Wzorce projektowe

Fabryki

Fabryki jako wzorzec

- Fabryki należą do grupy wzorców konstrukcyjnych
- Umożliwiają tworzenie obiektów bez eksponowania szczegółów tego procesu
- Tworzą konkretny obiekt na podstawie przekazanych parametrów
- Dzięki przeniesieniu procesu tworzenia obiektów do dedykowanej klasy pisany kod będzie bardziej ogólny i elastyczny
- Wyróżniamy trzy typy fabryk: Simply Factory, Factory Method oraz Abstract Factory

Simple Factory
(prosta fabryka)

- Wzorzec należący do grupy wzorców konstrukcyjnych
- W zależności od dostarczonych danych, zwraca instancję jednej z możliwych klas
- Zwracane klasy dziedziczą z tej samej klasy podstawowej mając takie same metody, ale każda z nich wykonuje swoje zadania w inny sposób
- Pozwala na hermetyzację procesu tworzenia obiektów różnych klas
- Dzięki rozdzieleniu miejsca tworzenia obiektów od ich wykorzystania kod staje się elastyczniejszy

Simple Factory
(prosta fabryka)

- Założenia przykładu
  - Program obsługujący sprzedaż biletów lotniczych na pojedynczym lotnisku
  - Każdy bilet ma swoją nazwę, cenę, indywidualną metodę komunikującą się z danymi liniami lotniczymi send()
  - Każdy bilet ma status rezerwacji, który można zmienić

Simple Factory
(prosta fabryka)

```java
public class Ticket {

    private double price;
    private String name;
    private boolean reservation;

    public Ticket(String name, double price) {
        this.name = name;
        this.price = price;
        this.reservation = false;
    }

    public void send() {
    }

    public boolean isReservation() {
        return reservation;
    }

    public void setReservation() {
        this.reservation = true;
    }

    public double getPrice() {
        return price;
    }

    public String getName() {
        return name;
    }
}
```

```java
public class TicketLot extends Ticket{

    public TicketLot() {
        super("LOT", 123.0);
    }

    @Override
    public void send(){}

}
```

```java
public class TicketLufthansa extends Ticket{

    public TicketLufthansa() {
        super("Lufthansa", 234.0);
    }

    @Override
    public void send(){}
}
```

```java
public class TicketService {

    private SimpleFactory factory;

    public TicketService(SimpleFactory factory) {
        this.factory = factory;
    }

    public Ticket buyTicket(String type) {
        Ticket ticket = factory.createTicket(type);

        ticket.setReservation();
        ticket.send();

        return ticket;
    }
}
```

```java
public class SimpleFactory {

    public Ticket createTicket(String type) {

        if (type.equals("LOT")) {
            return new TicketLot();
        } else if (type.equals("Lufthansa")) {
            return new TicketLufthansa();
        } else {
            return new Ticket(type, 0);
        }
    }

}
```

Simple Factory
(prosta fabryka)

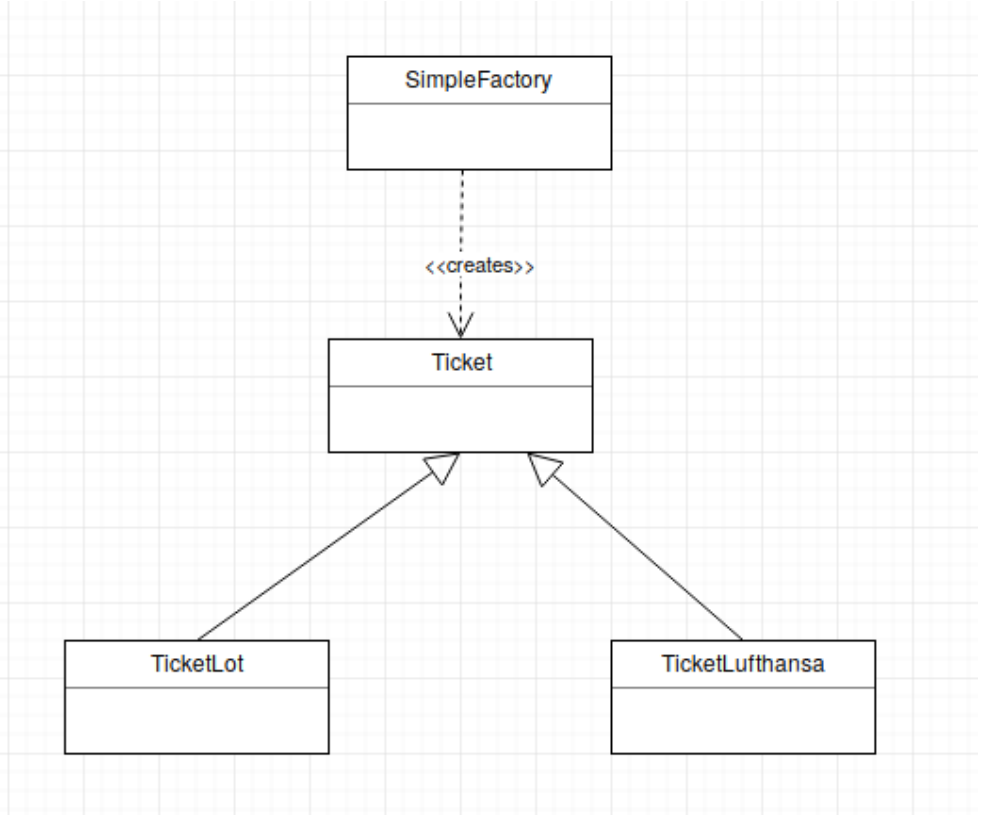

Diagram klas Simple Factory

Factory Method
(metoda fabrykująca)

- Rozwinięcie idei Prostej Fabryki likwidując pojedyncza klasę decyzyjną
- Klasy stojące wyżej w hierarchii dziedziczenia przenoszą podejmowanie decyzji do klas pochodnych

Factory Method
(metoda fabrykująca)

- Założenia przykładu
  - Program obsługujący sprzedaż biletów lotniczych **dla wielu lotnisk**
  - Każdy bilet ma swoją nazwę, cenę, indywidualną metodę komunikującą się z danymi liniami lotniczymi send()
  - Każdy bilet ma status rezerwacji, który można zmienić
  - **Cena biletu jest inna w zależności od linii lotniczych oraz lotniska**

Factory Method
(metoda fabrykująca)

- Klasę odpowiedzialną za tworzenie biletów modyfikujemy aby stała się abstrakcyjna – stanie się klasą ogólną
- Dziedziczące po niej klasy będą prezentować serwis biletów dla konkretnych lotnisk

```java
public class TicketService {

    private SimpleFactory factory;

    public TicketService(SimpleFactory factory) {
        this.factory = factory;
    }

    public Ticket buyTicket(String type) {
        Ticket ticket = factory.createTicket(type);

        ticket.setReservation();
        ticket.send();

        return ticket;
    }
}
```

Simple Factory

```java
public abstract class TicketService {

    public TicketService() {

    }

    protected abstract Ticket createTicket(String type);

    public Ticket buyTicket(String type) {
        Ticket ticket = createTicket(type);

        ticket.setReservation();
        ticket.send();

        return ticket;
    }
}
```

Factory Method

Factory Method
(metoda fabrykująca)

- Podklasy tworzące serwis biletów dla konkretnych lotnisk

```java
public class WarsawTicketService extends TicketService {

    @Override
    protected Ticket createTicket(String type) {
        if (type.equals("LOT")) {
            return new WTicketLot();
        } else {
            return new Ticket(type, 0);
        }
    }

}
```

```java
public class BerlinTicketService extends TicketService{

    @Override
    protected Ticket createTicket(String type) {
        if (type.equals("LOT")) {
            return new BTicketLot();
        } else if (type.equals("Lufthansa")) {
            return new BTicketLufthansa();
        } else {
            return new Ticket(type, 0);
        }
    }

}
```

- Oraz ich wywołanie

```java
public class P1_factoryMethod {


    public static void main(String[] args) {
        TicketService tcBerlin = new BerlinTicketService();
        System.out.println(tcBerlin.buyTicket("LOT").getPrice());


        TicketService tcWarsaw = new WarsawTicketService();
        System.out.println(tcWarsaw.buyTicket("LOT").getPrice());
    }

}
```

Factory Method
(metoda fabrykująca)

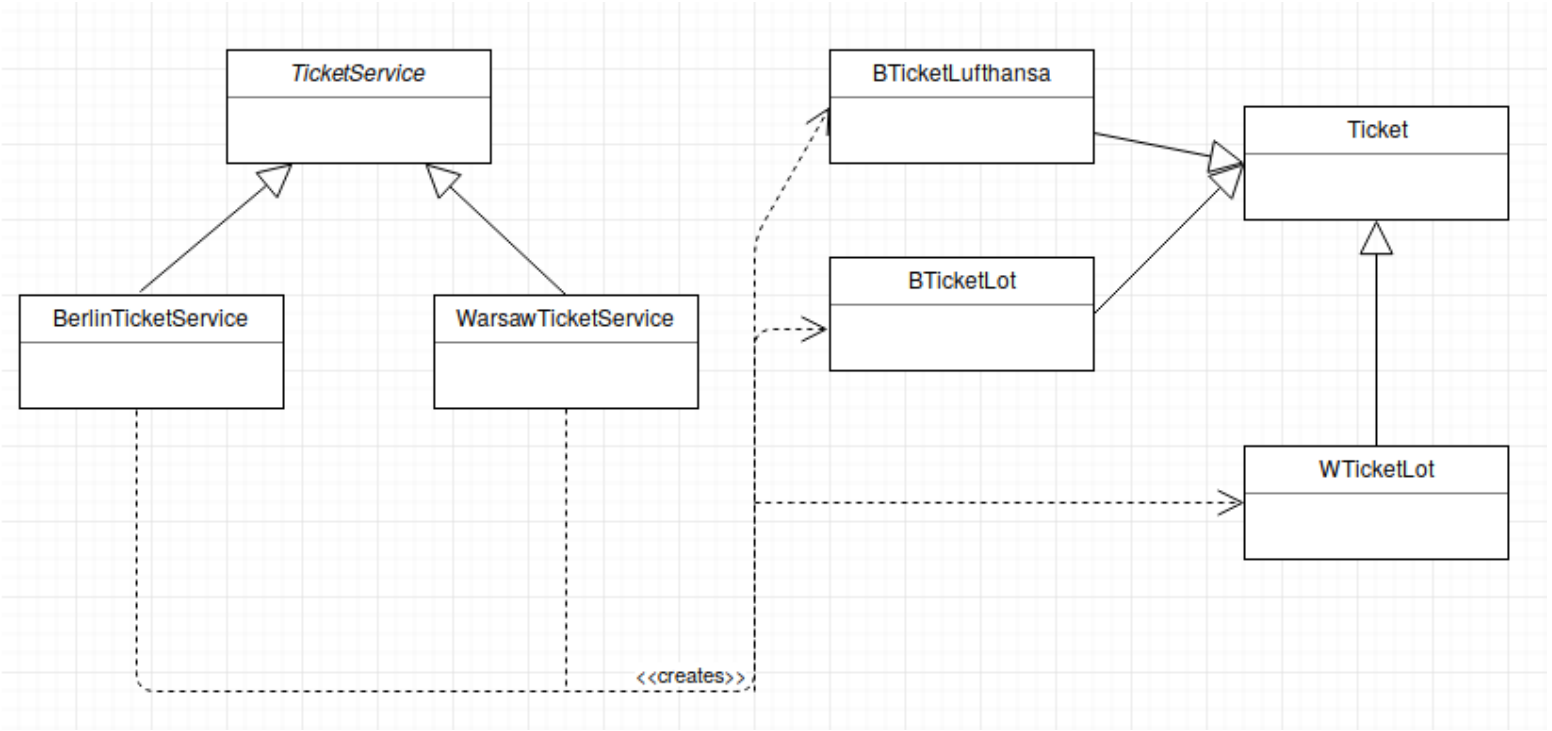

Diagram klas Factory Method

Abstract Factory
(fabryka abstrakcji)

- Wzorzec należący do grupy wzorców konstrukcyjnych będący modyfikacją Factory Method
- Celem wzorca jest tworzenie grupy powiązanych ze sobą obiektów
- Przykładem zastosowania jest implementacja obsługi wielu odmian interfejsów użytkownika – gdy „poinformujemy” naszą fabrykę, że chcemy otrzymać interfejs np. dla systemu Windows, otrzymamy obiekty przeznaczone dla tego środowiska (przyciski, pola wyboru, okna)
- Głównym celem wzorca jest tworzenie lepszego odizolowania od siebie generowanych klas

Abstract Factory
(fabryka abstrakcji)

- Założenia przykładu:
  - Aplikacja sprzedaży komputerów
  - Posiadamy dwa rodzaje komputerów: standard dla zwykłego klienta oraz wersja serwer

Abstract Factory
(fabryka abstrakcji)

- Fabryka abstrakcyjna, która tworzy podzespoły komputera w zależności od jego typu

```
public interface ComputerElementsFactory {
    Memory buyMemory();
    Procesor buyProcesor();
    Disc buyDisc();
}
```

- Podzespoły komputera – fabryka i podzespoły są abstrakcjami

```
public interface Disc {
    public void createDisc();
}
```

```
public interface Memory {
    public void createMemory();
}
```

```
public interface Procesor {
    public void createProcesor();
}
```

Abstract Factory
(fabryka abstrakcji)

- Dla każdego typu komputera tworzymy osobne implementacje

```
public class ServerComputerFactory implements  ComputerElementsFactory{

    @Override
    public Memory buyMemory() {
        return new ServerMemory();
    }

    @Override
    public Procesor buyProcesor() {
        return new ServerProcesor();
    }

    @Override
    public Disc buyDisc() {
        return new ServerDisc();
    }

}
```

```
public class StandardComputerFactory implements  ComputerElementsFactory{

    @Override
    public Memory buyMemory() {
        return new StandardMemory();
    }

    @Override
    public Procesor buyProcesor() {
        return new StandardProcesor();
    }

    @Override
    public Disc buyDisc() {
        return new StandardDisc();
    }

}
```

Abstract Factory
(fabryka abstrakcji)

- Wywołanie fabryki

```
public class P2_abstractFactory {

    public static void main(String[] args) {
        createComputer(new StandardComputerFactory());
        createComputer(new ServerComputerFactory());
    }

    private static void createComputer(ComputerElementsFactory cef)
    {
        Disc disc = cef.buyDisc();
        disc.createDisc();

        Memory memory = cef.buyMemory();
        memory.createMemory();

        Procesor procesor = cef.buyProcesor();
        procesor.createProcesor();
    }

}
```

- Posiadamy jedną metodę createComputer z jednym parametrem. Dzięki wykorzystaniu abstrakcji, możemy wykorzystując ten sam kod wywołać różne implementacje fabryki

Abstract Factory
(fabryka abstrakcji)

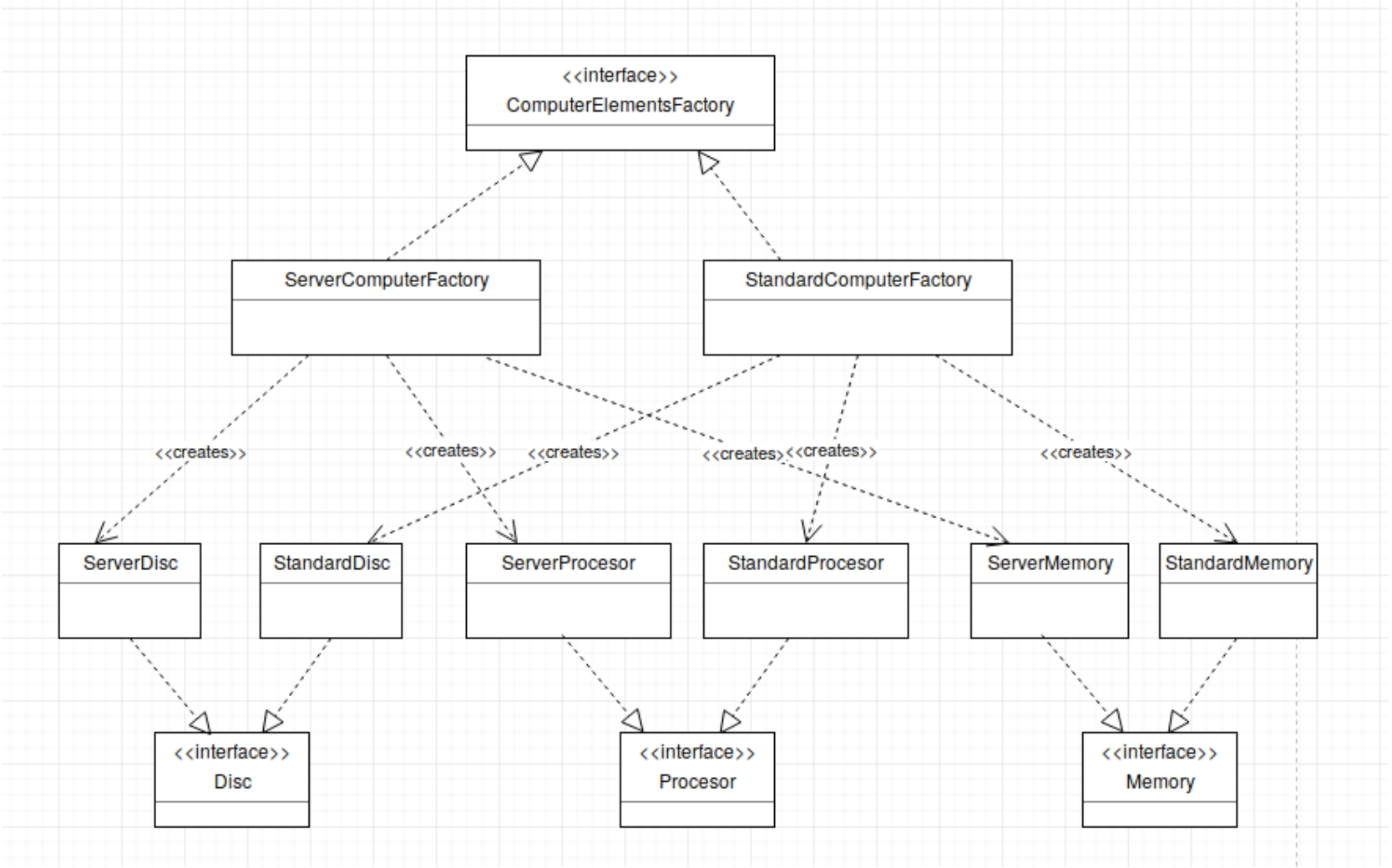

Diagram klas Abstract Factory