

# Proces Refaktoryzacji kodu

Refaktoryzacja (ang. *Refactoring*) to proces w którym nie wytwarza się nowej funkcjonalności oprogramowania, lecz poprawia się jakość już istniejącego kodu. Często ten proces ogranicza redundancję (nadmiarowość) kodu oraz wprowadza nowe standardy. Jest to kosztowny i na pierwszy rzut oka niepotrzebny proces, jednak dobrze przeprowadzona refaktoryzacja pozwala w przyszłości na łatwiejsze modyfikowanie kodu oraz zwiększa jego czytelność.

W projekcie przeprowadzono refaktoryzację polegającą na utworzeniu bazowego serwisu, który zapewnia potomnym serwisom podstawowe operacje CRUD. Przeniesiono również z kontrolerów do serwisów fragmenty kodu odpowiedzialne za inicjowanie oraz aktualizację modeli. Opisywane zmiany przedstawia commit: 40f4cd60777b85feb838b4a552c9a82c0bfdbf50 z dnia 05.01.2017r.

Pierwszym i najtrudniejszym krokiem tego procesu było utworzenie abstrakcyjnej klasy BaseService, korzystającej z typów generycznych. Miała zapewnić podstawowe operacje CRUD, więc wykorzystano interfejs CrudRepository z modułu Spring-data. Ostateczną wersję klasy przedstawiono poniżej:

```
package piotrek.k.flats.Service;

imports ...

@Service
public abstract class BaseService<T extends CrudRepository<M, Long>,M> {

    @Autowired
    protected T daoInterface;

    public M findById(Long id) {
        return daoInterface.findOne(id);
    }

    public List<M> findAll() {
        return (List<M>) daoInterface.findAll();
    }

    public void addOrUpdate(M model) {
        daoInterface.save(model);
    }

    public void delete(Long id) {
        daoInterface.delete(id);
    }

}
```

Taka implementacja serwisu zapewnia podstawowe operacje CRUD na dowolnym modelu M, pozwala również pobrać listę wszystkich modeli typu M z bazy. Dodatkowo w każdym potomku dziedziczącym po tym serwisie dostępny jest już odpowiedni interfejs DAO pod nazwą daoInterface, w którym powinna się znaleźć odpowiednia implementacja dla wybranego modelu.

Jako przykład zastosowania takiego rozwiązania posłużę się serwisem odpowiedzialnym za nieruchomości. Przed implementacją powyższego abstrakcyjnego serwisu, serwis nieruchomości wyglądał następująco:

```
@Service
public class RealEstateService {

    @Autowired
    private IRealEstate iRealEstate;

    public RealEstate findById (Long id){
        return iRealEstate.findById(id);
    }

    public List<RealEstate> findByUser (User user){
        return iRealEstate.findByUser(user);
    }

    public void addRealEstate(RealEstate realEstate){
        iRealEstate.save(realEstate);
    }

    public void deleteRealEstate(Long id){
        iRealEstate.delete(id);
    }

    public boolean itIsMyRealEstate(User user, RealEstate realEstate){
        return realEstate.getUser().getId().equals(user.getId());
    }
}
```

Po zastosowaniu klasy bazowej kod można skrócić do postaci:

```
@Service
public class RealEstateService extends BaseService<IRealEstateInterface, RealEstate> {

    public List<RealEstate> findByUser (User user){
        return daoInterface.findByUser(user);
    }

    public boolean itIsMyRealEstate(User user, RealEstate realEstate) {
        return realEstate.getUser().getId().equals(user.getId());
    }
}
```

Kolejnym celem refaktoryzacji było uproszczenie, a co za tym idzie zwiększenie czytelności kodu w kontrolerach. Przed tym procesem w kontrolerach odbywało się inicjalizowanie i aktualizacja pól modelu. Bardzo podobny kod występował dwa razy w obrębie jednego kontrolera. Postanowiłem to zadanie powierzyć odpowiedniemu serwisowi. Poniżej kod dwóch metod kontrolera nieruchomości sprzed refaktoryzacji:

```

@RequestMapping(value = "/add", method = RequestMethod.POST)
public String realEstateFormPost(@ModelAttribute("form") @Valid RealEstateDTO form,
BindingResult result) {
    if (result.hasErrors())
        return "realEstate/realEstateForm";
    else {
        RealEstate realEstate = new RealEstate();
        realEstate.setLocation(form.getLocation());
        realEstate.setRealEstateType(form.getRealEstateType());
        realEstate.setFloorArea(form.getFloorArea());
        realEstate.setPrice(form.getPrice());
        realEstate.setNumberOfRooms(form.getNumberOfRooms());
        realEstate.setHowOld(form.getHowOld());
        realEstate.setGarage(form.getGarage());
        realEstate.setParking(form.getParking());
        realEstate.setGarden(form.getGarden());
        realEstate.setCellar(form.getCellar());
        realEstate.setFloor(form.getFloor());
        realEstate.setMonitoring(form.getMonitoring());
        realEstate.setLift(form.getLift());
        realEstate.setOwnContribution(form.getOwnContribution());
        realEstate.setKmPerDay(form.getKmPerDay());
        realEstate.setMaintenanceCosts(form.getMaintenanceCosts());
        realEstate.setAccessToPublicTransport(form.getAccessToPublicTransport());
        realEstate.setAveragePriceInArea(form.getAveragePriceInArea());
        realEstate.setAdvertisementsLink(form.getAdvertisementsLink());
        realEstate.setNotes(form.getNotes());

        ZonedDateTime zdt = ZonedDateTime.now();
        Date date = Date.from(zdt.toInstant());
        realEstate.setSupplementDate(date);

        User user =
userService.getByUsername(SecurityContextHolder.getContext().getAuthentication().getName());
;

        realEstate.setUser(user);

        realEstateService.addRealEstate(realEstate);
        return "redirect:../realEstate/";
    }
}

@RequestMapping(value = "/edit-{id}", method = RequestMethod.POST)
public String editRealEstatePOST(@PathVariable("id") Long id, @ModelAttribute("form")
@Valid RealEstateDTO form, BindingResult result) {
    if (result.hasErrors()) {
        return "realEstate/realEstateForm";
    } else {
        RealEstate realEstate = realEstateService.findById(id);
        realEstate.setLocation(form.getLocation());
        realEstate.setRealEstateType(form.getRealEstateType());
        realEstate.setFloorArea(form.getFloorArea());
        realEstate.setPrice(form.getPrice());
        realEstate.setNumberOfRooms(form.getNumberOfRooms());
        realEstate.setHowOld(form.getHowOld());
        realEstate.setGarage(form.getGarage());
        realEstate.setParking(form.getParking());
        realEstate.setGarden(form.getGarden());
        realEstate.setCellar(form.getCellar());
        realEstate.setFloor(form.getFloor());
        realEstate.setMonitoring(form.getMonitoring());
        realEstate.setLift(form.getLift());
        realEstate.setOwnContribution(form.getOwnContribution());
        realEstate.setKmPerDay(form.getKmPerDay());
        realEstate.setMaintenanceCosts(form.getMaintenanceCosts());
        realEstate.setAccessToPublicTransport(form.getAccessToPublicTransport());
        realEstate.setAveragePriceInArea(form.getAveragePriceInArea());
        realEstate.setAdvertisementsLink(form.getAdvertisementsLink());
        realEstate.setNotes(form.getNotes());

        realEstateService.addRealEstate(realEstate);
        return "redirect:../realEstate/details-{id}";
    }
}

```

Po refaktoryzacji te metody wyglądały następująco:

```
@RequestMapping(value = "/add", method = RequestMethod.POST)
public String realEstateFormPost(@ModelAttribute("form") @Valid RealEstateDTO form,
BindingResult result) {
    if (result.hasErrors())
        return "realEstate/realEstateForm";
    else {
        realEstateService.addRealEstate(form);
        return "redirect:../realEstate/";
    }
}

@RequestMapping(value = "/edit-{id}", method = RequestMethod.POST)
public String editRealEstatePOST(@PathVariable("id") Long id, @ModelAttribute("form")
@Valid RealEstateDTO form, BindingResult result) {
    if (result.hasErrors()) {
        return "realEstate/realEstateForm";
    } else {
        RealEstate realEstate = realEstateService.findById(id);
        realEstateService.updateRealEstate(realEstate, form);
        return "redirect:../realEstate/details-{id}";
    }
}
```

Jak widać znacznie skrócił się kod w kontrolerze, co zwiększyło jego czytelność.

Funkcjonalności zostały przeniesione do stosownego serwisu, a ich implementacja wyglądała następująco:

```
public void addRealEstate(RealEstateDTO form) {
    RealEstate realEstate = new RealEstate();
    realEstate = initialize(realEstate, form);

    ZonedDateTime zdt = ZonedDateTime.now();
    Date date = Date.from(zdt.toInstant());
    realEstate.setSupplementDate(date);

    User user =
userService.getByUsername(SecurityContextHolder.getContext().getAuthentication().getName());
    realEstate.setUser(user);

    addOrUpdate(realEstate);
}

public void updateRealEstate(RealEstate realEstate, RealEstateDTO form) {
    realEstate = initialize(realEstate, form);
    addOrUpdate(realEstate);
}

private RealEstate initialize(RealEstate realEstate, RealEstateDTO form) {
    realEstate.setLocation(form.getLocation());
    realEstate.setRealEstateType(form.getRealEstateType());
    realEstate.setFloorArea(form.getFloorArea());
    realEstate.setPrice(form.getPrice());
    realEstate.setNumberOfRooms(form.getNumberOfRooms());
    realEstate.setHowOld(form.getHowOld());
    realEstate.setGarage(form.getGarage());
    realEstate.setParking(form.getParking());
    realEstate.setGarden(form.getGarden());
    realEstate.setCellar(form.getCellar());
    realEstate.setFloor(form.getFloor());
    realEstate.setMonitoring(form.getMonitoring());
    realEstate.setLift(form.getLift());
    realEstate.setOwnContribution(form.getOwnContribution());
    realEstate.setKmPerDay(form.getKmPerDay());
    realEstate.setMaintenanceCosts(form.getMaintenanceCosts());
    realEstate.setAccessToPublicTransport(form.getAccessToPublicTransport());
    realEstate.setAveragePriceInArea(form.getAveragePriceInArea());
    realEstate.setAdvertisementsLink(form.getAdvertisementsLink());
    realEstate.setNotes(form.getNotes());
    return realEstate;
}
```

Przeprowadzony proces refaktoryzacji poprawił jakość kodu i nie zmienił jego funkcjonalności. Statystyki commit'a wskazują, że udało się zmniejszyć objętość kodu:



Refaktoryzacja wpływa pozytywnie nie tylko na kod, na którym jest przeprowadzana, lecz poprawia umiejętności programisty, który przeprowadza ten proces. Aplikacje pisane przez niego w przyszłości będą wyglądały lepiej i nie będzie musiał „tracić czasu” na poprawianie tego typu błędów.

Celowo frazę „tracić czasu” ująłem w cudzysłów, ponieważ uważam, że czas spędzony nad refaktoryzacją nie jest czasem straconym, mimo że nie przynosi żadnych widocznych efektów.