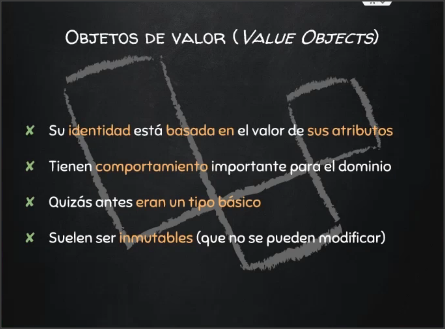
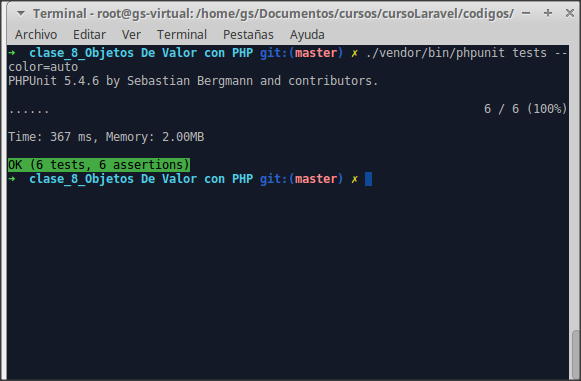
# Objetos De Valor con PHP

La diferencia en objetos es que algunos poseen un valor definido, no cambiará tan radicalmente como un usuario, se da el ejemplo de una cuerda que contiene una guitarra, sea cual sea no existirá diferencia notoria entre una cuerda y otra, quizás en calidad. Por ejemplo un mail es parte tal vez de un objeto usuario, pero este mail posee un comportamiento que hay que programar cada vez que se solicite, y para evitar estar duplicando el código una y otra vez, tal vez una solución sea crear un objeto de no tanto valor como un objeto mail, en donde se manejen las validaciones correspondientes antes de incluirlo en nuestra base de datos.



Se puede notar lo siguiente, si ingreso un mail y este no es un mail si no solo texto se debe corregir de alguna forma, por tanto a pesar de ser un tipo básico, el mail posee un comportamiento.

Las pruebas:



Códigos:

|  |
| --- |
| <?php  namespace PlatziPHP;  use Doctrine\Instantiator\Exception\InvalidArgumentException;  /\*\*  \* Created by PhpStorm.  \* User: gs  \* Date: 22-06-16  \* Time: 15:29  \*/  class Email  {  private $address;  public function \_\_construct( $address )  {  // validador nativo PHP  $valid\_email = filter\_var($address, FILTER\_VALIDATE\_EMAIL);  if( ! $valid\_email )  {  // si no llevara la barra atras iria a buscar a PlatziPhp/  throw new \InvalidArgumentException(  "Invalid email address: [".$address."] "  );  }  $this->address = $address;  }  public function getAddress()  {  return $this->address;  }  } |

|  |
| --- |
| <?php  use PlatziPHP\Email;  class EmailTest extends PHPUnit\_Framework\_TestCase  {  function test\_email\_is\_valid()  {  $email = new Email('fake@mail.dev');  $this->assertInstanceOf(  Email::class,  $email  );  }  function test\_email\_is\_invalid()  {  $this->setExpectedExceptionRegExp(  \InvalidArgumentException::class  );  $email = new Email('this is not an email');  }  } |