Materiały powtórzeniowe

Singleton

Witaj w materiałach powtórzeniowych przygotowanych specjalnie dla Ciebie. Znajdziesz tutaj powtórkę materiału z ostatniej lekcji oraz zadania do wykonania dla chętnych. Z pewnością sobie poradzisz. Powodzenia!

Cel lekcji

Celem lekcji jest zrozumienie, czym są wzorce projektowe i dlaczego są istotne w programowaniu. Poznamy zasady działania i zastosowanie wzorca Singleton.

Powtórzenie wiadomości

1. Co jest interfejs?

Interfejs to rodzaj „kontraktu” który wymusza na klasie zaimplementowanie określonych metod, właściwości lub zdarzeń. Jedna klasa może implementować **wiele** interfejsów, w przeciwieństwie do dziedziczenia po klasie, gdzie jest tylko **jedna** klasa bazowa.

2. Co to jest klasa abstrakcyjna?

Klasa abstrakcyjna to rodzaj klasy (a więc pewnego „przepisu” na obiekt), która **nie może być zainicjalizowana** (nie można utworzyć jej instancji bezpośrednio przez new Class). Co najmniej jedna z metod w klasie abstrakcyjnej jest metodą *abstrakcyjną*, czyli taką, której **nie ma ciała** (implementacji) – musi ona zostać zaimplementowana w klasie dziedziczącej.

Czym są wzorce projektowe?

- To zestawy sprawdzonych rozwiązań typowych problemów programistycznych.
- Przykładowe kategorie wzorców: kreacyjne (np. Singleton), strukturalne (np. Adapter), behawioralne (np. Observer).
- **Dlaczego warto je znać?** Ułatwiają komunikację w zespole, przyspieszają proces tworzenia oprogramowania, pomagają pisać lepszy, bardziej zrozumiały kod.

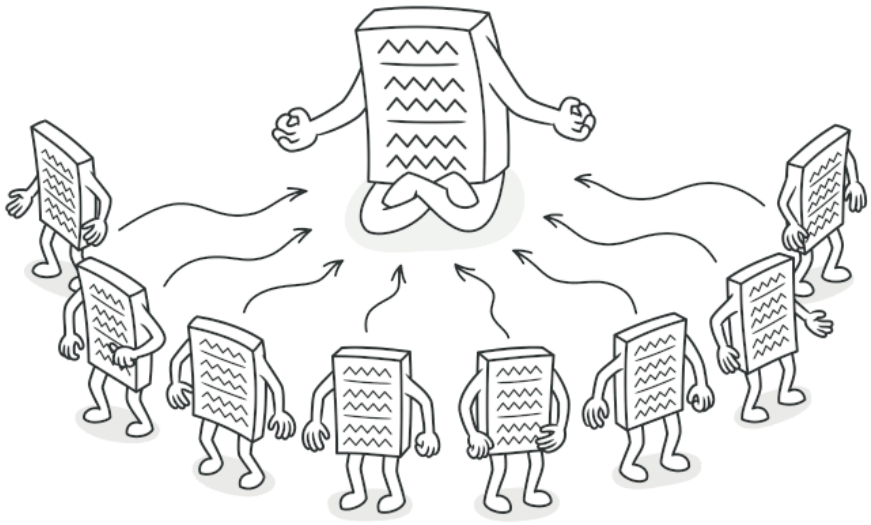
Po więcej wzorców projektowych:

https://refactoring.guru/pl/design-patterns/csharp

Singleton

Singleton jest kreacyjnym wzorcem projektowym, który pozwala zapewnić istnienie wyłącznie jednej instancji danej klasy. Ponadto daje globalny punkt dostępowy do tejże instancji.

Singleton to jeden z najprostszych wzorców projektowych, który dobrze wprowadza w świat wzorców jako takich. Łatwo go pokazać w kodzie i zrozumieć jego ideę – prywatny konstruktor, jedna instancja, globalny punkt dostępu. Dzięki temu od razu widać, jak wzorzec projektowy rozwiązuje konkretny problem (kontrola liczby tworzonych obiektów), co pomaga zrozumieć motywację do stosowania wzorców w ogóle.



Kiedy warto go użyć?

- Gdy chcemy kontrolować dostęp do wspólnego zasobu (np. logi systemowe, konfiguracja, połączenie z bazą danych).
- Gdy potrzebujemy scentralizować pewne funkcje (np. menedżer stanu aplikacji).

Jakie ma wady i zalety?

Zalety:

- Łatwy dostęp do instancji z dowolnego miejsca w kodzie.
- Kontrola nad liczbą tworzonych obiektów.

Wady:

- Może prowadzić do trudności w testowaniu (trudniej zamockować dostęp do Singletona). W testach jednostkowych często „podmieniamy" obiekty rzeczywiste (np. komunikujące się z bazą danych) na tzw. **mocki** (sztuczne obiekty), by kontrolować zachowanie i zależności.
- Niekiedy bywa nadużywany – w dużych projektach może utrudniać utrzymanie kodu.

Wielowątkowość

Gdy nasz program działa w **kilku wątkach** jednocześnie (np. kilka rzeczy dzieje się na raz), może się zdarzyć, że dwa wątki w tym samym momencie spróbują stworzyć nowy obiekt Singletona. Aby temu zapobiec, używamy specjalnego obiektu _lock i instrukcji lock(_lock).

Kiedy jeden wątek wejdzie do sekcji lock(_lock), inne wątki muszą zaczekać na swoją kolej (nie mogą wejść do tego fragmentu kodu). Dzięki temu **nie** powstanie sytuacja, w której dwie różne kopie Singletona zostaną utworzone w tym samym czasie. To zapewnia, że zawsze mamy tylko **jedną** instancję klasy Singleton.