# Flyweight

# Flyweight - Motivace

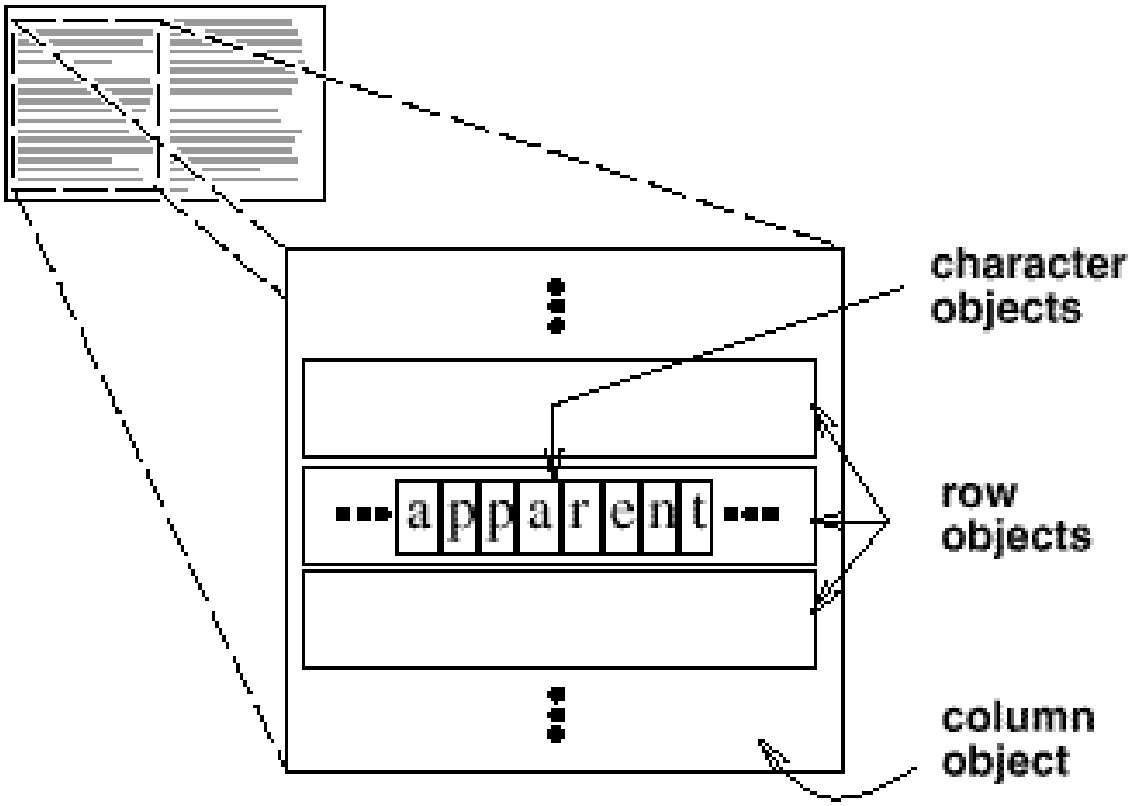
- **Vytvořit textový editor s podporou množství netradičních znakových sad**
  - snadno rozšiřitelný o nové znakové sady
- **Řešení:**
  - abstraktní třída Glyph
  - metody draw(), setColor(), setFont(), setSize() ...
  - Každá znaková sada realizována pomocí konkrétního potomka
  - Informace, o jaký znak dané sady se jedná, bude uložena v atributech objektu
- **Výhoda:**
  - i obrázky mohou být podtřídou Glyph
  - jednotné zacházení
- **Problém:**
  - pro každé písmeno musíme vytvořit objekt
  - paměťově náročné

Kleopátra névgyűrűje (kartusa)

# Flyweight – Motivace

Textový editor využívající objekty

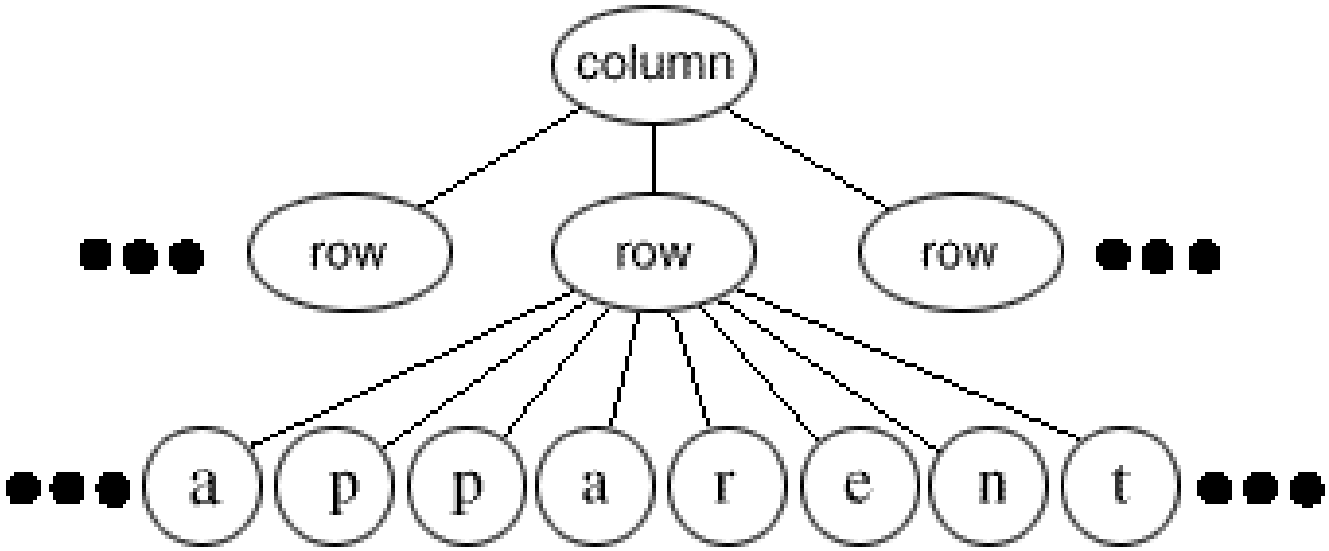
naivní implementace je příliš paměťově náročná



# Flyweight – Muší váha

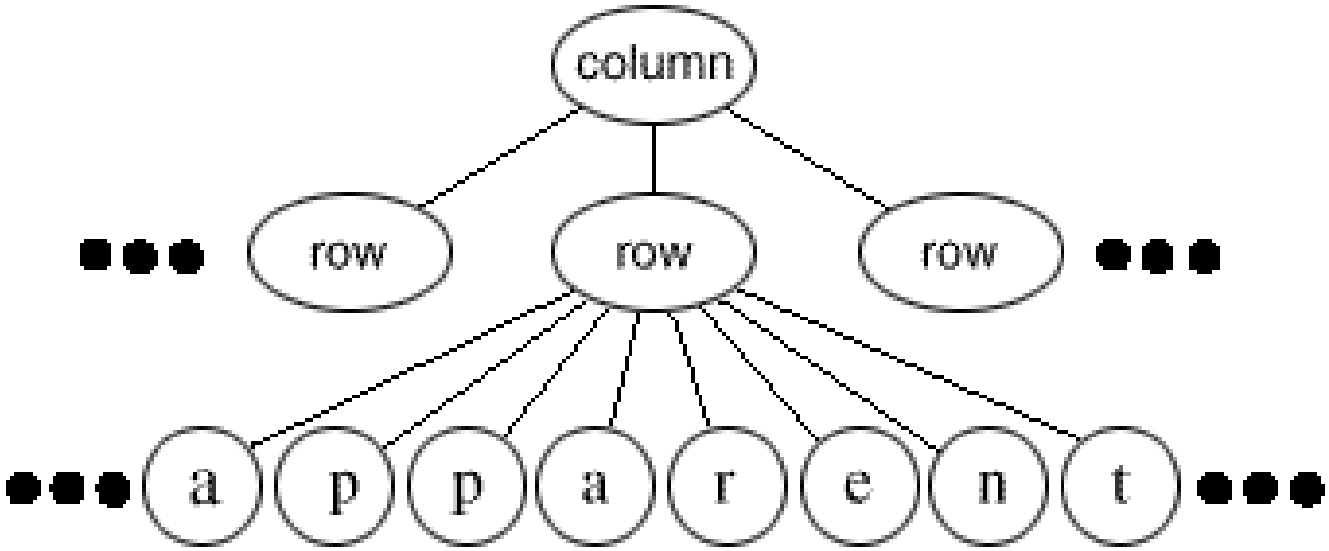
- **Účel**
  - kategorie: strukturální vzory
  - umožňuje úsporu paměti
  - účinná podpora většího počtu jednoduchých (fine-grained) objektů
- **Motivace**
  - textový editor ve kterém jsou všechny elementy objekty
  - řádkové objekty, sloupcové objekty, každý znak je objektem
  - naivní řešení je příliš náročné na paměť
  - je potřeba objekty stejného druhu vytvořit jen jednou a sdílet
- **Použitelnost**
  - aplikace používá velký počet objektů
  - náklady na úložný prostor jsou vysoké z důvodu kvantity objektů
  - část objektového stavu lze učinit externím
  - větší počet skupin objektů lze nahradit relativně malým počtem sdílených objektů, jakmile se odstraní externí stav
  - aplikace nezávisí na objektové totožnosti

# Flyweight – struktura textového editoru



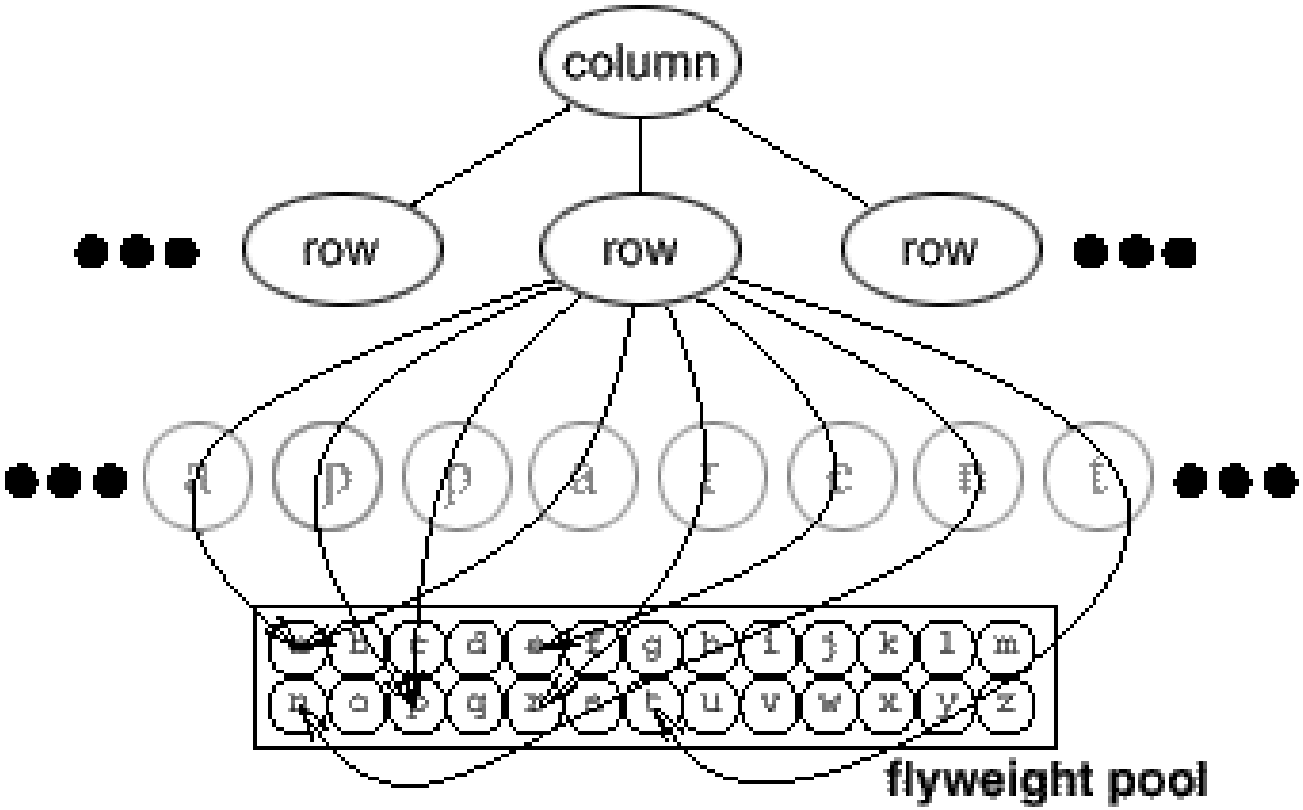

Takto vypadá naivní implementace textového editoru pomocí objektů

# Flyweight – struktura textového editoru




… a takto vypadá logická struktura objektů při použití Flyweight

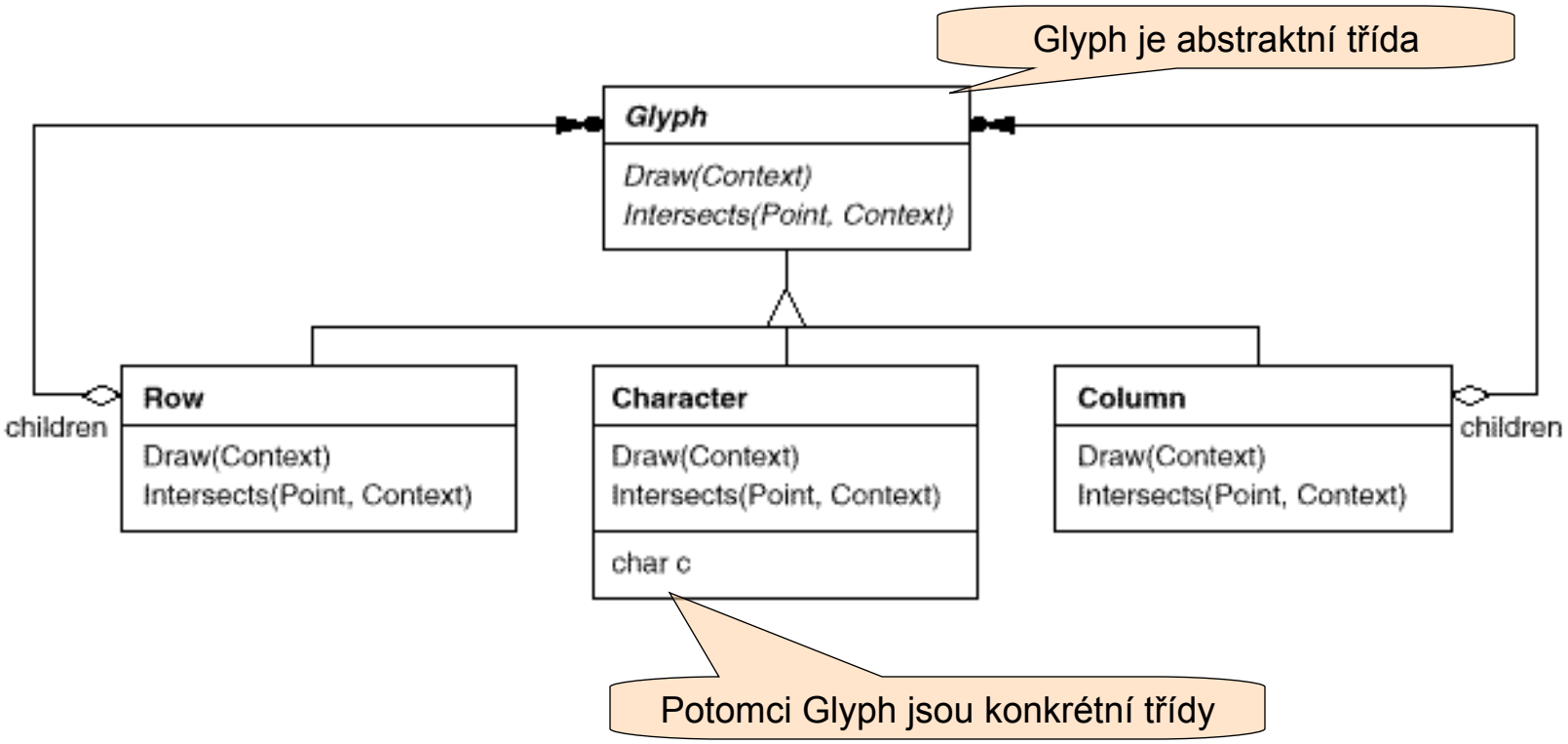
# Flyweight – struktura textového editoru




Fyzická struktura při využití muší váhy

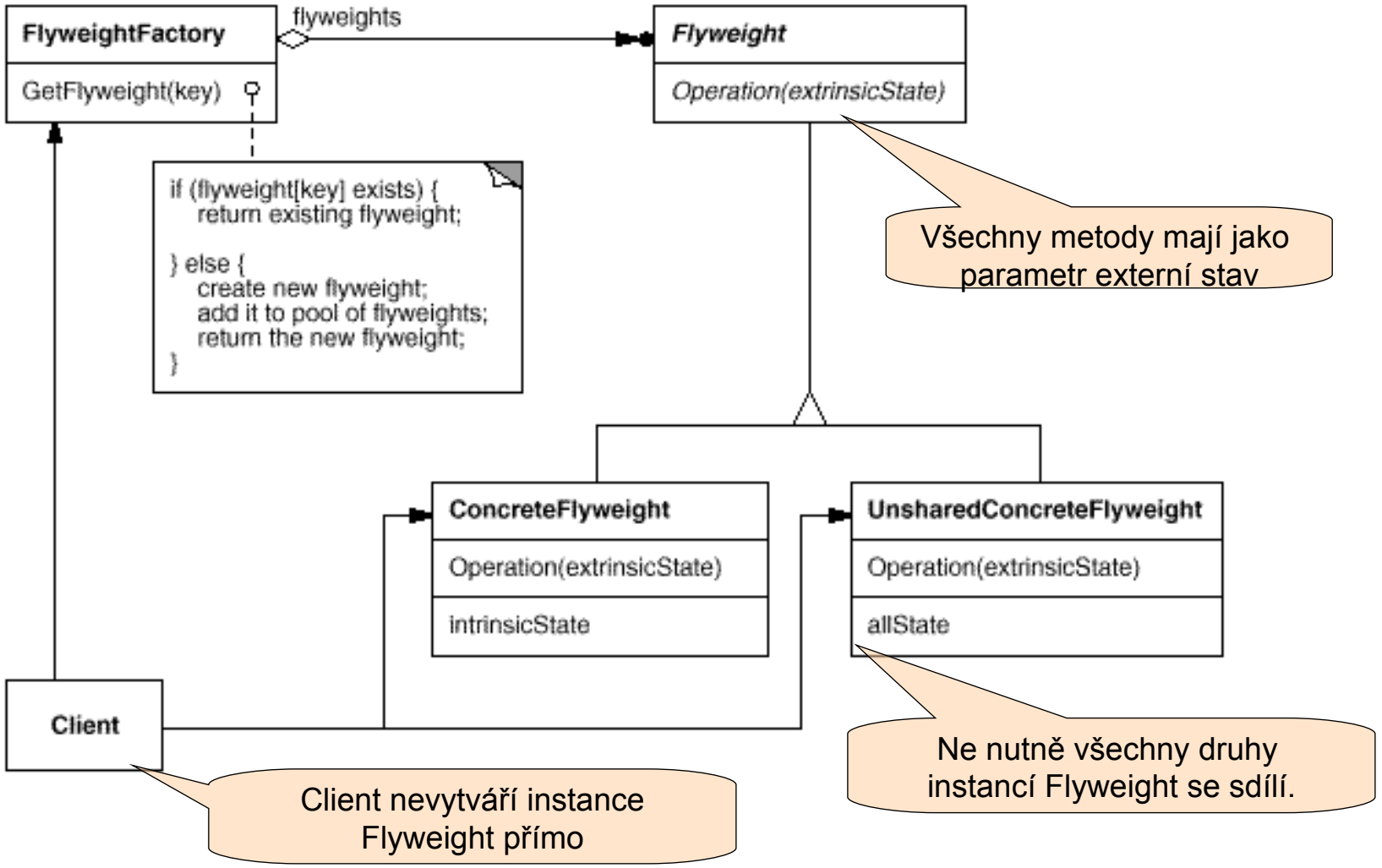
# Flyweight - struktura

- **Struktura hierarchie vytvářených objektů**



# Flyweight - struktura

- **Struktura vztahů mezi třídami**

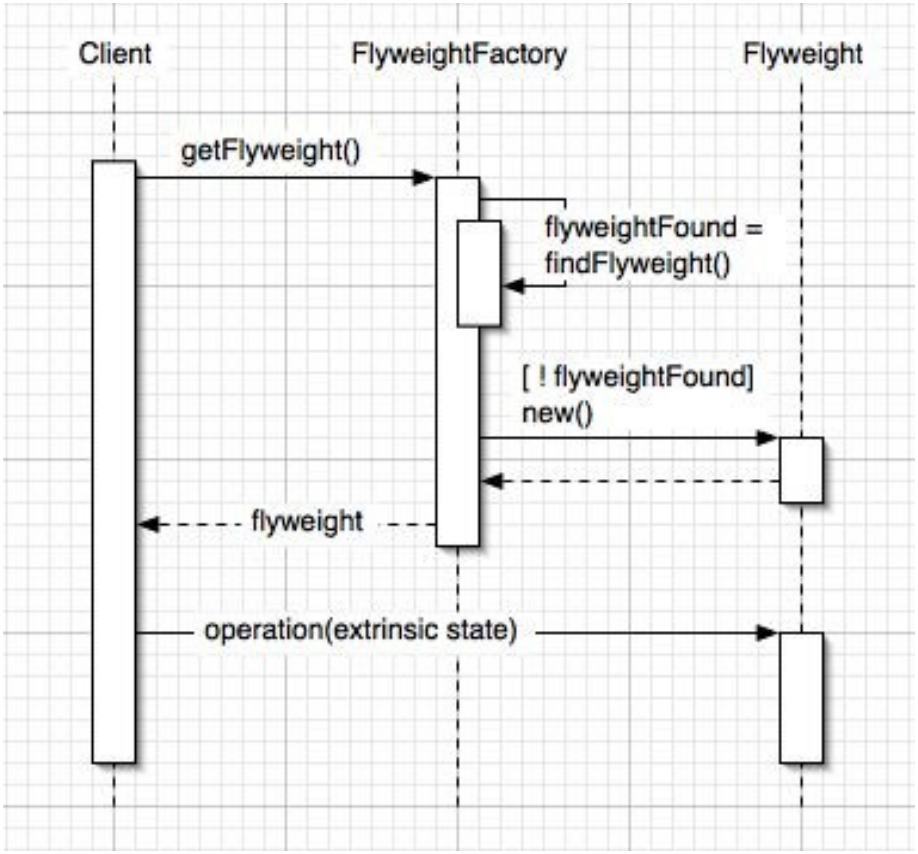


# Flyweight - struktura

- **Účastníci**
  - Flyweight (Glyph)
    - deklaruje rozhraní, které pracuje s vnějším (extrinsic) stavem
  - ConcreteFlyweight (Character)
    - implementuje rozhraní Flyweight a přidává vnitřní (intrinsic) stav
  - UnsharedConcreteFlyweight (Row, Column)
    - všechny podtřídy Flyweight se sdílet nemusí
    - jedná se o jedinečné objekty, kterých není mnoho
  - FlyweightFactory
    - vytváří a spravuje objekty muších vah
    - zajišťuje řádné sdílení muších vah
  - Client
    - Udržuje odkaz na muší váhu či váhy
    - počítá nebo ukládá vnější stav muší váhy či vah

# Flyweight – pořadí volání metod

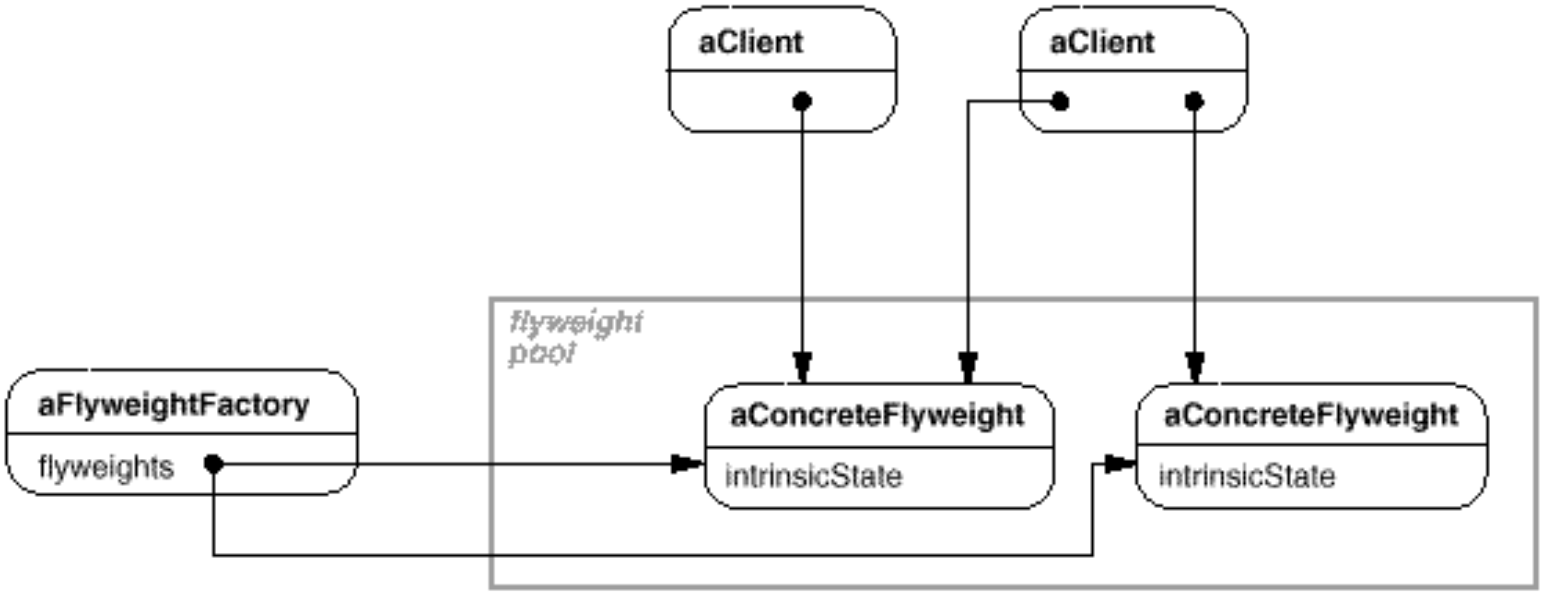
- **Třída FlyweightFactory vytváří instance potomků Flyweight**
- **Pokud už byla požadovaná instance vytvořena v minulosti, nová instance se nevytváří a továrna vrátí referenci na již vytvořenou instanci**
- **Client nesmí instance Flyweight vytvářet přímo, aby bylo zajištěné řádné sdílení**
- **Správce objektů Flyweight je Client, který musí oznámit továrně až objekty přestane využívat**
- **Objekty Flyweight by měly být neměnné z pohledu Clienta**
- **Client spravuje externí stav muší váhy a předává ho při volání metod**



# Flyweight - souvislosti

- **Souvislosti**
  - objekty Flyweight mají dva druhy stavů – vnitřní a vnější
    - vnitřní stav se ukládá do objektu `ConcreteFlyweight`
    - vnější stav ukládají nebo počítají objekty `Client`
  - klienti by neměli tvořit instance `ConcreteFlyweight` přímo
    - je potřeba si `Flyweight` vyžádat od `FlyweightFactory`
    - tím se zajistí řádné sdílení objektů

# Flyweight - souvislosti




Továrna pro muší váhy je často implementována jako Singleton.
Prostřednictvím jedné továrny mohou být muší váhy sdíleny i mezi více klienty.

# Flyweight - Známé použití

- **Java – řetězce během kompilace**
- **Java – rámečky komponent**
- **textový či jiný editor**

# Java – Swing borders

```
Border border = BorderFactory.createRaisedBevelBorder();
Border border2 = BorderFactory.createRaisedBevelBorder();

if(border == border2)
    System.out.println("bevel borders are shared");
else
    System.out.println("bevel borders are NOT shared");
```

```
public void paintBorder(Component c,
    Graphics g,
    int x,
    int y,
    int width,
    int height)
```

# Flyweight - implementace

- **Implementace**
  - Odstranění vnějšího stavu
    - je potřeba aby vnější stav byl oddělitelný
    - vnější stav bude ve struktuře s menšími požadavky na prostor
  - Správa sdílených objektů
    - Client nevytváří instance objektů přímo
    - vytváření pomocí FlyweightFactory
    - továrna umožní korektní sdílení objektů
    - nevyužívané sdílené instance je třeba rušit

# Flyweight – příklad

- **Nakreslení 10 000 linek v různých barvách**
  - Řešení využívající Flyweight:

```
Graphics g = canvas.getGraphics();
for(int i=0; i < 10000; ++i) {
     Line line = LineFactory.getLine(getRandomColor());
     line.draw(g, getRandomX(), getRandomY(), getRandomX(), getRandomY());
}
```

```
public class LineFactory {
     private static final HashMap linesByColor = new HashMap();
     public static Line getLine(Color color) {
          Line line = (Line)linesByColor.get(color);
          if(line == null) {
               line = new Line(color);
               linesByColor.put(color, line);
          }
          return line;
     }
}
```

```
public class Line {
     private Color color;
     public Line(Color color) {
          this.color = color;
     }
     public void draw(Graphics g, int x, int y, int x2, int y2) {
          g.setColor(color);
          g.drawLine(x, y, x2, y2);
     }
}
```

# Flyweight - související NV

- **Související NV**
  - Composite
    - často se Flyweight používá k implementaci logicky hierarchické struktury vyjádřené acyklickým grafem se sdílenými listovými uzly
  - State
    - často implementováno pomocí Flyweight
  - Strategy
    - často implementováno pomocí Flyweight