INF011 – Padrões de Projeto

14 – *Flyweight*

Sandro Santos Andrade
sandroandrade@ifba.edu.br

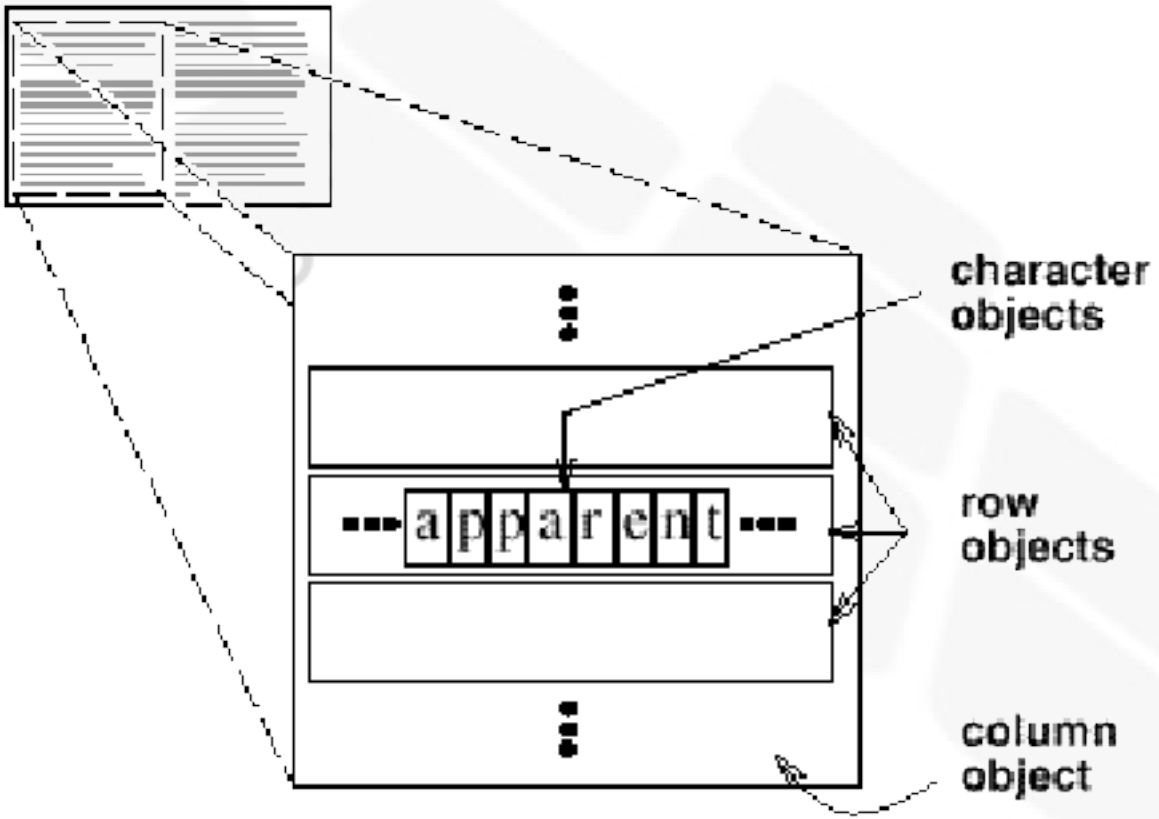
Instituto Federal de Educação, Ciência e Tecnologia da Bahia
Departamento de Tecnologia Eletro-Eletrônica
Graduação Tecnológica em Análise e Desenvolvimento de Sistemas

Flyweight

- Propósito:
  - Utilizar compartilhamento para suportar eficientemente um grande número de objetos com estado reduzido
- Motivação:
  - Em um editor de textos certas vantagens são obtidas ao representar cada caractere como objeto
  - Entretanto, seria necessário a instanciação de um número consideravelmente alto de objetos, com consumo excessivo de memória e *overhead* de execução

Flyweight

- Motivação:

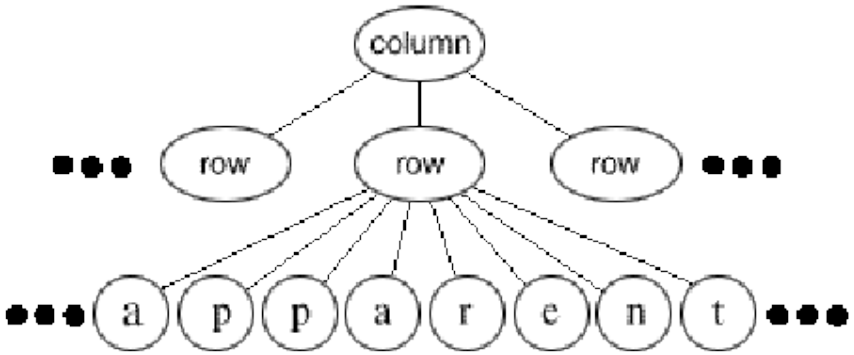


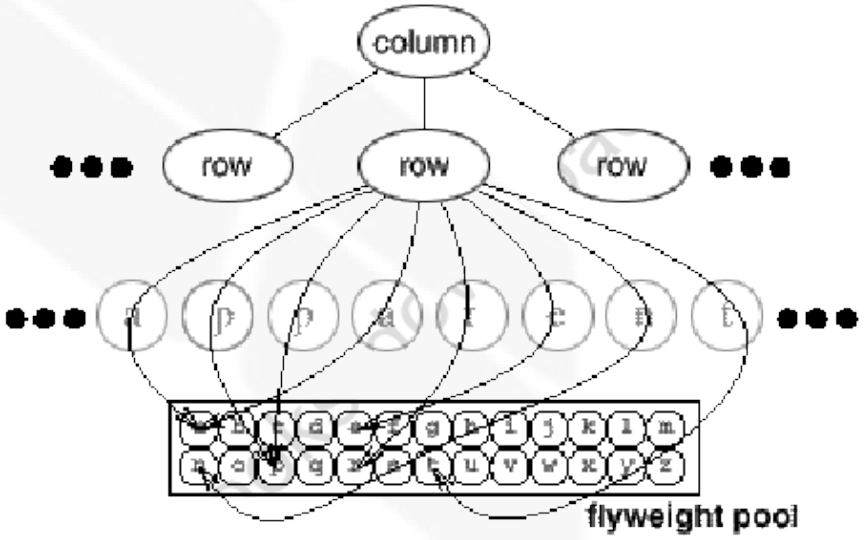
Flyweight

- Motivação:
  - Um *flyweight* é um objeto compartilhado que pode ser utilizado simultaneamente em múltiplos contextos
  - Atua como um objeto independente em cada contexto, de modo que os clientes não são cientes do compartilhamento
  - O conceito chave é a distinção entre **estado intrínseco** e **estado extrínseco**:
    - O estado intrínseco é armazenado no *flyweight* e consiste de informações independentes do contexto e, portanto, compartilháveis
    - O estado extrínseco depende de e varia com o contexto e, portanto, não pode ser compartilhado. Os clientes são responsáveis por passar o estado extrínseco para o *flyweight*, quando necessário

Flyweight

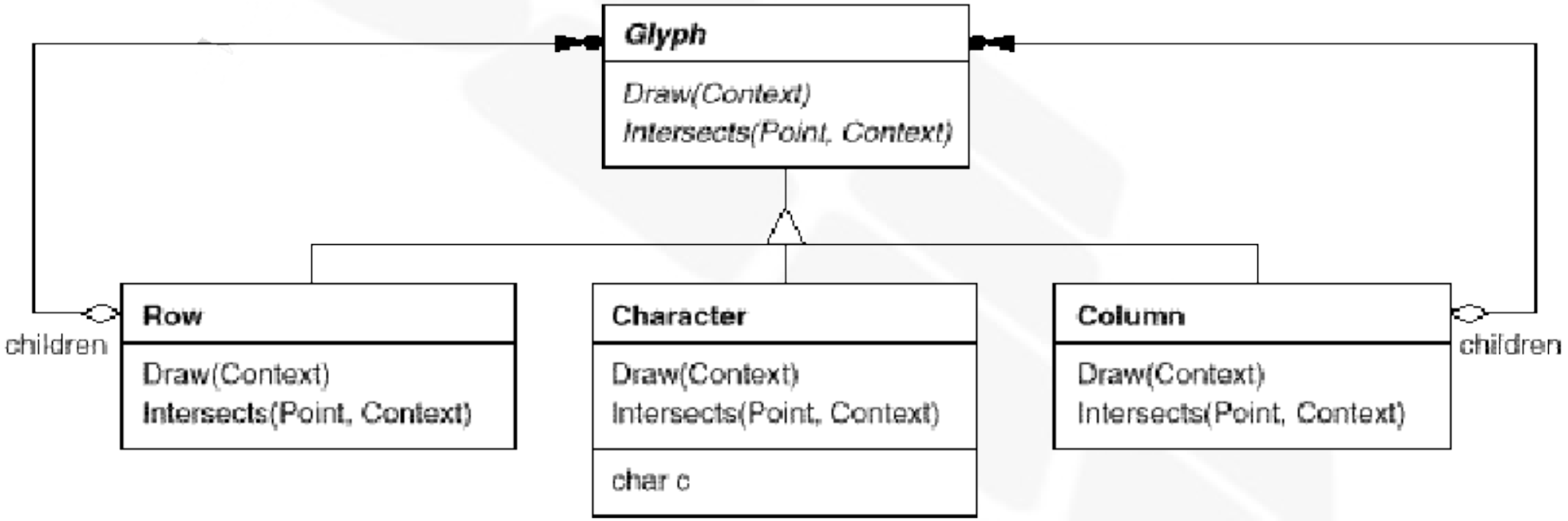
- Motivação:
  - Ex: *flyweight* para representação de cada caractere do texto:
    - Estado intrínseco: código do caractere
    - Estado extrínseco: posição no texto e estilo tipográfico





Flyweight

- Motivação:

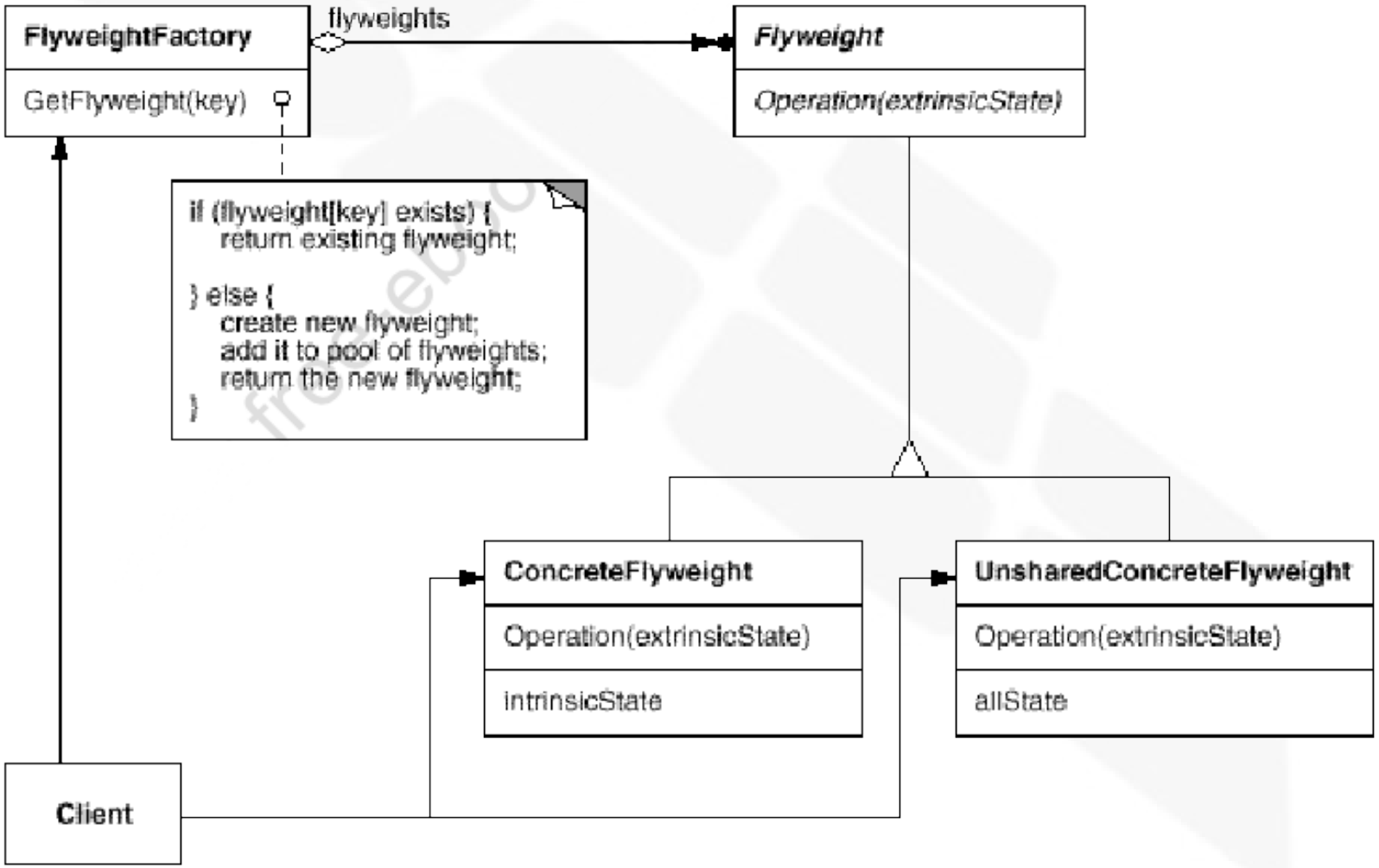


Flyweight

- Aplicabilidade:
  - A aplicação usa um grande número de objetos
  - Custos de armazenamento são altos devido ao grande número de objetos
  - O estado da maior parte dos objetos pode ser definida de forma extrínseca
  - Se o estado extrínseco for removido muitos grupos de objetos podem ser substituídos por um número relativamente pequeno de objetos compartilhados
  - A aplicação não depende da identidade dos objetos – testes de identidade retornarão verdadeiro para objetos conceitualmente diferentes

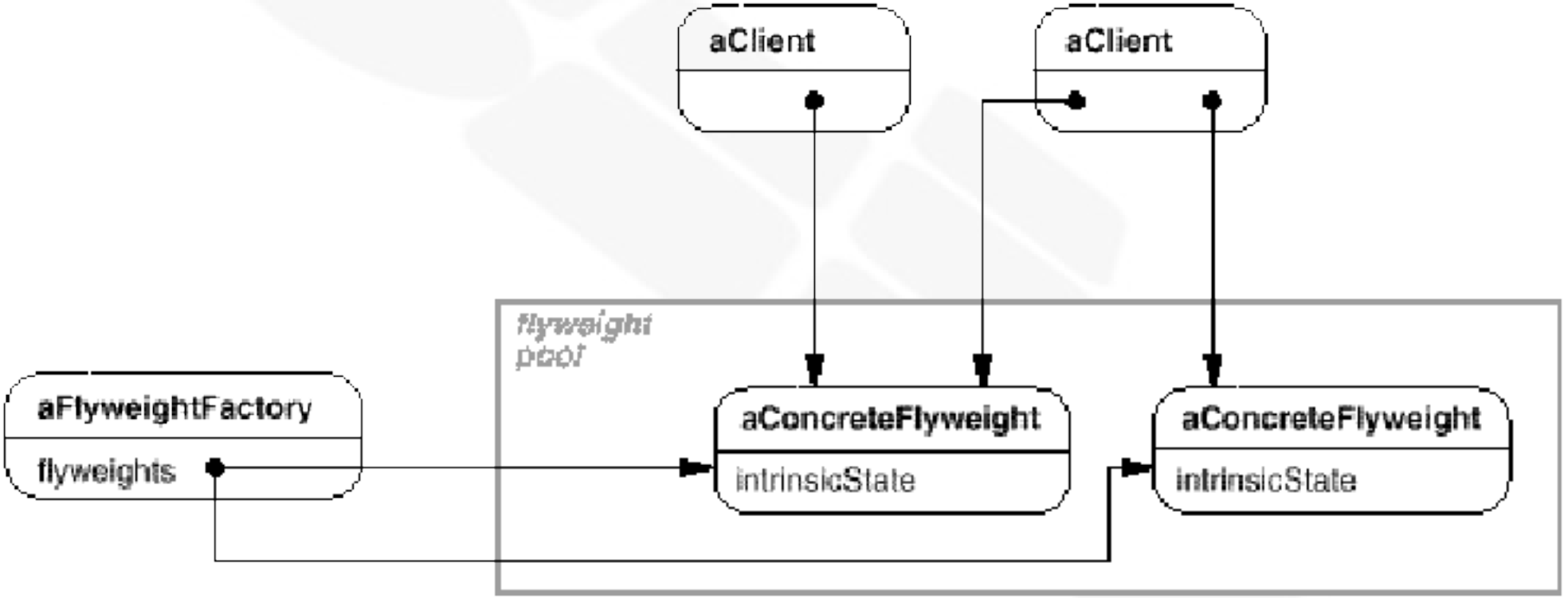
Flyweight

- Estrutura:



Flyweight

- Estrutura:



Flyweight

- Participantes:
  - *Flyweight*:
    - Declara uma interface através da qual *flyweights* podem receber estado extrínseco e atuar com base nele
  - *ConcreteFlyweight* (Caractere):
    - Implemente a interface *Flyweight* e adiciona o armazenamento do estado intrínseco, se existir. O objeto deve ser compartilhável – qualquer estado que ele armazenar deve ser independente do contexto do *ConcreteFlyweight*

Flyweight

- Participantes:
  - *UnsharedConcreteFlyweight* (Row, Column):
    - Nem todas as sub-classes de *Flyweight* precisam ser compartilhadas
    - É comum que objetos *UnsharedConcreteFlyweight* contenham, como filhos, objetos *ConcreteFlyweight* de algum nível da hierarquia de *Flyweights*
  - *FlyWeightFactory*:
    - Cria e gerencia objetos *Flyweight*
    - Garante que os *Flyweights* são compartilhados de forma apropriada. Quando o cliente solicita um *Flyweight* ele devolve um já existente ou cria um novo
  - *Client*:
    - Mantém referências a *Flyweights*
    - Calcula ou armazena o estado extrínseco dos *Flyweights*

Flyweight

- Colaborações:
  - O estado de que um *Flyweight* precisa para funcionar deve ser classificado como intrínseco ou extrínseco:
    - Estado intrínseco é armazenado no *ConcreteFlyweight*
    - Estado extrínseco é armazenado ou computado pelo cliente, que repassa essa informação ao *Flyweight* ao invocar suas operações
  - Clientes não devem instanciar *ConcreteFlyweights* diretamente e sim através do *FlyweightFactory*, garantindo dessa forma o compartilhamento apropriado

Flyweight

- Consequências:
  - Podem adicionar custos de *run-time*, devido à transferência, descoberta ou computação do estado extrínseco
  - Tais custos são, entretanto, compensados pelas economias no armazenamento, que é influenciado por alguns fatores:
    - Redução do número total de instâncias propiciada pelo compartilhamento
    - Tamanho do estado intrínseco por objeto
    - Forma de obtenção do estado extrínseco (armazenado ou calculado)

Flyweight

- Consequências:
  - Quanto maior o compartilhamento maior a economia no armazenamento
  - Quanto maior o tamanho do estado intrínseco (compartilhado) do objeto maior a economia no armazenamento
  - Esta economia é ainda maior quando o estado extrínseco é computado ao invés de armazenado

Flyweight

- Implementação:
  - Reduzindo o estado extrínseco:
    - Remover estado extrínseco não irá ajudar a reduzir o custo de armazenamento se existirem tantos tipos diferentes de estado quanto o número de instâncias antes do compartilhamento
    - O estado extrínseco pode ser computado a partir de uma estrutura separada, com menores demandas de armazenamento:
      - Ex: A fonte e o estilo de cada caracter do texto podem ser armazenados em um mapa separado que rastreia os caracteres com os mesmos atributos tipográficos
      - Como os textos utilizam uma quantidade menor de fontes e estilos do que caracteres diminui-se a demanda por armazenamento

Flyweight

- Implementação:
  - Gerenciando objetos compartilhados:
    - Clientes não devem instanciar objetos diretamente e sim através do *FlyweightFactory*
    - Geralmente a fábrica utiliza um *container* associativo (ex: mapa cuja chave é o caractere e o valor é o *Flyweight*)
    - Pode ser necessário *reference counting* ou *garbage collection* quando o número de *Flyweights* não é fixo e não é pequeno

Flyweight

- Código exemplo:

```
class Glyph {
public:
    virtual ~Glyph();

    virtual void Draw(Window*, GlyphContext&);

    virtual void SetFont(Font*, GlyphContext&);
    virtual Font* GetFont(GlyphContext&);

    virtual void First(GlyphContext&);
    virtual void Next(GlyphContext&);
    virtual bool IsDone(GlyphContext&);
    virtual Glyph* Current(GlyphContext&);

    virtual void Insert(Glyph*, GlyphContext&);
    virtual void Remove(GlyphContext&);
protected:
    Glyph();
};
```

Flyweight

- Código exemplo:

```
class Character : public Glyph {
public:
    Character(char);

    virtual void Draw(Window*, GlyphContext&);
private:
    char _charcode;
};
```

Flyweight

- Código exemplo:

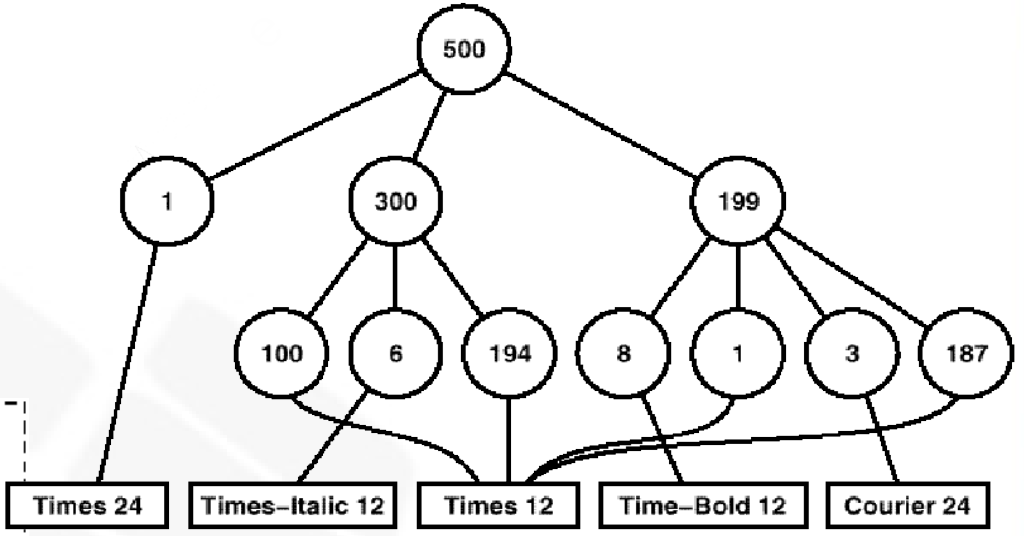
```
class GlyphContext {
public:
    GlyphContext();
    virtual ~GlyphContext();

    virtual void Next(int step = 1);
    virtual void Insert(int quantity = 1);

    virtual Font* GetFont();
    virtual void SetFont(Font*, int span = 1);
private:
    int _index;
    BTree* _fonts;
};
```

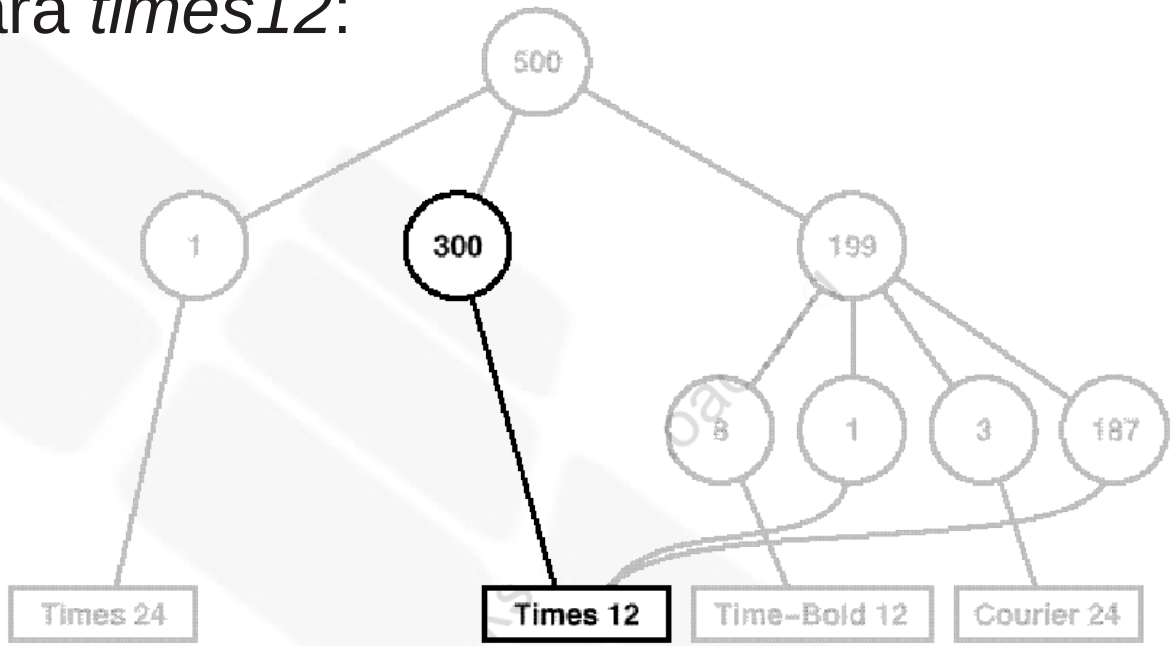
Flyweight

- Código exemplo:



Flyweight

- Código exemplo:
  - Modificando *expect* para *times12*:

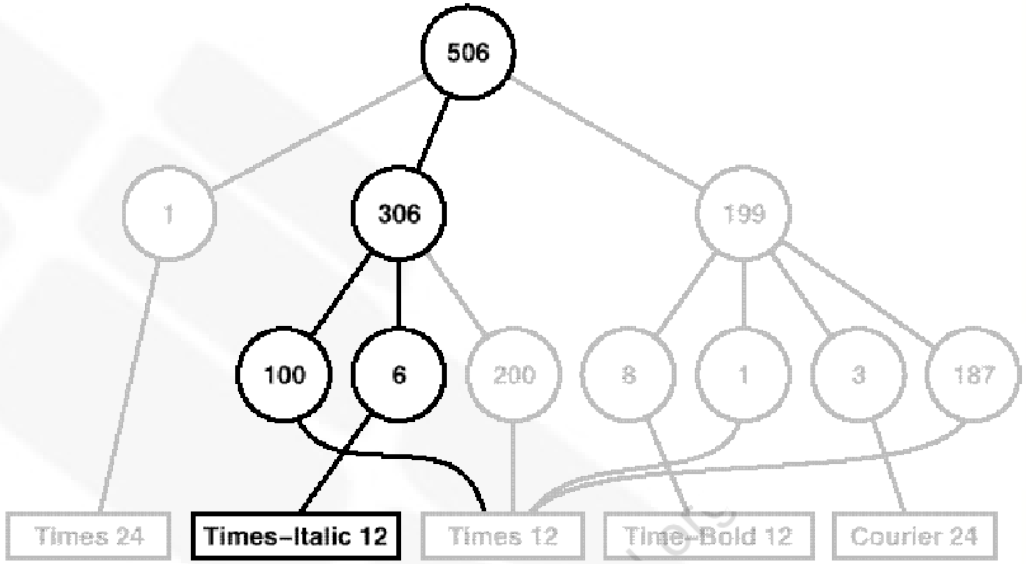



```
GlyphContext gc;
Font* times12 = new Font("Times-Roman-12");
Font* timesItalic12 = new Font("Times-Italic-12");
// ...

gc.SetFont(times12, 6);
```

Flyweight

- Código exemplo:
  - Adicionando “don't ” em *times-italic12* antes de *expect*




```
gc.Insert(6);
gc.SetFont(timesItalic12, 6);
```

Flyweight

- Código exemplo:

```
const int NCHARCODES = 128;

class GlyphFactory {
public:
    GlyphFactory();
    virtual ~GlyphFactory();

    virtual Character* CreateCharacter(char);
    virtual Row* CreateRow();
    virtual Column* CreateColumn();
    // ...
private:
    Character* _character[NCHARCODES];
};
```

Flyweight

- Código exemplo:

```
GlyphFactory::GlyphFactory () {
    for (int i = 0; i < NCHARCODES; ++i) {
        _character[i] = 0;
    }
}
```

```
Character* GlyphFactory::CreateCharacter (char c) {
    if (!_character[c]) {
        _character[c] = new Character(c);
    }

    return _character[c];
}
```

```
Row* GlyphFactory::CreateRow () {
    return new Row;
}

Column* GlyphFactory::CreateColumn () {
    return new Column;
}
```

Flyweight

- Usos conhecidos:
  - *InterViews* 3.0
  - Doc
  - ET++

Flyweight

- Padrões relacionados:
  - Frequentemente combinado com um *Composite*
  - Frequentemente é melhor implementar os padrões *State* e *Strategy* como *Flyweights*

INF011 – Padrões de Projeto

14 – *Flyweight*

Sandro Santos Andrade
sandroandrade@ifba.edu.br

Instituto Federal de Educação, Ciência e Tecnologia da Bahia
Departamento de Tecnologia Eletro-Eletrônica
Graduação Tecnológica em Análise e Desenvolvimento de Sistemas