Podręczna Checklista +30 Dobrych Praktyk

Czy wykonane?	Dobra Praktyka	Krótki opis				
	Nazewnictwo					
☐	Czy nazwy dobrze reprezentują odpowiednie modele i zachowania?	Najważniejszym, a często także najtrudniejszym zadaniem jest **odpowiednie nazwanie zmiennej**, metody lub klasy. Warto poświęcić na to trochę więcej czasu. Na etapie tworzenia nie zawsze mamy pełną jasność co do roli danej konstrukcji. Dlatego późniejsza refaktoryzacja nazw **nie powinna budzić obaw** — jest to prosta operacja. Kiedy już lepiej zrozumiemy, co dana rzecz dokładnie robi, łatwiej będzie znaleźć odpowiednią nazwę. Ważne, aby zastanowić się, **czy dana nazwa będzie zrozumiała** dla innego programisty, który nie zna szczegółów projektu. Na przykład, czy bez zaglądania do wnętrza metody będzie wiedział, co się w niej dzieje.				
	Projektowanie metod					
☐	Metoda powinna wykonywać jedno, dobrze zdefiniowane zadanie	Nie chodzi o to, żeby metoda składała się z jednej linijki kodu, lecz o to, by **realizowała jedną, spójną koncepcyjnie** czynność.				
☐	Metoda powinna być na jednym tym samym poziomie abstrakcji	Na poziomie jednej metody **nie chcemy mieszać mało istotnych szczegółów** implementacyjnych z kluczowymi koncepcyjnie elementami, ponieważ **utrudnia to zrozumienie** kodu. Nie ma potrzeby czytania szczegółów implementacji, gdy **wystarczy zrozumieć ogólną koncepcję.** Jeśli będziemy chcieli poznać więcej szczegółów, **możemy zajrzeć głębiej.** Warto wynieść te elementy, które nie pasują do aktualnego poziomu abstrakcji metody.				
☐	Metoda powinna mieć jak najmniej argumentów	Najlepsze są **metody z jak najmniejszą liczbą argumentów**. Im więcej argumentów, **tym trudniej zrozumieć** z samej sygnatury, co metoda robi. Metody z więcej niż dwoma argumentami są trudniejsze do odczytania. **Warto dążyć do upraszczania metod**, redukując liczbę argumentów, aby były łatwiejsze do zrozumienia.				
☐	Konwertuj metody dwuargumentowe na jednoargumentowe, gdy tylko jest to możliwe	Jeśli mamy metodę w klasie serwisowej lub pomocniczej z dwoma argumentami, warto **rozważyć przeniesienie jej do klasy jednego z tych argumentów**, ponieważ to może tam **znajduje się najwięcej informacji** potrzebnych do wykonania zadania. Dzięki temu będziemy mogli wywoływać metodę bezpośrednio na obiekcie, redukując liczbę argumentów do jednego.				
☐	Unikaj argumentów wyjściowych w parametrach metody	Jeśli metoda przyjmuje jeden argument i nic nie zwraca, to możemy domyślać się, że **coś się wydarzyło**, np. dane zostały zapisane na bazie. Natomiast jeśli metoda **miałaby więcej argumentów i nie zwracałaby wartości**, mogłoby być **trudno określić** po sygnaturze, który z argumentów został zmodyfikowany i w jaki sposób. Dlatego chcemy unikać agumentów wyjściowych.				
☐	Wydzielaj metody, nie tylko po to, aby je reużywać	Dzięki temu możemy nazwać daną czynność, **przekazać intencję** danego kawałka kodu, co sprawi, że Ty lub inni programiści w przyszłości **łatwiej zrozumieją**, co dzieje się w tym fragmencie kodu.				
☐	Korzystaj ze statycznych metod fabrycznych	W Javie obiekty możemy tworzyć przy użyciu statycznych metod fabrycznych, które są statycznymi metodami **zwracającymi nowe instancje** klasy. Zalety takiego podejścia to **możliwość nadania konkretnej** nazwy np. Distance.ofMiles(), Distance. ofKilometers(), **tworzenia wielu** różnych "konstruktorów", czy zwracania już istniejących instancji zamiast tworzenia nowych np. Boolean.TRUE. Przykłady popularnych nazw to: of, from, czy valueOf, a **ich użycie zwiększa czytelność i elastyczność kodu**.				
☐	Upraszczaj logikę	Możemy **przeanalizować dane rozwiązanie** i spróbować zapisać je w prostszy sposób. Często da się to zrobić. Możemy spróbować rozbić złożone metody na mniejsze, bardziej zrozumiałe jednostki. Możemy zwracać dobrze opisane obiekty. Możemy tak **organizować kod, aby było widać co krok po kroku dzieje się** w danej metodzie bez wchodzenia w szczegóły. Możemy dążyć do tego, aby można było **czytać kod jak książkę.**				
☐	Nie bój się jednolinijkowców	Dobrym przykładem są **złożone warunki if**. Często są one trudne do zrozumienia, dlatego **warto przenieść je do osobnej metody**. Dzięki temu, ktoś od razu będzie wiedział, co się tam dzieje, **bez konieczności zagłębiania się** w skomplikowaną logikę warunkową.				