Лекция 3.

Структура и оформление консольного PHP-приложения

Создание приложения

Нужно написать консольную игру на PHP, данные будут храниться в БД SQLite.
В дальнейшем приложение нужно перевести на другую среду выполнения (веб-браузер) и ЯП (JavaScript).

1. **Настройка рабочей среды**

2. **Программирование задачи**

Создание приложения

Нужно написать консольную игру на PHP, данные будут храниться в БД SQLite.
В дальнейшем приложение нужно перевести на другую среду выполнения (веб-браузер) и ЯП (JavaScript).

1. **Настройка рабочей среды**
   - Настройка Git и GitHub, создание репозитория
   - Установка СУБД SQLite
   - Установка PHP CLI
   - Установка менеджера зависимостей Composer
   - Публикация игры на Packagist
   - Выбор IDE или редактора
   - Настройка линтера кода (*PHP Code Sniffer*)

2. **Программирование задачи**

Создание приложения

Нужно написать консольную игру на PHP, данные будут храниться в БД SQLite.
В дальнейшем приложение нужно перевести на другую среду выполнения (веб-браузер) и ЯП (JavaScript).

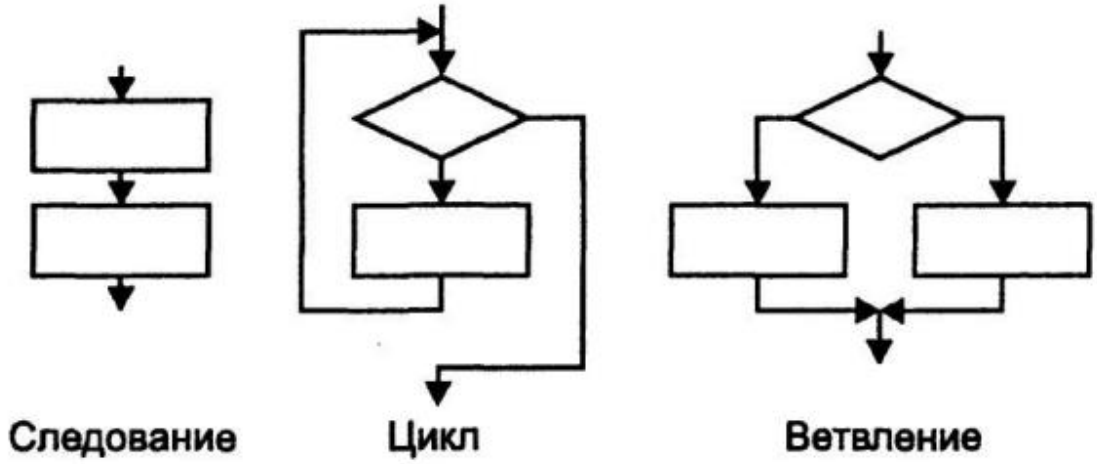
1. **Настройка рабочей среды**

2. **Программирование задачи**
   - Разработка алгоритма
   - *Выбор технологии и языка программирования*
   - Проектирование архитектуры приложения
   - Кодирование и оформление кода
   - Тестирование

Выбор технологии программирования

Будем писать приложение в императивном процедурном стиле, придерживаясь структурного подхода.

- Проектирование сверху вниз
- Структурное кодирование
- Модульное программирование



Базовые конструкции структурного программирования

Проектирование архитектуры приложения

- Разделение логики и данных от визуального представления
- Выделение модулей и подпрограмм
- Разделение кода на модули, содержащие описания функций, и скрипты с исполняемым кодом
- Создание файловой структуры приложения

Проектирование архитектуры приложения

- Отделение логики и данных от визуального представления
- Выделение модулей и подпрограмм
- Разделение кода на модули, содержащие описания функций, и скрипты с исполняемым кодом
- Создание файловой структуры приложения

Специфика PHP:

- Настройка автозагрузки модулей с помощью Composer
- Соблюдение стандартов кодирования PSR-1, PSR-12 (PHP Code Sniffer, PHP Coding Standard Fixer)

Подпрограммы

Почему нужно писать подпрограммы и когда это нужно делать?

Подпрограммы

Почему нужно писать подпрограммы и когда это нужно делать?

- **Снижение сложности за счет формирования понятной промежуточной абстракции.** Выделение фрагмента кода в удачно названный метод — один из лучших способов документирования его цели.

Подпрограммы

Почему нужно писать подпрограммы и когда это нужно делать?

- **Снижение сложности за счет формирования понятной промежуточной абстракции.** Выделение фрагмента кода в удачно названный метод — один из лучших способов документирования его цели.

```
Assign(f, "my.ini");
Open(f);
While (not eof(f)) do
  Begin
    Readln(f, s);
    If substr(s, 1,5) = "user" then UserName:= substr(s,
6,len(s)-5);
  End;

UserName := GetUserNameFromIniFile;
```

Подпрограммы

Почему нужно писать подпрограммы и когда это нужно делать?

- Снижение сложности за счет формирования понятной промежуточной абстракции.
- **Предотвращение дублирования кода.** Проще изменять код, выше надежность (меньше вероятность ошибиться).

Подпрограммы

Почему нужно писать подпрограммы и когда это нужно делать?

- Снижение сложности за счет формирования понятной промежуточной абстракции.
- Предотвращение дублирования кода.
- **Упрощение переноса приложения на другие платформы.** Выделение и изоляция непортируемого кода (нестандартные возможности языка, зависимости от оборудования и операционной системы и т. д.).

Подпрограммы

Почему нужно писать подпрограммы и когда это нужно делать?

- Снижение сложности за счет формирования понятной промежуточной абстракции.
- Предотвращение дублирования кода.
- Упрощение переноса приложения на другие платформы.
- **Упрощение сложных логических проверок.** Сокрытие деталей проверок + описательное имя метода позволяет лучше охарактеризовать суть проверки.

Кодирование и оформление кода

Программирование - раздел лингвистики.
ЯП - это искусственный язык, на котором мы общаемся с машиной (10%) и другими программистами (90%).

Код должен быть самодокументируемым и понятным

Основная задача - **создание кода, читая который можно воспроизвести задачу в исходных терминах**

Кодирование и оформление кода

- Осмысленные и произносимые имена сущностей
- Только английские слова
- Не должно быть магических чисел (нужны константы)

Именование функций

Функция - это действие, название должно быть глаголом

Действие-Объект
Глагол-Существительное

`getUser`, `printPage`

Командлет Get-Verb в PowerShell

Именование функций

- **Описывайте все, что метод выполняет.**

```
Date newDate = date.add(5);
```

Именование функций

- **Описывайте все, что метод выполняет.**

```
Date newDate = date.add(5);
```

1. Что прибавляется к дате (дни, часы, недели)?

2. Изменяется ли экземпляр date, или функция возвращает новое значение Date без изменения старого?

Именование функций

- **Описывайте все, что метод выполняет.**

```
Date newDate = date.add(5);
```

Если функция прибавляет пять дней с изменением date, то она должна называться `addDaysTo` или `increaseByDays`.

Если функция возвращает новую дату, смещенную на пять дней, но не изменяет исходного экземпляра `date`, то она должна называться `daysLater` или `daysSince`.

Именование функций

- Описывайте все, что метод выполняет.
- **Избегайте невыразительных глаголов**.

```
HandleCalculation(),PerformServices(),OutputUser(), ProcessInput()
```

HandleOutput() – непонятно. Лучше FormatAndPrintOutput().

Именование функций

- Описывайте все, что метод выполняет.
- Избегайте невыразительных глаголов.
- **Не используйте для различения имен методов исключительно номера.**

Именование функций

- Описывайте все, что метод выполняет.
- Избегайте невыразительных глаголов.
- Не используйте для различения имен методов исключительно номера.
- **Для именования функции используйте описание возвращаемого значения**.

```
customerId.Next(), printer.IsReady(), pen.CurrentColor()
```

Именование функций

- **Аккуратно используйте антонимы.**

```
add / remove
increment / decrement
open / close
begin / end
insert / delete
show / hide
create / destroy
lock / unlock
source / target
first / last
min / max
start / stop
get / put
next / previous
up / down
get / set
old / new
```

Именование функций

- **Аккуратно используйте антонимы.**
- **Определяйте правила именования часто используемых операций и придерживайтесь их.**

Метод, возвращающий уникальный идентификатор объекта:

```
employee.id.Get()
dependent.GetId()
candidate.id()
```

Именование функций

- Для именования процедуры используйте выразительный глагол, дополняя его объектом.
- Аккуратно используйте антонимы.
- Определяйте правила именования часто используемых операций и придерживайтесь их.
- **Разделяйте команды и запросы.**

Функция должна что-то делать или отвечать на какой-то вопрос, но не одновременно. Либо функция изменяет состояние объекта, либо возвращает информацию об этом объекте.

```
function Set(attribute: string, value: string): boolean;
…
if Set("username", "popov-av") …   // ???
```

Именование функций

- Для именования процедуры используйте выразительный глагол, дополняя его объектом.
- Аккуратно используйте антонимы.
- Определяйте правила именования часто используемых операций и придерживайтесь их.
- **Разделяйте команды и запросы.**

Функция должна что-то делать или отвечать на какой-то вопрос, но не одновременно. Либо функция изменяет состояние объекта, либо возвращает информацию об этом объекте.

```
function Set(attribute: string, value: string): boolean;
…
if Set("username", "popov-av") …   // ???
```

Лучше:

```
if attributeExists("username") then setAttribute("username",
"popov-av");
```

Аргументы функций

Количество аргументов нужно минимизировать.

Идеальный (самый выразительный) случай – аргументов нет. Проще понимать и тестировать.

Следует избегать использования аргументов-флагов.

Возврат из подпрограмм

Упрощайте сложную обработку ошибок с помощью досрочных return

Неудачный код (Visual Basic):

```
If file.validName() Then
    If file.Open() Then
        If encryptionKey.valid() Then
            If file.Decrypt( encryptionKey ) Then
                ' Много кода.
                ...
            End If
        End If
    End If
End If
```

Возврат из подпрограмм

Упрощайте сложную обработку ошибок с помощью досрочных return

Так лучше:

```
If Not file.validName() Then Exit Sub
If Not file.Open() Then Exit Sub
If Not encryptionKey.valid() Then Exit Sub
If Not file.Decrypt( encryptionKey ) Then Exit Sub
’ Много кода.
...
```

Возврат из подпрограмм

Упрощайте сложную обработку ошибок с помощью досрочных return

Реалистичный код:

```
If Not file.validName() Then
    errorStatus = FileError_InvalidFileName
    Exit Sub
End If

If Not file.Open() Then
    errorStatus = FileError_CantOpenFile
    Exit Sub
End If

If Not encryptionKey.valid() Then
    errorStatus = FileError_InvalidEncryptionKey
    Exit Sub
End If

If Not file.Decrypt( encryptionKey ) Then
    errorStatus = FileError_CantDecryptFile
    Exit Sub
End If

' Много кода.
...
```

Схемы именования переменных

- Визуальное выделение составных частей имени
- Понимание сути переменной, ее назначение и тип (именованная константа; элементарная переменная; тип, определенный пользователем, или класс).

Схемы именования переменных

CamelCase (CamelCase, PascalCase). Применяется для имен классов Java, в методах Windows API и .NET.

camelCase. Применяется для членов классов в Java, для переменных в JavaScript.

snake_case (змеиная нотация). Принят в PHP.

kebab-case (шашлычная нотация)

UPPERCASE_WITH_UNDERSCORES. Для констант.

_name. Переменная объявлена как private или protected и недоступна вне класса.

Имена переменных

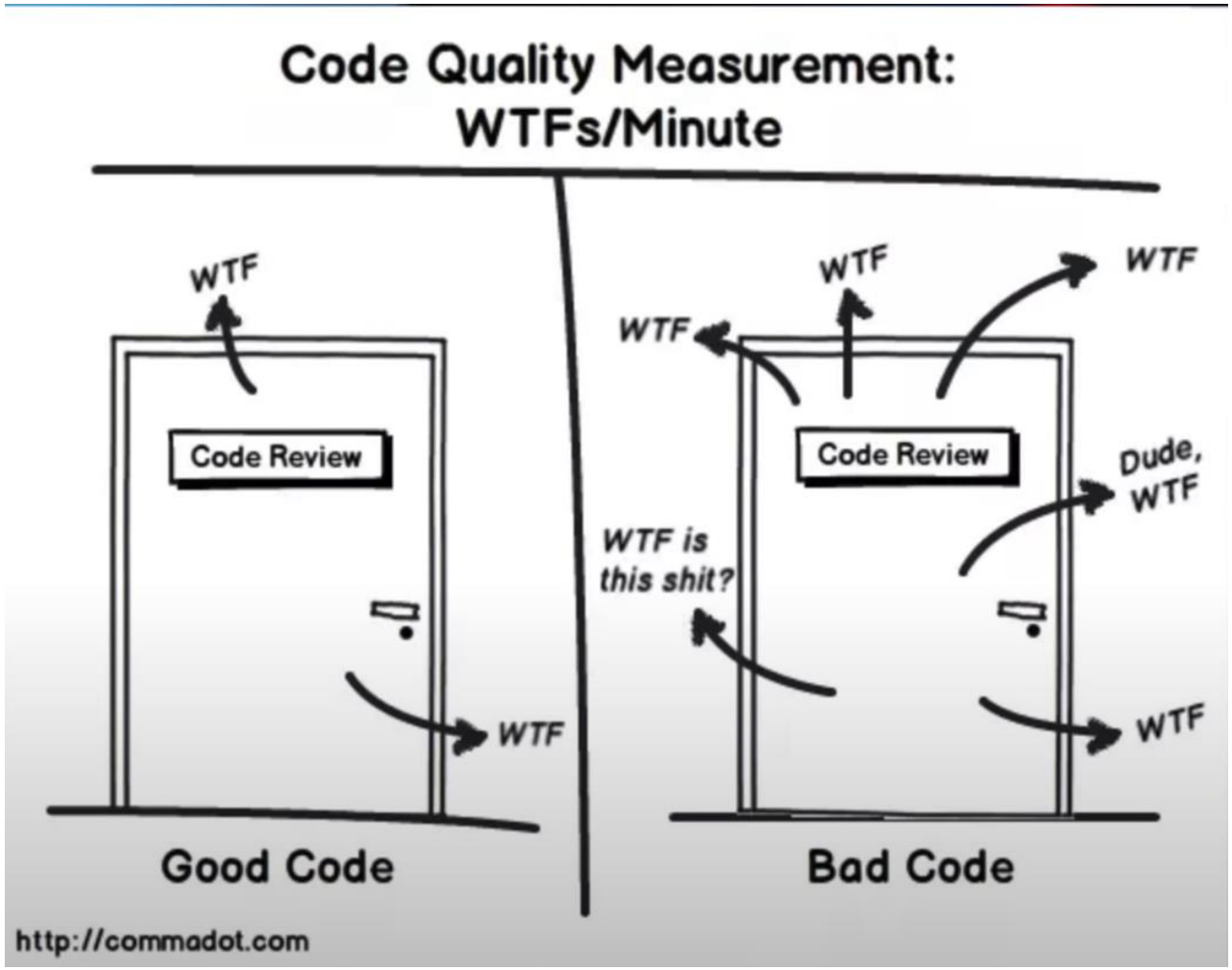
- **Имена должны быть максимально конкретны.** Для важных переменных следует избегать имен x, j, temp, … . Короткие имена допустимы в качестве счетчиков в циклах и т.п.

- **Следует отказаться от сокращения имен.** Нужно выбирать имена так, чтобы они облегчали чтение кода, даже за счет удобства его написания.

- Пример: *В переменной хранится текущая дата. Как назвать переменную?*

  `CD`, `CurDate`, `CurrentDate`

Принцип наименьшего удивления



Имена переменных

Хорошее имя чаще всего описывает проблему, а не ее решение (отвечает на вопрос ***что?*** а не ***как?***)

- Запись данных о сотруднике: `inputRecord` или `employeeData`
- Статус принтера: `bitFlag` или **printerReady**

Имена переменных

Следует избегать имен:

- Общие термины
  `data`, `value`, `temp`
- Синонимы
  `number` и `nomer`
- Имеют разную суть, но похожие имена
  `clientRecs`, `clientReps`
- Содержат цифры (лучше массивы)
- Отличаются регистром символов
- Совпадают со стандартными типами, переменными и методами
  `integer`, `string`
- Не связаны со смыслом переменных.

Никогда не использовать символы `I`, `l`, `O` как однобуквенные идентификаторы!

Имена переменных

Хорошие программисты...

- Понимают важность выбора имен и занимаются этой проблемой
- Не забывают о выборе правильного имени для каждого создаваемого объекта
- Учитывают многие факторы: длину имени, понятность, контекст и т. д.
- Видят общую картину и выбирают имена в рамках проекта (или проектов)

Плохие программисты...

- Не заботятся о понятности своего кода
- Пишут *одноразовый* код, плохо продумывая его в погоне за скоростью
- Игнорируют естественную идиоматику языка
- Не стремятся к единообразию в именах
- Не заботятся об общей картине и не интересуются согласованностью своего кода с проектом в целом

Имена переменных

Хорошо ли выбраны имена?

a. `int apple_count`
b. `char foo`
c. `bool apple_count`
d. `char *string`
e. `int loop_counter`

Имена переменных

Что делает этот код?

```
void bsrt(int a[], int n)
{
   for (int i = 0; i < n1; i++)
      for (int j = n1; j > i; j)
         if (a[j1] > a[j])
         {
            int tmp = a[j1];
            a[j1] = a[j];
            a[j] = tmp;
         }
}
```

Имена переменных

Что делает этот код?

```
void bsrt(int a[], int n)
{
   for (int i = 0; i < n1; i++)
      for (int j = n1; j > i; j)
         if (a[j1] > a[j])
         {
            int tmp = a[j1];
            a[j1] = a[j];
            a[j] = tmp;
         }
}
```

```
void swap(int *first, int *second)
{
   int temp = *first;
   *first = *second;
   *second = temp;
}

void bubbleSort(int items[], int size)
{
   for (int pos1 = 0; pos1 < size1; pos1++)
      for (int pos2 = size1; pos2 > pos1; pos2)
         if (items[pos21] > items[pos2])
            swap(&items[pos21],
&items[pos2]);
}
```

Базовые структуры управления

- **Последовательность** – операторы, перечисленные в определенном порядке. Ее выполнение состоит в выполнении каждого из этих операторов в том же порядке.
- **Цикл**, который содержит последовательность операторов, выполняемую многократно.
- **Выбор**, состоящий из условия и двух последовательностей операторов.

Дополнительные структуры управления

- **Оператор перехода (goto).** Любая программа, использующая его, имеет эквивалент, выраженный в терминах стандартных структур управления.

- **Обработка исключений**, обеспечивающая способы восстановления после возникновения событий, прерывающих нормальный поток управления (переполнение, void-вызовы и т.п.) . Try … Except …

Последовательность

- Главный принцип организации последовательного кода — упорядочение зависимостей.
- Зависимости должны быть сделаны явными с помощью хороших имен методов, списков параметров, комментариев и вспомогательных переменных.
- Если порядковые зависимости в коде отсутствуют, старайтесь размещать взаимосвязанные выражения как можно ближе друг к другу.

Программа должна быть написана так, чтобы ее можно было читать сверху вниз, а не перескакивая с места на место.

Последовательность

Неудачный код (C++)

```
MarketingData marketingData;
SalesData salesData;
TravelData travelData;

travelData.ComputeQuarterly();
salesData.ComputeQuarterly();
marketingData.ComputeQuarterly();

salesData.ComputeAnnual();
marketingData.ComputeAnnual();
travelData.ComputeAnnual();

salesData.Print();
travelData.Print();
marketingData.Print();
```

Как рассчитывается marketingData?

Последовательность

Хороший последовательный код (C++)

```
MarketingData marketingData;
marketingData.ComputeQuarterly();
marketingData.ComputeAnnual();
marketingData.Print();

SalesData salesData;
salesData.ComputeQuarterly();
salesData.ComputeAnnual();
salesData.Print();

TravelData travelData;
travelData.ComputeQuarterly();
travelData.ComputeAnnual();
travelData.Print();
```

Упоминания каждой переменной располагаются вместе — они «локализованы».

Код можно разбить на отдельные методы для данных по маркетингу, продажам и поездкам.

Выбор

- Для последовательностей `if … then …else` и операторов `case` выбирайте порядок, позволяющий улучшить читабельность

- Избегайте вложенных проверок и сложных выражений в условиях (лучше определить для них функции)

Циклы

- Циклы сложны для понимания, поэтому они должны быть максимально простыми. Следует избегать экзотических видов циклов, минимизировать вложенность, делать очевидными входы и выходы цикла.

- Называйте индексы цикла понятно и используйте только с одной целью.