

W31

Programmation web côté serveur

Pierre Kraemer

kraemer@unistra.fr

POO en PHP

```
class Personne
{
    private $_nom;

    public function nom()
    {
        return $this->_nom;
    }

    public function setNom($n)
    {
        if (is_string($n)) {
            $this->_nom = $n;
        } else {
            $this->_nom = '';
        }
    }
}
```

POO en PHP

```
class Personne
{
    private $_nom;

    public function nom()
    {
        return $this->_nom;
    }

    public function setNom($n)
    {
        if (is_string($n)) {
            $this->_nom = $n;
        } else {
            $this->_nom = '';
        }
    }
}
```

membre privé

méthode publique (getter)

méthode publique (setter)

Méthodes __magiques

```
$p = new Personne;  
$p->setNom( 'Pierre' );
```



crée une instance par défaut,
puis initialise le nom

Méthodes __magiques

```
$p = new Personne;  
$p->setNom('Pierre');
```



crée une instance par défaut,
puis initialise le nom

```
class Personne  
{  
    ...  
    public function __construct($nom)  
    {  
        $this->setNom($nom);  
    }  
    ...  
}
```



constructeur
(le destructeur s'appelle
__destruct)

Méthodes __magiques

```
$p = new Personne;  
$p->setNom('Pierre');
```



crée une instance par défaut,
puis initialise le nom

```
class Personne  
{  
    ...  
    public function __construct($nom)  
    {  
        $this->setNom($nom);  
    }  
    ...  
}
```

constructeur
(le destructeur s'appelle
__destruct)

```
$p = new Personne('Pierre');
```



crée une instance initialisée
par le constructeur

Méthodes __magiques

```
$p = new Personne;  
$p->setNom('Pierre');
```



crée une instance par défaut,
puis initialise le nom

```
class Personne  
{
```

```
...  
public function __construct($nom)  
{  
    $this->setNom($nom);  
}  
...
```



constructeur
(le destructeur s'appelle
__destruct)

```
}
```

\$p est de type *object* et
instanceof Personne

```
$p = new Personne('Pierre');
```



crée une instance initialisée
par le constructeur


Méthodes __magiques

```
class Personne
{
    ...
    public function __toString()
    {
        return 'Je m\'appelle ' . $this->_nom;
    }
    ...
}
```


Méthodes __magiques

```
class Personne
{
    ...
    public function __toString()
    {
        return 'Je m\'appelle ' . $this->_nom;
    }
    ...
}
```


appelée lors d'une conversion
implicite comme : `echo $p;`



Méthodes __magiques

```
class Personne
{
    ...
    public function __toString()
    {
        return 'Je m\'appelle ' . $this->_nom;
    }
    ...
}
```

appelée lors d'une conversion
implicite comme : `echo $p;`



Autres méthodes magiques :

`__get()`, `__set()`, `__isset()`, `__unset()`, `__call()`, `__clone()`, ...

Héritage - Interfaces

```
interface Drawable
{
    public function draw(Canvas $c);
}

abstract class Forme implements Drawable
{
    protected $_position;
    protected $_couleur;

    abstract public function draw(Canvas $c);
    abstract public function area();
}

class Forme_Cercle extends Forme
{
    private $_rayon;

    public function __construct($r) {
        $_rayon = (double) $r;
    }
}
```

Héritage - Interfaces


```
interface Drawable
{
    public function draw(Canvas $c);
}

abstract class Forme implements Drawable
{
    protected $_position;
    protected $_couleur;

    abstract public function draw(Canvas $c);
    abstract public function area();
}

class Forme_Cercle extends Forme
{
    private $_rayon;

    public function __construct($r) {
        $_rayon = (double) $r;
    }
}
```



vérifie que le
paramètre est de type
object et instanceof
Canvas

Héritage - Interfaces

```
interface Drawable
{
    public function draw(Canvas $c);
}
```

vérifie que le
paramètre est de type
object et **instanceof**
Canvas

```
abstract class Forme implements Drawable
{
    protected $_position;
    protected $_couleur;

    abstract public function draw(Canvas $c);
    abstract public function area();
}
```

ne peut pas
être instanciée

```
class Forme_Cercle extends Forme
{
    private $_rayon;

    public function __construct($r) {
        $_rayon = (double) $r;
    }
}
```

Héritage - Interfaces

```
abstract class Forme implements Drawable
{
    protected $_position;
    protected $_couleur;

    abstract public function draw(Canvas $c);
    abstract public function area();
}

class Forme_Cercle extends Forme
{
    private $_rayon;

    public function __construct($r) {
        $_rayon = (double) $r;
    }

    public function draw(Canvas $c) {
        $c->trace(..., $this->_position, $this->_couleur, ...);
    }

    public function area() {
        return M_PI * $this->_rayon * $this->_rayon;
    }
}

class Forme_Carre extends Forme
{
    private $_cote;
```

Héritage - Interfaces

```
abstract class Forme implements Drawable
{
    protected $_position;
    protected $_couleur;

    abstract public function draw(Canvas $c);
    abstract public function area();
}
```

accessible
aux classes
filles

```
class Forme_Cercle extends Forme
{
    private $_rayon;

    public function __construct($r) {
        $_rayon = (double) $r;
    }

    public function draw(Canvas $c) {
        $c->trace(..., $this->_position, $this->_couleur, ...);
    }

    public function area() {
        return M_PI * $this->_rayon * $this->_rayon;
    }
}
```

```
class Forme_Carre extends Forme
{
    private $_cote;
```

Héritage - Interfaces

```
}
    public function draw(Canvas $c) {
        $c->trace(..., $this->_position, $this->_couleur, ...);
    }
    public function area() {
        return M_PI * $this->_rayon * $this->_rayon;
    }
}

class Forme_Carre extends Forme
{
    private $_cote;

    public function __construct($c) {
        $_cote = (double) $c;
    }
    public function draw(Canvas $c) {
        $c->trace(..., $this->_position, $this->_couleur, ...);
    }
    public function area() {
        return $this->_cote * $this->_cote;
    }
}

class Dessin
{
    private $_formes;
```


Héritage - Interfaces

```
}  
    public function area() {  
        return $this->_cote * $this->_cote;  
    }  
}  
  
class Dessin  
{  
    private $_formes;  
    private $_canvas;  
  
    public function __construct() {  
        $this->_formes = array();  
        $this->_canvas = new Canvas;  
    }  
    public function addForme(Forme $f) {  
        $this->_formes[] = $f;  
    }  
    public function update() {  
        $this->_canvas->clear();  
        foreach ($this->_formes as $f) {  
            $f->draw($this->_canvas);  
        }  
    }  
}  
  
$d = new Dessin();
```

Héritage - Interfaces

```
}  
    public function area() {  
        return $this->_cote * $this->_cote;  
    }  
}
```

```
class Dessin
```

```
{  
    private $_formes;  
    private $_canvas;  
  
    public function __construct() {  
        $this->_formes = array();  
        $this->_canvas = new Canvas;  
    }  
    public function addForme(Forme $f) {  
        $this->_formes[] = $f;  
    }  
    public function update() {  
        $this->_canvas->clear();  
        foreach ($this->_formes as $f) {  
            $f->draw($this->_canvas);  
        }  
    }  
}
```

tableau de
formes
génériques

```
$d = new Dessin();
```

Héritage - Interfaces

```
        $this->_canvas = new Canvas;
    }
    public function addForme(Forme $f) {
        $this->_formes[] = $f;
    }
    public function update() {
        $this->_canvas->clear();
        foreach ($this->_formes as $f) {
            $f->draw($this->_canvas);
        }
    }
}

$d = new Dessin();

$cercle = new Forme_Cercle(2.3);
$carre = new Forme_Carre(4.7);

$d->addForme($cercle);
$d->addForme($carre);

$d->update();
```

Références et clônage

```
$c1 = new Forme_Cercle(1.6);
```

Références et clônage

`$c1 = new Forme_Cercle(1.6);`

référence vers un
objet de type
Forme_Cercle

Références et clônage

`$c1 = new Forme_Cercle(1.6);`

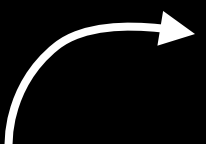
référence vers un
objet de type
Forme_Cercle

`$c2 = $c1;`

Références et clônage

```
$c1 = new Forme_Cercle(1.6);
```

référence vers un
objet de type
Forme_Cercle



```
$c2 = $c1;
```

nouvelle référence
vers le même objet



Références et clônage

```
$c1 = new Forme_Cercle(1.6);
```

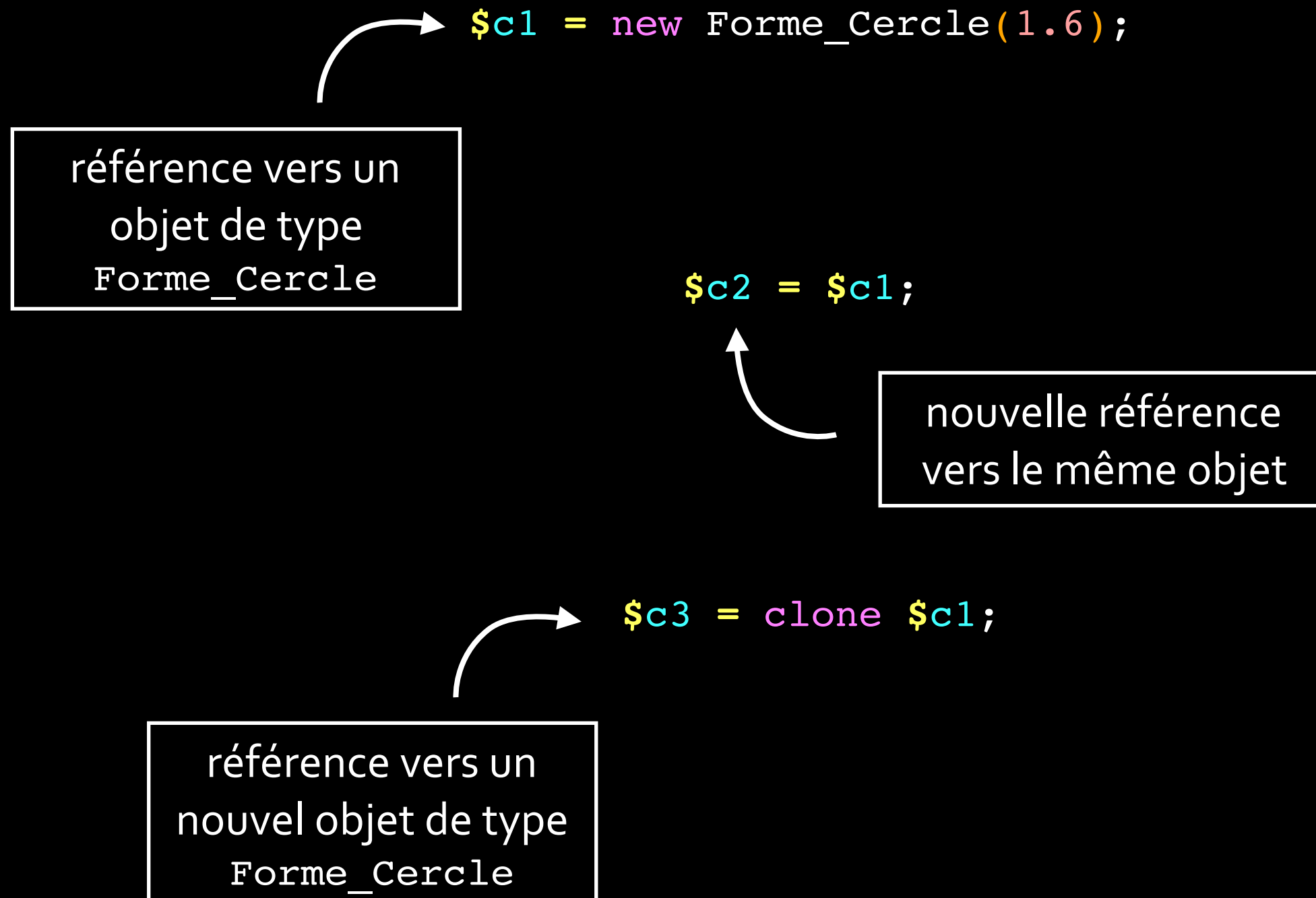
référence vers un
objet de type
Forme_Cercle

```
$c2 = $c1;
```

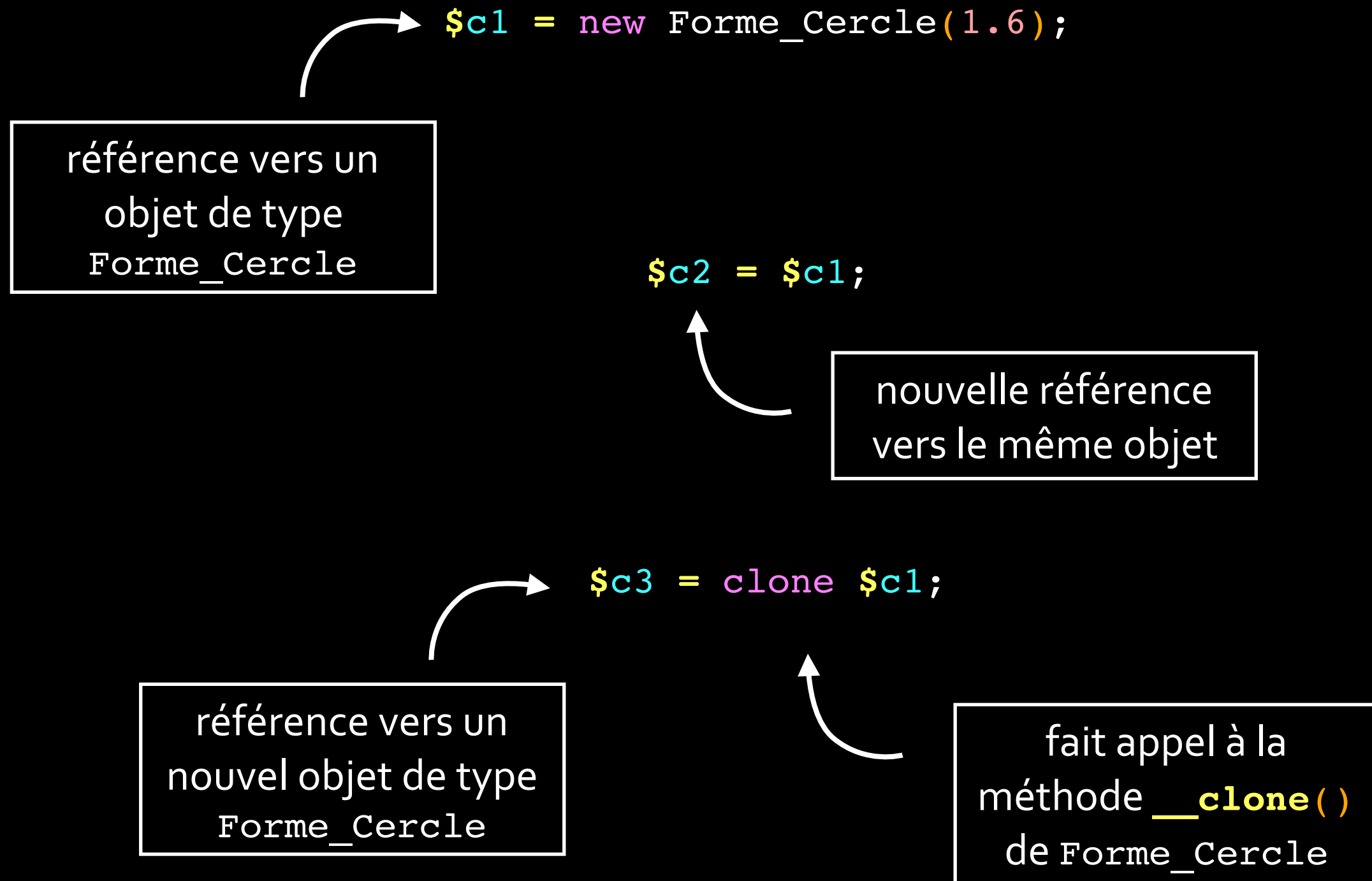
nouvelle référence
vers le même objet

```
$c3 = clone $c1;
```


Références et clônage



Références et clônage



Accès aux BDD

Accès aux BDD

ORACLE®

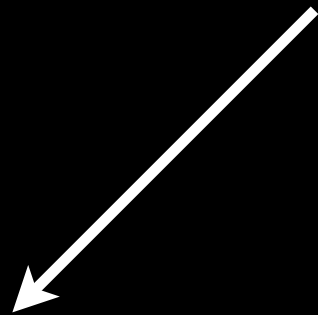


PostgreSQL

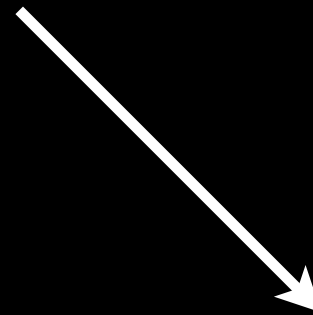


Accès aux BDD

PDO
PHP Data Objects



ORACLE®



PostgreSQL



Accès aux BDD

```
define( 'SQL_DSN',      'mysql:host=localhost;dbname=w31' );  
define( 'SQL_USERNAME', 'login' );  
define( 'SQL_PASSWORD', 'password' );
```


```
try {  
    $db = new PDO(SQL_DSN, SQL_USERNAME, SQL_PASSWORD);  
}  
catch(Exception $e) {  
    echo 'Erreur de connexion à la BDD : ' . $e->getMessage();  
    exit;  
}
```

```
$result = $db->query( 'SELECT * FROM formes' );
```

```
while($data = $result->fetch(PDO::FETCH_ASSOC)) {  
    foreach($data as $key => $value) {
```

Accès aux BDD

paramètres de la
connexion au serveur



```
define( 'SQL_DSN',      'mysql:host=localhost;dbname=w31' );  
define( 'SQL_USERNAME', 'login' );  
define( 'SQL_PASSWORD', 'password' );
```

```
try {  
    $db = new PDO( SQL_DSN, SQL_USERNAME, SQL_PASSWORD );  
}  
catch( Exception $e ) {  
    echo 'Erreur de connexion à la BDD : ' . $e->getMessage();  
    exit;  
}
```

```
$result = $db->query( 'SELECT * FROM formes' );
```

```
while( $data = $result->fetch( PDO::FETCH_ASSOC ) ) {  
    foreach( $data as $key => $value ) {
```

Accès aux BDD

paramètres de la
connexion au serveur

```
define( 'SQL_DSN',      'mysql:host=localhost;dbname=w31' );  
define( 'SQL_USERNAME', 'login' );  
define( 'SQL_PASSWORD', 'password' );
```

création de l'objet PDO
(établit la connexion)

```
try {  
    $db = new PDO(SQL_DSN, SQL_USERNAME, SQL_PASSWORD);  
}  
catch(Exception $e) {  
    echo 'Erreur de connexion à la BDD : ' . $e->getMessage();  
    exit;  
}
```

```
$result = $db->query( 'SELECT * FROM formes' );
```

```
while($data = $result->fetch(PDO::FETCH_ASSOC)) {
```

```
    foreach($data as $key => $value) {
```


Accès aux BDD

```
        exit;
    }

    $result = $db->query('SELECT * FROM formes');

    while($data = $result->fetch(PDO::FETCH_ASSOC)) {
        foreach($data as $key => $value) {
            echo $key . ' -> ' . $value;
        }
    }


    $nb = $db->exec("DELETE FROM formes WHERE couleur = 'green'");

    echo 'Nombre d\'éléments effacés : ' . $nb;
```

Accès aux BDD


id	position	couleur
1	4,6	blue
2	-12,2	green
3	3,7	red

```
exit;  
} requête de récupération  
de données
```



```
$result = $db->query('SELECT * FROM formes');
```

```
while($data = $result->fetch(PDO::FETCH_ASSOC)) {  
    foreach($data as $key => $value) {  
        echo $key . ' -> ' . $value;  
    }  
}  
id -> 1  
position -> 4,6  
couleur -> blue
```



parcours des
données obtenues

```
$nb = $db->exec("DELETE FROM formes WHERE couleur = 'green'");
```

```
echo 'Nombre d\'éléments effacés : ' . $nb;
```

Accès aux BDD

id	position	couleur
1	4,6	blue
2	-12,2	green
3	3,7	red

```
exit;  
} requête de récupération  
de données
```




```
$result = $db->query('SELECT * FROM formes');
```

```
while($data = $result->fetch(PDO::FETCH_ASSOC)) {  
    foreach($data as $key => $value) {  
        echo $key . ' -> ' . $value;  
    }  
}
```



parcours des
données obtenues

```
requête de mise-à-  
jour des données
```



```
$nb = $db->exec("DELETE FROM formes WHERE couleur = 'green'");
```

```
echo 'Nombre d\'éléments effacés : ' . $nb;
```



nombre de lignes
affectées

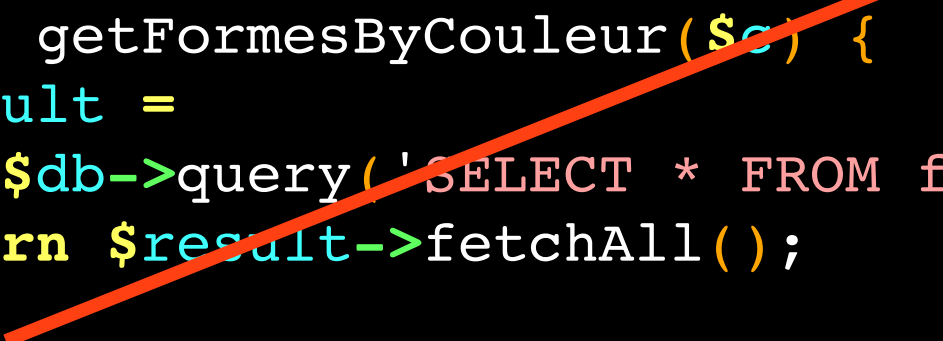
Accès aux BDD

```
function getFormesByCouleur($c) {  
    $result =  
        $db->query('SELECT * FROM formes WHERE couleur = ' . $c);  
    return $result->fetchAll();  
}
```

```
function getFormesByCouleur($c) {  
    $statement =  
        $db->prepare('SELECT * FROM formes WHERE couleur = :c');  
    $statement->bindValue(':c', $c, PDO::PARAM_STR);  
    $statement->execute();  
    return $statement->fetchAll();  
}
```

Accès aux BDD

```
function getFormesByCouleur($c) {  
    $result =  
        $db->query('SELECT * FROM formes WHERE couleur = ' . $c);  
    return $result->fetchAll();  
}
```



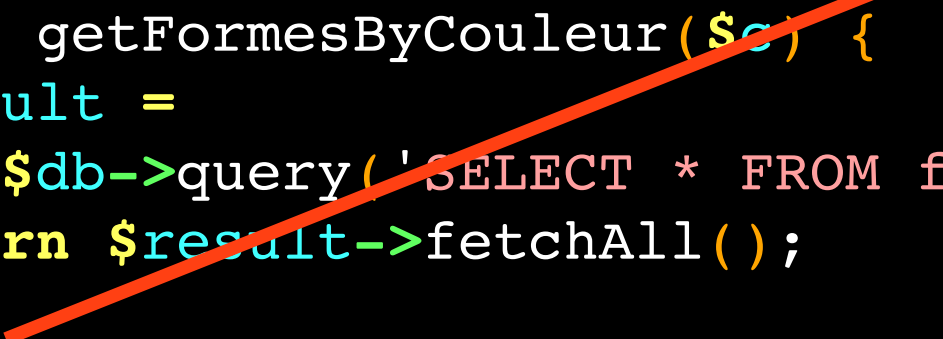
pas bien : risque
d'injection SQL



```
function getFormesByCouleur($c) {  
    $statement =  
        $db->prepare('SELECT * FROM formes WHERE couleur = :c');  
    $statement->bindValue(':c', $c, PDO::PARAM_STR);  
    $statement->execute();  
    return $statement->fetchAll();  
}
```

Accès aux BDD

```
function getFormesByCouleur($c) {  
    $result =  
        $db->query('SELECT * FROM formes WHERE couleur = ' . $c);  
    return $result->fetchAll();  
}
```




pas bien : risque
d'injection SQL

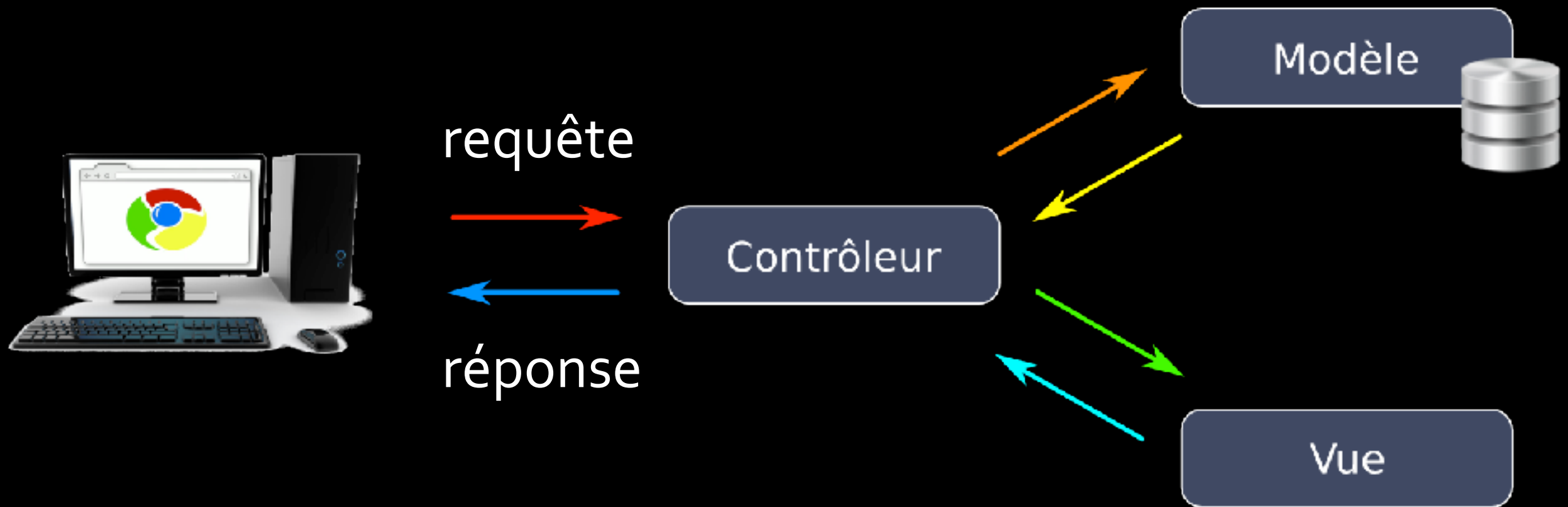


```
function getFormesByCouleur($c) {  
    $statement =  
        $db->prepare('SELECT * FROM formes WHERE couleur = :c');  
    $statement->bindValue(':c', $c, PDO::PARAM_STR);  
    $statement->execute();  
    return $statement->fetchAll();  
}
```

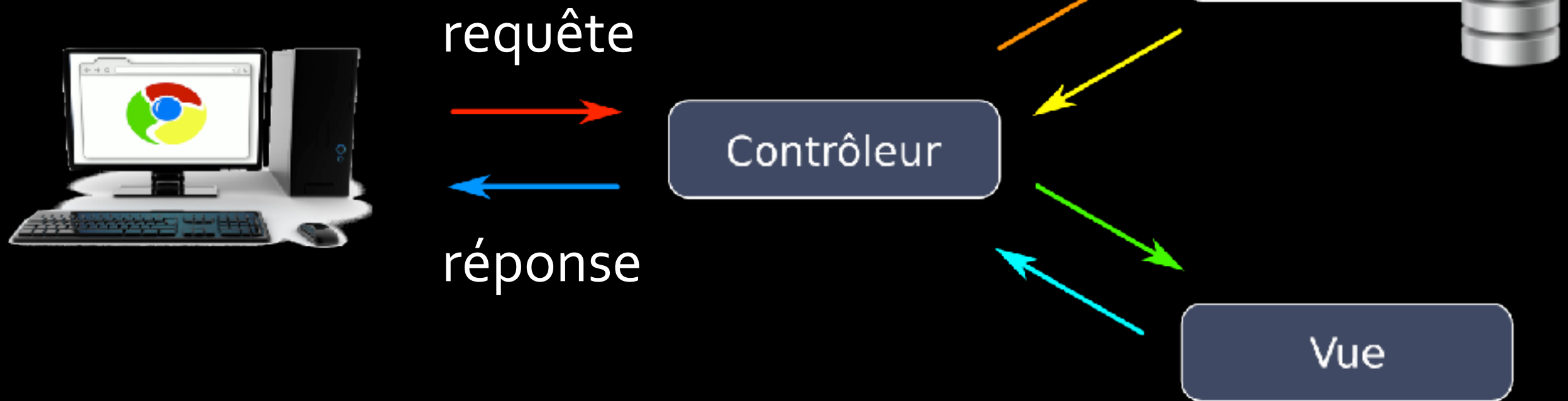
bien : PDO prépare
la requête et vérifie
les paramètres



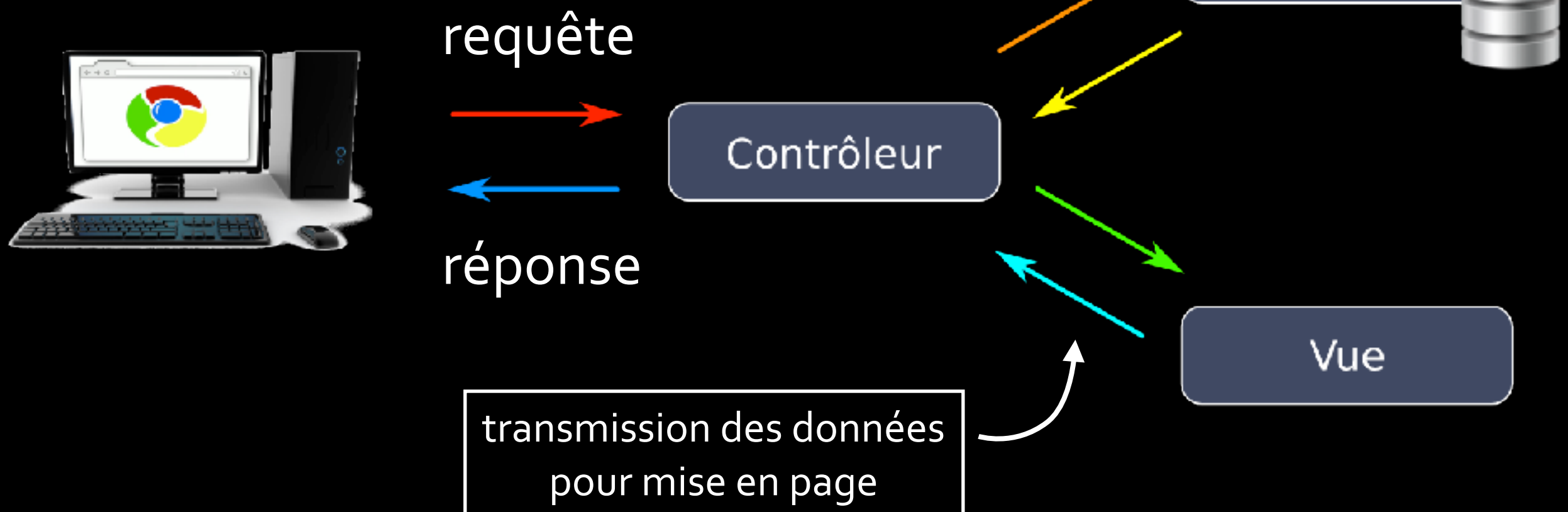
Le Pattern MVC



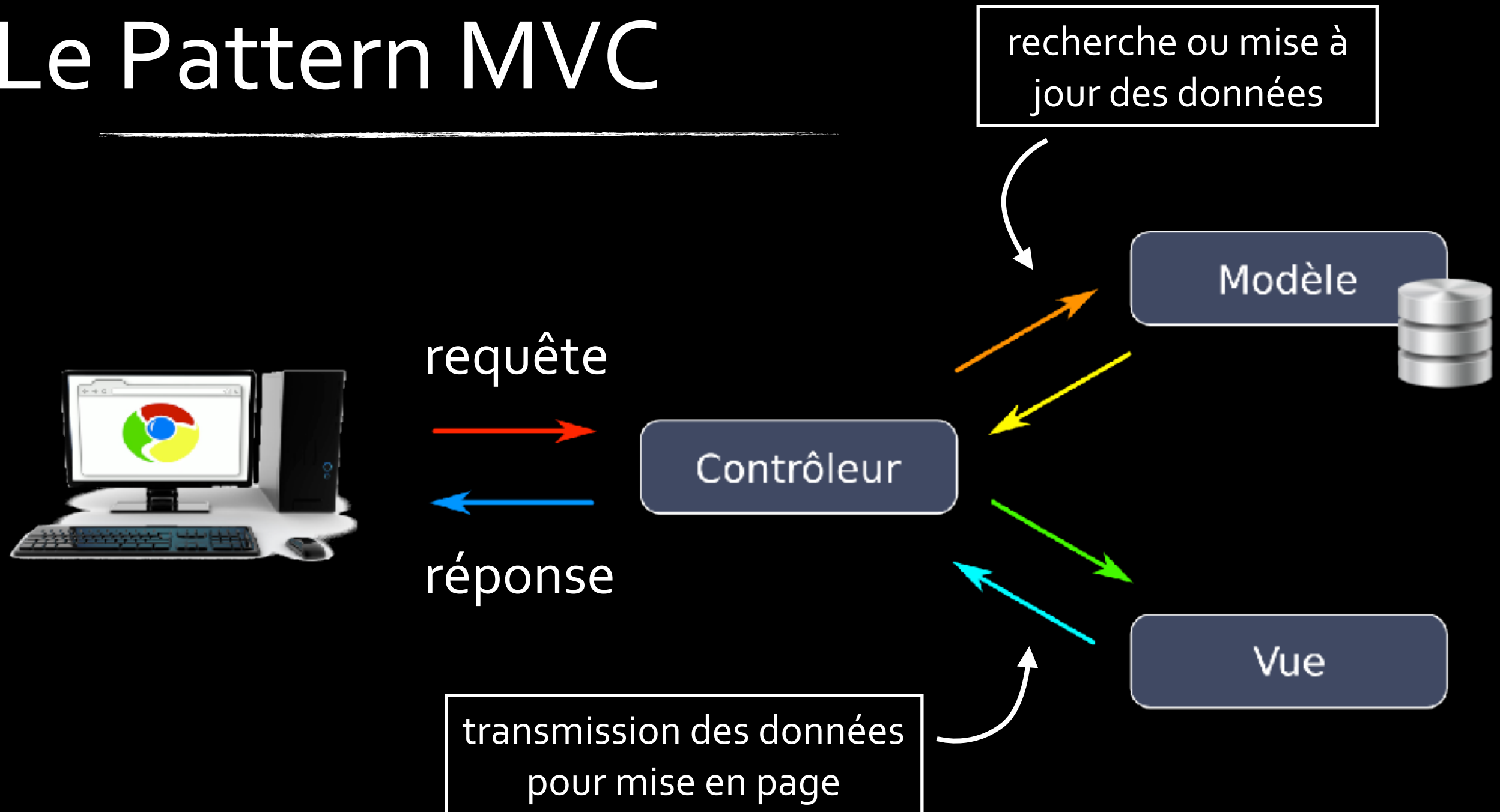
Le Pattern MVC



Le Pattern MVC



Le Pattern MVC



- Séparation des tâches
- Organisation du code
- Code plus propre et plus maintenable

Modèles

Classes représentant les entités de la BDD


Modèles

Classes représentant les entités de la BDD

```
class Forme extends Model_Base  
{
```

```
    private $_id;  
    private $_position;  
    private $_couleur;
```

champs privés correspondant aux
colonnes de la table en BDD



```
    public function set_id($id) {  
        $id = (int) $id;  
        if ($id > 0) {  
            $this->_id = $id;  
        }  
    }  
}
```

setters



```
    public function id() {  
        return $this->_id;  
    }  
}
```

getters



...

```
    public function save() {
```

Modèles

Classes représentant les entités de la BDD

...

```
public function save() {  
    if(! is_null($this->_id)) {  
        $q = self::$_db->prepare('UPDATE formes SET position = :p,  
                                couleur = :c WHERE id = :id');  
        $q->bindValue(':p', $this->_position, PDO::PARAM_STR);  
        $q->bindValue(':c', $this->_couleur, PDO::PARAM_STR);  
        $q->bindValue(':id', $this->_id, PDO::PARAM_INT);  
        $q->execute();  
    }  
}
```

champ statique PDO hérité
de la classe mère commune
aux modèles

```
public function delete() {  
    $q = self::$_db->prepare('DELETE FROM formes WHERE id = :id');  
    $q->bindValue(':id', $this->_id);  
    $q->execute();  
    $this->_id = NULL;  
}
```

...

Modèles

Classes représentant les entités de la BDD

...

```
public static function create($position, $couleur) {
    $f = new Forme();

    $q = self::$_db->prepare('INSERT INTO formes SET
                                position = :p, couleur = :c');
    $q->bindValue(':p', $position, PDO::PARAM_STR);
    $q->bindValue(':c', $couleur, PDO::PARAM_STR);
    $q->execute();

    $f->set_id(self::$_db->lastInsertId());
    $f->set_position($position);
    $f->set_couleur($couleur);
    return $f;
}

public static function getById($id) {
    $id = (int) $id;
    $q = self::$_db->prepare('SELECT * FROM formes WHERE id = :id');
    $q->bindValue(':id', $id, PDO::PARAM_INT);
```

Modèles

Classes représentant les entités de la BDD

```
}  
  
public static function getById($id) {  
    $id = (int) $id;  
    $q = self::$_db->prepare('SELECT * FROM formes WHERE id = :id');  
    $q->bindValue(':id', $id, PDO::PARAM_INT);  
    $q->execute();  
    if($data = $q->fetch(PDO::FETCH_ASSOC)) {  
        $f = new Forme();  
        $f->set_id($data['id']);  
        $f->set_position($data['position']);  
        $f->set_couleur($data['couleur']);  
        return $f;  
    } else {  
        return NULL;  
    }  
}  
  
public static function getAll() {  
    $formes = array();  
    $q = self::$_db->query('SELECT * FROM formes');
```

Modèles

Classes représentant les entités de la BDD

```
}  
  
public static function getAll() {  
    $formes = array();  
    $q = self::$_db->query('SELECT * FROM formes');  
    while ($data = $q->fetch(PDO::FETCH_ASSOC)) {  
        $f = new Forme();  
        $f->set_id($data['id']);  
        $f->set_position($data['position']);  
        $f->set_couleur($data['couleur']);  
        $formes[] = $f;  
    }  
    return $formes;  
}  
}
```

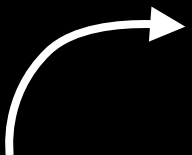

Vues

Code HTML (+ JavaScript) pouvant exploiter des données provenant des modèles

Vues

Code HTML (+ JavaScript) pouvant exploiter des données provenant des modèles

tableau
contenant toutes
les formes



```
<section>

<p>Couleurs des formes :</p>
<ul>
<?php
foreach($formes as $f) {
    echo '<li>' . $f->couleur() . '</li>';
}
?>
</ul>

</section>
```

Contrôleurs

Concentrent la logique de l'application web

Contrôleurs

Concentrent la logique de l'application web

```
require_once 'models/formes.php';

class Controller_Formes
{
    public function viewAll()
    {
        $formes = Forme::getAll();
        include 'views/viewFormes.php';
    }

    public function view($id)
    {
        $forme = Forme::getById($id);
        include 'views/viewForme.php';
    }

    ...
}
```

récupération de la
liste des formes



chargement de la vue
correspondante

Contrôleur d'entrée (routeur)

Point d'entrée de l'application web

Client → index.php

Contrôleur d'entrée (routeur)

Point d'entrée de l'application web

Client → `index.php?section=forme&action=view`

Contrôleur d'entrée (routeur)

Point d'entrée de l'application web

Client → index.php?section=forme&action=view

```
<?php
```

```
$found = false;
```

```
$controller_file = 'controllers/' . $_GET['section'] . '.php';  
if (is_file($controller_file)) {  
    require_once $controller_file;  
    $controller_name = 'Controller_' . ucfirst($_GET['section']);  
    if (class_exists($controller_name)) {  
        $c = new $controller_name;  
        if (method_exists($c, $_GET['action'])) {  
            $c->$_GET['action']();  
            $found = true;  
        }  
    }  
}
```

Contrôleur d'entrée (routeur)

Point d'entrée de l'application web

Client → index.php?section=forme&action=view

```
<?php
```

```
$found = false;
```

```
$controller_file = 'controllers/' . $_GET['section'] . '.php';  
if (is_file($controller_file)) {  
    require_once $controller_file;  
    $controller_name = 'Controller_' . ucfirst($_GET['section']);  
    if (class_exists($controller_name)) {  
        $c = new $controller_name;  
        if (method_exists($c, $_GET['action'])) {  
            $c->$_GET['action']();  
            $found = true;  
        }  
    }  
}
```

inclusion du fichier
contenant le contrôleur :
controller_forme.php

Contrôleur d'entrée (routeur)

Point d'entrée de l'application web

Client → index.php?section=forme&action=view

```
<?php

$found = false;

$controller_file = 'controllers/' . $_GET['section'] . '.php';
if (is_file($controller_file)) {
    require_once $controller_file;
    $controller_name = 'Controller_' . ucfirst($_GET['section']);
    if (class_exists($controller_name)) {
        $c = new $controller_name;
        if (method_exists($c, $_GET['action'])) {
            $c->$_GET['action']();
            $found = true;
        }
    }
}
```

inclusion du fichier
contenant le contrôleur :
controller_forme.php

instanciation de l'objet
contrôleur :
Controller_Forme

Contrôleur d'entrée (routeur)

Point d'entrée de l'application web

Client → index.php?section=forme&action=view

```
<?php
```

```
$found = false;
```

```
$controller_file = 'controllers/' . $_GET['section'] . '.php';  
if (is_file($controller_file)) {  
    require_once $controller_file;  
    $controller_name = 'Controller_' . ucfirst($_GET['section']);  
    if (class_exists($controller_name)) {  
        $c = new $controller_name;  
        if (method_exists($c, $_GET['action'])) {  
            $c->$_GET['action']();  
            $found = true;  
        }  
    }  
}
```

inclusion du fichier
contenant le contrôleur :
`controller_forme.php`

instanciation de l'objet
contrôleur :
`Controller_Forme`

appel de la méthode
demandée : `view()`

Contrôleur d'entrée (routeur)

Point d'entrée de l'application web

Client → index.php?section=forme&action=view

```
$found = false;

$controller_file = 'controllers/' . $_GET['section'] . '.php';
if (is_file($controller_file)) {
    require_once $controller_file;
    $controller_name = 'Controller_' . ucfirst($_GET['section']);
    if (class_exists($controller_name)) {
        $c = new $controller_name;
        if (method_exists($c, $_GET['action'])) {
            $c->$_GET['action']();
            $found = true;
        }
    }
}

if (!$found) {
    http_response_code(404);
    include 'views/errors/404.php';
}
```

Contrôleur d'entrée

Point d'entrée de l'application web

Client → index.php

Contrôleur d'entrée

Point d'entrée de l'application web

Client → index.php / forme / view / 12

Contrôleur d'entrée

Point d'entrée de l'application web

Client → index.php / forme / view / 12

```
<?php
```

```
$found = false;
```

```
$args = explode('/', $_SERVER['PATH_INFO']);
```

```
$controller = $args[1];
```

```
$method = $args[2];
```

```
$params = array();
```

```
for($i=3; $i<count($args); $i++) {
```

```
    $params[] = $args[$i];
```

```
}
```

```
$controller_file = 'controllers/' . $controller . '.php';
```

```
if (is_file($controller_file)) {
```

```
    require_once $controller_file;
```

```
    $controller_name = 'Controller_' . ucfirst($controller);
```

```
    if (class_exists($controller_name)) {
```

```
        $c = new $controller_name;
```

```
        if (method_exists($c, $method)) {
```

Contrôleur d'entrée

Point d'entrée de l'application web

Client → index.php / forme / view / 12

```
<?php
```

```
$found = false;
```

```
$args = explode('/', $_SERVER['PATH_INFO']);
```

```
$controller = $args[1];
```

```
$method = $args[2];
```

```
$params = array();
```

```
for($i=3; $i<count($args); $i++) {
```

```
    $params[] = $args[$i];
```

```
}
```

```
$controller_file = 'controllers/' . $controller . '.php';
```

```
if (is_file($controller_file)) {
```

```
    require_once $controller_file;
```

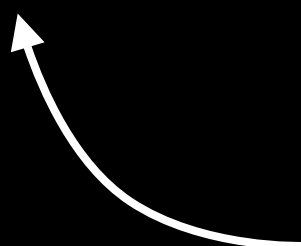
```
    $controller_name = 'Controller_' . ucfirst($controller);
```

```
    if (class_exists($controller_name)) {
```

```
        $c = new $controller_name;
```

```
        if (method_exists($c, $method)) {
```

contient la chaîne :
"/ forme / view / 12"



Contrôleur d'entrée

Point d'entrée de l'application web

Client → index.php / forme / view / 12

```
$controller_file = 'controllers/' . $controller . '.php';
if (is_file($controller_file)) {
    require_once $controller_file;
    $controller_name = 'Controller_' . ucfirst($controller);
    if (class_exists($controller_name)) {
        $c = new $controller_name;
        if (method_exists($c, $method)) {
            call_user_func_array(array($c, $method), $params);
            $found = true;
        }
    }
}

if (!$found) {
    http_response_code(404);
    include 'views/errors/404.php';
}
```

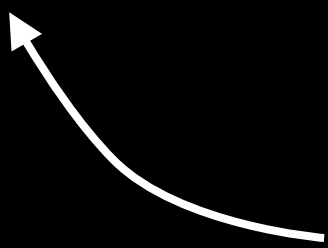

Contrôleur d'entrée

Point d'entrée de l'application web

Client → index.php / forme / view / 12

```
$controller_file = 'controllers/' . $controller . '.php';
if (is_file($controller_file)) {
    require_once $controller_file;
    $controller_name = 'Controller_' . ucfirst($controller);
    if (class_exists($controller_name)) {
        $c = new $controller_name;
        if (method_exists($c, $method)) {
            call_user_func_array(array($c, $method), $params);
            $found = true;
        }
    }
}

if (!$found) {
    http_response_code(404);
    include 'views/errors/404.php';
}
```



appel de la méthode `$method` de
l'objet `$c` avec les paramètres
contenus dans le tableau `$params`