**Dependency injection**

dependency injection is a software design pattern that implements inversion of control for software libraries.

 We can use Dependency Injection to write modular, testable and maintainable code:

* ****Modular****: The Dependency Injection helps create completely self-sufficient classes or modules
* ****Testable****: It helps write testable code easily eg unit tests for example
* ****Maintainable****: Since each class becomes modular, it becomes easier to manage it

**Problem**

**<?php**

**class User**

**{**

**private $database = null;**

**public function \_\_construct() {**

**$this->database = new database('host', 'user', 'pass', 'dbname');**

**}**

**public function getUsers() {**

**return $this->database->getAll('users');**

**}**

**}**

**$user = new User();**

**$user->getUsers();**

**?>**

If we wanted to change database credentials, we need to edit the User class which is not good; every class should be completely ****modular**** or black box. If we need to operate further on it, we should actually use its public properties and methods instead of editing it again and again.

The class User has implicit dependency on the specific database.

**Soln**

**<?php**

**class User**

**{**

**private $database = null;**

**//injecting it into the constructor**

**public function \_\_construct(Database $database) {**

**$this->database = $database;**

**}**

**public function getUsers() {**

**return $this->database->getAll('users');**

**}**

**}**

**$database = new Database('host', 'user', 'pass', 'dbname');**

**$user = new User($database);**

**$user->getUsers(); //passing database instance**

**?>**