Budowniczy

Budowniczy (ang. builder) - kreacyjny wzorzec projektowy, którego celem jest rozdzielenie sposobu tworzenia obiektów od ich reprezentacji inaczej mówiąc wzorzec ten stosowany jest do konstruowania obiektów poprzez wcześniejsze stworzenie jego fragmentów. Składamy od szczegółu do ogółu (np. budowanie domu). Obiekty mogą być rozmaitych postaci, a wszystko opiera się na jednym procesie konstrukcyjnym. W konkretnych budowniczych decydujemy o tym, jak dany obiekt jest tworzony. Na koniec wywołujemy wszystkie metody poszczególnych budowniczych i otrzymujemy obiekt końcowy. Wzorzec ten często występuje z wzorcami fabryki oraz kompozytu.

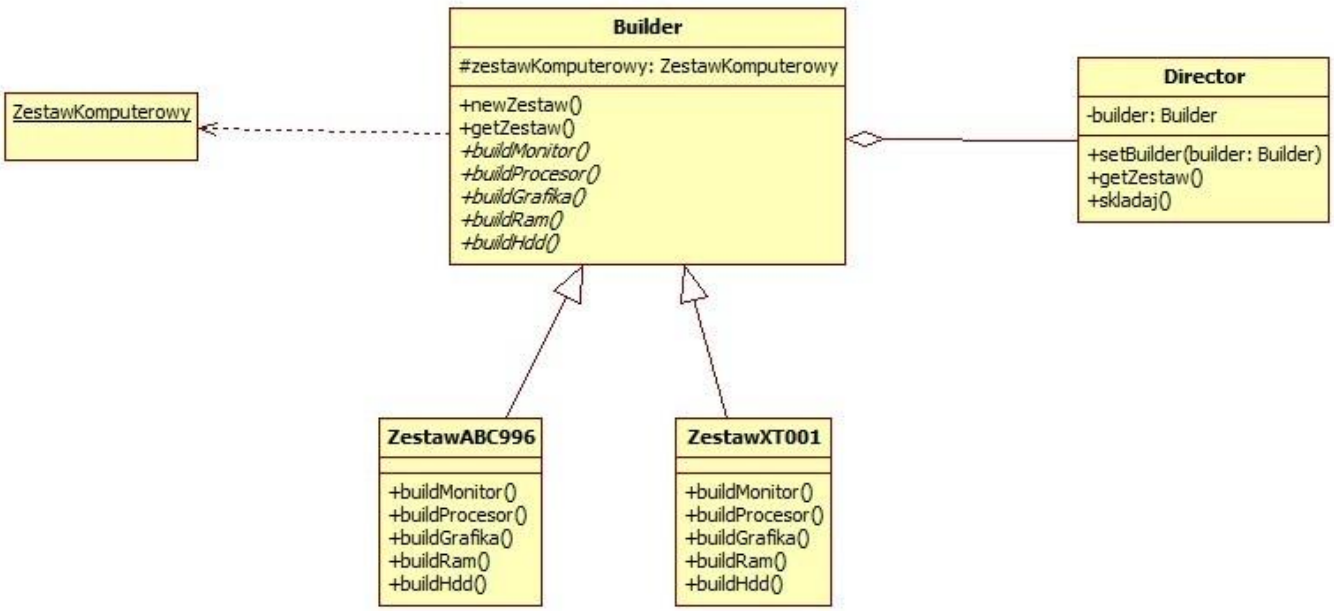
Oddziela tworzenie złożonego obiektu od jego reprezentacji, dzięki czemu ten sam proces konstrukcji może prowadzić do powstania różnych reprezentacji.

Wzorca Budowniczy należy używać gdy:

- algorytm tworzenia obiektu złożonego powinien być niezależny od składników tego obiektu i sposobu ich łączenia;
- proces konstrukcji musi umożliwiać tworzenia różnych reprezentacji generowanego obiektu.

Budowa wzorca

Standardowo wzorzec składa się z dwóch podstawowych obiektów. Pierwszy z nich oznaczony jest jako Budowniczy – jego celem jest dostarczenie interfejsu do tworzenia obiektów nazywanych w tym kontekście produktami. Drugim obiektem jest obiekt oznaczony jako Konkretny Budowniczy, a jego celem jest tworzenie konkretnych reprezentacji produktów przy pomocy zaimplementowanego interfejsu Budowniczego. W Konkretnym Budowniczym zawarte są procedury odpowiedzialne za konstrukcje i inicjalizację obiektu. Strukturę wzorca uzupełnia obiekt Kierownika (czasami nazywany także Dyrektorem, Nadzorcą), który zleca konstrukcję produktów poprzez obiekt Budowniczego dbając o to, aby proces konstrukcyjny przebiegał w odpowiedniej kolejności.



Implementacja przykładu

```java
package wzorce.budowniczy;

/* nasz glowny interface */
abstract class Builder {

    protected ZestawKomputerowy zestawKomputerowy;

    public void newZestaw() {
        zestawKomputerowy = new ZestawKomputerowy();
    }

    public ZestawKomputerowy getZestaw() {
        return zestawKomputerowy;
    }

    public abstract void buildMonitor();
    public abstract void buildProcesor();
    public abstract void buildGrafika();
    public abstract void buildRam();
    public abstract void buildHdd();
}
```

```java
package wzorce.budowniczy;

/* keirownik */
class Director {

    private Builder builder;

    public void setBuilder(Builder builder) {
        this.builder = builder;
    }

    public ZestawKomputerowy getZestaw() {
        return builder.getZestaw();
    }

    public void skladaj() {
        builder.newZestaw();
        builder.buildMonitor();
        builder.buildProcesor();
        builder.buildHdd();
        builder.buildRam();
        builder.buildGrafika();
    }
}
```

```java
package wzorce.budowniczy;

/* produkt koncowy */
class ZestawKomputerowy {

    private String monitor;
    private String procesor;
    private String grafika;
    private String ram;
    private String hdd;

    public void setMonitor(String monitor) {
        this.monitor = monitor;
    }
    public void setProcesor(String procesor) {
        this.procesor = procesor;
    }
    public void setGrafika(String grafika) {
        this.grafika = grafika;
    }
    public void setRam(String ram) {
        this.ram = ram;
    }
    public void setHdd(String hdd) {
        this.hdd = hdd;
    }
    public void show() {
        if (monitor != null) {
            System.out.println("Monitor = " + monitor);
        }
        if (procesor != null) {
            System.out.println("Procesor = " + procesor);
        }
        if (grafika != null) {
            System.out.println("Grafika = " + grafika);
        }
        if (ram != null) {
            System.out.println("RAM = " + ram);
        }
        if (hdd != null) {
            System.out.println("HDD = " + hdd);
        }
    }
}
```

```java
package wzorce.budowniczy;

public class ZestawABC996 extends Builder {

    @Override
    public void buildMonitor() {
        zestawKomputerowy.setMonitor("LG");
    }
    @Override
    public void buildProcesor() {
        zestawKomputerowy.setProcesor("INTEL");
    }
    @Override
    public void buildGrafika() {
//zestaw nie obejmuje karty graficznej
    }
    @Override
    public void buildRam() {
        zestawKomputerowy.setRam("DDR4");
    }
    @Override
    public void buildHdd() {
        zestawKomputerowy.setHdd("SanDisk");
    }
}
```

```java
package wzorce.budowniczy;

import java.util.Scanner;

public class ZestawXT001 extends Builder {

    @Override
    public void buildMonitor() {
        zestawKomputerowy.setMonitor("Benq 19");
    }
    @Override
    public void buildProcesor() {
        zestawKomputerowy.setProcesor("amd");
    }
    @Override
    public void buildGrafika() {
        zestawKomputerowy.setGrafika("ATI");
    }
    @Override
    public void buildRam() {
        zestawKomputerowy.setRam("DDR3");
    }
    @Override
    public void buildHdd() {
        Scanner in = new Scanner(System.in);
        int t;
        while (true) {
            System.out.println("Dysk do wyboru: (1) SanDisc, (2)
Segate, (3) WesternDigital");
            t = in.nextInt();
            if (t > 0 && t < 4) {
                break;
            }
        }
        String wynik = "";
        switch (t) {
            case 1:
                wynik = "SanDisc";
                break;
            case 2:
                wynik = "Segate";
                break;
            case 3:
                wynik = "WesternDigital";
                break;
            default:
                break;
        }
        zestawKomputerowy.setHdd(wynik);
    }
}
```

```java
package wzorce.budowniczy;

public class Main {

    public static void main(String[] args) {

        Director szef = new Director();
        Builder builder = new ZestawXT001();
        Builder builder2 = new ZestawABC996();

        System.out.println("\nZESTAW1");
        szef.setBuilder(builder);
        szef.skladaj();
        ZestawKomputerowy zestaw1 = szef.getZestaw();
        szef.setBuilder(builder2);
        szef.skladaj();
        ZestawKomputerowy zestaw2 = szef.getZestaw();
        zestaw1.show();
        System.out.println("\n\nZESTAW2");
        zestaw2.show();
    }
}
```

ZAD.1.

Utwórz model realizujący zamawianie pizzy. Zaimplementuj klasę budującą pizzę – ***PizzaBuilder*** (abstrakcyjny budowniczy) posiadającą pole typu ***Pizza*** oraz możliwość pobierania pizzy, tworzenia nowego produktu – ***Pizza***, i możliwość definiowania atrybutów pizzy takich jak rodzaj ciasta, sosu i zestawu składników(metody abstrakcyjne). Ponadto projekt musi zawierać klasę ***Kelner*** będącą nadzorcą procesu powstawania pizzy(produktu). Posiada on pole klasy ***PizzaBuilder*** , może też pobrać pizzę oraz ją utworzyć. Produkt ***Pizza*** powinien posiadać:

- Ciasto (np. cienkie, grube, średnie);
- Sos (np. pikantny, łagodny);
- Zestaw składników (np. pepperoni+salami, boczek+kukurydza, ananas+szynka).
- Przesłoniętą metodę toString wypisującą dane dotyczące zamówionej pizzy

(metody zwracające łańcuch znaków np.

```
public void buildSauce(){
    return „mild”;
}
```

Zaprojektować dwóch konkretnych budowniczych (np. ***PizzaHawajska*** i ***PizzaPepperoni***). W klasie reprezentującej proces zamawiania pizzy utwór obiekt kelnera, dwóch konkretnych budowniczych pizzy oraz zamówić którąś z pizz u kelnera (wywołać metodę ustawiającą rodzaj zamówionej pizzy na rzecz obiektu kelner) oraz zlecić jej wykonanie kelnerowi a następnie pobrać wybraną pizzę i wyświetlić jej właściwości.