# 02 - Composite, Strategy

Credits: Askar Gafurov

***DU #1 bude zverejnena v Teams pocas tohto tyzdna (na riesenie bude 10 dni)***

## Uzitocne materialy:

- Slajdy z prednasky z roku 2019
- Slajdy z prednasky z roku 2020
- Poznamky ku Composite a Strategy z minulych rokov
- Head First Design Patterns, kap. 1, 8, 9

# Composite

Pouzivame, ked chceme reprezentovat hierarchicke struktury a pracovat s lubovolnym podstromom hierarchie rovnakym sposobom (t.j. napriklad s celkom alebo jednotlivcom).

## Zakladna struktura:

```
public interface Component {
    void do_stuff();
}

public class LeafComponent implements Component {
    @Override
    public void do_stuff() {...}
}

// notice that concrete classes have no references on
// another concrete classes, only interfaces or abstract classes
public class CompositeComponent implements Component {
    private final ArrayList<Component> children = new ArrayList<>();

    @Override
    public void do_stuff() {
        for (Component c: children) {
            // do stuff on children
        }
        // use the subresults to produce the final result
    }

    public void addChild(Component child) {
        children.add(child);
```

```
    }

    // also can add `removeChild`
}
```

## Pouzitie:

```
class Demo {
    void demo() {
        Component c1 = new LeafComponent();
        Component c2 = new LeafComponent();
        CompositeComponent c3 = new CompositeComponent();
        c3.addChild(c1);
        c3.addChild(c2);
        CompositeComponent c5 = new CompositeComponent();
        c5.addChild(c3); // sic! We are adding a CompositeComponent
                         // inside another CompositeComponent
        c5.addChild(new LeafComponent()); // this is also possible

        c5.do_stuff();
    }
}
```

## Ulohy:

U: Implementujte v baliku `workbalance` strukturu pracovnikov vo firme, kde:

- Kazdy pracovnik (`Worker`) ma meno a plat
- Kazdy pracovnik patri do nejakej organizacnej jednotky (`Team`), ktora ma nazov
- Organizacne jednotky mozu sucastou vacsej organizacnej jednotky

(Rozhranie mozete nazvat napriklad `WorkUnit`)

Zaistite, ze triedy su verejne. **Vsetky ich metody a atributy maju mat najprisnejsi mozny pristup (tento postup plati pocas celeho semestra).**

U: Pridajte jednotlivym komponentom metodu `getSalary()`, ktora pre jednotlivcov vrati ich plat, a pre organizacne jednotky vrati sucet platov vsetkych pracovnikov v danej strukture (aj v podstrukturach)

U: Pridajte metodu `String repr()`, ktora vrati plnu strukturu daneho podstromu, t.j. vsetky jednotky a mena zamestnancov spolu s ich platmi, napriklad takto:

```
Company (total salary: 5700):
    R&D department (total salary: 2300):
        Rick (salary: 1000)
        David (salary: 1300)
    M&M department (total salary: 3400):
        Michael (salary: 1400)
        Maximilian (salary: 1000)
```

```
        Q&A subdepartment (total salary: 1000):
            Quentin (salary: 500)
            Ashley (salary: 500)
```

Pre zaistenie spravneho odsadenia pridajte pomocnu metodu `repr(int offset)`, kde parameter `offset` urcuje potrebne odsadenie (napr. +4 medzery pre kazdu novu uroven vnorenia).

U: Pridajte komponentom verejne metody `getCurrentWorkload()` a `addWorkload(int amount)`, pricom pri jednotlivych pracovnikoch to respektivne vrati alebo navysi o prislusne cislo interne pocitadlo `int workload` (pociatocnu hodnotu nastavte na nulu), a pri organizacnych jednotkach to vzdy respektivne uvedie sucet zatazi jednotlivych pracovnikov alebo najde v ramci celej podstruktury pracovnika s najmensou zatazou a prida tu pracu jemu.
Skuste to implementovat pridanim do rozhrania pomocnej metody `Worker getWorkerWithLowestWorkload()`, ktora vrati referenciu na pracovnika s najmensou aktualnou zatazou v pod strome. Ktorý stupeň prístupu je najprísnejší možný pre danú metódu?
(*Tu sa da pozorovat rozdiel medzi abstraktnou triedou a rozhranim. Totiz metoda `getWorkerWithLowestWorkload` je potrebna len na interne ucely, ale kvoli architekture ju musime definovat ako verejnu. Pri pouziti abstraktnej triedy by sme mohli definovat tuto pomocnu metodu ako "package-level" access, pripadne aj ako "protected"*).

U**: Aka je casova zlozitost priamociarej implementacie metody `addWorkload` z predoslej ulohy? Implementuje tieto metody s casovou zlostitostou umernou sucinu hlbky stromu a maximalneho poctu deti pri vrchole (napr. ukladanim referencie na pracovnika s najmensou zatazou v podstrome). Porovnajte rychlost priamociarej implementacie s novou na stromoch hlbky 10, kde kazdy vnutorny vrchol ma 2 deti (implementujte teda aj generator takychto stromov).

# Strategy

Pouzivame, ked chceme oddelit cast logiky (napriklad algoritmus) a/alebo chceme moct menit tuto logiku dynamicky pocas behu algoritmu.

## Zakladna struktura:

```
public interface Strategy {
    void do_stuff(int params);
}

public class ConcreteStrategy implements Strategy {
    @Override
    public void do_stuff(int params) {...}
}
```

```
// once again, the client class doesn’t have a reference to a concrete
// class, only a reference to an interface
public class Client {
    private Strategy strategy;

    public void setStrategy(Strategy strategy) {
       // may also be assigned in a constructor
        this.strategy = strategy;
    }

    public void doSomething() {
        // using the strategy
        strategy.do_stuff(47);
    }
}
```

## Pouzitie:

```
class Demo {
    void demo() {
        Client client = new Client();
        Strategy strategy = new ConcreteStrategy();
        client.setStrategy(strategy);
        client.doSomething();

        client.setStrategy(new ConcreteStrategy2());
        client.doSomething();
    }
}
```

## Ulohy:

U: Implementujte v baliku `dnd` rozhranie `MyRandom` s metodou `long nextLong()`, ktora bude vracat dalsie nahodne cislo.

U: Implementujte triedu `StardardRandom` splnajucu rozhranie `MyRandom`, ktora si bude interne udrziavat objekt triedy `java.util.Random` a bude presmerovavat jej hodnoty metode `nextLong()`. `StandardRandom` je teda **konkretnou strategiou**.

U: Implementujte triedu `D6`, simulujucu seststennu kocku, ktora bude mat verejne metody

- `int nextValue()`, ktora vrati dalsie nahodne cislo z rozsahu 0 .. 5
- `int getSidesCount()`, ktora vrati pocet stien kocky, teda 6

Ako zdroj nahody bude prijmat objekt s rozhranim `MyRandom`. Implementujte to pomocou navrhoveho vzoru Strategy. Navratovu hodnotu metody `nextValue()` vypocitajte ako zvysok nahodneho cisla po deleni 6.
Vypiste hodnoty 10 hodov pri pouziti strategie `StandardRandom`.

U: Implementute rozhranie `NDie`, ktore bude kopirovat verejne rozhranie triedy `D6`, upravte `D6` tak, aby implementovalo to prostredie (v zasade len pridat riadok `implements NDie` do definicie triedy.

U: Pridajte triedu `D4` s rozhranim `NDie`, ktora bude simulovat stvorstennu kocku. Vyskusajte ju.

U*: Vytvorte triedu `Dn` s rozhranim `NDie` pre simulaciu kocky s lubovolnym poctom stien, ktora bude dostavat pocet stien ako parameter konstruktoru.

U: Pre ucely testovania nahodnosti kociek vytvorte triedu `DieTester` so statickou metodou `void testDistribution(NDie die, int tries)`, kde `tries` udava pocet hodov pri testovani. Metoda ma zaznamenat, ktore hodnoty padli ako casto, a tuto informaciu vypisat na standardny vystup.
Vyskusajte kocky d4 a d6 so standardnym generatorom na 100 hodoch.

U: Pridajte do metody `testDistribution()` vypis entropie vysledkov hodu. Samotny vypocet entropie vyclente do samostatnej statickej sukromnej metody `evalEntropy`

> **Entropia** vysledkov a_1, a_2,...,a_k so suctom N je definovana ako
> -a_1/N * log(a_1/N) - a_2/N * log(a_2/N) … - a_k/N * log(a_k/N),
> pricom logaritmus je pri zaklade 2, a cleny s nulovym a_i su vynechane.

Aku hodnotu entropie ocakavame pri vysledkoch dokonalo nahodnej seststennej kocky? Stvorstennej kocky?

U*: Do triedy `DieTester` pridajte funkciu `testConsecutivePairs` na analyzu dvojich za sebou iducich hodov. Vypiste tabulku NxN s frekvenciami vyskytov danych parov. Pridajte taktiez aj vypocet entropie danej tabulky. Viete pri tom pouzit povodnu funkciu `evalEntropy`?

U: Implementujte triedu `LCG` s rozhranim `MyRandom`, ktora bude implementovat linearny kongruentny nahodny generator, pricom vsetky parametre dostane v ramci konstruktora. Otestujte kocky d4 a d6 pomocou triedy `DieTester` na generatoroch

1. LCG(A=65539, C=0, M=2^31, seed=47) (RANDU settings),
2. LCG(A=1103515245, C=12345, M=2^31, seed=47) (GLIBC settings),

pricom pouzite metodu `setGenerator()` na zmenu generatora uz existujuceho objektu.

U*: Implementute inverzny kongruentny generator, pricom modularny inverz x^(-1) pocitajte cez Eulerovu vetu ako x^{m-2}. Otestujte generator na nastaveniach:

1. ICG(A=849, C=1, M=1031, seed=47)
2. ICG(A=1048, C=1, M=2027, seed=47)
3. ICG(A=858993221, C=1, M=2147483053, seed=47)

U**: Implementujte vypocet modularneho inverzu v case O(ln(m)) pomocou techniky rychleho umocnovania.

U*: Implementujte triedu `Hand` s rozhranim `NDie`, ktora bude pri inicializacii prijmat lubovolny pocet lubovolnych kociek a bude vracat sucet hodnot z ich hodov. Otestujte ju pomocou triedy `DieTester`.