DEPRECANDO SINGLETON(GOF)

Uma abordagem mais apropriada para janelas filhas

27 de Novembro de 2018

Ricardo Cosme

Instituto Tecgraf de Desenvolvimento de
Software Técnico-Científico da PUC-Rio
Tecgraf/PUC-Rio

PADRÃO SINGLETON

PADRÃO SINGLETON

Objetivo

Instância **única** de uma classe que é obtida através de um ponto de acesso **global**. [1]

[1] Design Patterns: Elements Of Reusable Object-Oriented Software (1994, Addison Wesley)

EM SUMA

Singleton = global + instância única

GLOBAIS

Globais não são bem-vindas

→ Compartilhamento de estado

→ Condições de corrida (computação moderna)

GLOBAIS

Globais não são bem-vindas

- → Compartilhamento de estado
  - → Condições de corrida (computação moderna)
  - → Ineficiência na alocação e liberação de recursos

GLOBAIS

Globais não são bem-vindas

- → Compartilhamento de estado
  - → Condições de corrida (computação moderna)
  - → Ineficiência na alocação e liberação de recursos
  - → Não há garantia de ordem na inicialização entres TUs

GLOBAIS

Globais não são bem-vindas

- → Compartilhamento de estado
  - → Condições de corrida (computação moderna)
  - → Ineficiência na alocação e liberação de recursos
  - → Não há garantia de ordem na inicialização entres TUs
  - → Impossibilidade de reasoning do código

INSTÂNCIAS

Classes não deveriam ser responsáveis pelo lifetime de instâncias

→ Lógica estranha em C++

INSTÂNCIAS

Classes não deveriam ser responsáveis pelo lifetime de instâncias

→ Lógica estranha em C++

→ Hoje é **uma** instância, amanhã pode ser **N**

INSTÂNCIAS

Classes não deveriam ser responsáveis pelo lifetime de instâncias

- → Lógica estranha em C++
- → Hoje é **uma** instância, amanhã pode ser **N**
- → Por que sou obrigado a usar alocação dinâmica?

MAU USO

→ Janelas filhas (Qual é a janela pai(dono)?)

MAU USO

- → Janelas filhas (Qual é a janela pai(dono)?)
- → Managers (Projeto como dono?)

PLEASE!

→ Não use Singletons indiscriminadamente

PLEASE!

- → Não use Singletons indiscriminadamente
- → Suícidio não é bom na vida real nem na programação
  - → `delete this;`

JANELAS FILHA (SINGLE)

JANELA FILHA (SINGLE)

```
1  class ToolXPresenter {
2    ToolXPresenter() = default;
3    static ToolXPresenter* _instance;
4  public:
5    static ToolXPresenter& instance() {
6      if(!_instance)
7        _instance = new ToolXPresenter();
8      return *_instance;
9    }
10   void close() { delete this; }
11 };
12 void MainPresenter::openToolX()
13 { ToolXPresenter::instance().show(); }
```

JANELA FILHA (SINGLE)

```
1  class ToolXPresenter {
2    ToolXPresenter() = default;
3    static ToolXPresenter* _instance;
4  public:
5    static ToolXPresenter& instance() {
6      if(!_instance)
7        _instance = new ToolXPresenter();
8      return *_instance;
9    }
10   void close() { delete this; }
11 };
12 void MainPresenter::openToolX()
13 { ToolXPresenter::instance().show(); }
```

SOLUÇÃO ADHOC

```
1  class ToolXPresenter {
2    function<void()> _onClose;
3  public:
4    template<typename FunctionObject>
5    void onClose(FunctionObject cbk)
6    { _onClose = std::move(cbk); }
7  };
```

SOLUÇÃO ADHOC

```
1  class ToolXPresenter {
2    function<void()> _onClose;
3  public:
4    template<typename FunctionObject>
5    void onClose(FunctionObject cbk)
6    { _onClose = std::move(cbk); }
7  };
```

SOLUÇÃO ADHOC

```
1  struct MainPresenter {
2    unique_ptr<ToolXPresenter> toolx;
3  };
4
5  void MainPresenter::openToolX() {
6    if(!toolx) {
7      unique_ptr<ToolXPresenter> o
8        (new ToolXPresenter(view));
9      o->onClose([this]{ toolx.release(); });
10     toolx = std::move(o);
11   }
12   toolx.show();
13 }
```

SOLUÇÃO ADHOC

```
1  struct MainPresenter {
2    unique_ptr<ToolXPresenter> toolx;
3  };
4
5  void MainPresenter::openToolX() {
6    if(!toolx) {
7      unique_ptr<ToolXPresenter> o
8        (new ToolXPresenter(view));
9      o->onClose([this]{ toolx.release(); });
10     toolx = std::move(o);
11   }
12   toolx.show();
13 }
```

SOLUÇÃO ADHOC

```
1  struct MainPresenter {
2    unique_ptr<ToolXPresenter> toolx;
3  };
4
5  void MainPresenter::openToolX() {
6    if(!toolx) {
7      unique_ptr<ToolXPresenter> o
8        (new ToolXPresenter(view));
9      o->onClose([this]{ toolx.release(); });
10     toolx = std::move(o);
11   }
12   toolx.show();
13 }
```

SOLUÇÃO ADHOC

```
1  struct MainPresenter {
2    unique_ptr<ToolXPresenter> toolx;
3  };
4
5  void MainPresenter::openToolX() {
6    if(!toolx) {
7      unique_ptr<ToolXPresenter> o
8        (new ToolXPresenter(view));
9      o->onClose([this]{ toolx.release(); });
10     toolx = std::move(o);
11   }
12   toolx.show();
13 }
```

SOLUÇÃO ADHOC

```
1  struct MainPresenter {
2    unique_ptr<ToolXPresenter> toolx;
3  };
4
5  void MainPresenter::openToolX() {
6    if(!toolx) {
7      unique_ptr<ToolXPresenter> o
8        (new ToolXPresenter(view));
9      o->onClose([this]{ toolx.release(); });
10     toolx = std::move(o);
11   }
12   toolx.show();
13 }
```

SOLUÇÃO ADHOC - POR VALOR

```
1  //MainPresenter models SemiRegular
2  struct MainPresenter {
3    ToolXPresenter toolx;
4  };
5
6  void MainPresenter::openToolX() {
7    if(toolx == ToolXPresenter()) {
8      toolx = ToolXPresenter(view);
9      toolx.onClose
10       ([this]{ toolx = ToolXPresenter(); });
11   }
12   toolx->show();
13 }
```

SOLUÇÃO ADHOC - POR VALOR

```
1  //MainPresenter models SemiRegular
2  struct MainPresenter {
3    ToolXPresenter toolx;
4  };
5
6  void MainPresenter::openToolX() {
7    if(toolx == ToolXPresenter()) {
8      toolx = ToolXPresenter(view);
9      toolx.onClose
10       ([this]{ toolx = ToolXPresenter(); });
11   }
12   toolx->show();
13 }
```

SOLUÇÃO ADHOC - POR VALOR

```
1  //MainPresenter models SemiRegular
2  struct MainPresenter {
3    ToolXPresenter toolx;
4  };
5
6  void MainPresenter::openToolX() {
7    if(toolx == ToolXPresenter()) {
8      toolx = ToolXPresenter(view);
9      toolx.onClose
10       ([this]{ toolx = ToolXPresenter(); });
11   }
12   toolx->show();
13 }
```

SOLUÇÃO ADHOC - POR VALOR

```
1  //MainPresenter models SemiRegular
2  struct MainPresenter {
3    ToolXPresenter toolx;
4  };
5
6  void MainPresenter::openToolX() {
7    if(toolx == ToolXPresenter()) {
8      toolx = ToolXPresenter(view);
9      toolx.onClose
10       ([this]{ toolx = ToolXPresenter(); });
11   }
12   toolx->show();
13 }
```

JANELAS FILHAS (COLEÇÃO)

JANELAS FILHAS (COLEÇÃO)

```
1  struct ToolXPresenter {
2    ToolXPresenter(view) {}
3    //Não posso alocar ToolXPresenter na pilha
4    void close() { delete this; }
5  };
6
7  void MainPresenter::openToolX() {
8    //Memory leak?! Quem é o dono?
9    auto toolx = new ToolXPresenter(view);
10   toox->show();
11 }
```

JANELAS FILHAS (COLEÇÃO)

```
1  struct ToolXPresenter {
2    ToolXPresenter(view) {}
3    //Não posso alocar ToolXPresenter na pilha
4    void close() { delete this; }
5  };
6
7  void MainPresenter::openToolX() {
8    //Memory leak?! Quem é o dono?
9    auto toolx = new ToolXPresenter(view);
10   toox->show();
11 }
```

SOLUÇÃO ADHOC

```
1  struct MainPresenter {
2    list<unique_ptr<ToolXPresenter>> toolxs;
3  };
4
5  void MainPresenter::openToolX() {
6    auto it = toolxs.emplace
7      (toolxs.end(), new ToolXPresenter(view));
8    it->get()->onClose
9      ([this, it]{ toolxs.erase(it); });
10   it->get()->show();
11 }
```

SOLUÇÃO ADHOC

```
1  struct MainPresenter {
2    list<unique_ptr<ToolXPresenter>> toolxs;
3  };
4
5  void MainPresenter::openToolX() {
6    auto it = toolxs.emplace
7      (toolxs.end(), new ToolXPresenter(view));
8    it->get()->onClose
9      ([this, it]{ toolxs.erase(it); });
10   it->get()->show();
11 }
```

SOLUÇÃO ADHOC

```
1  struct MainPresenter {
2    list<unique_ptr<ToolXPresenter>> toolxs;
3  };
4
5  void MainPresenter::openToolX() {
6    auto it = toolxs.emplace
7      (toolxs.end(), new ToolXPresenter(view));
8    it->get()->onClose
9      ([this, it]{ toolxs.erase(it); });
10   it->get()->show();
11 }
```

SOLUÇÃO ADHOC

```
1  struct MainPresenter {
2    list<unique_ptr<ToolXPresenter>> toolxs;
3  };
4
5  void MainPresenter::openToolX() {
6    auto it = toolxs.emplace
7      (toolxs.end(), new ToolXPresenter(view));
8    it->get()->onClose
9      ([this, it]{ toolxs.erase(it); });
10   it->get()->show();
11 }
```

SOLUÇÃO GENÉRICA

SINGLE - ALOCAÇÃO DINÂMICA - IMPLEMENTAÇÃO

```
1  template<typename Presenter>
2  class dyn_presenter {
3    std::unique_ptr<Presenter> _presenter;
4  public:
5    dyn_presenter() = default;
6
7    Presenter* operator->() const noexcept
8    { return _presenter.get(); }
9
10   template<typename... Args>
11   Presenter& instance(Args&&... args) {
12     if(!_presenter) {
13       std::unique_ptr<Presenter> o(
14           new Presenter(std::forward<Args>(args)...));
15       o->onClose([this]{ _presenter.release(); });
16       _presenter = std::move(o);
17     }
18     return *_presenter;
19   }
20 };
```

SINGLE - ALOCAÇÃO DINÂMICA - USO

```
1 struct MainPresenter {
2   dyn_presenter<ToolXPresenter> toolx;
3 };
4
5 void MainPresenter::openToolX()
6 { toolx.instance(view).show(); }
```

SINGLE - ALOCAÇÃO DINÂMICA - USO

```
1 struct MainPresenter {
2   dyn_presenter<ToolXPresenter> toolx;
3 };
4
5 void MainPresenter::openToolX()
6 { toolx.instance(view).show(); }
```

SINGLE - POR VALOR - USO

```
1 struct MainPresenter {
2   presenter<ToolXPresenter> toolx;
3 };
4
5 void MainPresenter::openToolX()
6 { SACI_SINGLE_OF(toolx, view).show(); }
```

COLEÇÃO - ALOCAÇÃO DINÂMICA - IMPLEMENTAÇÃO

```
1  template<typename Presenter>
2  class dyn_presenters {
3    std::list<std::unique_ptr<Presenter>> _presenters;
4  public:
5    dyn_presenters() = default;
6
7    template<typename... Args>
8    Presenter& instance(Args&&... args) {
9      auto it = _presenters.emplace
10       (_presenters.end(),
11        new Presenter(std::forward<Args>(args)...));
12     it->get()->onClose([this, it]{ _presenters.erase(it); });
13     return **it;
14   }
15 };
```

COLEÇÃO - ALOCAÇÃO DINÂMICA - USO

```
1 struct MainPresenter {
2   dyn_presenters<ToolXPresenter> toolxs;
3 };
4
5 void MainPresenter::openToolX()
6 { toolxs.instance(view).show(); }
```

COLEÇÃO - POR VALOR - USO

```
1 struct MainPresenter {
2   presenters<ToolXPresenter> toolxs;
3 };
4
5 void MainPresenter::openToolX()
6 { toolxs.instance(view).show(); }
```

CONCLUSÃO

GUIDELINE

→ Classe não cria instâncias próprias

GUIDELINE

→ Classe não cria instâncias próprias

→ Classe não é responsável por sua destruição

GUIDELINE

- → Classe não cria instâncias próprias
- → Classe não é responsável por sua destruição
- → Classe não assume se será instanciada dinamicamente ou não

GUIDELINE

- → Classe não cria instâncias próprias
- → Classe não é responsável por sua destruição
- → Classe não assume se será instanciada dinamicamente ou não
- → Procure energicamente por um pai(dono) para o objeto

OBRIGADO

- Apresentação
  - github.com/ricardocosme/presentations
- Solução genérica
  - github.com/ricardocosme/saci