Type-Object-Pattern

Carsten Gips (HSBI)

Motivation: Monster und spezialisierte Monster

```java
public abstract class Monster {
    protected int attackDamage;
    protected int movementSpeed;

    public Monster(int attackDamage, int movementSpeed) { ... }
    public void attack(Monster m)  { ... }
}

public class Rat extends Monster {
    public Rat() { super(10, 10); }  // Ratten haben 10 Damage und 10 Speed
    @Override public void attack(Monster m)  { ... }
}

public class Gnoll extends Monster { ... }


public static void main(String[] args) {
    Monster harald = new Rat();
    Monster eve = new Gnoll();
    ...
}
```

Vereinfachen der Vererbungshierarchie (mit Enums als Type-Object)

```java
public enum Species { RAT, GNOLL, ... }

public final class Monster {
    private final Species type;
    private int attackDamage;
    private int movementSpeed;

    public Monster(Species type) {
        switch (type) {
            case RAT: attackDamage = 10; movementSpeed = 10; break;
            ...
        }
    }
    public void attack(Monster m)  { ... }
}


public static void main(String[] args) {
    Monster harald = new Monster(Species.RAT);
    Monster eve = new Monster(Species.GNOLL);
    ...
}
```

Monster mit Strategie

```java
public final class Species {
    private final int attackDamage;
    private final int movementSpeed;
    private final int xp;

    public Species(int attackDamage, int movementSpeed, int xp) { ... }
    public void attack(Monster m)  { ... }
}

public final class Monster {
    private final Species type;
    private int xp;

    public Monster(Species type) { this.type = type;  xp = type.xp(); }
    public int movementSpeed() { return type.movementSpeed(); }
    public void attack(Monster m)  { type.attack(m); }
}


public static void main(String[] args) {
    final Species RAT = new Species(10, 10, 4);
    final Species GNOLL = new Species(...);

    Monster harald = new Monster(RAT);
    Monster eve = new Monster(GNOLL);
}
```

Fabrikmethode für die Type-Objects

```java
public final class Species {
    ...

    public Monster newMonster() {
        return new Monster(this);
    }
}


public static void main(String[] args) {
    final Species RAT = new Species(10, 10, 4);
    final Species GNOLL = new Species(...);

    Monster harald = RAT.newMonster();
    Monster eve = GNOLL.newMonster();
}
```

Vererbung unter den Type-Objects

```java
public final class Species {
    ...

    public Species(int attackDamage, int movementSpeed, int xp) {
        this.attackDamage = attackDamage;  this.movementSpeed = movementSpeed;  this.xp = xp;
    }
    public Species(Species parent, int attackDamage) {
        this.attackDamage = attackDamage;
        movementSpeed = parent.movementSpeed;  xp = parent.xp;
    }
}


public static void main(String[] args) {
    final Species RAT = new Species(10, 10, 4);
    final Species BOSS_RAT = new Species(RAT, 100);
    final Species GNOLL = new Species(...);

    Monster harald = RAT.newMonster();
    Monster eve = GNOLL.newMonster();
}
```

Erzeugen der Type-Objects dynamisch über eine Konfiguration

```
{
    "Rat": {
        "attackDamage": 10,
        "movementSpeed": 10,
        "xp": 4
    },
    "BossRat": {
        "parent": "Rat",
        "attackDamage": 100
    },
    "Gnoll": {
        "attackDamage": ...,
        "movementSpeed": ...,
        "xp": ...
    }
}
```

Wrap-Up

Type-Object-Pattern: Implementierung eines eigenen Objekt-Modells

- Ziel: Minimierung der Anzahl der Klassen
- Ziel: Erhöhung der Flexibilität

- Schiebe “Typen” in ein eigenes Objekt-Modell
- Type-Objects lassen sich dynamisch über eine Konfiguration anlegen
- Objekte erhalten eine Referenz auf “ihr” Type-Object
- “Vererbung” unter den Type-Objects möglich

LICENSE