CSCI 253

Object Oriented Design:
Flyweight Pattern
George Blankenship

Overview

Creational Patterns
- Singleton
- Abstract factory
- Factory Method
- Prototype
- Builder

Structural Patterns
- Composite
- Façade
- Proxy
- Flyweight
- Adapter
- Bridge
- Decorator

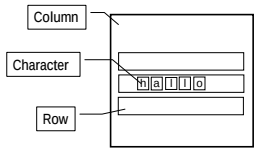
Behavioral Patterns
- Chain of Respons.
- Command
- Interpreter
- Iterator
- Mediator
- Memento
- Observer
- State
- Strategy
- Template Method
- Visitor

The Elements of a Design Pattern

- A pattern name
- The problem that the pattern solves
  - Including conditions for the pattern to be applicable
- The solution to the problem brought by the pattern
  - The elements (classes-objects) involved, their roles, responsibilities, relationships and collaborations
  - Not a particular concrete design or implementation
- The consequences of applying the pattern
  - Time and space trade off
  - Language and implementation issues
  - Effects on flexibility, extensibility, portability

The Flyweight Pattern: The Problem

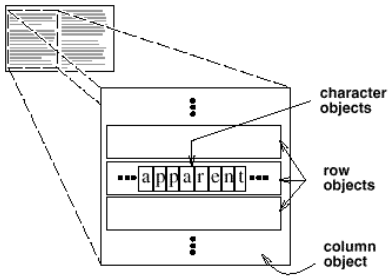
Some applications benefit from using objects in their design but a naïve implementation is prohibitively expensive because of the large number of objects



- use an object for each character in a text document editor
- use a layout object for each widget in a GUI

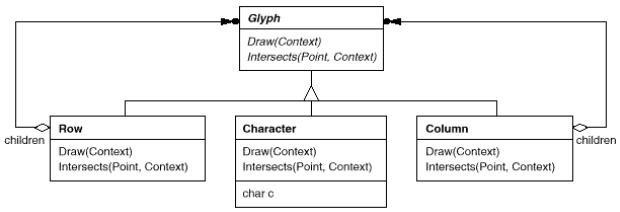
Flyweight Pattern George Blankenship 4

Page Objects



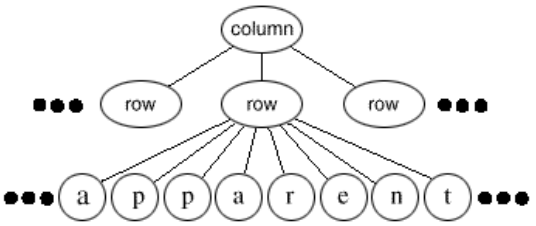
Flyweight Pattern George Blankenship 5

Page Classes

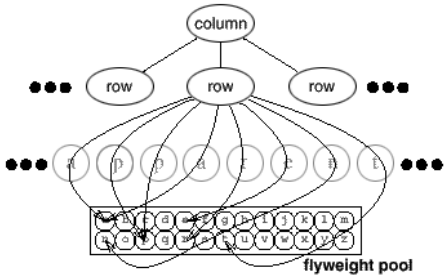


Flyweight Pattern George Blankenship 6

Object Hierarchy



Flyweight Objects



Java String

- Java Strings are flyweighted by the compiler wherever possible
- Can be flyweighted at runtime with the `intern` method

```
public class StringTest {
  public static void main(String[] args) {
    String fly  = "fly", weight  = "weight";
    String fly2 = "fly", weight2 = "weight";

    System.out.println(fly == fly2);                  // true
    System.out.println(weight == weight2);            // true

    String distinctString = fly + weight;
    System.out.println(distinctString == "flyweight"); // false

    String flyweight = (fly + weight).intern();
    System.out.println(flyweight == "flyweight");     // true
  }
}
```

StateVariable

```
public class StateVariable extends ListEntry {
public StateVariable(Trace t, StateMachine f, String s, StateVariable r, long ms) {
public StateMachine getFSM() {return fsm;}
public int getIndex() {return index;}
public String getDescription() {return description;}
public void setTime(long t) {time = t;}      // set start time of state
public long getTime() {return time;}         // fetch start time of state
public long getNormalTimeout() {return normalTimeout;}// fetch normal timeout of state
public void setTimeout(long t) {timeout=t;}  // change the timeout for the state
public long getTimeout() {return timeout;}   // fetch the timeout for the state
public void setMissed(boolean b) {missed = b;}
public boolean wasMissed() {return missed;}  // state missed (timeout occurred)
public boolean setNextState(StateVariable s) {
public StateVariable getNextState() {
public String getShortSummary() {
public String getSummary() {
```

Flyweight Pattern George Blankenship 10

The Flyweight Pattern Applicability

- Apply flyweight when ALL of the following are true:
  - An application uses a large number of objects
  - Storage cost is high because of the quantity of objects
  - Most objects can be made extrinsic
  - Many groups of objects can be replaced by relatively few shared objects once extrinsic state is removed
  - The application does not depend on object identity

Flyweight Pattern George Blankenship 11

The Flyweight Pattern Participants

- Flyweight
  - Declares an interface through which flyweights can receive and act upon extrinsic state
- Concrete Flyweight
  - Implements the flyweight interface and adds storage for intrinsic state
  - A concrete flyweight object must be sharable, i.e. all state must be intrinsic
- Unshared Concrete Flyweight
  - Not all flyweights subclasses need to be shared, unshared concrete flyweight objects have concrete flyweight objects at some level in the flyweight object structure

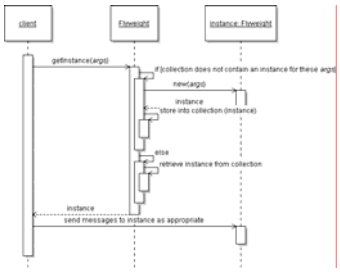
Flyweight Pattern George Blankenship 12

The Flyweight Pattern Participants

- Flyweight Factory
  - Creates and manages flyweight objects
  - Ensures that flyweights are shared properly; when a client requests a flyweight the flyweight factory supplies an existing one from the pool or creates one and adds it to the pool
- Client
  - Maintains a reference to flyweight(s)
  - Computes or stores the extrinsic state of flyweight(s)

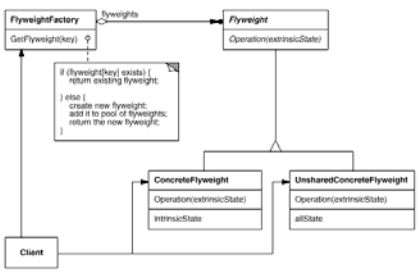
Flyweight Pattern George Blankenship 13

Sequence Diagram



Flyweight Pattern George Blankenship 14

FlyweightFactory



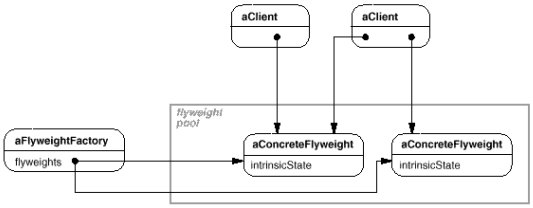
Flyweight Pattern George Blankenship 15

The Flyweight Pattern Collaborations

- State that a flyweight needs to function must be characterised as either intrinsic or extrinsic. Intrinsic state is stored in the concrete flyweight object.; extrinsic state is stored or computed by client objects. Clients pass this state to the flyweight when invoking operations.
- Clients should not instantiate concrete flyweights directly. Clients must obtain concrete flyweight objects exclusively from the flyweight factory object to enshure that they are shared properly

Flyweight Pattern George Blankenship 16

FlyweightFactory Participants



Flyweight Pattern George Blankenship 17

The Flyweight Pattern Consequences

- - Flyweights may introduce run-time costs associated with transferring, finding, and/or computing extrinsic state
- + The increase in run-time cost are offset by storage savings which increase
  - as more flyweights are shared
  - as the amount of intrinsic state is considerable
  - as the amount of extrinsic state is considerable but can be computed
- - The flyweight pattern is often combined with the composite pattern to build a graph with shared leaf nodes. Because of the sharing, leaf nodes cannot store their parent which has a major impact on how the objects in the hierarchy communicate

Flyweight Pattern George Blankenship 18

The Flyweight Pattern Implementation

- Removing extrinsic state:
  - Identify extrinsic state and remove it from the shared objects
  - Removing extrinsic state will not help if there are as many different kinds of extrinsic state as there are objects before sharing
  - Ideally extrinsic state can be computed from a separate object structure with far smaller storage requirements
- Managing shared objects:
  - Use an associative store with the flyweight factory to let clients locate a particular flyweight
  - Some form of reference counting or garbage collection is needed to reclaim a flyweight's storage when it is no longer needed
  - When the number of flyweights is small and fixed, consider to initialise the pool and keep the flyweights around permanently

Flyweight Pattern George Blankenship 19