REGULI DE BUNĂ PRACTICĂ ÎN PROGRAMARE

Jack of all trades, master of none

- Java
- Javascript
- Python
- Ruby on Rails
- Go
- C++
- C
- Ansible

1. Dependency Management

- Almost any project needs libraries
- Dependency Hell
- Almost every programming language has a build system

  Java – Maven or Gradle

  Ruby – Bundler

  PHP – Composer

PHP without Composer

```php
<?php

require 'Psr/Log/LoggerInterface.php';
require 'Monolog/Handler/HandlerInterface.php';
require 'Monolog/Handler/Handler.php';
require 'Monolog/Handler/ProcessableHandlerTrait.php';
require 'Monolog/Handler/ProcessableHandlerInterface.php';
require 'Monolog/Handler/FormattableHandlerTrait.php';
require 'Monolog/Handler/FormattableHandlerInterface.php';
require 'Monolog/ResettableInterface.php';
require 'Monolog/DateTimeImmutable.php';
require 'Monolog/Formatter/FormatterInterface.php';
require 'Monolog/Formatter/NormalizerFormatter.php';
require 'Monolog/Formatter/LineFormatter.php';
require 'Monolog/Handler/AbstractHandler.php';
require 'Monolog/Handler/AbstractProcessingHandler.php';
require 'Monolog/Logger.php';
require 'Monolog/Handler/StreamHandler.php';

use Monolog\Logger;
use Monolog\Handler\StreamHandler;



$log = new Logger('name');
$log->pushHandler(new StreamHandler('path/to/your.log', Logger::WARNING));

$log->warning('Foo');
$log->error('Bar');
```

PHP with Composer

```php
<?php

require 'vendor/autoload.php';

use Monolog\Logger;
use Monolog\Handler\StreamHandler;

$log = new Logger('name');
$log->pushHandler(new StreamHandler('path/to/your.log', Logger::WARNING));

$log->warning('Foo');
$log->error('Bar');
```

2. Model View Controller in web-applications

- Use a web framework:
  - Ruby - Ruby On Rails
  - PHP - Laravel or Symfony
  - Python - Django or Flask

Without Model View Controller

http://localhost:8000/show_student.php?id=1

```php
<?php

$servername = '127.0.0.1';
$username = 'root';
$password = 'password';
$database = 'presentation';

$id = $_GET['id'];

$conn = new mysqli($servername, $username, $password, $database);

$stmt = $conn->prepare("SELECT id, first_name, last_name FROM students WHERE id = ?");
$stmt->bind_param('i', $id);
$stmt->execute();
$stmt->bind_result($id, $first_name, $last_name);
$stmt->fetch();
$stmt->close();
?>

<h3>Student: <?php echo "$first_name $last_name" ?></h3>
```

Model View Controller

```php
# app/Student.php

<?php

namespace App;
use Illuminate\Database\Eloquent\Model;

class Student extends Model
{

    function fullName()
    {
        return "$this->first_name  $this->last_name";
    }

}
```

```php
# app/Http/Controllers/StudentController.php
<?php

namespace App\Http\Controllers;

use App\Student;

class StudentController
{
    public function show($id)
    {
        $student = Student::findOrFail($id);
        return view('students.profile', ['student' => $student ]);
    }

}
```

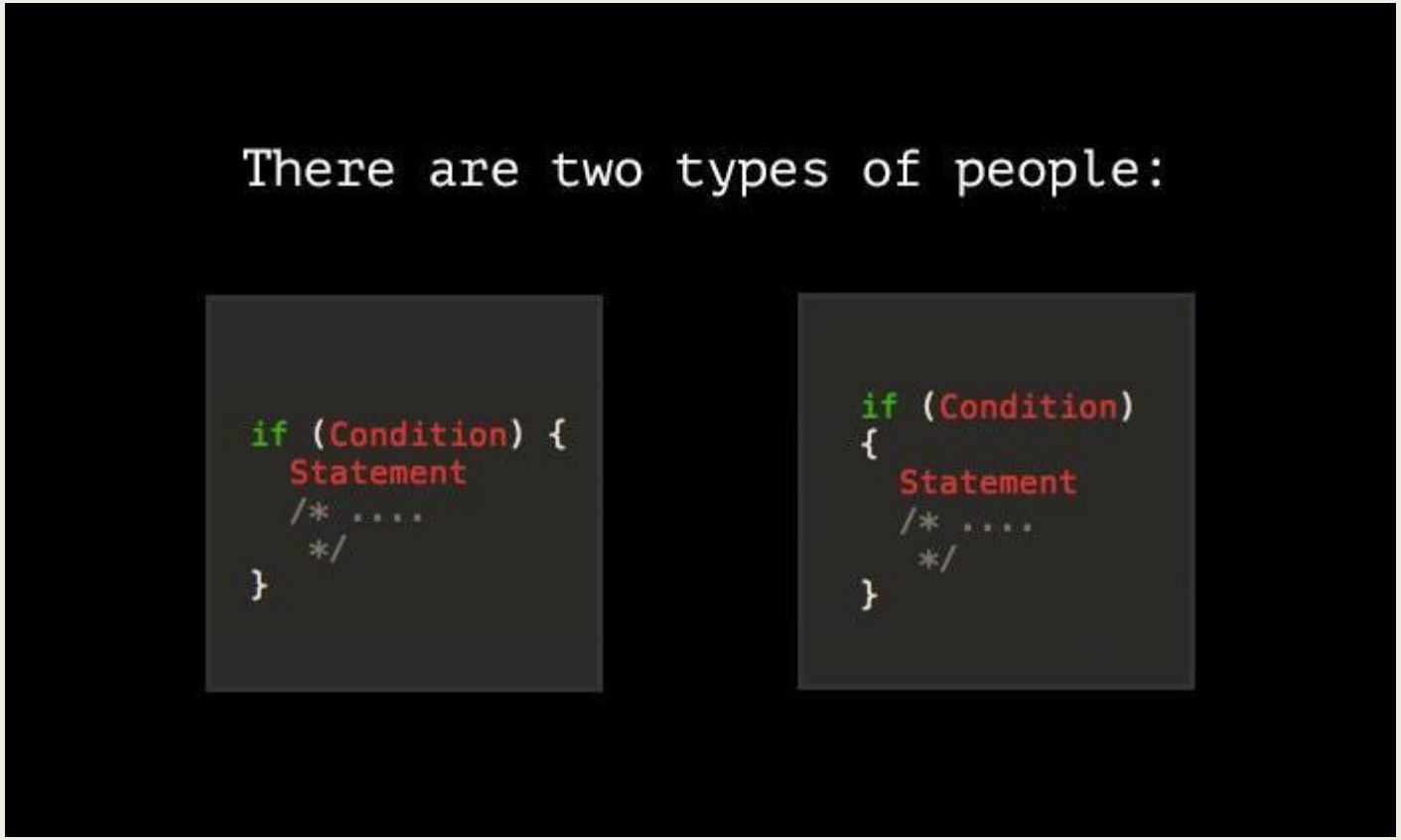
```blade
# resources/views/students/profile.blade.php

<h3> Student: {{ $student->fullName() }} </h3>
```

3. Source Control

- CVS, GIT, SVN
- Share code
- Revert to a known version
- Safe to delete code
- Branch
- Deploy

4. Code Conventions

- Class names, variables, white spaces, curly brackets



Broken windows theory

The broken windows theory is a criminological theory that visible signs of crime, anti-social behavior, and civil disorder create an urban environment that encourages further crime and disorder, including serious crimes.

4. Code Conventions

- Python doesn’t even run if not indented properly

```
if True:
print "Hello!"

#   File "test.py", line 2
#     print "Hello!"
#         ^
# IndentationError: expected an indented block
```

5. Write good code

- Over the span of a year or two, teams that were moving very fast at the beginning of a project can find themselves moving at a snail's pace.
- Changes take longer as you try to understand the system and find the duplicate code.
- Every change they make to the code breaks two or three other parts of the code.
- As the mess builds, the productivity of the team continues to decrease, asymptotically approaching zero.

We are the @authors

- Authors are responsible for communicating well with their readers.
- The ratio of time spent reading vs. writing is well over 10:1.
- We want the reading of code to be easy, even if it makes the writing harder.
- Good programmers write code that is easy to understand and maintain.

The boy scout rule

"Leave the campground cleaner than you found it."

- doesn't have to be a big change (change a variable name, break up a big function, eliminate some duplication)
- the code will improve over time
- https://www.refactoring.com/catalog/index.html has a list of precise rules that can be used

Functions - should be small

- since the early days of programming people have realized that the longer a procedure is, the more difficult it is to understand.
- should be small, no longer than 20 lines
- should do one thing, each one of them should tell a story
- mixing one level of abstraction is confusing, readers will not be able to tell whether a particular expression is an essential concept or a detail.

Levels of abstraction

```
public List<Integer> sortNumbers(String filename) throws FileNotFoundException {

    List<Integer> numbers = new ArrayList<>();

    try (Scanner scanner = new Scanner(new FileInputStream(filename))) {

        while (scanner.hasNext()) {
            numbers.add(scanner.nextInt());
        }

    }

    Collections.sort(numbers);

    return numbers;
}
```

Levels of abstraction

```
public List<Integer> sortNumbers(String filename) throws FileNotFoundException {

    List<Integer> numbers = readNumbers("sample.txt");
    Collections.sort(numbers);

    return numbers;
}
```

Use descriptive names

- "You know you are working on clean code when each routine turns out to be pretty much what you expected“ - Ward Cunningham
- Don't be afraid to spend time choosing a name
- There are only two hard things in Computer Science: cache invalidation and naming things.

Avoid Cognitive breaks

- prefer exceptions to returning error codes
- avoid negative conditionals

```
if (!buffer.shouldNotCompact())
```

- output arguments are counterintuitive

```
void appendFooter (final StringBuffer s) { }
```

If your function must change the state of something, have it change the state of the object it is called on.

```
result.appendFooter();
```

Argument guidelines

- arguments are hard to understand, take a lot of conceptual power
- they are difficult to use
- they are even harder from a testing point of view
- the ideal number of arguments to a function is zero
- avoid flag arguments : passing a Boolean into a function proclaims that the method does more than one thing

Classes - small

- we should have many small classes, and each of them should have one single responsibility
- we should describe the name without using :”if”, “and”, “or” or “but”.
- class names including weasel words like Processor or Manager or Super often hint at aggregation of responsibilities.
- do you want your tools organized into toolboxes with many small drawers each containing well-defined and well-labeled components? Or do you want a few drawers that you just toss everything into?
- Single Responsibility Principle - one responsibility - one reason to change

Single Responsibility Principle

```
class Person {
    public name : string;
    public surname : string;
    public email : string;
    constructor(name : string, surname : string, email : string){
        this.surname = surname;
        this.name = name;
        if(this.validateEmail(email)) {
            this.email = email;
        }
        else {
            throw new Error("Invalid email!");
        }
    }
    validateEmail(email : string) {
        var re = /^([\w-]+(?:\.[\w-]+)*)@((?:[\w-]+\.)*\w[\w-]{0,66})\.([a-z]{2,6}(?:\.[a-z]{2})?)$/i;
        return re.test(email);
    }
    greet() {
        alert("Hi!");
    }
}
```

Single Responsibility Principle

```
class Email {
    public email : string;
    constructor(email : string){
        if(this.validateEmail(email)) {
            this.email = email;
        }
        else {
            throw new Error("Invalid email!");
        }
    }
    validateEmail(email : string) {
        var re = /^([\w-]+(?:\.[\w-]+)*)@((?:[\w-]+\.)*\w[\w-]{0,66})\.([a-z]{2,6}(?:\.[a-z]{2})?)$/i;
        return re.test(email);
    }
}

class Person {
    public name : string;
    public surname : string;
    public email : Email;
    constructor(name : string, surname : string, email : Email){
        this.email = email;
        this.name = name;
        this.surname = surname;
    }
    greet() {
        alert("Hi!");
    }
}
```

Class Cohesion

- Classes should have a small number of instance variables
- Each of the methods of a class should manipulate one or more of those variables.
- The more variables a method manipulates, the more cohesive that method is to its class.

Stack.java

```
public class Stack {
  private int topOfStack = 0;
  List<Integer> elements = new LinkedList<Integer>();
  public int size() {
    return topOfStack;
  }
  public void push(final int element) {
    topOfStack++;
    elements.add(element);
  }
  public int pop() throws PoppedWhenEmpty {
    if (topOfStack == 0) {
        throw new PoppedWhenEmpty();
    }
    final int element = elements.get(--topOfStack);
       elements.remove(topOfStack);
    return element;
  }
```

COMMENTS

Don't comment bad code,
rewrite it.

Comments are at best, necessary evil

- The proper use of comments is to compensate for our failure to express ourselves in code
- When you feel the need to write a comment, first try to refactor the code so that any comment becomes superfluous.
- If you need a comment to explain what a block of code does, try to extract a method.
- If the method is already extracted but you still need a comment to explain what it does, rename that method

Express yourself in code

```
// Check to see if the employee is eligible for full benefits
if ((employee.flags & HOURLY_FLAG) && (employee.age > 65))
```

versus

```
if (employee.isEligibleForFullBenefits())
```

Good Comments

- Legal comments
- Informative comments (regex patterns)

```
// format matched kk:mm:ss EEE, MMM dd, yyyy
```

- Explanation of intent

```
// This is our best attempt to get a race condition by creating large number of threads
```

- Clarification

```
assertTrue(a.compareTo(a) == 0); // a == a
```

- Warnings of consequences

```
//  SimpleDateFormat is not thread safe
```

- Amplification

```
// the trim is real important. It removes the starting spaces that could cause the item to be
recognized as another list
```

Bad Comments

- Mandated comments

```
/**
* Returns the day of the month.
* @return the day of the month.
*/
public int getDayOfMonth() {
    return dayOfMonth;
 }
```

Noise comments

```
/**
* Default constructor.
*/
protected AnnualDateRule() {
}
```

6. Automated tests

- Unit tests
  - *First line of defense against bugs*
  - *Easy to write, closer to the code. TDD*

- Functional Tests
  - *Test system as a whole*
  - *Need less maintenance*
  - *More reliable*

6. Unit tests

```
require 'spec_helper'

describe Post do

  let(:post) { FactoryGirl.create(:post) }

  it "should not be valid without title" do
    post.title = ' '
    expect(post).not_to be_valid
  end

  it "should not be valid without body" do
    post.body = nil
    expect(post).not_to be_valid
  end

end
```

6. Integration Tests

```
class PostFlowTest < Capybara::Rails::TestCase
  def setup
    @one = posts :one
    @two = posts :two
  end

  test 'post index' do
    visit posts_path

    assert page.has_content?(@one.title)
    assert page.has_content?(@two.title)
  end
end
```

7. Dev-ops

- Automate you work, rule of 3
- Know Linux systems basics, configuration, services, permissions

7. Devops - Docker

- Docker useful for developing, testing, deployment production

```
docker run -e 'ACCEPT_EULA=Y'
        -e 'SA_PASSWORD=yourStrong(!)Password'
        -p 1433:1433
        -d microsoft/mssql-server-linux:2017-CU8
```

7. Devops - Ansible

```yaml
---
- name: Install nginx
  hosts: host.name.ip
  become: true

  tasks:
  - name: Add epel-release repo
    yum:
      name: epel-release
      state: present

  - name: Install nginx
    yum:
      name: nginx
      state: present

  - name: Insert Index Page
    template:
      src: index.html
      dest: /usr/share/nginx/html/index.html

  - name: Start NGiNX
    service:
      name: nginx
      state: started
```

8. Code reviews

- Find bugs
- Share knowledge
- Enforce a uniform approach across the project

rs017991 committed 3 days ago Verified

commit 1c21e628ab2bf79cdef9993f5b78e5cea3a7c65e

2 CHANGES.txt

Show comments | Copy path | View file

```diff
@@ -24,7 +24,7 @@

  2.0.11
   * Added case in StringConverter to properly output InetAddress (SPARKC-559)
-  * Added java.time.Instant -> java.util.Data conversion (SPARKC-560)
+  * Added java.time.Instant -> java.util.Date conversion (SPARKC-560)
   * RegularStatements not Cached by SessionProxy (SPARKC-558)
   * Fix CassandraSourceRelation Option Parsing in Spark 2.0 (SPARKC-551)

```

Write | Preview

this looks good to me

Attach files by dragging & dropping, selecting or pasting them.

Styling with Markdown is supported

Cancel | Add single comment | Start a review