LEKCJA 11 – Debugowanie

Debugowanie, z ang. odrobaczanie, polega na sprawdzaniu wykonywania programu (czy jest zgodne z oczekiwaniami) poprzez jego zatrzymywanie, w wybranych miejscach. Jest to funkcja każdego dobrego IDE. Znacznie ułatwia ona pracę programisty, a w szczególności wyszukiwanie błędów w kodzie. Bez debugera (narzędzia do debugowania) trzeba ręcznie wypisywać wszystkie wartości, które chcemy sprawdzić, do konsoli, co wydłuża proces i zaciemnia obraz programu. Narzędzie to jest tym bardziej potrzebne, że debugowanie zajmuje przeciętnie 80 procent czasu pracy programisty. Można spróbować go jednak nieco skrócić poprzez stosowanie testów.

# Debuger w Visual Studio

## Utworzenie/usunięcie punktu zatrzymania programu (ang. *Breakpoint*)

1. Kliknięcie myszką na lewym pasku okna edytora kodu, na wysokości linii (polecenia/instrukcji) w której chcemy wstawić/usunąć *Breakpoint*. Jeżeli w danej linii znajduje się kilka instrukcji to wykonywanie programu zostanie wstrzymane przed pierwszą z nich. Tym sposobem można wstawić w linii tylko jeden *Breakpoint*.
2. Alternatywnie można użyć skrótu F9, co utworzy/usunie punkt zatrzymania przed poleceniem, w którym się właśnie znajdujemy. Jeżeli w jednej linii znajduje się wiele instrukcji, to tym sposobem można w niej utworzyć kilka punktów zatrzymania, maksymalnie po jednym na każdą instrukcję. Można również utworzyć tylko *Breakpoint* dla kolejnej (nie pierwszej) instrukcji w linii.
3. Ostatnią metodą jest kliknięcie na wybraną instrukcję (instrukcję przed którą chcemy zatrzymać wykonywanie programu) prawym przyciskiem myszki i wybranie opcji *Breakpoint* > *Insert Breakpoint*. Sposób ten działa analogicznie do użycia skrótu F9.

Polecenie którego dotyczy wstawiony punkt zatrzymania programu zaznaczone będzie w oknie edytora kodu czerwoną ramką. Wykonywanie programu, uruchomionego w trybie *Debug*, zostanie zatrzymane w miejscu w którym utworzyliśmy *Breakpoint*. Punktów zatrzymania można wstawić w programie dowolną ilość (maksymalnie jeden na polecenie). Można je również tworzyć i usuwać podczas pracy programu, a działania te zostaną zachowane również po jego zakończeniu. Działają one jednak tylko w IDE w którym je utworzyliśmy. Podczas działania programu polecenie przed którego wykonaniem praca aplikacji została w danym momencie wstrzymana jest zaznaczone w oknie edytora kodu żółtą ramką.

## Kontynuacja debugowania

Kiedy praca naszego programu została już wstrzymana możemy ją kontynuować na kilka sposobów.

### Przejście do kolejnego polecenia

Możemy przejść do kolejnego polecenia (wykonać tylko jedną instrukcję i ponownie zatrzymać program). W tym celu możemy użyć skrótu F10, odpowiedniego przycisku z paska narzędzi (*Step Over*), opcji z menu głównego (*Debug* > *Step Over*) lub sposobu opisanego niżej jako „Kontynuowanie działania programu do wybranej linii”, wybierając kolejną linię.

### Wejście do wnętrza klasy/metody

Zamiast przechodzić do kolejnego polecenia można wejść do wnętrza metody/klasy użytej w instrukcji przed którą program został wstrzymany. W tym celu możemy użyć skrótu F11, odpowiedniego przycisku z paska narzędzi (*Step Into*), opcji z menu głównego (*Debug* > *Step Into*) lub sposobu opisanego niżej jako „Kontynuowanie działania programu do wybranej linii” wybierając pierwszą linię implementacji tej metody.

### Wyjście z wnętrza klasy/metody

Debuger Visual Studio umożliwia również wyjście z metody/klasy w której właśnie się znajdujemy. W tym celu możemy użyć skrótu *Shift* + F11, odpowiedniego przycisku z paska narzędzi (*Step Out*) lub opcji z menu głównego (*Debug* > *Step Out*). Powoduje to przejście do miejsca w kodzie po wykonaniu metody, w której właśnie się znajdujemy. Tak więc jeżeli znajdujemy się w metodzie *Main* spowoduje to zakończenie działania programu (kontynuację jego pracy do kolejnego *Breakpointa* lub innego zatrzymania niezwiązanego z debugerem). W innym wypadku przechodzimy do poleceń metody znajdującej się o poziom wyżej. Załóżmy, że zatrzymaliśmy program przed instrukcją przypisania wyniku działania jakiejś metody do zmiennej. Np.:

Person me = new Person("Małgorzata", "Mielczarek");

Przeszliśmy następnie do wnętrza konstruktora *Person*. Po sprawdzeniu w nim co chcieliśmy wyszliśmy z jego wnętrza. Wykonywanie metody *Person* zostało więc zakończone. Krok wyjścia spowodował przeniesienie wykonywania programu przed operację przypisania (przed powyższą przykładową instrukcję).

### Kontynuowanie działania programu do wybranej linii

Możemy również kontynuować wykonywanie programu zatrzymując go dopiero w wybranym miejscu. W tym celu możemy oczywiście wstawić kolejny punkt zatrzymania programu (zgodnie z opisem z punktu „Utworzenie/usunięcie punktu zatrzymania programu (ang. *Breakpoint*)”) i kontynuować jego wykonywanie zgodnie ze sposobami opisanymi poniżej w punkcie „Kontynuowanie działania programu”. Można to również zrobić bez wstawiania nowych *Breakpointów*. Podczas działania programu w konfiguracji *Debug* po najechaniu wskaźnikiem myszki na linijki kodu na początku linii pojawia się ikonka ( *Run execution to here*). Jej kliknięcie powoduje zatrzymanie działania programu przed rozpoczęciem wykonywania znajdujących się w niej instrukcji. Ten sposób sprawdza się, gdy chcemy coś jednorazowo sprawdzić, gdyż nie będziemy musieli wówczas kasować *Breakpointa*. Jeżeli jednak kilkukrotnie będziemy chcieli wykonać program zatrzymując go za każdym razem w tym samym miejscu, warto wstawić tam punkt zatrzymania pracy programu.

### Kontynuowanie działania programu

Wykonywanie programu można również kontynuować do momentu napotkania kolejnego punktu zatrzymania lub innego wydarzenia powodującego wstrzymanie programu (np. polecenia pobierającego dane od użytkownika). W tym celu można użyć skrótu F5, odpowiedniego przycisku z paska narzędzi (zielona strzałka *Continue*) lub opcji z menu głównego (*Debug* > *Continue*). Jeżeli znajdujemy się w funkcji *Main*, to możemy również skorzystać z opisanego wcześniej sposobu na wyjście z wnętrza metody („Wyjście z wnętrza klasy/metody”). Jeżeli w pozostałej części programu nie ma już żadnych przyczyn powodujących zatrzymanie pracy programu, to wówczas jego działanie zostaje zakończone.

## Sprawdzanie wartości zmiennych podczas debugowania

Podczas działania programu w konfiguracji *Debug*, możemy sprawdzać wartości zmiennych w danym momencie wykonywania programu. Ich zmieniające się wartości możemy podglądać w monitorze zmiennych. W zakładce *Locals* znajdziemy aktualne wartości wszystkich zmiennych, które istnieją w naszym programie. Istnieją, oznacza, że zostały zainicjowane, zadeklarowane i wciąż przetrzymują jakąś wartość. Jeżeli wyjdziemy z jakiejś metody/klasy to, o ile nie wskażemy inaczej, utworzone w niej zmienne przestają istnieć, więc znikają z zakładki *Locals*. Poza tym wartości zmiennych można podejrzeć najeżdżając wskaźnikiem myszki na nazwy zmiennych w kodzie programu w czasie debugowania. Jeżeli zmienna w danym momencie istnieje przy wskaźniku pojawi nam się okienko z nazwą zmiennej i jej aktualną wartością.