Exceptions

План занятия

- Исключения
- Шаблоны проектирования

Исключения

Что такое исключения?

Исключения являются специальными условиями, обычно в случае ошибки, которые проявляются или могут быть специально созданы программой. Они указывают на то, что что-то в процессе отличается от предполагаемого хода событий. В большинстве случаев подобные ситуации требуют выполнения специальных инструкций, чтобы предотвратить неконтролируемое завершение процесса.

Как работают исключения?

Исключения могут быть брошены (thrown) и пойманы (caught). Когда исключение брошено, то значит, произошло что-то выходящее за рамки нормального течения процесса работы программы, и требуется выполнение какой-то другой функции. Подхват исключения осуществляется в специальной функции, которая сообщает остальной программе о том, что она готова обработать исключение.

Исключения

Давайте сразу взглянем на пример сгенерированного исключения (и впоследствии перехваченного):

```
<?php

//Пробуем (try) что-либо сделать.
try{
    //Очевидно, 1 никогда не будет равняться 2...
    if(1 !== 2){
        //Генерируем исключение.
        throw new Exception('1 не равняется 2!');
    }
}
//Перехватываем (catch) исключение, если что-то идет не так.
catch (Exception $ex) {
    //Выводим сообщение об исключении.
    echo $ex->getMessage();
}
```

Исключения

В приведенном выше примере было продемонстрировано использование TRY и CATCH, в котором исключение всегда сгенерировано (только ради примера):

1. Внутри блока TRY мы проверяем, равняется ли цифра 1 цифре 2. Так как она не равняется (и никогда не будет равняться), мы генерируем исключение с сообщением “1 не равняется 2!”;
2. Внутри блока CATCH мы перехватываем исключение и выводим соответствующее сообщение.

Исключения

Выводы:

- TRY: внутри блока PHP try мы задаем логику приложения. Этот блок содержит код, который может или не может сгенерировать исключение;
- CATCH: блок CATCH будет перехватывать любые исключения, проявившиеся в предыдущем блоке TRY. Код внутри блока CATCH будет исполнен только в случае обнаружения исключения;
- FINALLY: если вы используете PHP 5.5 и выше, то вы можете использовать блок FINALLY. Расположенный в нем код исполняется всегда, вне зависимости от того, было ли обнаружено исключение.

Исключения

Исключения реализуются также как и любой другой объект:

```
$exception = new Exception();
throw $exception;
```

Так же как и любой объект, они имеют методы, которые можно вызвать. Данные методы облегчают формирование реакции на исключение. Например, функция getMessage() позволяет получить сообщение об ошибке, которое может быть записано в журнал или выведено в браузер.

Вот список методов исключений:

- getMessage() – получает сообщение исключения;
- getCode() – возвращает числовой код, который представляет исключение;
- getFile() – возвращает файл, в котором произошло исключение;
- getLine() – возвращает номер строки в файле, где произошло исключение;
- getTrace() – возвращает массив *backtrace()* до возникновения исключения;
- getPrevious() – возвращает исключение, произошедшее перед текущим, если оно было;
- getTraceAsString() – возвращает массив *backtrace()* исключения в виде строки;
- __toString() –возвращает все исключение в виде строки. Данную функцию можно переписать.

Исключения

Полная иерархия исключений в PHP7 имеет такой вид:

```
\Throwable
├── \Exception (implements \Throwable)
│   ├── \LogicException (extends \Exception)
│   │   ├── \BadFunctionCallException (extends \LogicException)
│   │   │   └── \BadMethodCallException (extends \BadFunctionCallException)
│   │   ├── \DomainException (extends \LogicException)
│   │   ├── \InvalidArgumentException (extends \LogicException)
│   │   ├── \LengthException (extends \LogicException)
│   │   └── \OutOfRangeException (extends \LogicException)
│   └── \RuntimeException (extends \Exception)
│       ├── \OutOfBoundsException (extends \RuntimeException)
│       ├── \OverflowException (extends \RuntimeException)
│       ├── \RangeException (extends \RuntimeException)
│       ├── \UnderflowException (extends \RuntimeException)
│       └── \UnexpectedValueException (extends \RuntimeException)
└── \Error (implements \Throwable)
    ├── \AssertionError (extends \Error)
    ├── \ParseError (extends \Error)
    └── \TypeError (extends \Error)
```

Исключения

Здесь PHP7 добавил базовый тип Throwable, тип Error и его подтипы AssertionError, ParseError и TypeError. Остальные типы доступны с версии PHP 5.1.0 или еще раньше.

Итак, рассмотрим их по порядку:

Throwable

Throwable — это даже не исключение, а интерфейс, который реализуют все остальные рассматриваемые классы. Добавлен в PHP7

Исключения

Exception

Базовый класс для исключений. Есть две группы исключений, два надкласса: для исключений в логике: **LogicException** и исключений времени исполнении **RuntimeException**.

LogicException

Используется, когда ваш код возвращает значение, которое не должен возвращать. Часто вызывается при разных багах в коде. Потомки этого класса используются в более специализированных ситуациях. Если ни одна из них не подходит под ваш случай, можно использовать LogicException.

Исключения

BadFunctionCallException

Используется, когда вызываемой функции физически не существует или когда в вызове используется неверное число аргументов. Редко бывает нужно.

BadMethodCallException

Подкласс **BadFunctionCallException**. Аналогично ему используется для методов, которые не существуют или которым передано неверное число параметров. Всегда используйте внутри __call(), в основном для этого оно и применяется.

Исключения

DomainException

Если в коде подразумеваются некие ограничения для значений, то это исключение можно вызывать, когда значение выходит за эти ограничения. Например, у вас дни недели обозначаются числами от 1 до 7, а ваш метод получает внезапно на вход 0 или 9, или, скажем, вы ожидаете число, обозначающее количество зрителей в зале, а получаете отрицательное значение. Вот в таких случаях и вызывается **DomainException**. Также можно использовать для разных проверок параметров, когда параметры нужных типов, но при этом не проходят проверку на значение. Например,

```
if ($a>5){

    throw new DomainException ("a должно быть меньше 5");

}
```

Исключения

InvalidArgumentException

Вызываем, когда ожидаемые аргументы в функции/методе некорректно сформированы. Например, ожидается целое число, а на входе строка или ожидается GET, а пришел POST и т.п.

Пример:

```
public function foo($number) {
    if(!is_numeric($number)) {
        throw new InvalidArgumentException('На входе ожидалось число!');
    }
}
```

Исключения

LengthException

Вызываем, если длина чего-то слишком велика или мала. Например, имя файла слишком короткое или длина массива слишком большая.

RuntimeException

Исключения времени выполнения нужно вызывать, когда код самостоятельно не может справиться с некой ситуацией во время своего выполнения. Подклассы этого класса сужают область применения, но, если ни один из них не подходит для вашей ситуации, смело пользуйтесь этим классом

Исключения

OutOfBoundsException

Вызываем, когда обнаружили попытку использования неправильного ключа, например, в ассоциативном массиве или при реализации ArrayAccess. Используется тогда, когда ошибка не может быть обнаружена до прогона кода. То есть, например, когда то, какие именно ключи будут легитимными, определяется динамически уже во время выполнения.

Вот пример использования в реализации ArrayAccess:

```
public function offsetGet($offset) {
    if(!isset($this->objects[$offset])) {
        throw new OutOfBoundsException("Смещение '$offset' вышло из заданного диапазона");
    }

    return $this->objects[$offset];
}
```

Исключения

OutOfRangeException

Используется, когда встречаем некорректный индекс, но на этот раз ошибка должна быть обнаружена ещё до прогона кода, например, если мы пытаемся адресовать элемент массива, который в принципе не поддерживается. То есть если функция, возвращающая день недели по его индексу от 1 до 7, получает внезапно 9, то это **DomainException** — ошибка логики, а если у нас есть массив с днями недели с индексами от 1 до 7, а мы пытаемся обратиться к элементу с индексом 9, то это уже **OutOfRangeException**.

OverflowException

Исключение вызываем, когда есть переполнение. Например, имеется некий класс-контейнер, который может принимать только 5 элементов, а мы туда пытаемся записать шестой.

Исключения

UnderflowException

Обратная **OverflowException** ситуация, когда, например, класс-контейнер имеет недостаточно элементов для осуществления операции. Например, когда он пуст, а вы пытаетесь удалить элемент.

RangeException

Вызывается, когда значение выходит за границы некоего диапазона. Похоже на **DomainException**, но используется при возврате из функции, а не при входе. Если мы не можем вернуть легитимное значение, мы выбрасываем это исключение. То есть, к примеру, функция у вас принимает целочисленный индекс и использует другую функцию, чтоб получить некое значение по этой сущности. Та функция вернула null, но ваша функция не имеет права возвращать Null. В таком случае можно применить это исключение. То есть между ними примерно такая же разница, как между OutOfBoundsException и OutOfRangeException.

Исключения

UnexpectedValueException

Используется, когда значение выходит из ряда ожидаемых значений. Часто применяется, когда то, что вернулось из вызываемой функции, не соответствует тому, что мы от нее ожидаем в ответе по типу или значению. Сюда не относятся арифметические ошибки или ошибки, связанные с буфером. Важно, что, в отличие от InvalidArgumentException, здесь мы имеем дело, в основном, с возвращаемыми значениями. Часто мы заранее не можем быть уверены в том, что придет в ответе от функции (особенно сторонней). Скажем, мы используем некую стороннюю функцию, использующую API ВКонтакте, и возвращающую количество постов для пользователя. Она всегда возвращала целое неотрицательное число, и вдруг неожиданно возвращает отрицательное число. Это не соответствует документации. Соответственно, чтобы подстраховаться от таких ситуаций, мы можем проверять результат такого вызова и, если он отличается от ожидаемого, выбрасывать UnexpectedValueException.

Исключения

Error

Добавлено в PHP7 для обработки фатальных ошибок. То есть многие из ошибок, которые раньше приводили к Fatal Error, в PHP7 могут обрабатываться в блоках try/catch. Эти ошибки вызываются самим PHP, нет нужды их вызывать, как Exception. Класс **Error** имеет три подкласса:

AssertionError

Вызывается, когда условие, заданноле методом assert(), не выполняется.

ParseError

Для ошибок парсинга, когда подключаемый по include/require код вызывает ошибку синтаксиса, ошибок функции eval() и т.п.

Исключения

Пример:

```
try {
    require 'file-with-syntax-error.php';
} catch (ParseError $e) {
    // обработка ошибки
}
```

TypeError

Используется для ошибок несоответствия типов данных. В PHP7 введена опциональная строгая типизация. Вот для поддержки ошибок, связанных с ней, и служит этот класс. Например, если функция ожидает на входе аргумент типа int, а вы ее вызываете со строковым аргументом.

Исключения

Пример:

```
try {
    require 'file-with-syntax-error.php';
} catch (ParseError $e) {
    // обработка ошибки
}
```

TypeError

Используется для ошибок несоответствия типов данных. В PHP7 введена опциональная строгая типизация. Вот для поддержки ошибок, связанных с ней, и служит этот класс. Например, если функция ожидает на входе аргумент типа int, а вы ее вызываете со строковым аргументом.

Домашнее задание

1. Создайте обработку исключения открытия несуществующего файла. При попытке открыть несуществующий файл должно появиться сообщение: Файл с именем Name не существует, где Name- имя файла.
2. Создайте класс User с полями id (должно содержать только число) и password (длина поля должна быть не более 8 символов) и методом getUserData, который возвращает id и email. Обработайте исключения, когда id содержит не число и password содержит более 8 символов, при этом выведите сообщение исключения, файл в котором данное исключение возникло и номер строки.