Interfaces

An interface enforces behavior that must be implemented in the classes that ‘implement’ that interface:

interface Animal

{

    public function communicate();

}

//implement the interface

class Dog implements Animal

{

    public function communicate()

    {

        return 'bark';

    }

}

class Cat implements Animal

{

    public function communicate()

    {

        return 'meow';

    }

}

\*note all animal classes must have a communicate() method (it’s enforced by the interface they implement);

As a generic rule, whenever a task has multiple possible implementations, we should program them into an interface which makes it easier to change/modify subsequent implementations;

2nd exemple Interface Logger

interface Logger

{

    public function execute($message);

}

class LogToFile implements Logger

{

    public function execute($message)

    {

        var\_dump('log the message to a file ' . $message);

    }

}

class LogToDatabase implements Logger //this may not even exist initially

{

    public function execute($message)

    {

        var\_dump('log the message to a database ' . $message);

    }

}

class UsersController

{

protected $logger;

// public function \_\_construct(LogToFile $logger)  //note that intially we had passed in an implementation of the interface as argument

    public function \_\_construct(Logger $logger)  //now the interface only specifies that it needs a type of logger (so instead of having to pass in the LogToFile class obligatory, we can pass any class that implements the Logger interface)

    {

        $this->logger = $logger;

    }

    public function show()

    {

        $user = 'John Doe';

        //log information

        $this->logger->execute($user);

    }

}

$controller = new UsersController(new LogToDatabase);

$controller->show();

\*see above: an interface dictates how the classes that implement it should behave (in this case that they should have a log() method, that accepts a $message variable as param;

\*however, if in the first case in the UsersController, in the constructor we would pass an implementation of the interface (LogToFile or LogToDatabase) instead of passing the interface (Logger), we would have to modify the logic of the UsersController class (if for instance we would have to replace the LogToFile log() method with the LogToDatabase one.

\*in this case, an interface is perfect to use in order to enforce implementation logic and to de-couple the functionality of the app (we do not need to touch the UsersController but we can modify its behavior by passing it a different Logger interface class);

Interface 3rd example

//interface 3rd example (class able to cast a JSON)

interface CastsToJson

{

    public function toJson();

}

class User implements CastsToJson

{

}

class Collection implements CastsToJson

{

}

Interface 4th example (Repository)

//interface 4th example (Repository);

interface Repository

{

    public function save();

}

class MongoRepository implements Repository   //class that queries a database store

{

    public function save($data)

    {

    }

}

class FileRepository implements Repository   //class that queries a database store

{

    public function save($data)

    {

    }

}

5th example:

//interface 5th example Filterable (data can befiltered)

interface Filterable

{

    public function filter();

}

class Favourited implements Filterable

{

}

class UnreadPosts implements Filterable

{

}

class DifficultyFilter implements Filterable

{

}