Paradoksalnie sytuacje wyjątkowe zdarzają się bardzo często. Z sytuacją wyjątkową mamy do czynienia wtedy, gdy program w określonych warunkach zachowa się nie tak, jak to przewidzieliśmy czy zaprogramowaliśmy. Sytuacja wyjątkowa będzie np. wtedy, gdy zechcemy otworzyć plik (za pomocą np. funkcji fopen()), a tymczasem okaże się, że plik nie istnieje.

W wersji 5. PHP pojawiła się obsługa wyjątków, znana z języków programowania wysokiego poziomu, takich jak Java, C++ czy C#. Jest jednak jeden problem: tylko niektóre elementy PHP wspierają strukturalną obsługę wyjątków, większość elementów języka działa tak, jak kiedyś (przykład z plikiem — funkcja fopen() nie generuje wyjątku, po prostu zgłaszany jest błąd).

W rozdziale przedstawiono strukturalną obsługę wyjątków oraz pokazano, jak tworzyć własne funkcje, które będą wspierały obsługę wyjątków w stylu PHP w wersji 5.

Instrukcja try/catch

Pisząc kod, często jesteśmy w stanie przewidzieć, który jego fragment jest najbardziej narażony na wystąpienie nieprzewidzianej sytuacji. Np. pisząc skrypt, który nawiązuje połączenie z serwerem baz danych, zdajemy sobie sprawę, że najwięcej problemów może przysporzyć funkcja otwierająca połączenie — serwer baz danych może być wyłączony, baza może być niedostępna, mogliśmy podać złą nazwę użytkownika itp. Dalej w skrypcie prawdopodobnie będą funkcje, które już operują na otwartym połączeniu, wykonują zapytania i odczytują wyniki. Każda z tych funkcji może spowodować sytuację wyjątkową — przecież następuje połączenie z serwerem przez sieć (nawet wtedy, gdy serwer działa na tej samej maszynie, nie możemy być pewni, że w danym momencie działa i może odpowiadać na polecenia wysyłane przez nasz skrypt). Można w takim przypadku, korzystając z instrukcji if, badać wynik każdej z tych funkcji i podejmować odpowiednie działania, gdy nie było niepowodzenia, tyle że spowoduje to powstanie kaskad kodu, dość trudnych do analizy (wystarczy zajrzeć do takiego kodu po dwóch tygodniach przerwy…).

Najbardziej odpowiednim rozwiązaniem problemu obsługi wyjątków jest zastosowanie instrukcji try/catch, która ma postać:

```
try {

  // kod, który prawdopodobnie spowoduje sytuację wyjątkową

} catch (KlasaWyjątku $obiekt) {

  // kod obsługi wyjątku

}
```

Pomiędzy słowami try i catch, wewnątrz nawiasów klamrowych umieszczamy kod, co do którego mamy obawy, że może wygenerować sytuację wyjątkową. Kod ten może się składać z większej liczby wierszy, z tym że im więcej wierszy w tym bloku, tym trudniej jest jednoznacznie zidentyfikować źródło błędu. W nawiasie okrągłym po słowie catch umieszcza się deklarację obiektu klasy ogólnej Exception lub klas pochodnych od Exception, specyficznych dla danego elementu (np. rozszerzenia) PHP. Ponieważ mało które rozszerzenie czy mało który element PHP obsługuje (na razie) wyjątki, w naszych przykładach będziemy się posługiwać klasą ogólną Exception. W nawiasach klamrowych po słowie catch umieszczany jest kod obsługi wyjątku — najczęściej jest to kod wyświetlający komunikat o błędzie.

Jak działa ta instrukcja? Otóż PHP próbuje (ang. try) wykonać kod umieszczony w nawiasach klamrowych po

słowie try. Jeżeli kod ten zostanie wykonany bez żadnego problemu czy błędu, blok catch zostanie pominięty. Jeżeli jednak w bloku try wystąpi błąd (sytuacja wyjątkowa), zostanie on przechwycony (ang. catch) jako obiekt klasy Exception i sterowanie przekazane zostanie do bloku catch. W bloku tym możemy wykorzystać metody udostępniane przez obiekt klasy Exception. Metody te umożliwiają uzyskanie informacji o błędzie (wyjątku), które to informacje można przedstawić w sposób elegancki użytkownikowi.

W instrukcji try/catch może występować więcej sekcji catch — wszystko zależy od tego, ilu i jakiej klasy wyjątków się spodziewamy. Taką sytuację schematycznie można przedstawić jako:

```
try{

  //instrukcje mogące spowodować wyjątek

}

catch(KlasaWyjątku1 $identyfikatorWyjątku1){

  //obsługa wyjątku 1

}

catch(KlasaWyjątku2 $identyfikatorWyjątku2){

  //obsługa wyjątku 2

}

/*

... dalsze bloki catch ...

*/

catch(KlasaWyjątkuN $identyfikatorWyjątkuN){

  //obsługa wyjątku N

}
```

Po wygenerowaniu wyjątku sprawdzane jest, czy jest on klasy KlasaWyjątku1 — jeśli tak, wykonywane są instrukcje jego obsługi, a blok try...catch jest opuszczany. Jeżeli jednak wyjątek nie jest klasy KlasaWyjątku1, sprawdzane jest, czy jest on klasy KlasaWyjątku2 itd.

Przykład zastosowania instrukcji obsługi wyjątków pokazany został w następnym punkcie. Mimo iż mechanizm obsługi wyjątków został wprowadzony w wersji 5. PHP, a obecna wersja to 5.3.0, niewiele rozszerzeń wspiera wyjątki — są to głównie nowe, napisane całkowicie obiektowo rozszerzenia, np. do obsługi wymiany danych z bazą MySQL czy do obsługi obiektowego modelu dokumentu XML. PHP umożliwia jednak pisanie funkcji (metod), które mogą w razie potrzeby generować wyjątki, co stanowić może furtkę dla programistów — będą oni mogli tworzyć klasy w PHP w pełni obsługujące mechanizm przechwytywania wyjątków. PHP umożliwia też tworzenie własnych klas wyjątków, ale to już całkiem inny temat.

Generowanie (rzucanie) wyjątków

Na szczęście PHP daje nam możliwość zgłoszenia wyjątku (inaczej wyrzucenia, od ang. throw — rzucać), czyli wygenerowania wyjątku w dowolnej sytuacji, którą — projektując aplikację — uznamy za wyjątkową. Zgłoszenie wyjątku to nic innego jak informacja dla aparatu wykonawczego PHP, że wystąpiła sytuacja nadzwyczajna, która

wymaga specjalnej obsługi. Z reguły jest to informacja o błędzie. Weźmy jako przykład klasyczny błąd — dzielenie przez zero — i prostą funkcję realizującą dzielenie dwóch liczb:

```
function divide($a, $b)

{

  return $a / $b; // wynik = a / b

}
```

Taka funkcja przyjmuje dwa parametry: $a, czyli dzielną, i $b, czyli dzielnik. Funkcja ta darzy program ją wywołujący absolutnym zaufaniem co do wartości zmiennej $b, ponieważ wewnątrz funkcji nie ma kontroli, czy zmienna $b jest różna od zera. Wprowadźmy zatem kontrolę wartości zmiennej $b. Można to zrobić na dwa sposoby (a może i więcej — wszystko zależy od pomysłowości programisty): albo w przypadku, gdy $b będzie równa zero, funkcja zwróci wartość false (byłoby to trochę niekonsekwentne — w końcu funkcja ta powinna zwracać liczbę, a nie wartość logiczną — wiele funkcji działa jednak w podobny sposób), albo funkcja po prostu wyświetli na ekranie komunikat o błędzie dzielenia przez zero (jest to rozwiązanie dyskusyjne — z jednej strony dobrym nawykiem jest, aby funkcje do tego nieprzeznaczone nie wypisywały niczego na wyjście, z drugiej strony, stosując to rozwiązanie, nie dawalibyśmy informacji programowi wywołującemu o rezultacie działania tej funkcji).

Spróbujemy, zostawiając dwa przedyskutowane rozwiązania, zastosować generowanie wyjątku przez funkcję w momencie, gdy wartość $b będzie równa zero:

```
function divide($a, $b)

{

  if ($b == 0)

    throw new Exception("Dzielenie przez zero!");

  return $a / $b;

}
```

Funkcja będzie działała w następujący sposób: najpierw badana jest wartość zmiennej $b — jeżeli zmienna ta nie zawiera zera, obliczony zostanie i zwrócony wynik dzielenia. Jeśli jednak $b zawiera zero, zostanie zgłoszony wyjątek — używamy w tym celu instrukcji throw, po której następuje utworzenie obiektu klasy Exception (lub jednej z klas pochodnych) z parametrem przekazywanym konstruktorowi tej klasy, zawierającym komunikat o błędzie. Mając tak skonstruowaną funkcję, możemy jej wywołanie umieścić w bloku try instrukcji try/catch:

```
$wynik = 0;

try {

  $wynik = divide(12, 0);

} catch (Exception $e) {

  print($e->getMessage());

}
```

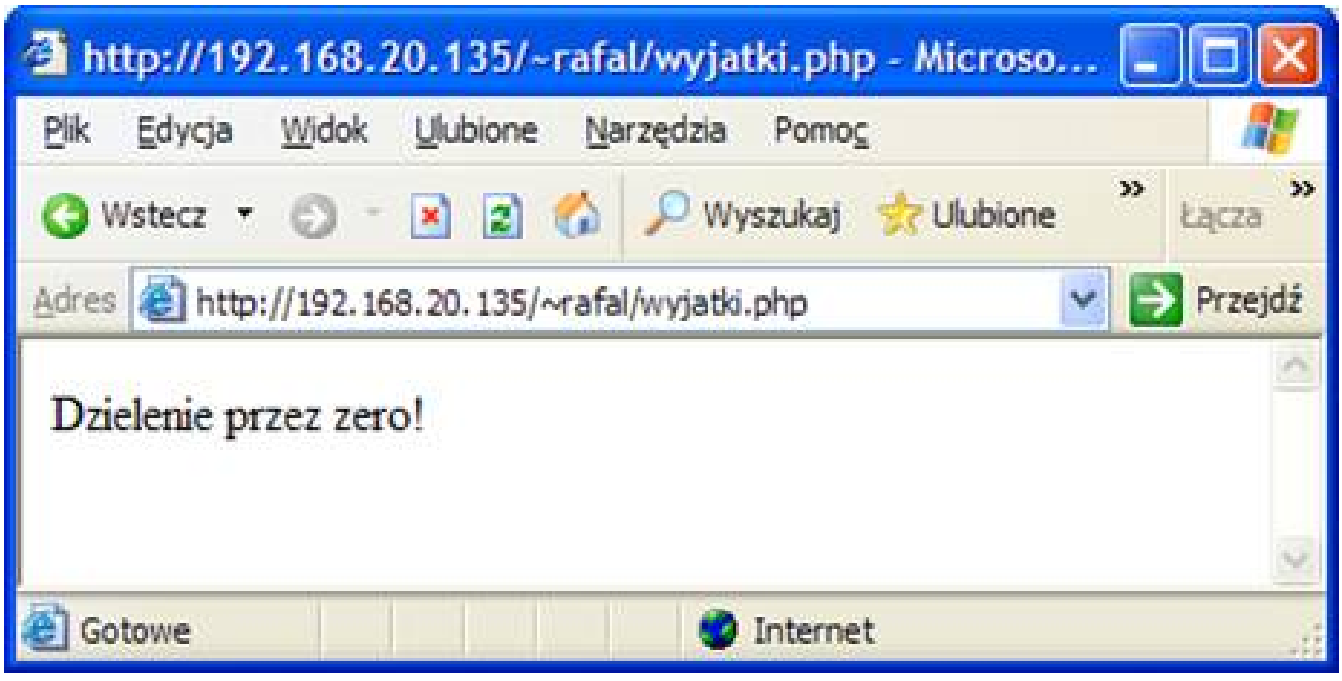
Kod ten zawiera tak dobrane dane, aby można było zaobserwować wyjątek. Listing 9.1 zawiera pełny kod skryptu

demonstrujący obsługę wyjątku dzielenia przez zero.

Listing 9.1. Obsługa wyjątku dzielenia przez zero

```
<?php
  function divide($a, $b) {
    if ($b == 0)
      throw new Exception("Dzielenie przez zero!");
    return $a / $b;
  }


  $wynik = 0;
  try {
    $wynik = divide(12, 0);
    print($wynik);
  } catch (Exception $e) {
    print($e->getMessage());
  }
?>
```

Efekt wykonania tego skryptu widoczny jest na rysunku 9.1.



Rysunek 9.1. Komunikat o błędzie dzielenia przez zero

Zauważmy, że blok catch zawiera instrukcję, która powoduje wyświetlenie w oknie przeglądarki wyniku działania metody getMessage() obiektu klasy Exception. Jako komunikat (wynik działania getMessage()) wyświetlony zostaje tekst, który przekazaliśmy konstruktorowi klasy Exception w momencie „wyrzucania" wyjątku. Klasa Exception zawiera kilka przydatnych metod, które mogą być wykorzystane do generowania szczegółowego komunikatu o błędzie. Niektóre z tych metod wymienione zostały w tabeli 9.1.

Tabela 9.1. Metody klasy Exception

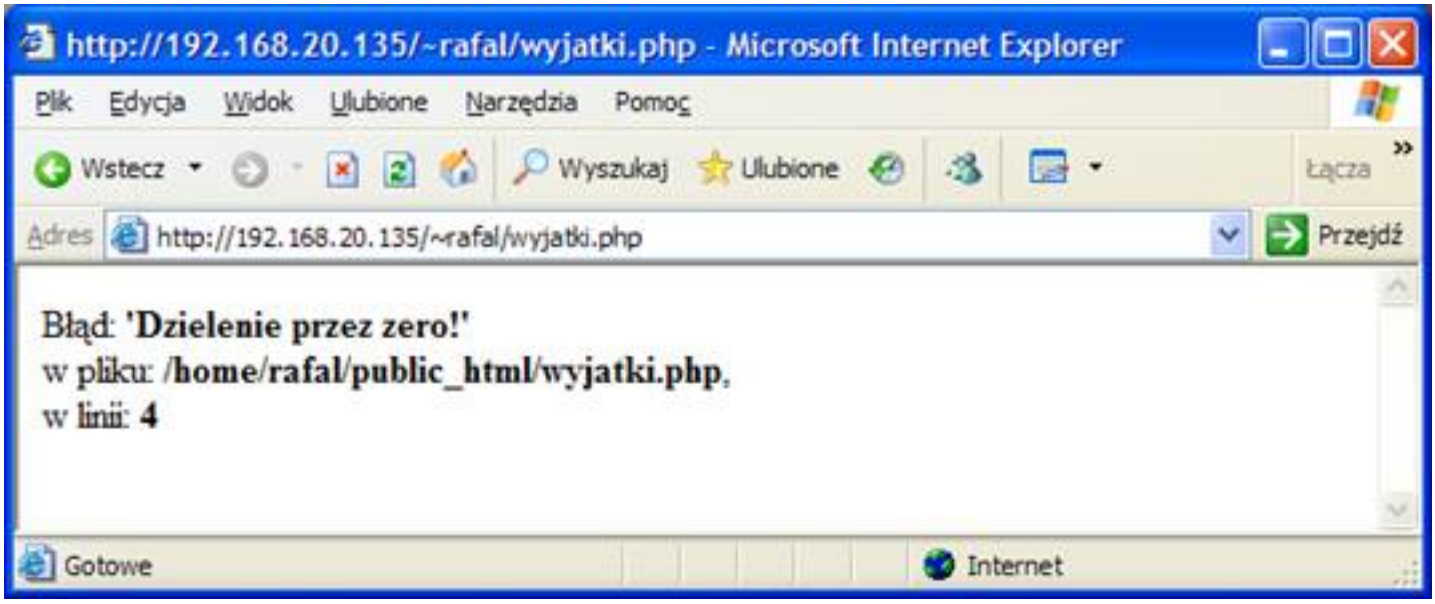
Metoda	Opis
getCode()	Zwraca kod błędu.
getFile()	Zwraca nazwę pliku, w którym wystąpił błąd.
getLine()	Zwraca numer linii pliku, w którym wystąpił błąd.
getMessage()	Zwraca komunikat o błędzie.
getTrace()	Zwraca tablicę zawierającą informacje debugowe (tzw. backtrace).
getTraceAsString()	Zwraca backtrace jako drukowalny ciąg znaków.

Listing 9.2 zawiera kod skryptu z rozbudowaną sekcją catch — komunikat, który otrzymuje użytkownik, jest bardziej uszczegółowiony. Efekt działania tego skryptu przedstawiony został na rysunku 9.2.

Listing 9.2. Obsługa wyjątku dzielenia przez zero (ze szczegółowym komunikatem o błędzie)

```
<?php
```

```
 function divide($a, $b)

 {

   if ($b == 0)

     throw new Exception("Dzielenie przez zero!");

   return $a / $b;

 }


 $wynik = 0;

 try {

   $wynik = divide(12, 0);

   print($wynik);

 } catch (Exception $e) {

   print("Błąd: <b>'".$e->getMessage()."'</b><br />

         w pliku: <b>".$e->getFile()."</b>,<br />

         w linii: <b>".$e->getLine()."</b>");

 }

?>
```



Rysunek 9.2. Rozbudowany komunikat o błędzie

Dobrym nawykiem programistów jest unikanie prezentowania użytkownikom końcowym bardzo szczegółowych komunikatów o błędach. Użytkownicy, których nie interesują szczegóły techniczne serwisu, pewnie i tak nie przeczytają takiego komunikatu, natomiast użytkownicy, których zainteresowania skierowane są na tworzenie stron WWW w PHP (ze szczególnym naciskiem na bezpieczeństwo serwisów i serwerów), bardzo wnikliwie przeanalizują każdy taki komunikat, szczególną radość objawiając na widok ścieżek dostępu do plików i nazw plików zawierających skrypty…

Własne wyjątki

Jeśli dostępne standardowo predefiniowane typy wyjątków są dla nas niewystarczające, nic nie stoi na przeszkodzie, abyśmy zdefiniowali własne. Utworzenie własnego wyjątku polega na wyprowadzeniu nowej klasy od klasy Exception lub dowolnej klasy od niej pochodnej. Możemy np. utworzyć wyjątek, który miałby być generowany w przypadku, kiedy argument jest mniejszy od zera. Zapewne nazwalibyśmy go: ArgumentMniejszyOdZeraException. Najprostsza definicja takiej klasy to:

```
class ArgumentMniejszyOdZeraException extends Exception

{

}
```

Jak widać, nie musi ona zawierać żadnego dodatkowego kodu. Jeśli teraz założymy, że w skrypcie ma się pojawić funkcja o nazwie dodatniArgument, która będzie generowała wyjątek ArgumentMniejszyOdZeraException w przypadku otrzymania argumentu o wartości mniejszej od zera, to wykorzystanie takiej klasy wyglądałoby tak, jak na listingu 9.3.

Listing 9.3. Użycie wyjątku ArgumentMniejszyOdZeraException

```
<?php

 class ArgumentMniejszyOdZeraException extends Exception

 {

 }



 function dodatniArgument($arg)

 {

  if($arg < 0){

   throw new ArgumentMniejszyOdZeraException("Argument mniejszy od zera: $arg.");

  }

 }



 try{
```

```
    dodatniArgument(-1);

  }

  catch(ArgumentMniejszyOdZeraException $e){

    echo "Wystąpił błąd: ", $e->getMessage();

  }
```

?> Podsumowanie

Stosując któreś z rozszerzeń PHP, zawsze należy sprawdzić, czy obsługuje ono wyjątki w stylu PHP 5 — jeśli tak jest, to kod, który napiszemy, może być o wiele bardziej przejrzysty i łatwy w pielęgnacji, po prostu nie pogubimy się w nim. Pisząc własne klasy czy funkcje, pamiętajmy też, że w PHP 5 jest strukturalna obsługa wyjątków.

Przechwytywanie wyjątków jest opisane w dokumentacji PHP na stronie http://www.php.net/manual/pl/language.exceptions.php.