InterBit
Training & Consulting Ltd.

Flyweight Patterns

Chapter Content

InterBit
Training & Consulting Ltd.

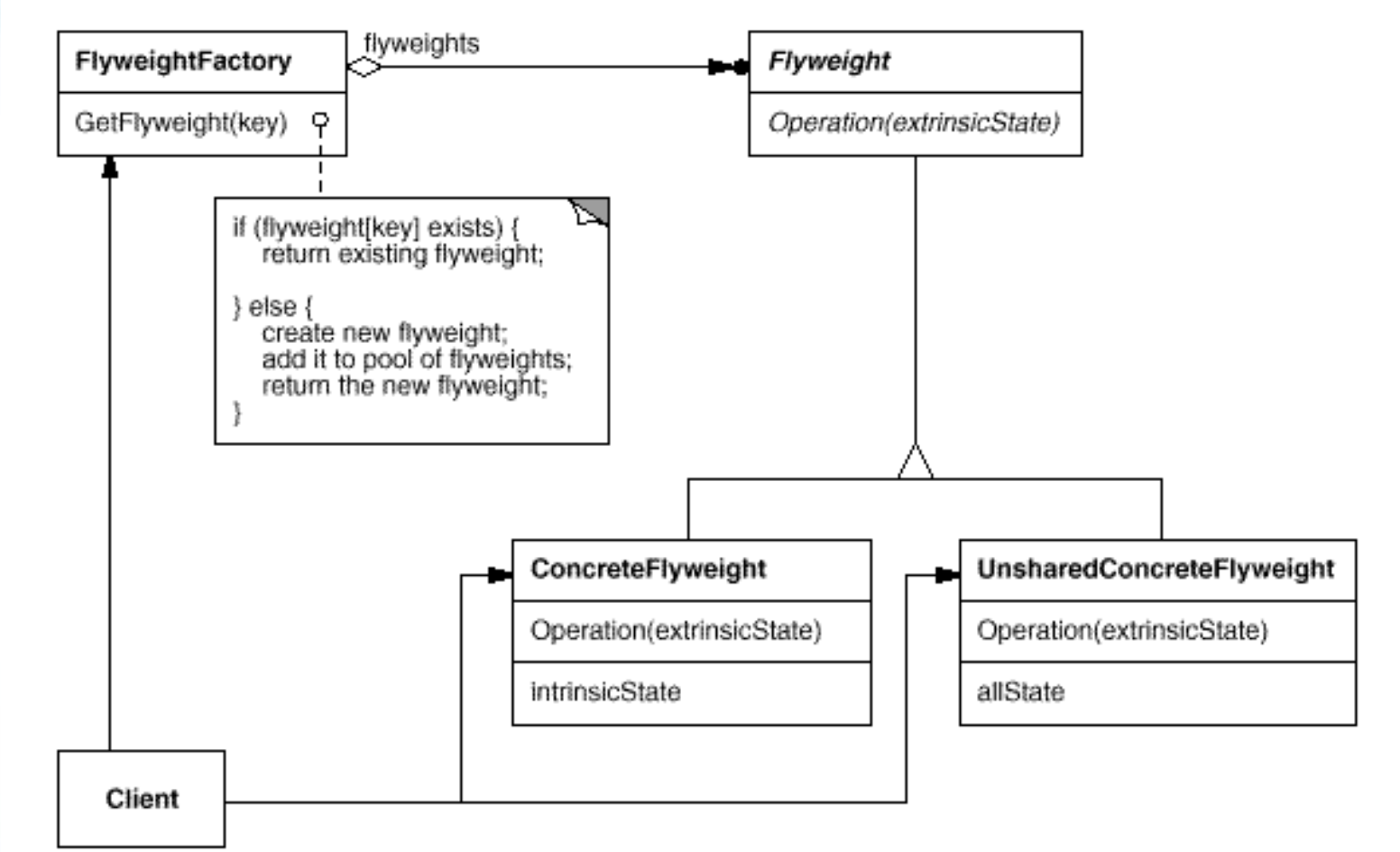
- Flyweight Pattern Overview
- Flyweight Pattern UML Diagram
- Flyweight Usage
- With & Without flyweight
- The factory
- Usage
- Flyweight Pattern Examples

Flyweight Pattern Overview

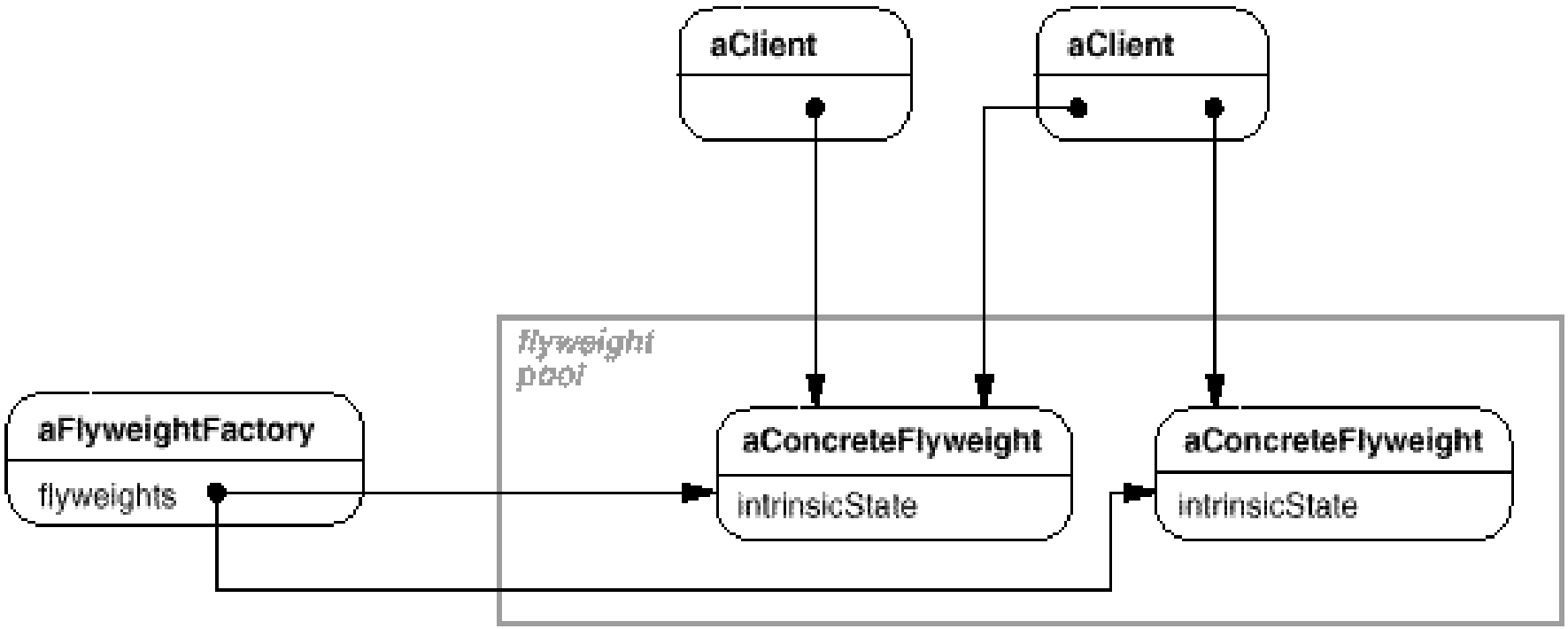
InterBit Training & Consulting Ltd.

- Share instances, to save memory allocations.
  - Especially useful when a program uses a large number of small, similar instances.
- To make instances sharable:
  - Remove some class (state) variables.
  - Those will now be passed as method parameters.
  - This is referred to as making data *extrinsic* instead of *intrinsic.*
- The Flyweight Pattern naturally relies on a Factory.

Flyweight Pattern UML Diagram



Flyweight Usage



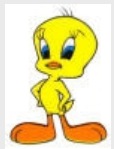
The middle ConcreteFlyweight is shared by 2 clients, but this fact is transparent to clients.

There was still a need to generate another ConcreteFlyweight (the right one) since it has a different intrinsic state.

Without flyweight

- Non-sharable objects; much intrinsic data.

```
class MyImage {
     private String imageFilename;
     private Point location;
     private Dimension size;
     public void draw() {
        // draw image in give location and given size
     }
     ...
}
```



Usage:

```
MyImage img1= new MyImage(“tweety.gif”, new Point(20,10), new Dimension(60,60);
MyImage img2= new MyImage(“tweety.gif”, new Point(25,15), new Dimension(50,50);
MyImage img3= new MyImage(“sylvester.gif”, new Point(55,15), new Dimension(90,90);
```

With flyweight

- Sharable objects; some data made extrinsic.

```
public class MyImage {
     private String imageFilename;

     public void draw(Point location, Dimension size) {
        // draw image in given location and size
     }


     // constructor may be non-public (not mandatory)
     MyImage(String imageFilename){
         this.imageFilename=imageFilename;
     }
     ...
}
```

We chose to leave imageFilename intrinsic;

e.g. we could be caching data from file.

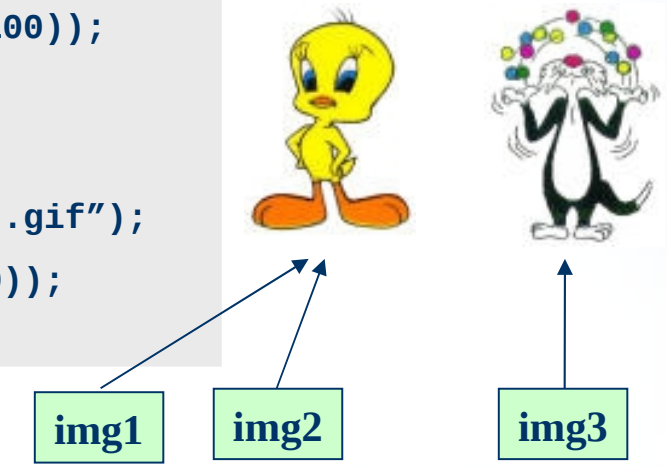
Size & location became extrinsic

The Factory

```
// For this implementation we happened to make the following choices:
// Use hashing for fast object location
// Use lazy initialization (one may also allocate objects in advance)
public class MyImageFactory {

    public Map<MyImage> map = new HashMap<MyImage>();

    public synchronized MyImage create(String fileName) {
        MyImage myImage = get(fileName);
        if(myImage == null) {
            // if no such image exits, allocate it
            myImage = new MyImage(fileName);
            map.put(fileName, myImage);
        }
        return myImage;
    }
}
```

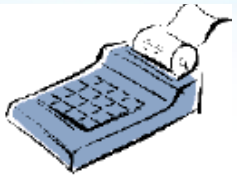
Usage

```
// called from one part of the program (e.g. thread 1):
MyImage img1 = MyImageFactory.create("tweety.gif");
img1.draw(new Point(20,10), new Dimension(100,100));

// called from another part of the prog (e.g. thread 2):
MyImage img2 = MyImageFactory.create("tweety.gif");
Img2.draw(new Point(20,10), new Dimension(100,100));

// called from another part (e.g. thread 3):
MyImage img3 = MyImageFactory.create("sylvester.gif");
Img3.draw(new Point(55,15), new Dimension(90,90));
```



Another example

- Our old Fourier calculator:



```
abstract class Calculator {
  protected float angle;
  public Calculator(float angle) {
      this.angle = angle;
  }
  abstract public void execute(Complex x, Complex y);
}
```

- A sharable calculator:

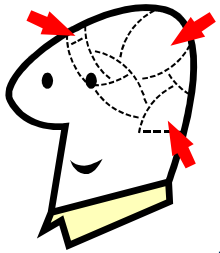
```
abstract class Calculator {
    abstract public void execute(float angle , Complex x, Complex y);
}
```

- In this extreme case, calculator can become a singleton (no intrinsic data)

The time/memory tradeoff

InterBit Training & Consulting Ltd.

- When using flyweight, our class may need to re-calculate information (which could have been cached if we used a non-flyweight approach).



- Example:
  - Non-sharable Calculator may cache sin value.
  - Non-sharable Image could cache graphical context, scaling calculations, etc.

Flyweight – example #3

- With EJB's, you may choose between Stateless & Stateful SessionBeans.
  - The former may be shared, but this is transparent to clients.