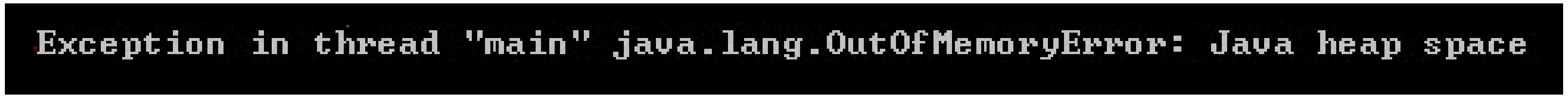
Flyweight

A structural pattern

Learning goals

1. Learn the idea, structure, and Java implementation of the Flyweight design pattern.
2. Learn to apply the Flyweight DP in your own programming.

Idea of Flyweight



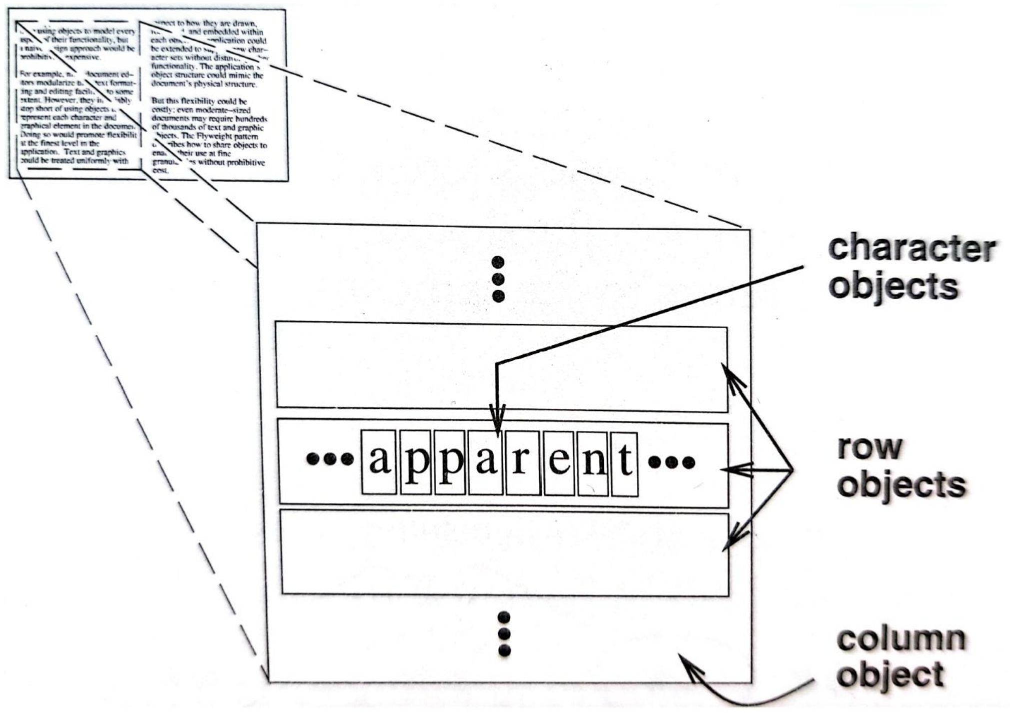
```
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
```

- The purpose of the Flyweight DP is to **optimize memory usage**.
- Consider an application with a very large number of objects that are big in size.
  - In Java VM, the objects are stored in the shared heap memory segment.
  - The large number of large objects eventually makes the VM run out of heap memory.
- Still, some of the objects' instance variable values may be common to several objects.
  - In Flyweight, this shared part of an object's state is identified, and stored only once.
  - The shared part, expressed as shared objects, essentially becomes an object **cache**.

Intrinsic and extrinsic state

- The Flyweight DP relies on the concepts of intrinsic and extrinsic states.
- **Intrinsic State**:
  - Represents internal data shared across many objects.
  - Intrinsic state is immutable after it’s constructed (for safe sharing)
- **Extrinsic State**:
  - Consists of external, changeable states that vary from one object to another.
  - Complements the intrinsic state and makes the object usable in a particular context.

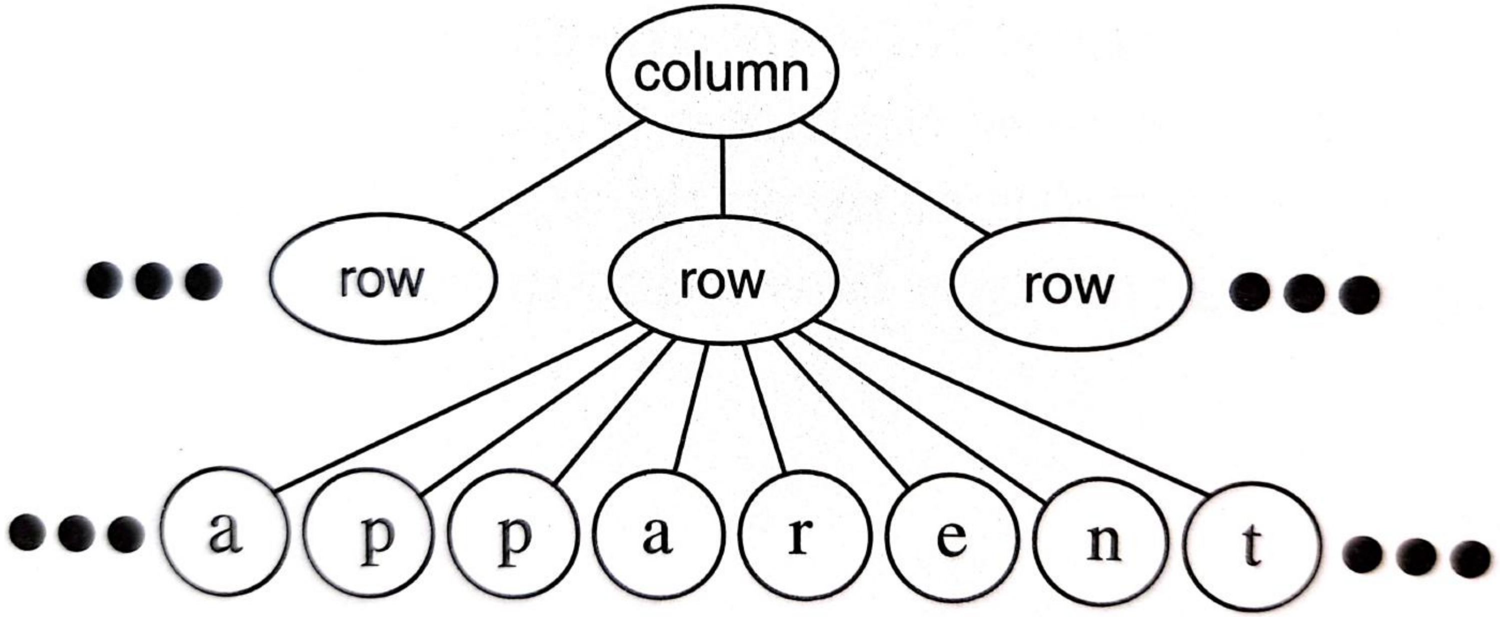
Example: text editor



- In a text editor, the rows, columns, and characters are ideally treated as objects.

Image: Gamma et al., Design Patterns. Elements of Reusable Object-Oriented Software. Addison Wesley Longman (1995), p. 195

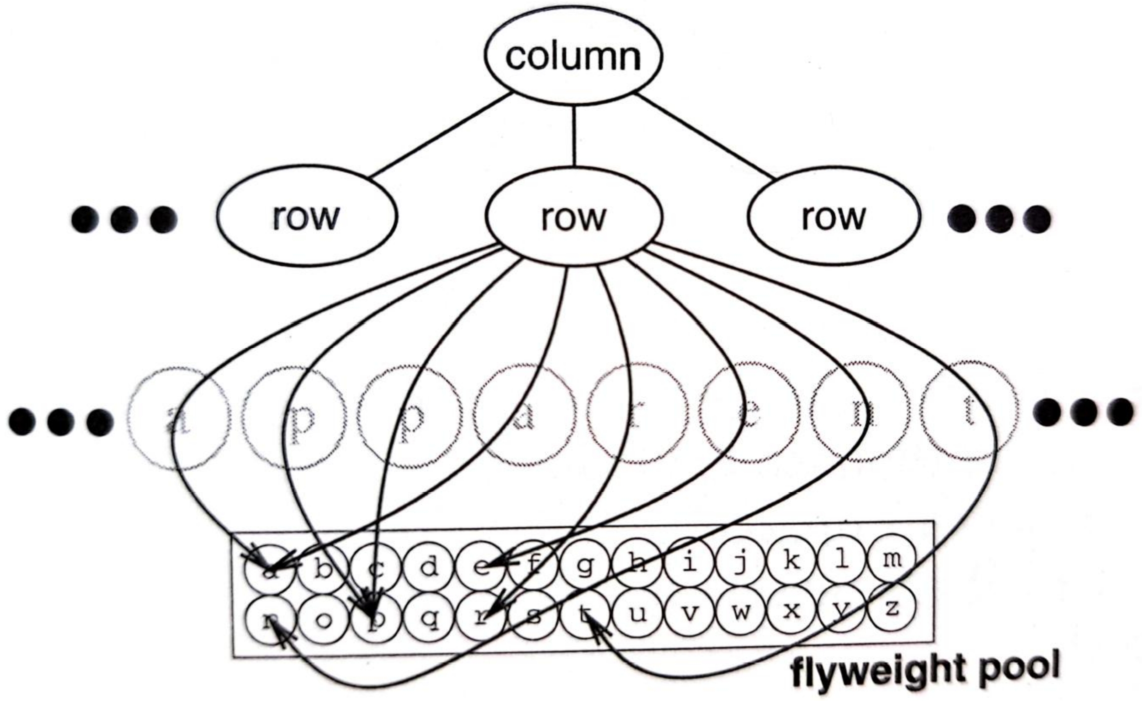
Example: text editor



- Representing each character as an object may pose a problem.
- There is a large number of characters.
- Each character can be big in size, if it contains complex data of the character.

Image: Gamma et al., Design Patterns. Elements of Reusable Object-Oriented Software. Addison Wesley Longman (1995), p. 196

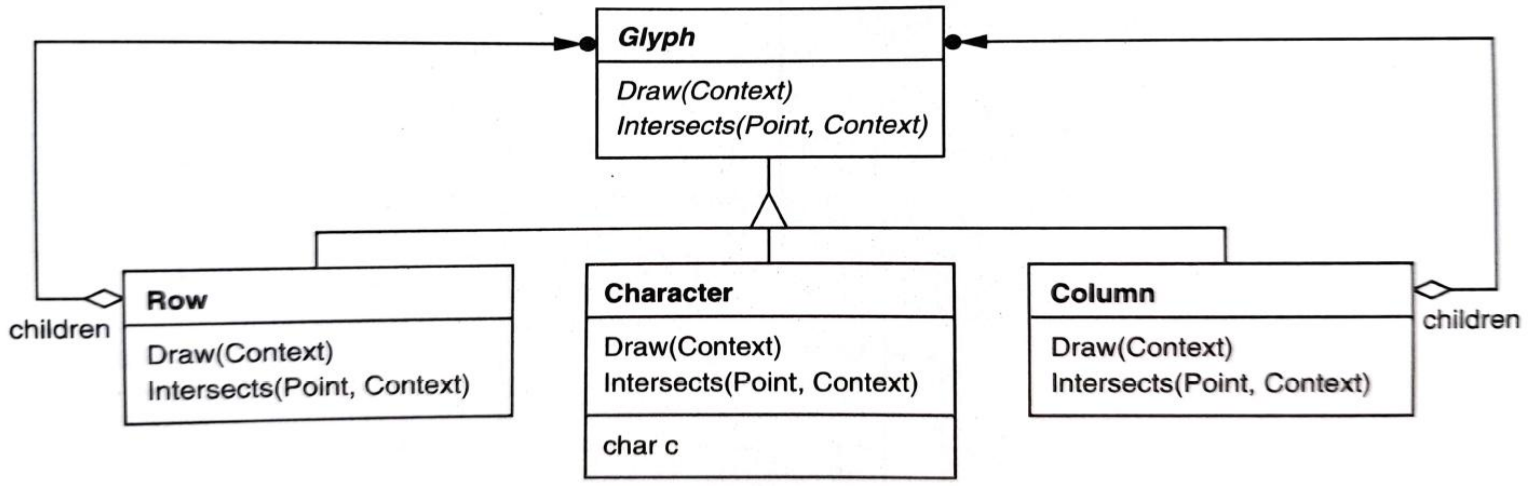
Example: text editor



- Following the idea of the Flyweight DP, a shared pool of character objects is generated.

Image: Gamma et al., Design Patterns. Elements of Reusable Object-Oriented Software. Addison Wesley Longman (1995), p. 196

Example: text editor



Glyph
Draw(Context)
Intersects(Point, Context)
Row
Draw(Context)
Intersects(Point, Context)
children
Character
Draw(Context)
Intersects(Point, Context)
char c
Column
Draw(Context)
Intersects(Point, Context)
children

- The Glyph interface contains all drawable objects as flyweights.
- For characters:
  - the intrinsic state contains the character code.
  - The extrinsic state contains the font and the location. That is passed as context parameter to the flyweight.

Image: Gamma et al., Design Patterns. Elements of Reusable Object-Oriented Software. Addison Wesley Longman (1995), p. 197

General structure

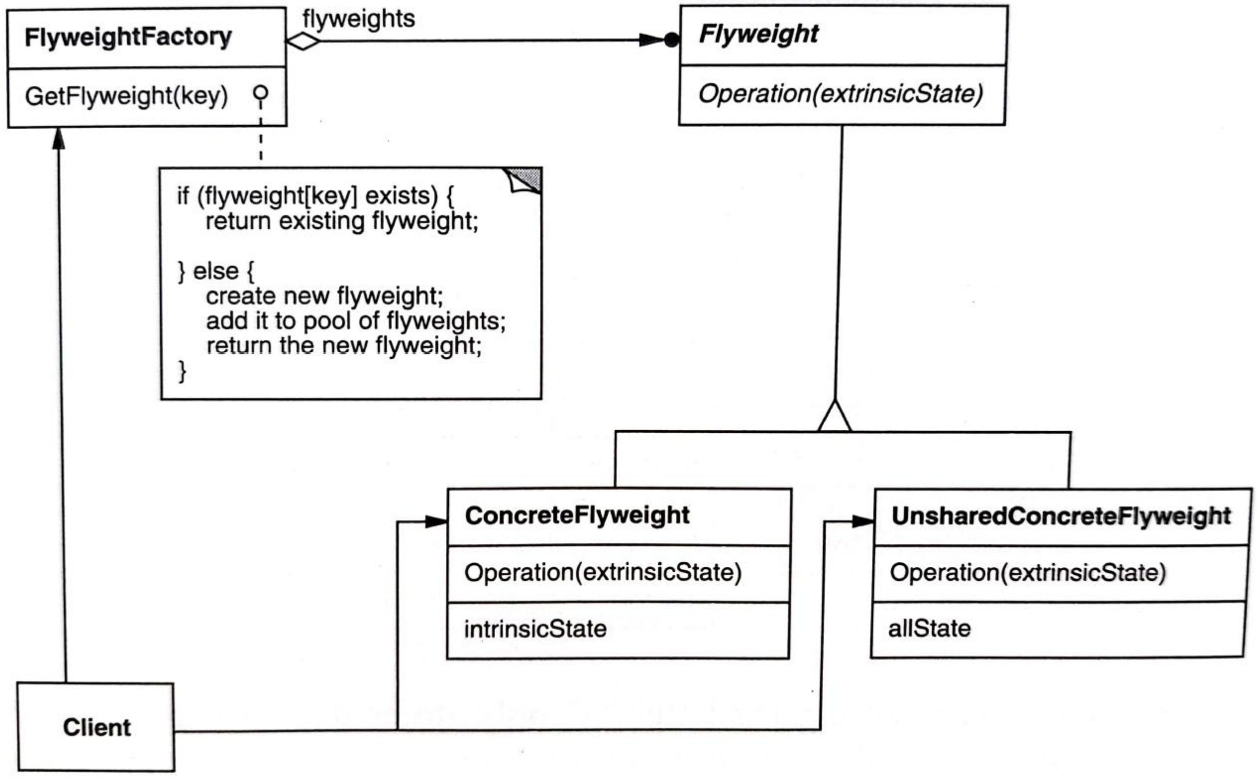

FlyweightFactory
GetFlyweight(key)
flyweights
Flyweight
Operation(extrinsicState)
if (flyweight[key] exists) {
return existing flyweight;
} else {
create new flyweight;
add it to pool of flyweights;
return the new flyweight;
}
ConcreteFlyweight
Operation(extrinsicState)
intrinsicState
UnsharedConcreteFlyweight
Operation(extrinsicState)
allState
Client

Image: Gamma et al., Design Patterns. Elements of Reusable Object-Oriented Software. Addison Wesley Longman (1995), p. 198

Roles

- **Flyweight**: Interface through which flyweights can receive and act on extrinsic states.
- **Concrete Flyweight**: Implements the Flyweight interface and stores intrinsic state.
  - Concrete Flyweights must be sharable and capable of operating on extrinsic state.
- **Unshared Concrete Flyweight**: Implements unique Flywieghts that must not be shared (if such exist).
- **Flyweight Factory**: Creates and manages the flyweight objects. Ensures that flyweights are shared properly.
  - When a client requests a flyweight, the Flyweight Factory objects handles an existing instance, or creates one if none exists.
- **Client**: Maintains a reference to flyweight(s). Computes or stores the extrinsic state of flyweight(s).
  - The client is responsible for passing the extrinsic state to the flyweight when it needs it.

When to use?

- Use Flyweight when all of the following criteria are satisfied:
  1. There is a large number of objects.
  2. The number of objects causes high storage costs (usually RAM consumption)
  3. Objects' extrinsic state must be separable from intrinsic state (not tightly coupled).
  4. Many groups of objects may be replaced by few shared objects after extrinsic state is removed.
  5. The application will not depend on object identity.

Practical issues

- The intrinsic state should be immutable to avoid side effects to other objects.
  - In Java, use **final** keyword for the reference to the intrinsic state.
- The Flyweight Factory is crucial in Java implementations to manage flyweights and ensure they are properly shared.
  - **HashMap** is often good for bookkeeping of existing intrinsic states.
  - Yet, it is prone to memory leaks: an item added in a HashMap needs to be explicitly removed.
- The DP adds a new layer of complexity, and should be used only when the memory usage may become a problem.