Tecniche di Progettazione: Design Patterns

GoF: Flyweight

Cara/caro Collega,

ti comunichiamo che è possibile la compilazione on line dei questionari di valutazione della didattica da parte degli studenti, relativi agli insegnamenti impartiti nel **primo semestre** del corrente anno accademico.

Ti ricordiamo che l'accesso è possibile dal portale **Valutami** (VALUTazione della didattica ed iscrizione agli esAMI) al link **https://esami.unipi.it**.

Le domande contenute nel questionario ripropongono l'obbligatorio schema ANVUR e sono analoghe a quelle relative agli insegnamenti dello scorso anno, con l'aggiunta della domanda relativa alla "valutazione complessiva del corso" e delle domande specifiche proposte dai Dipartimenti che ne hanno fatto richiesta.

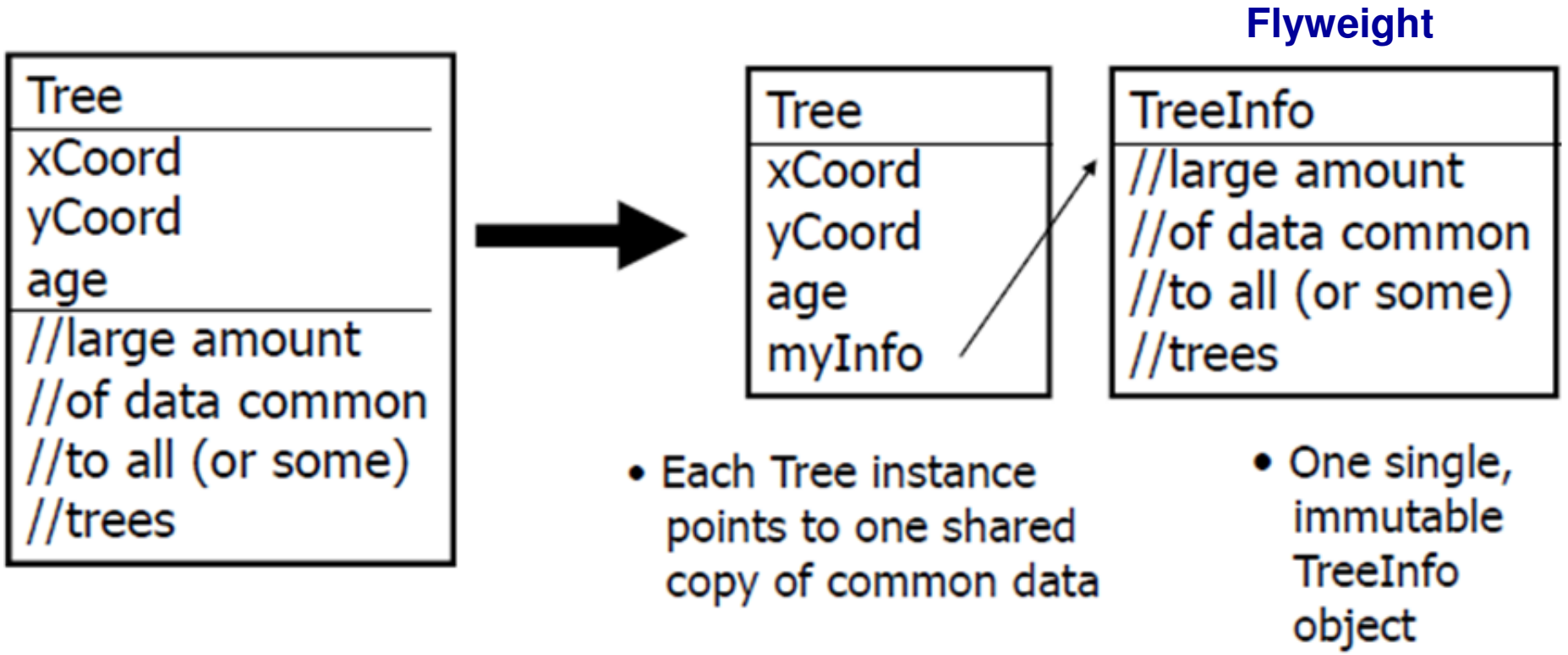
Come ti è noto, la compilazione del questionario è obbligatoria per gli studenti all'atto dell'iscrizione all'esame. E' tuttavia particolarmente importante, per la rilevazione dell'opinione degli studenti frequentanti, anche alla luce delle linee guida del nuovo sistema AVA, **che gli stessi accedano al sistema e provvedano alla compilazione del questionario durante lo svolgimento del corso, quando siano stati svolti almeno i 2/3 delle lezioni**.

Anche per la crescente importanza che l'Anvur sta attribuendo alla rilevazione dell'opinione degli studenti, diviene particolarmente importante proseguire nella adeguata pubblicizzazione e sensibilizzazione degli studenti sul significato dell'iniziativa ed in particolare sollecitare la corretta tempistica di compilazione. Per evitare che il questionario venga redatto solo all'ultimo momento utile, può essere di sicuro aiuto **una tua comunicazione fornita in aula agli studenti o anche la concessione di pochi minuti di sospensione della lezione per dare agli stessi il tempo di compilare il questionario dai loro dispositivi mobili.**
Ti segnaliamo infine che il Senato Accademico, nella sua ultima seduta, ha approvato specifiche linee guida per l'analisi e la diffusione dei risultati e delle conseguenti azioni migliorative, alle quali verrà data adeguata pubblicità

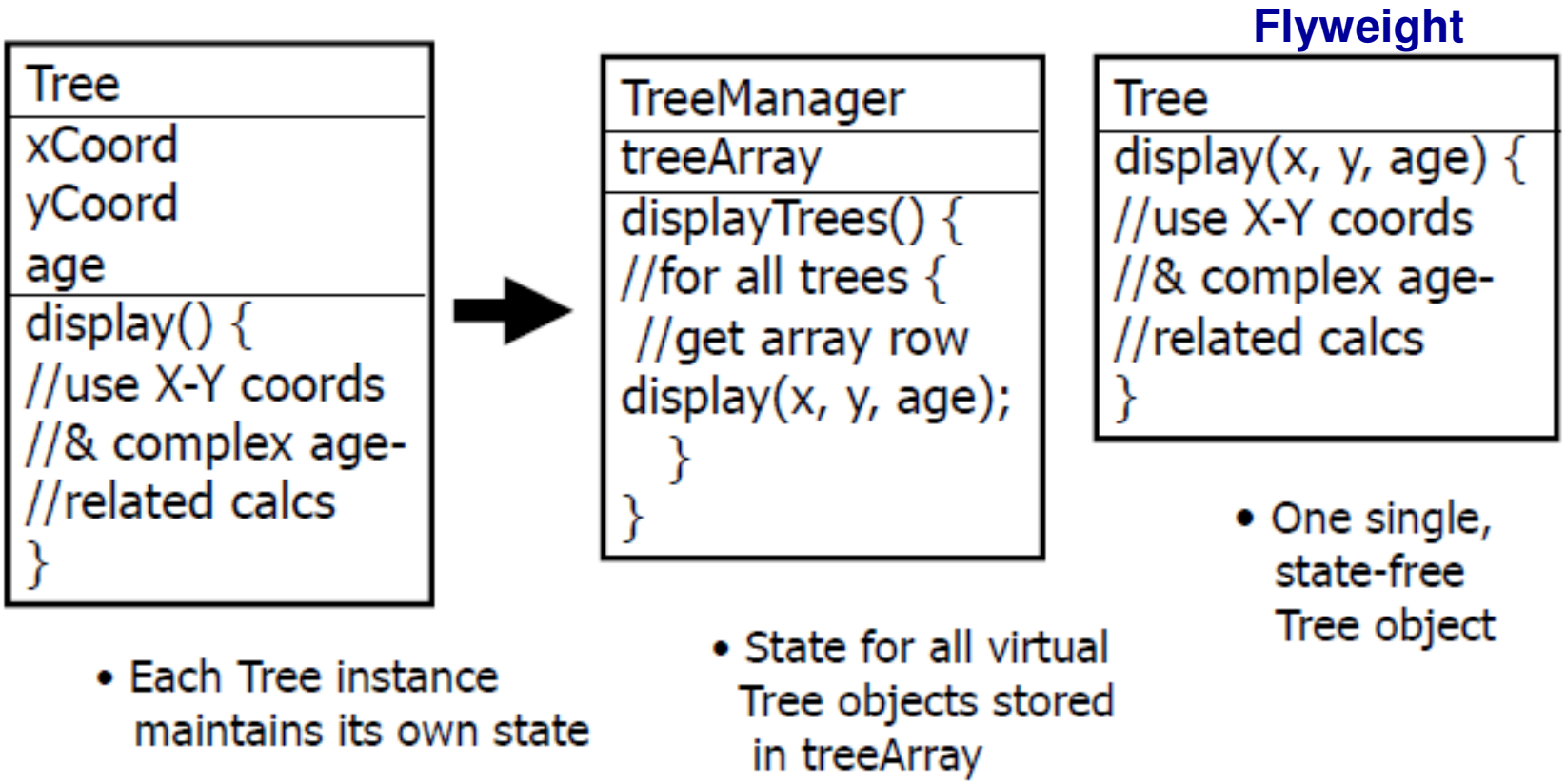
Flyweight Pattern

- Intent
  - Use sharing to support large numbers of fine-grained objects efficiently
- Motivation
  - Can be used when an application could benefit from using objects throughout their design, but a naïve implementation would be prohibitively expensive
    - Objects for each character in a document editor
      - Cost is too great!
    - Can use flyweight to share characters

Example


Flyweight
Tree
xCoord
yCoord
age
//large amount
//of data common
//to all (or some)
//trees
Tree
xCoord
yCoord
age
myInfo
TreeInfo
//large amount
//of data common
//to all (or some)
//trees
• Each Tree instance points to one shared copy of common data
• One single, immutable TreeInfo object
• Each Tree instance maintains its own copy of common data

Head first example

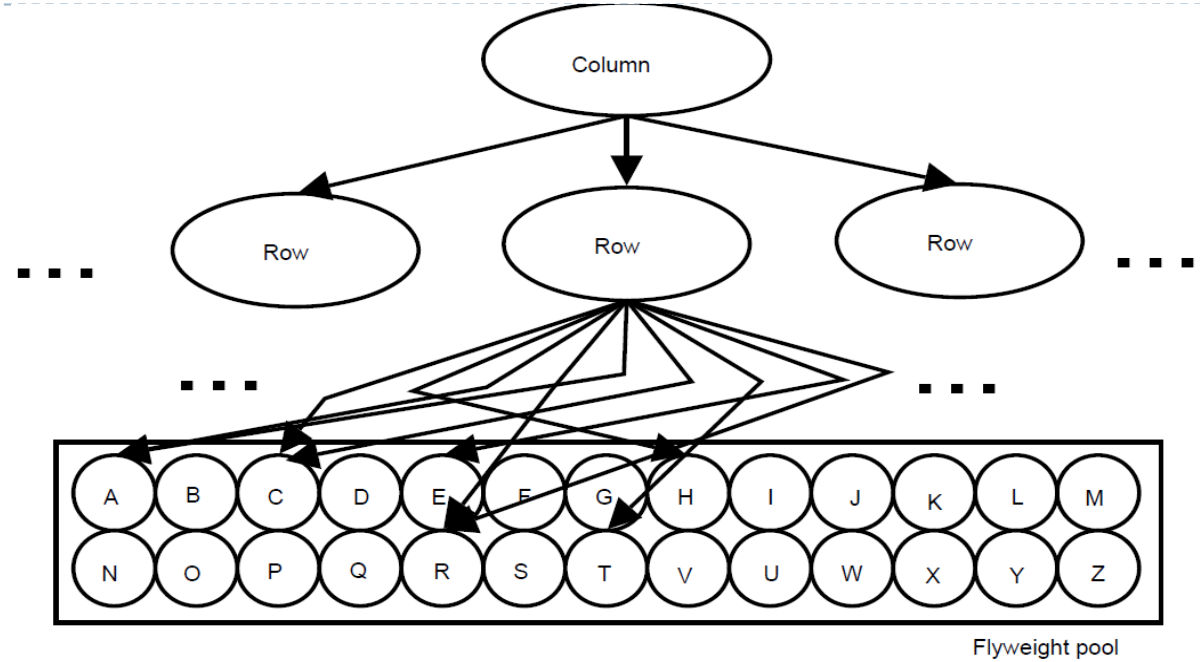


The manager is acting as a collecion more than a Flyweight

Intrinsic vs. Extrinsic

- Most objects would share a set of stateless information, this could be extracted from the main objects to be held in flyweight objects
- Intrinsic
  - The intrinsic data is held in the properties of the flyweight objects that are shared. This information is stateless and generally remains unchanged, as any changes would be effectively replicated amongst all of the objects that reference the flyweight
- Extrinsic
  - Extrinsic data can be stateful as it is held outside of a flyweight object. It can be passed to methods of a flyweight when needed but should never be stored within a shared flyweight object.

GoF Example

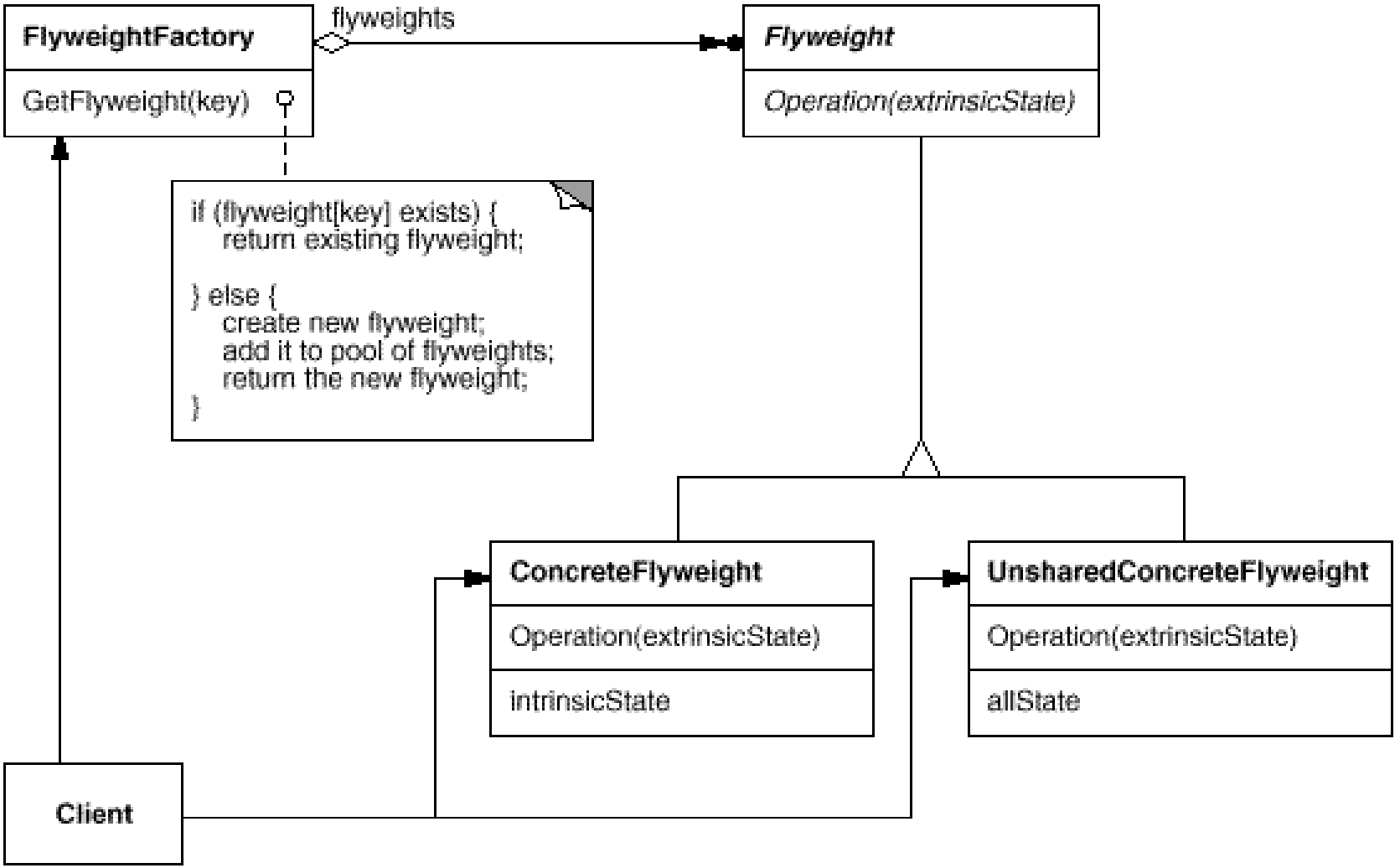


Creating a flyweight for each letter of the alphabet:

***Intrinsic state*: a character code**

***Extrinsic state*: coordinate position in the document**
typographic style (font, color)
is determined from the text layout algorithms and formatting commands in effect wherever the character appears

Flyweight: Structure



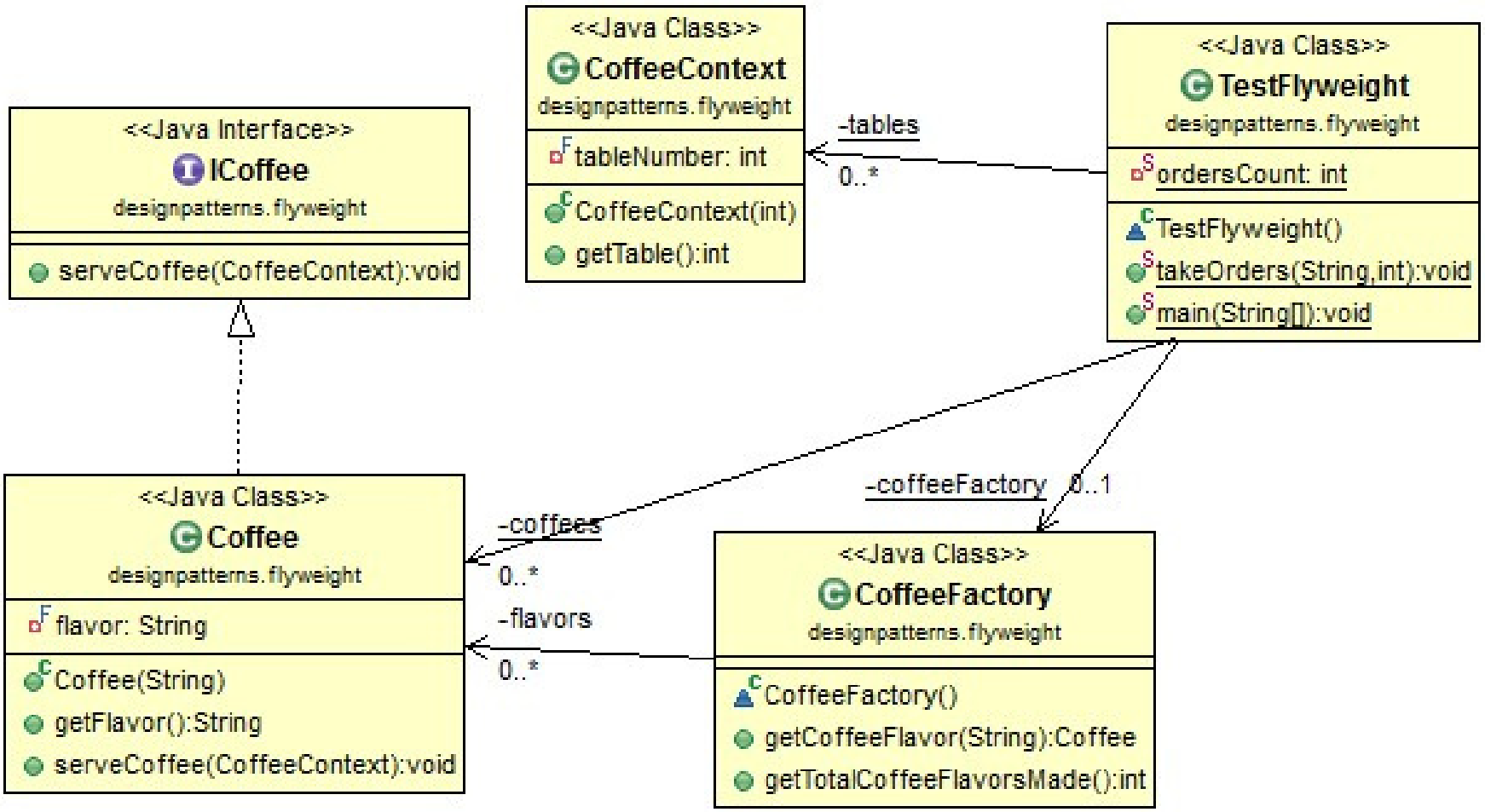
Flyweight: Participants

- Flyweight
  - Declares an interface through which flyweights can receive and act on extrinsic state
- ConcreteFlyweight
  - Implements the Flyweight interface and adds storage for intrinsic state, if any
  - Must be shareable
- UnsharedConcreteFlyweight
  - Although the flyweight design pattern enables sharing of information, it is possible to create instances of concrete flyweight classes that are not shared. In these cases, the objects may be stateful.

Flyweight: Participants

- FlyweightFactory
  - Creates and manages flyweight objects
  - Ensures that flyweights are shared properly
- Client
  - Maintains reference to flyweights
  - Computes or stores the extrinsic state of flyweights

Example


<<Java Interface>>
ICoffee
designpatterns.flyweight
serveCoffee(CoffeeContext):void
<<Java Class>>
CoffeeContext
designpatterns.flyweight
tableNumber: int
CoffeeContext(int)
getTable():int
-tables
0..*
<<Java Class>>
TestFlyweight
designpatterns.flyweight
ordersCount: int
TestFlyweight()
takeOrders(String,int):void
main(String[]):void
<<Java Class>>
Coffee
designpatterns.flyweight
flavor: String
Coffee(String)
getFlavor():String
serveCoffee(CoffeeContext):void
-coffees
0..*
-flavors
0..*
-coffeeFactory
0..1
<<Java Class>>
CoffeeFactory
designpatterns.flyweight
CoffeeFactory()
getCoffeeFlavor(String):Coffee
getTotalCoffeeFlavorsMade():int

Flyweight object interface

```
interface ICoffee {
    public void serveCoffee(CoffeeContext context);
}
```

Concrete Flyweight object

```
class Coffee implements ICoffee {
    private final String flavor;
    public Coffee(String newFlavor) {
        this.flavor = newFlavor;
        System.out.println("Coffee is created! - " + flavor);
    }
    public String getFlavor() {   return this.flavor;   }
    public void serveCoffee(CoffeeContext context) {
        System.out.println("Serving " + flavor + " to table " +
                    context.getTable());
    }}
```

A context, here is table number

```
class CoffeeContext {
   private final int tableNumber;

   public CoffeeContext(int tableNumber) {
      this.tableNumber = tableNumber;
   }

   public int getTable() {
      return this.tableNumber;
   }
}
```

Flyweight Factory

```
class CoffeeFactory {
    private HashMap<String, Coffee> flavors = new  HashMap<String, Coffee>();

    public Coffee getCoffeeFlavor(String flavorName) {
        Coffee flavor = flavors.get(flavorName);
        if (flavor == null) {
            flavor = new Coffee(flavorName);
            flavors.put(flavorName, flavor);
        }
        return flavor;
    }
     public int getTotalCoffeeFlavorsMade() {
        return flavors.size();
    }
}
```

Waitress (continues)

```
public class Waitress {
   //coffee array
   private static Coffee[] coffees = new Coffee[20];
   //table array
   private static CoffeeContext[] tables = new CoffeeContext[20];
   private static int ordersCount = 0;
   private static CoffeeFactory coffeeFactory;

   public static void takeOrder(String flavorIn, int table) {
            coffees[ordersCount] = coffeeFactory.getCoffeeFlavor(flavorIn);
            tables[ordersCount] = new CoffeeContext(table);
            ordersCount++;
   }
```

Waitress (continued)

```
public static void main(String[] args) {
        coffeeFactory = new CoffeeFactory();

        takeOrder("Cappuccino", 2);
        takeOrder("Cappuccino", 2);
        takeOrder("Regular Coffee", 1);
        takeOrder("Regular Coffee", 2);
        takeOrder("Regular Coffee", 3);
        for (int i = 0; i < ordersCount; ++i) {  coffees[i].serveCoffee(tables[i]); }

        System.out.println("\nTotal Coffee objects made: " +
                                                coffeeFactory.getTotalCoffeeFlavorsMade());
    }
}
```

Flyweight: Applicability

- Use the Flyweight pattern when ALL of the following are true
  - An application uses a large number of objects
  - Storage costs are high because of the sheer quantity of objects
  - Most object state can be made intrinsic
  - Many Groups of objects may be replaced by relatively few shared objects once extrinsic state is removed
  - The application doesn't depend on object identity

Flyweight: Consequences

- May introduce run-time costs associated with transferring, finding, and/or computing extrinsic state
  - Costs are offset by space savings
- Storage savings are a function of the following factors:
  - The reduction in the total number of instances that comes from sharing
  - The amount of intrinsic state per object
  - Whether extrinsic state is computed or stored
- Ideal situation
  - High number of shared flyweights
  - Objects use substantial quantities of both intrinsic and extrinsic state
  - Extrinsic state is computed

Implementation

- Removing extrinsic state
  - Success of pattern depends on ability to remove extrinsic state from shared objects
  - No help if there are many different kinds of extrinsic state
  - Ideally, state is computed separately
- Managing shared objects
  - Objects are shared so clients should not instantiate
  - FlyweightFactory is used to create and share objects
  - Garbage collection may not be necessary

Java Strings

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

Flyweight: Related Patterns

- Composite
  - Often combined with flyweight
  - Provides a logically hierarchical structure in terms of a directed-acyclic graph with shared leaf nodes
- State and Strategy
  - Can be implemented as flyweights

- Used in game programming
- Ex reference http://gameprogrammingpatterns.com/

Ethimology

- Flyweight is a boxing category, for light weight people.
- Flyweight pattern is for "light weight" objects (though many of them).

Flyweight: homework

- In Flyweight pattern, a Flyweight object has intrinsic state that cannot be changed. This also means that a Flyweight object cannot have any public-accessible *set()* method to set a new value for some instance variable of the object.

- Consider using the Flyweight pattern to allow a Flyweight object to have set() methods. When a set() method of a Flyweight object is called, the object becomes a non-Flyweight, non-shared object. This idea is similar to copy-on-write.

- Build an example of copy-on-write.
  - For instance: decorate the Christmas tree and then change colour to some decorations