Flyweight-Pattern

Carsten Gips (HSBI)

Motivation: Modellierung eines Levels

```
public enum Tile { WATER, FLOOR, WALL, ... }

public class Level {
    private Tile[][] tiles;

    public Level() {
        tiles[0][0] = Tile.WALL;  tiles[1][0] = Tile.WALL;   tiles[2][0] = Tile.WALL;  ...
        tiles[0][1] = Tile.WALL;  tiles[1][1] = Tile.FLOOR;  tiles[2][1] = Tile.FLOOR; ...
        tiles[0][2] = Tile.WALL;  tiles[1][2] = Tile.WATER;  tiles[2][2] = Tile.FLOOR; ...
        ...
    }

    public boolean isAccessible(int x, int y) {
        switch (tiles[x][y]) {
            case: WATER: return false;
            case: FLOOR: return true;
            ...
        }
    }
    ...
}
```

Motivation: Modellierung eines Levels (cnt.)

```
public abstract class Tile {
    protected boolean isAccessible;
    protected Texture texture;
    public boolean isAccessible() { return isAccessible; }
}
public class Floor extends Tile {
    public Floor() { isAccessible = true;  texture = Texture.loadTexture("path/to/floor.png"); }
}
...

public class Level {
    private final Tile[][] tiles;

    public Level() {
        tiles[0][0] = new Wall();  tiles[1][0] = new Wall();   tiles[2][0] = new Wall();  ...
        tiles[0][1] = new Wall();  tiles[1][1] = new Floor();  tiles[2][1] = new Floor(); ...
        tiles[0][2] = new Wall();  tiles[1][2] = new Water();  tiles[2][2] = new Floor(); ...
        ...
    }
    public boolean isAccessible(int x, int y) { return tiles[x][y].isAccessible(); }
}
```

Flyweight: Nutze gemeinsame Eigenschaften gemeinsam

```
public final class Tile {
    private final boolean isAccessible;
    private final Texture texture;
    public boolean isAccessible() { return isAccessible; }
}

public class Level {
    private static final Tile FLOOR = new Tile(true,  Texture.loadTexture("path/to/floor.png"));
    private static final Tile WALL  = new Tile(false, Texture.loadTexture("path/to/wall.png"));
    private static final Tile WATER = new Tile(false, Texture.loadTexture("path/to/water.png"));

    private final Tile[][] tiles;

    public Level() {
        tiles[0][0] = WALL;  tiles[1][0] = WALL;   tiles[2][0] = WALL;  ...
        tiles[0][1] = WALL;  tiles[1][1] = FLOOR;  tiles[2][1] = FLOOR; ...
        tiles[0][2] = WALL;  tiles[1][2] = WATER;  tiles[2][2] = FLOOR; ...
        ...
    }
    public boolean isAccessible(int x, int y) { return tiles[x][y].isAccessible(); }
}
```

Flyweight: Nutze gemeinsame Eigenschaften gemeinsam (cnt.)

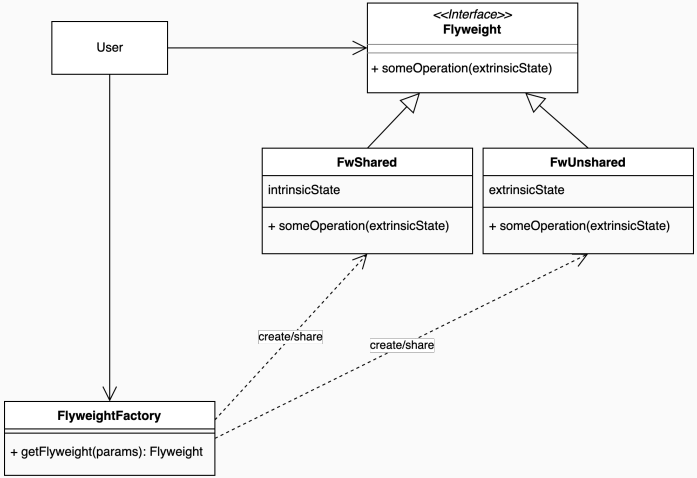
```
public final class TileModel {
    private final boolean isAccessible;
    private final Texture texture;
    public boolean isAccessible() { return isAccessible; }
}
public final class Tile {
    private boolean wasEntered;
    private final TileModel model;
    public boolean isAccessible() { return model.isAccessible(); }
    public boolean wasEntered() { return wasEntered; }
}

public class Level {
    private static final TileModel FLOOR = new TileModel(true,  Texture.loadTexture("path/to/floor.png"));
    ...

    private final Tile[][] tiles;

    public Level() {
        tiles[0][0] = new Tile(WALL);  tiles[1][0] = new Tile(WALL);   tiles[2][0] = new Tile(WALL);  ...
        tiles[0][1] = new Tile(WALL);  tiles[1][1] = new Tile(FLOOR);  tiles[2][1] = new Tile(FLOOR); ...
        tiles[0][2] = new Tile(WALL);  tiles[1][2] = new Tile(WATER);  tiles[2][2] = new Tile(FLOOR); ...
        ...
    }
    public boolean isAccessible(int x, int y) { return tiles[x][y].isAccessible(); }
}
```

Flyweight-Pattern: Begriffe

- **Intrinsic** State: invariant, Kontext-unabhängig, gemeinsam nutzbar
  => auslagern in gemeinsame Objekte

- **Extrinsic** State: variant, Kontext-abhängig und kann nicht geteilt werden
  => individuell modellieren

Flyweight-Pattern: Klassische Modellierung



Hinweis zum Beispiel: -Interface, -Factory, +Composite

Wrap-Up

Flyweight-Pattern: Steigerung der (Speicher-) Effizienz durch gemeinsame Nutzung von Objekten

- Lagere *Intrinsic State* in gemeinsam genutzte Objekte aus
- Modelliere *Extrinsic State* individuell

LICENSE