- Adicionar novas funcionalidades
- Melhorias no desempenho
- Correção de erros existentes
- Detecção de falhas de segurança

WRONG!

Refatoração

Então, quando fazer?

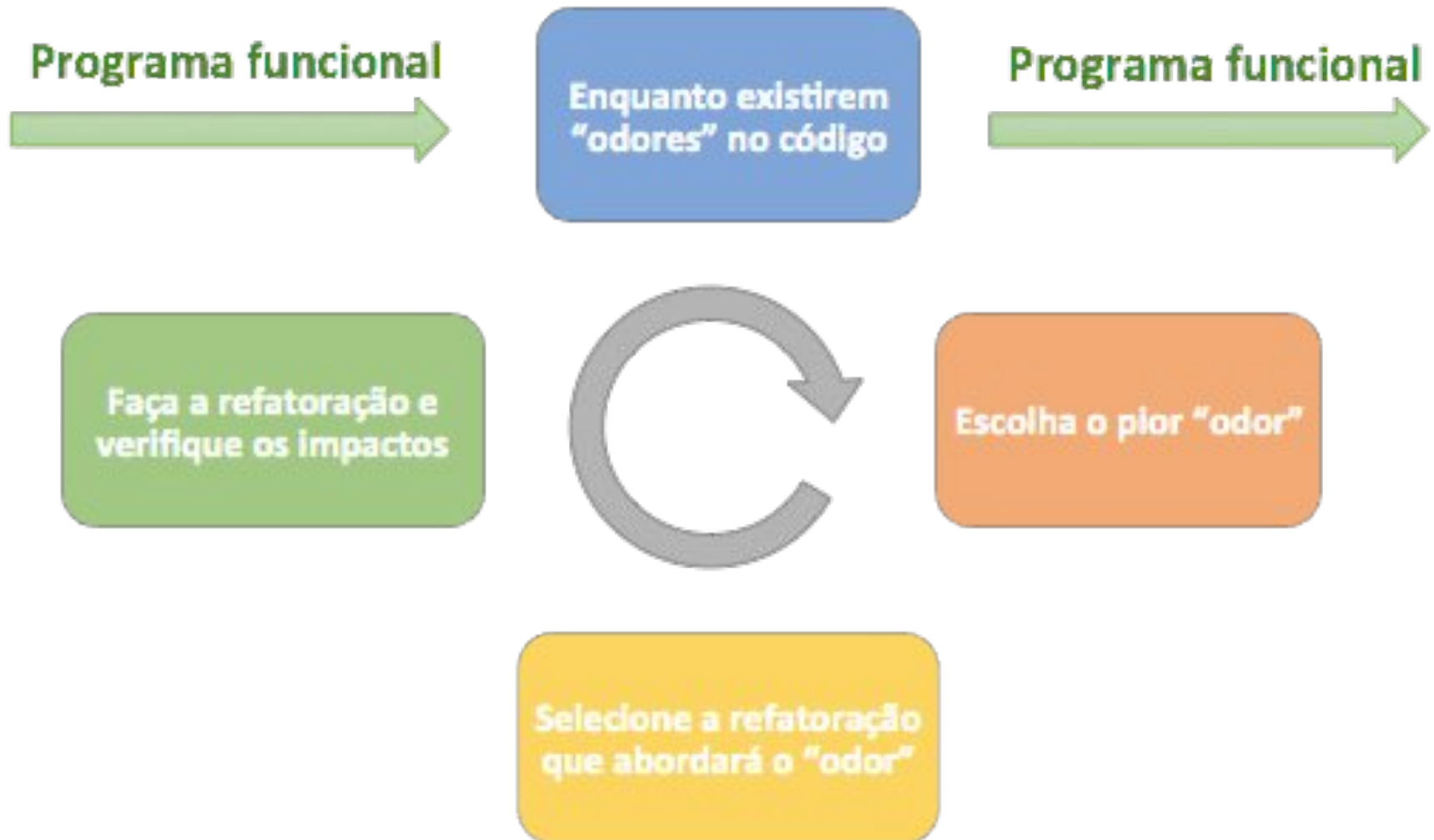
- Encontrou um “bad smell”
- Sabe uma maneira melhor de fazer as coisas
- Alteração não vai quebrar o código

E quando NÃO fazer?

- Código estável que não precisa mudar
- Prazo para entrega se aproximando
- Pouco conhecimento do código (terceiros)

Refatoração

Ciclo



Refatoração

- Geralmente são mudanças simples

- Operações sistemáticas e óbvias
- Catálogo de refatorações [Fowler, 1999]

- Localmente pode não ser tão perceptível, porém no todo o impacto é considerável

“If you want to refactor, the essential precondition is having solid tests.”

- Martin Fowler, Refactoring, p. 89

<https://refactoring.com/catalog/>

Alguns Odores Comuns

Código Duplicado	<ul style="list-style-type: none">- Criar método comum- Ou criar classe- Substituir código por chamada
Método Longo	<ul style="list-style-type: none">- Criar sub-métodos- Ou criar classe- Inserir chamadas
Classe Longa	<ul style="list-style-type: none">- Criar novas classes- Extrair super-classe ou interface- Re-adequar o código
Inveja de Features	<ul style="list-style-type: none">- Extrair método- Mover método- Composição
Muita Intimidade	<ul style="list-style-type: none">- Re-organizar dados- Mover métodos- Mover campos

Catálogo do Problemas e Soluções

- Existem diversos outros problemas
- Além de outras soluções

<https://blog.codinghorror.com/code-smells/>

<https://refactoring.com/catalog/>

Caso de Estudo

Sistema de alugueis de filmes do Google Play

Acabaram de chegar

Lançamentos do cinema

See more



Skyscraper
Action & Adventure

★★★★★ R\$25.90



Hotel Transylvania 3
Portuguese audio

★★★★★ R\$16.90



Ant-Man and the Wasp
Action & Adventure

★★★★★ R\$7.90



Jurassic World: Fallen Kingdom
Action & Adventure

★★★★★ R\$6.90

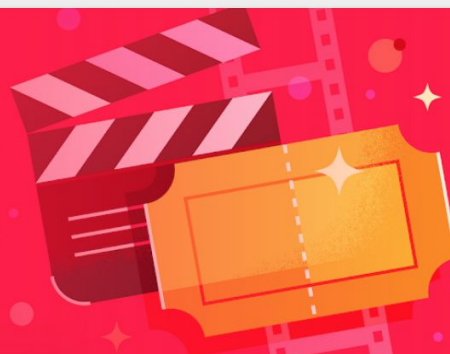


Solo
Action & Adventure

★★★★★ R\$7.90

Alugue qualquer filme por R\$3,90

Clique para resgatar

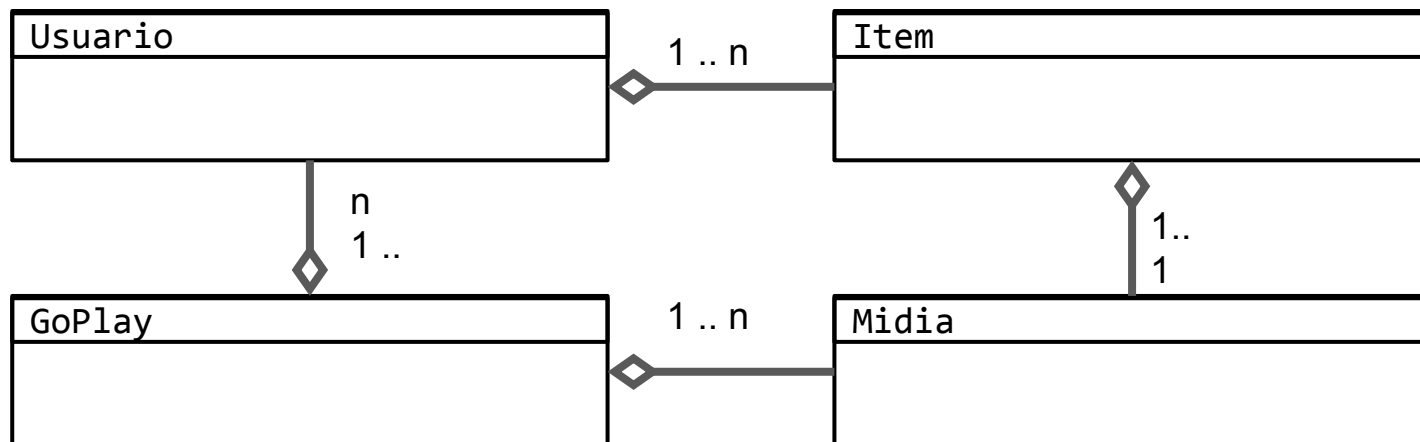


Modelando o Problema

Hands On:

<https://github.com/flaviovdv/programacao-2/tree/master/exemplos/aula15-refatoramento/01-codigo-ruim>

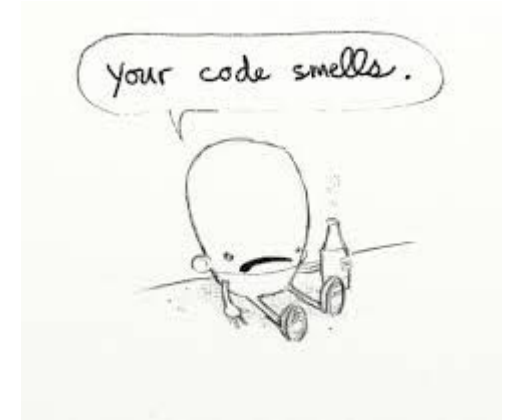
- Acompanhar código do GitHub
- Sistema um pouco complexo para slides



Foco do nosso problema

Um método que computa o total que um usuário gastou na sessão

```
double Usuario::total_gastos() {  
    std::vector<Item>::iterator it = this->_midias.begin();  
    std::vector<Item>::iterator ed = this->_midias.end();  
    double total_gastos = 0.0;  
    for (; it != ed; it++) {  
        Item item = *it;  
        Midia midia = item.get_midia();  
        switch(midia.get_tipo()) {  
            case Tipo::EPISODIO_SERIE:  
                if (item.foi_compra()) {  
                    total_gastos += 12.0;  
                } else {  
                    total_gastos += 3.50;  
                }  
                break;  
            case Tipo::LANCAMENTO:  
                if (item.foi_compra()) {  
                    total_gastos = 32.0;  
                } else {  
                    total_gastos += 16.00;  
                }  
                break;  
            case Tipo::NORMAL:  
                if (item.foi_compra()) {  
                    total_gastos = 15.0;  
                } else {  
                    total_gastos += 7;  
                }  
                break;  
        }  
    }  
    return total_gastos;  
}
```



Bad Smell #1

Método muito longo. Tá na cara. Como Resolver?

```
double Usuario::total_gastos() {  
    std::vector<Item>::iterator it = this->_midias.begin();  
    std::vector<Item>::iterator ed = this->_midias.end();  
    double total_gastos = 0.0;  
    for (; it != ed; it++) {  
        Item item = *it;  
        Midia midia = item.get_midia();  
        switch(midia.get_tipo()) {  
            case Tipo::EPISODIO_SERIE:  
                if (item.foi_compra()) {  
                    total_gastos += 12.0;  
                } else {  
                    total_gastos += 3.50;  
                }  
                break;  
            case Tipo::LANCAMENTO:  
                if (item.foi_compra()) {  
                    total_gastos = 32.0;  
                } else {  
                    total_gastos += 16.00;  
                }  
                break;  
            case Tipo::NORMAL:  
                if (item.foi_compra()) {  
                    total_gastos = 15.0;  
                } else {  
                    total_gastos += 7;  
                }  
                break;  
        }  
    }  
    return total_gastos;  
}
```

Solução: Bad Smell #1

Extração de métodos comuns. Parece que vai ajudar...

```
double Usuario::_preco_serie(Item &item) {  
    if (item.foi_compra()) {  
        return 12.0;  
    } else {  
        return 3.50;  
    }  
}  
  
double Usuario::_preco_lancamento(Item &item) {  
    if (item.foi_compra()) {  
        return 32.0;  
    } else {  
        return 16.00;  
    }  
}
```

Solução: Bad Smell #1

Método menor. Porém ainda temos problemas

```
double Usuario::total_gastos() {  
    std::vector<Item>::iterator it = this->_midias.begin();  
    std::vector<Item>::iterator ed = this->_midias.end();  
    double total_gastos = 0.0;  
    for (; it != ed; it++) {  
        Item item = *it;  
        Midia midia = item.get_midia();  
        switch(midia.get_tipo()) {  
            case Tipo::EPISODIO_SERIE:  
                total_gastos += this->_preco_serie(item);  
                break;  
            case Tipo::LANCAMENTO:  
                total_gastos += this->_preco_lancamento(item);  
                break;  
            case Tipo::NORMAL:  
                total_gastos += this->_preco_normal(item);  
                break;  
        }  
    }  
    return total_gastos;  
}
```

Solução: Bad Smell #1

Qual o problema aqui?

```
double Usuario::_preco_serie(Item &item) {  
    if (item.foi_compra()) {  
        return 12.0;  
    } else {  
        return 3.50;  
    }  
}  
  
double Usuario::_preco_lancamento(Item &item) {  
    if (item.foi_compra()) {  
        return 32.0;  
    } else {  
        return 16.00;  
    }  
}
```


Bad Smells #2, 3

Parece que Usuário sabe muito sobre os preços

- Note que a classe usuário sabe muito sobre como os preços são computados
- #2: Usuário é uma classe invejosa
- #3: Muita intimidade com o preço

Solução: Bad Smells #2, 3

Vamos mover o conhecimento para o local correto. Item.

```
double Item::_preco_serie() {  
    if (this->foi_compra()) {  
        return 12.0;  
    } else {  
        return 3.50;  
    }  
}
```

```
double Item::get_valor() {  
    switch(this->_midia.get_tipo()) {  
        case Tipo::EPISODIO_SERIE:  
            return this->_preco_serie();  
        case Tipo::LANCAMENTO:  
            return this->_preco_lancamento();  
        case Tipo::NORMAL:  
            return this->_preco_normal();  
    }  
}
```

Solução: Bad Smells #2, 3

Vamos mover o conhecimento para o local correto. Item.

- Estamos no caminho certo
- Olhe como o método do usuário fica

```
double Usuario::total_gastos() {  
    std::vector<Item>::iterator it = this->_midias.begin();  
    std::vector<Item>::iterator ed = this->_midias.end();  
    double total_gastos = 0.0;  
    for (; it != ed; it++) {  
        Item item = *it;  
        total_gastos += item.get_valor();  
    }  
    return total_gastos;  
}
```

Solução: Bad Smells #2, 3

Vamos mover o conhecimento para o local correto. Item.

- A classe nem sequer usa Midia
- Método curto e direto
- Vamos limpar a "sujeira" C++

```
double Usuario::total_gastos() {  
    std::vector<Item>::iterator it = this->_midias.begin();  
    std::vector<Item>::iterator ed = this->_midias.end();  
    double total_gastos = 0.0;  
    for (; it != ed; it++) {  
        Item item = *it;  
        total_gastos += item.get_valor();  
    }  
    return total_gastos;  
}
```

Solução: Bad Smells #2, 3

Vamos mover o conhecimento para o local correto. Item.

- A classe nem sequer usa Midia
- Método curto e direto
- Good, but not yet the best.

```
double Usuario::total_gastos() {  
    auto it = this->_midias.begin();  
    auto ed = this->_midias.end();  
    double total_gastos = 0.0;  
    for (; it != ed; it++) {  
        Item item = *it;  
        total_gastos += item.get_valor();  
    }  
    return total_gastos;  
}
```

Solução: Bad Smells #2, 3

Vamos mover o conhecimento para o local correto. Item.

- A classe nem sequer usa Midia
- Método curto e direto
- Melhor

```
double Usuario::total_gastos() {  
    double total_gastos = 0.0;  
    for (auto item : this->_midias) {  
        total_gastos += item.get_valor();  
    }  
    return total_gastos;  
}
```

Bad Smells #4, 5, 6

Os itens agora viraram o problema

- Note que o item agora é um problema
- Bad Smell #4: Código Repetido

```
double Item::_preco_serie() {  
    if (this->foi_compra()) {  
        return 12.0;  
    } else {  
        return 3.50;  
    }  
}
```

Bad Smells #4, 5, 6

Os itens agora viraram o problema

- Note que o item agora é um problema
- Bad Smell #4: Código Repetido
 - Note que o if/else abaixo se repete em três métodos diferentes

```
double Item::_preco_serie() {  
    if (this->foi_compra()) {  
        return 12.0;  
    } else {  
        return 3.50;  
    }  
}
```


Bad Smells #4, 5, 6

Os itens agora viraram o problema

- Note que o item agora é um problema
- Bad Smell #4: Código Repetido
- Bad Smell #5: Classe Longa
 - Os três métodos no estilo abaixo

```
double Item::_preco_serie() {  
    if (this->foi_compra()) {  
        return 12.0;  
    } else {  
        return 3.50;  
    }  
}
```

Bad Smells #4, 5, 6

Os itens agora viraram o problema

- Note que o item agora é um problema
- Bad Smell #4: Código Repetido
- Bad Smell #5: Classe Longa
- Bad Smell #6: Uso de Magic Numbers

```
double Item::_preco_serie() {  
    if (this->foi_compra()) {  
        return 12.0;  
    } else {  
        return 3.50;  
    }  
}
```

Bad Smell #6

Números mágicos são uma boa dica para refatorar

- Números mágicos são constantes "soltas"
- Veja o exemplo abaixo
 - Qual o significado do número 7?
 - Se tivermos que mudar por outro número?

```
for (int i = 0; i < 7; i++) {  
    cartas.push_back(deck.random());  
}
```

Solução: Bad Smell #6

Uso de constantes, re-organizar dados, move method

- Podemos definir constantes usando mais de uma forma em C++
 - `#define NUM_CARTAS 7;`
 - `static int const NUM_CARTAS;`
- Podemos também re-organizar os dados
 - No caso de estudo, os preços devem ficar junto com os itens
- Criar classes que manipulam os números

Pensando em um sistema real

Itens tem valores fixos, mídias não

- Note que no momento os valores estão atrelados aos itens. Porém note que:
 - Compramos uma única vez
- Parece que não faz sentido a classe item saber computar valores

Item ideal

Mais uma classe pequena =)

- Precisamos da lógica em algum local
- A classe Midia é um bom ponto de partida

```
Item::Item(Midia &midia, double valor, bool compra):  
    _midia(midia), _valor(valor), _compra(compra) {}
```

```
Midia Item::get_midia() {  
    return this->_midia;  
}
```

```
double Item::get_valor() {  
    return this->_valor;  
}
```

Nova Midia

```
Midia::Midia(std::string nome, double preco_aluguel, double preco_compra,
             Tipo tipo):
    _nome(nome), _preco_aluguel(preco_aluguel),
    _preco_compra(preco_compra), _tipo(tipo) {}

Tipo Midia::get_tipo() {
    return this->_tipo;
}

std::string Midia::get_nome() {
    return this->_nome;
}

double Midia::get_preco_aluguel() {
    return this->_preco_aluguel;
}

double Midia::get_preco_compra() {
    return this->_preco_compra;
}
```

Classe GoPlay

Uso de constantes para problemas de magic numbers

```
double const GoPlay::ALUGUEL_NORMAL = 7.0;
double const GoPlay::COMPRA_NORMAL = 15.0;

double const GoPlay::ALUGUEL_LANCAMENTO = 16.0;
double const GoPlay::COMPRA_LANCAMENTO = 32.0;

double const GoPlay::ALUGUEL_SERIE = 3.50;
double const GoPlay::COMPRA_SERIE = 12.0;

GoPlay::GoPlay() {
    this->_codigo_midia = 0;
    this->_codigo_usuario = 0;
}
```


Mais problemas

Parece que a classe GoPlay está com problemas

- Temos que limpar o código duplicado
- Porém note que:
 - Para um sistema de Filmes, Séries e Lançamentos
 - Onde o preço não muda ao longo do tempo
 - Estamos ok!
- Keep it simple.

Ainda não estamos prontos

Vamos agora evoluir o programa

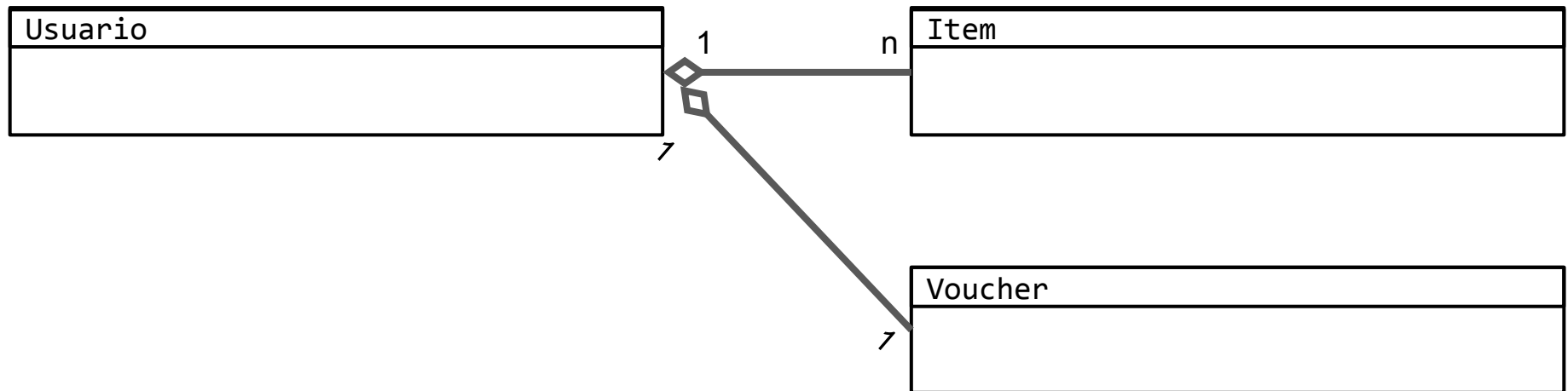
- Novas User Stories
 - Preços que mudam com o tempo
 - Aplicação de Vouchers de Preços



Sistema de Vouchers

Com o código atual, muda apenas a GoPlay e Usuario. Isto é bom!

- Assumindo que os vouchers colocam um preço único em tudo. Estilo acima.
- Cada usuário pode ter 1 voucher



Sistema de Vouchers

Com o código atual, muda apenas a GoPlay e Usuario. Isto é bom!

- Mudança ao setar o preço
- Verificamos se usuário tem voucher
- Se sim, preço novo
- Se não, preço antigo

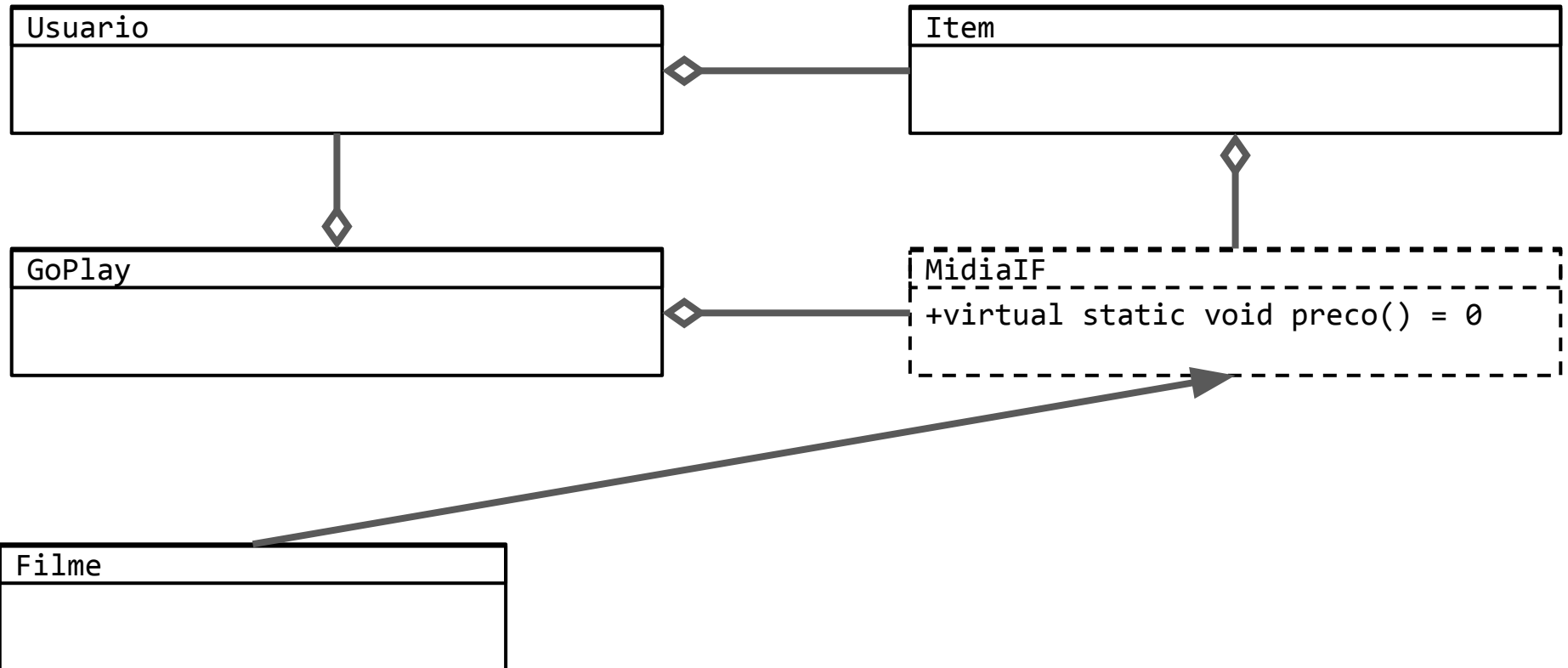
Nova Funcionalidade

Compra e Aluguel de Livros

- Ao invés de Bad Smells vamos adicionar novas Mídias no nosso Google Play
- Como resolver tal caso?
 - Note que não existem séries de livros
 - Pelo menos não no estilo seriados de TV
 - O Enum é um impecilho
 - Além de tal, temos um comportamento comum (os preços)

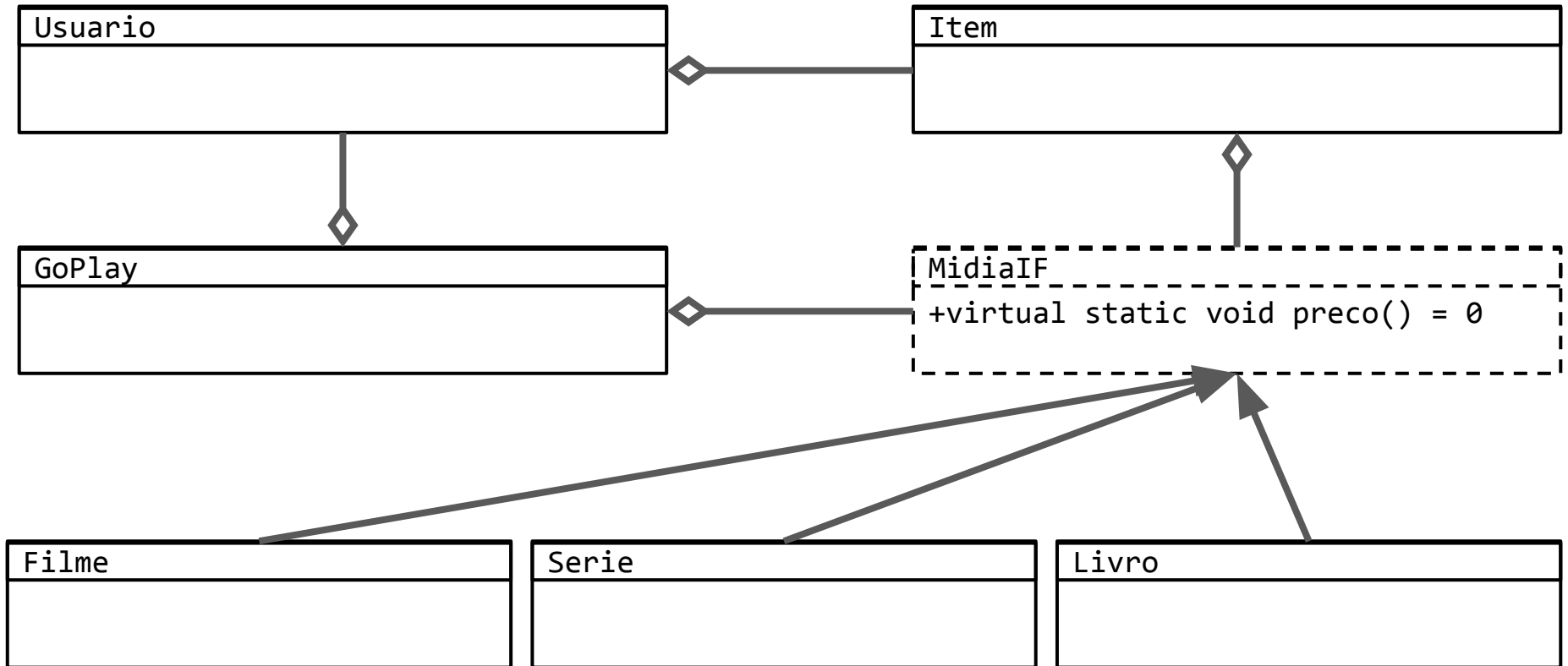
Refatorando: Diagrama Novo

Extraindo uma interface comum. Testar se deu certo



Evoluindo: Diagrama Novo

Depois de refatorar, implementar tipos novos



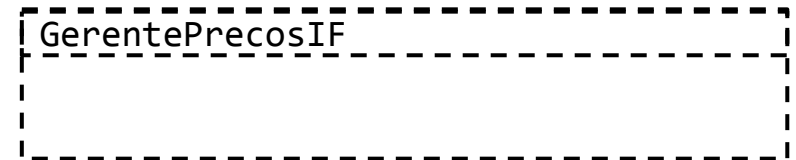
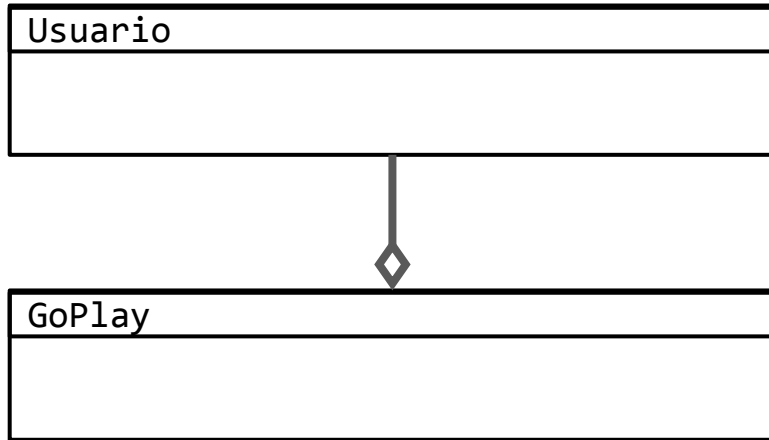
Nova Funcionalidade

Mercado de Preços

- Cada MidiaIF vai ter regras diferentes sobre como o preço muda
- Como resolver?

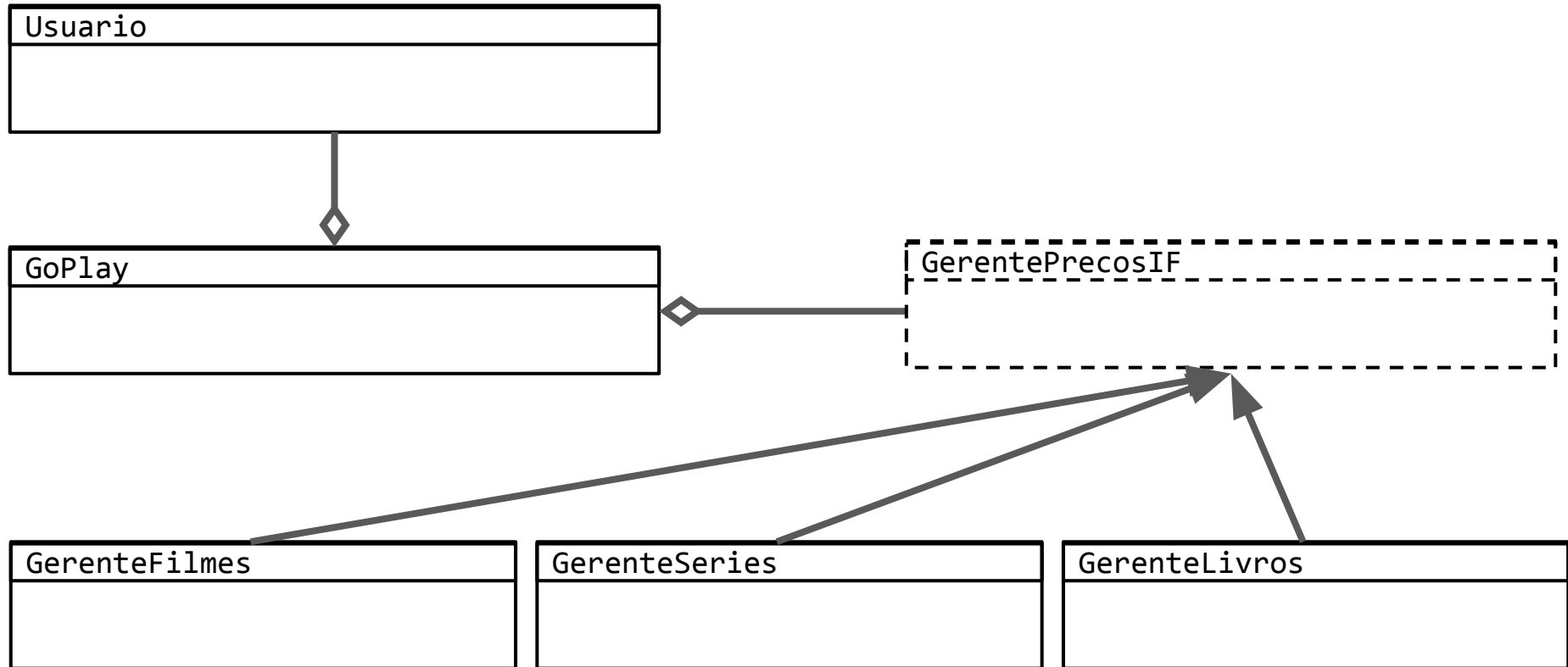
Solução

Refatorando



Ainda temos problemas

Evoluindo



Quando parar?

Keep it simple Stupid!

- Podemos continuar eternamente
- É bom para quando:
 - Nossas user stories são cumpridas
 - Iniciar novamente a partir de cada funcionalidade nova
 - Não faça over-designs desde o início
- Bons programadores escrevem código para outros programadores (humanos).

Considerações finais

- Benefícios desenvolvedores
 - Melhora manutenção
 - Aumenta reusabilidade
 - Facilita testes automatizados
 - Alta coesão, baixo acoplamento
 - Maior robustez do código
 - Facilita o trabalho em equipe

Considerações finais

- Benefícios negócios
 - Baixo índice de erros
 - Código de maior valor
 - Facilidade de adaptação de requisitos
 - Fácil adicionar novas features
 - Produto liberado mais rapidamente
 - Atacar desempenho/segurança