תיכנות מונחה עצמים

אובייקטים ומחלקות

התמיכה בתיכנות מונחה עצמים התווספה ל-PHP החל מגירסה 5.

הגדרת מחלקה ב-PHP נעשית באופן הבא:

```
class שם המחלקה
{
	...
}
```

דוגמא:

```
class Rectangle
{
	var $width;
	var $height;
	function area()
	{
		return $this->width * $this->height;
	}
}
```

בדוגמא זו הוגדרה המחלקה Rectangle. כל אובייקט שיווצר מהמחלקה הזו ייצג מלבן אחר. כל אובייקט שיווצר מהמחלקה הזו יכלול בתוכו שני משתנים: width ו-height. כל אובייקט ייצג מלבן אחר ולפיכך עשוי לכלול בתוך שני המשתנים הללו ערכים אחרים. כאשר מגדירים משתנים בתוך מחלקה במובן שכל אובייקט שיווצר ממנה יכלול בתוכו את שני המשתנים הללו יש להשתמש במילה השמורה var בהגדרה של כל אחד מהם (או לחילופין לציין את הרשאת הגישה כפי שיוסבר בהמשך).

כאשר פונים בתוך מתודה שהוגדרה במחלקה למשתנה שקיים בתוך האובייקט שכעת המתודה פועלת עליו יש להשתמש במילה השמורה $this באופן הבא:

$this → שם המשתנה

דוגמא:

$this → width

יש לשים לב שכאשר פונים למשתנה באובייקט מסויים באמצעות $this יש להשמיט את סימן ה-$ משמו של המשתנה שאליו פונים.

יצירת אובייקט חדש נעשית באמצעות המילה השמורה new באופן הבא:

שם משתנה = new שם המחלקה();

דוגמא:

```
$var = new Rectangle();
```

בדוגמא זו נוצר אובייקט חדש מהמחלקה Rectangle. לכל אובייקט יש reference. ה-reference הוא מעין הכתובת של האובייקט

בזיכרון. כאשר יוצרים אובייקט יש להכניס את ה-reference שלו אל תוך משתנה. השימוש שלנו באובייקט ייעשה באמצעות המשתנה. בדוגמא זו ה-reference של האובייקט החדש שנוצר מהמחלקה Rectangle נכנס אל תוך המשתנה var$.

כיוון שאנו עובדים עם references של אובייקטים ניסיון לבצע השמה של הערך של משתנה אחד (אשר מחזיק ב-reference של אובייקט מסויים) אל תוך משתנה אחר יגרום לכך ששני המשתנים יכילו את אותו reference.

$rec1 = new Rectangle();

$rec2 = $rec1l

גם המשתנה rec1 וגם המשתנה rec2 יחזיקו באותו reference לאותו אובייקט. כל שינוי שנבצע באובייקט באמצעות המשתנה rec1 יבוא לידי ביטוי גם כשנבדוק את האובייקט באמצעות המשתנה rec2.

באמצעות האופרטור '<-' ניתן לקרוא להפעלתה של מתודה (מתודה זו פונקציה שמוגדרת בתוך גבולות של מחלקה) שהוגדרה במחלקה על אובייקט מסויים שנוצר ממנה.

שם המתודה → שם המשתנה

דוגמא:

$rec1->area();

שורה זו תגרום להפעלת המתודה area על האובייקט שנוצר מהמחלקה Rectangle ושה-reference שלו נמצא בתוך המשתנה rec1.

הדוגמא הבאה מציגה הגדרה של המחלקה Rectanle, יצירת אובייקט ממנה כדי לתאר מלבן מסויים והפעלה של המתודה area עליו. כמו כן, הדוגמא גם כוללת הגדרה של המתודות setWidth ו-setHeight שבאמצעותן מוכנסים ערכים לתוך המשתנים שיש באובייקט שנוצר.

```
<?php
class Rectangle
{
	var $width;
	var $height;
	function area()
	{
		return $this->width * $this->height;
	}
	function setWidth($val)
	{
		if($val>0)
		{
			$this->width = $val;
		}
	}
```

```
	function setHeight($val)
	{
		if($val>0)
		{
			$this->height = $val;
		}
	}
}

$rec1 = new Rectangle();
$rec2 = $rec1;
$rec1 -> setHeight(30);
$rec1 -> setWidth(40);
echo $rec2 -> area();
?>
```

הפלט שנקבל הוא:

```
1200
```

כאשר מגדירים `class` ניתן לאתחל את המשתנים שאנו מגדירים בו. כאשר מאתחלים משתנה לא ניתן לאתחל בערך של ביטוי.

הדוגמא הבאה מציגה הגדרה של `class` שכל אובייקט שנוצר ממנו מתאר מעגל. כל אובייקט שנוצר ממנו כולל את המשתנה `radius` אשר מאותחל בערך `8` (ברירת מחדל).

```
class Circle
{
	var $radius = 8;
	function details()
	{
		echo "radius=";
		echo $this->radius;
	}
}
```

הורשה

ניתן להגדיר מחלקה שיורשת ממחלקה אחרת ובכך לגרום לשתי תוצאות:

1. כל המשתנים שהוגדרו במחלקה המורישה יימצאו גם בכל אובייקט שיווצר מהמחלקה המורישה וגם בכל אובייקט שיווצר מהמחלקה היורשת.
2. כל אחת מהמתודות שהוגדרו במחלקה המורישה ניתן יהיה להפעילה גם על כל אובייקט מהמחלקה המורישה וגם על כל אובייקט מהמחלקה היורשת.

הגדרת מחלקה שיורשת ממחלקה אחרת נעשית באמצעות המילה השמורה `extends` על פי המוצג בדוגמא הבאה:

```
class Person
{
	...
}

class Student extends Person
{
	...
}
```

כל אובייקט שיווצר מהמחלקה `Student` יכלול בתוכו את כל אחד מהמשתנים שהוגדרו במחלקה `Student` וגם את כל אחד מהמשתנים שהוגדרו במחלקה `Person`. כמו כן, כל מתודה שהוגדרה במחלקה `Person` ניתן יהיה להפעיל אותה גם על כל אובייקט מהמחלקה `Person` וגם על כל אובייקט מהמחלקה `Student`. דוגמת הקוד הבאה מציגה את יצירתו של אובייקט מטיפוס `Student`, מחלקה אשר יורשת מהמחלקה `Person`.

```
<?php
class Person
{
	var $name;
	var $id;
	function setName($str)
	{
		$this->name=$str;
	}
	function setId($val)
	{
		$this->id=$val;
	}
}
```

```
class Student extends Person
{
	var $average;
	function setAverage($val)
	{
		$this->average=$val;
	}
	function printDetails()
	{
		echo $this->name." ".$this->id." ".$this->average;
	}
}

$ob = new Student();
$ob->setId(123);
$ob->setName("David");
$ob->setAverage(88);
$ob->printDetails();
?>
```

הפלט של התכנית הוא:

```
David 123 88
```

כאשר מגדירים מחלקה שיורשת ממחלקה אחרת ניתן להגדיר מחדש מתודות שכבר הגיעו בהורשה ובכך לדרוס את הגירסאות שהגיעו בהורשה. המשמעות היא שכאשר תופעל מתודה על אובייקט הגירסה שתופעל תהיה הגירסה הקרובה ביותר בהגדרתה לטיפוס של האובייקט. כך למשל, אם מוגדרת המחלקה `Student` כמחלקה שיורשת מהמחלקה `Person` והמתודה `printDetails` מוגדרת גם במחלקה `Person` וגם במחלקה `Student` ואנו קוראים להפעלתה על אובייקט מטיפוס `Student` אז הגירסה של `printDetails` שתופעל היא הגירסה שהוגדרה במחלקה `Student`.

דוגמא:

```
<?php
class Person
{
	var $name;
	var $id;
	function setName($str)
	{
```

```
		$this->name=$str;
	}
	function setId($val)
	{
		$this->id=$val;
	}
	function printDetails()
	{
		echo "(person) ".$this->name." ".$this->id;
	}
}

class Student extends Person
{
	var $average;
	function setAverage($val)
	{
		$this->average=$val;
	}
	function printDetails()
	{
		echo "(student) ".$this->name." ".$this->id." ".$this->average;
	}
}

$ob = new Student();
$ob->setId(123);
$ob->setName("David");
$ob->setAverage(88);
$ob->printDetails();
?>
```

הפלט שנקבל למסך הוא:

```
(student) David 123 88
```

באמצעות 'parent::' ניתן לפנות להפעלתה של מתודה בגירסה שנדרסה לאחר שהגדרנו אותה מחדש. ניתן לעשות בזה שימוש לצורך הגדרתה של מתודה מחדש תוך שימוש בגירסה שנדרסה. הדוגמא הבאה מציגה זאת.

```
<?php
class Person
{
	var $name;
	var $id;
	function setName($str)
	{
		$this->name=$str;
	}
	function setId($val)
	{
		$this->id=$val;
	}
	function printDetails()
	{
		echo $this->name." ".$this->id;
	}
}
class Student extends Person
{
	var $average;
	function setAverage($val)
	{
		$this->average=$val;
	}
	function printDetails()
	{
		parent::printDetails();
		echo " ".$this->average;
	}
}

$ob = new Student();
$ob->setId(123);
$ob->setName("David");
```

```
$ob->setAverage(88);
$ob->printDetails();
?>
```

הפלט שנקבל הוא:

```
David 123 88
```

בדומה לשימוש ב-::parent ניתן גם להשתמש ב-:: בלבד (תוך ציון שם של מחלקה מסויימת) ובדרך זו להורות על הפעלתה של מתודה מסויימת בגירסה שהוגדרה במחלקה מסויימת. הדוגמא הבאה מציגה זאת.

```
<?php
class Aaa
{
	function doSomething()
	{
		echo "a something";
	}
}
class Bbb extends Aaa
{
	function doSomething()
	{
		parent::doSomething();
	}
}
class Ccc extends Aaa
{
	function doSomething()
	{
		Aaa::doSomething();
	}
}
$ob_b = new Bbb();
$ob_b->doSomething();
?>
```

המילה השמורה $this מאפשרת לפנות לאובייקט הנוכחי. היא מכילה את ה-reference של האובייקט הנוכחי. כאשר רוצים לפנות למשתנה מסויים בתוך האובייקט הנוכחי או כאשר רוצים לקרוא להפעלת מתודה מסויימת על האובייקט הנוכחי יש להשתמש במילה השמורה $this בצירוף האופרטור <-.

הדוגמא הבאה מציגה זאת.

```
<?php
class Star
{
	function print_star()
	{
		echo "x";
	}
	function print_line($num)
	{
		for($i=1; $i<=$num; $i++)
		{
			$this -> print_star();
		}
	}
}
$rec = new Star();
$rec->print_line(4);
?>
```

ה-constructor וה-destructor

כאשר יוצרים אובייקט חדש מופעל ה-constructor. ה-constructor היא פונקציה מיוחדת שמופעלת פעם אחת במהלך חייו של אובייקט. היא מופעלת בעת יצירתו. ה-constructor משמש לאיתחול משתניו של האובייקט. ניתן לשלוח ל-constructor ערכים כדי שיעשה בהם שימוש לצורך איתחולו של האובייקט החדש.

ה-constructor היא מתודה מיוחדת בעלת השם __constructor.

הדוגמא הבאה מציגה הגדרה של מחלקה בשם Star עם constructor אחד מופעל בעת יצירת אובייקט ממנה.

```
<?php
	class Star
	{
		var $tav;
		function __construct()
		{
			$this->tav = '*';
		}
		function print_star()
		{
			echo $this->tav;
		}
		function print_line($num)
		{
			for($i=1; $i<=$num; $i++)
			{
				$this -> print_star();
			}
		}
	}
	$rec = new Star();
	$rec->print_line(4);
?>
```

הפלט שנקבל למסך הוא:

```
****
```

ניתן להגדיר את ה-constructor גם עם שם שזהה לשם של המחלקה. הדוגמא הבאה מציגה זאת.

```
<?php
	class Star
	{
		var $tav;
		function Star()
		{
			$this->tav = '*';
		}
		function print_star()
		{
			echo $this->tav;
		}
		function print_line($num)
		{
			for($i=1; $i<=$num; $i++)
			{
				$this -> print_star();
			}
		}
	}
	$rec = new Star();
	$rec->print_line(4);
?>
```

ה-destructor היא פונקציה מיוחדת שמופעלת על האובייקט רגע לפני שהוא מסיים את חייו. אובייקט מסיים את חייו כאשר כל ה-reference אליו כבר לא מוחזק באף משתנה. ה-destructor שימושי לביצוע פעולות שמשמעותן שיחרור של משאבים שתופסים זיכרון (שיחרור חיבור לבסיס נתונים, מחיקת קבצים וכדומה). ל-destructor שם מיוחד: __destructor.

הדוגמא הבאה מציגה בפעולת מתי מופעל ה-constructor ומתי מופעל ה-destructor.

```
<?php
class Star
{
	var $tav;
	function __construct()
	{
```

```
            echo __METHOD__."<BR>";
      }
      function __destruct()
      {
            echo __METHOD__."<BR>";
      }
}
new Star();
?>
```

הפלט שנקבל הוא:

```
Star::__construct
Star::__destruct
```

הרשאות גישה

כאשר מגדירים בתוך מחלקה מתודות ומשתנים ניתן לקבוע לכל אחד מהם הרשאת גישה. הרשאות הגישה האפשריות הן:

public
ניתן לגשת למשתנה/מתודה מכל מקום. זו ברירת המחדל.

private
ניתן לגשת למשתנה/מתודה רק בגבולות המחלקה שבה הם מוגדרים.

protected
ניתן לגשת למשתנה/מתודה רק בגבולות המחלקה שבה הם מוגדרים או בגבולותיה של מחלקה אחרת שיורשת ממנה.

ברוב המקרים אנחנו נשתמש בהרשאת הגישה private בהגדרתם של משתנים במחלקה כדי להבטיח שהגישה אליהם תיעשה באופן עקיף באמצעות הפעלת מתודה אשר גם תבטיח שערכו של המשתנה יישאר חוקי.

הדוגמא הבאה כוללת הגדרה של המחלקה Rectangle ובתוכה מוגדרים המשתנים width ו-height עם הרשאת הגישה private כדי להבטיח שכל ניסיון לשנות את הערכים שלהם ייעשה אך ורק באמצעות מתודות אשר בודקות במהלך פעולתן את הערך שמנסים להכניס כדי להבטיח שלא יוכנס ערך שלילי או אפס.

```
<?php
class Rectangle
{
	private $width;
	private $height;
	function area()
	{
		return $this->width * $this->height;
	}
	function setWidth($val)
	{
		if($val>0)
		{
			$this->width = $val;
		}
	}
	function setHeight($val)
```

```
	{
		if($val>0)
		{
			$this->height = $val;
		}
	}
}

$rec1 = new Rectangle();
$rec2 = $rec1;
$rec1 -> setHeight(30);
$rec1 -> setWidth(40);
echo $rec2 -> area();

?>
```

המילה השמורה final

באמצעות הוספת המילה השמורה final להגדרה של class אנו יכולים להבטיח שלא ניתן יהיה לרשת ממנה. במניעת האפשרות לרשת ממחלקה שאנחנו מגדירים אנחנו יכולים להבטיח את ההתנהגות של אובייקטים מהטיפוס שהגדרנו ולמנוע כל ניסיון לשנותה באמצעות הגדרת class חדש שמתבסס על ה-class שהגדרנו.

באמצעות הוספת המילה השמורה final להגדרה של method אנו יכולים להבטיח שלא ניתן יהיה לעשות לה overriding. בדרך זו אנחנו יכולים להבטיח שלא ניתן יהיה לשנות התנהגותה של מתודה מסויימת באמצעות overriding.

משתנים ומתודות סטטיים

באמצעות הוספת המילה השמורה static להגדרה של משתנה, המשתנה נהפך למשתנה סטטי. משתנה סטטי הוא משתנה תופס בזיכרון מקום אחד בלבד. הוא איננו נמצא בתוך האובייקט והוא מחזיק ערך שמתאר את המחלקה כולה במקום ערך שמתאר אובייקט מסויים (כגון width ו-height שחוזרים על עצמם בכל אובייקט ובכל אובייקט הם מחזיקים ערכים ספציפיים לאותו אובייקט).

פניה למשתנה סטטי נעשית באמצעות שם המחלקה בתוספת האופרטור '::' ושם המשתנה הסטטי מייד אחריו. להבדיל משימוש במשתנה רגיל שמוגדר במחלקה כאשר משתמשים במשתנה סטטי לא משמיטים את סימן ה-$.

באמצעות הוספת המילה השמורה static להגדרה של מתודה, המתודה נהפכת למתודה סטטית. כאשר קוראים להפעלתה של מתודה סטטית היא איננה פועלת על אובייקט מסויים.

קריאה להפעלת מתודה סטטית נעשית באמצעות שם המחלקה בתוספת האופרטור '::' ושם המתודה הסטטית מייד אחריו.

הדוגמא הבאה מציגה class אשר כולל בתוכו שתי מתודות סטטיות ומשתנה סטטי יחיד.

```
<?php
class Utils
{
	static function sum($a,$b)
	{
		return $a+$b;
	}
	static function multiply($a,$b)
	{
		return $a*$b;
	}
	public static $PI = 3.14;
}
echo Utils::sum(4,5);
echo "<BR>";
echo Utils::multiply(4,5);
echo "<BR>";
echo Utils::$PI;
?>
```

ניתן לכלול בהגדרה של class את ההגדרה של constants, קבועים שיכולים לשמש אותנו בהקשר של אותו class. המעמד של constant דומה למעמד של משתנה סטטי. בשני המקרים מדובר במשתנה שתופס מקום אחד בזיכרון. משתנה שאיננו חוזר על עצמו בכל אחד מהאובייקטים שנוצרים מה-class. משתנה שנשמר בזיכרון באותו מקום שבו שמורה ההגדרה של ה-class.

כדי לייצור constant יש להשתמש במילה השמורה const לפני השם של הקבוע שרוצים לייצור. כמו כן, בשורה שבה יוצרים קבוע יש גם לאתחל אותו.

כדי לגשת ל-constant שהגדרנו יש לרשום את שם ה-class בצירוף האופרטור '::' ושם ה-constant מייד אחריו.

הדוגמא הבאה מציגה שימוש בסיסי ב-class constant.

```
<?php
class Countries
{
      const US = "United States";
      const UK = "United Kingdom";
      const IL = "Israel";
      const RU = "Russia";
      const IT = "Italy";
}

echo "<BR>".Countries::US;
echo "<BR>".Countries::UK;
echo "<BR>".Countries::IL;
echo "<BR>".Countries::RU;
?>
```

הפלט שמתקבל הוא:

```
United States
United Kingdom
Israel
Russia
```

מחלקה אבסטרקטית

מחלקה אבסטרקטית היא מחלקה שלא ניתן לייצור ממנה אובייקטים כיוון שהיא כוללת מתודה אבסטרקטית אחת (או יותר). מתודה אבסטרקטית היא מתודה ללא גוף (כותרת בלבד). בשורת הכותרת שמגדירה מחלקה אבסטרקטית יש להוסיף את המילה השמורה abstract. בשורת הכותרת של מתודה אבסטרקטית גם יש להוסיף את המילה השמורה abstract.

כדי להשתמש במחלקה אבסטרקטית יש תחילה להגדיר מחלקה חדשה שיורשת ממנה ובמחלקה החדשה יש לעשות overriding למתודה (או המתודות) האבסטרקטית/יות שמגיעות ממנה בהורשה כדי שנוכל לייצור ממנה אובייקטים.

הדוגמא הבאה כוללת הגדרה של המחלקה האבסטרקטית Shape ושל מחלקות שיורשות ממנה.

```
<?php
abstract class Shape
{
	abstract function area();
}
class Rectangle extends Shape
{
	private $width;
	private $height;
	public function __construct($wval,$hval)
	{
		$this->width = $wval;
		$this->height = $hval;
	}
	public function area()
	{
		return $this->width * $this->height;
	}
}
class Circle extends Shape
{
	private $radius;
	public function __construct($num)
	{
		$this->radius = $num;
	}
```

```
	public function area()
	{
		return $this->radius * $this->radius * 3.14;
	}
}
$rec = new Rectangle(5,2);
echo $rec->area();
echo "<BR>";
$circ = new Circle(4);
echo $circ->area();
echo "<BR>";
?>
```

הגדרת interface

בדומה ל-class, ניתן להגדיר interface. במקום להשתמש במילה השמורה class משתמשים במילה השמורה interface. בתוך הגדרה של interface ניתן להגדיר מתודות אבסטרקטיות בלבד. לא ניתן להגדיר בתוך interafce מתודות עם גוף, constructors, destructor וגם לא ניתן להגדיר משתנים.

ניתן להגדיר class ולציין שהוא מיישם interface מסויים. עושים זאת באמצעות המילה השמורה implements. באמצעות interfaces ניתן להגדיר בנפרד את הפונקציונליות הנדרשת ממחלקות שונות ולהניח לכל מחלקה ליישם אותה באופן שונה.

להבדיל מ-class שיכול לרשת מ-class אחד בלבד, ניתן להגדיר class שמיישם מספר interfaces.

```
class Something implements Driveable, Cloneable, Printable
{
	..
	..
}
```

הדוגמא הבאה כוללת הגדרה של interface אשר כולל מתודה אחת ומספר classes אשר מיישמים אותו.

```
<?php

interface Helloable
{
	function sayHello();
}

class Cat implements Helloable
{
	function sayHello()
	{
		echo "miau miau";
	}
}

class Dog implements Helloable
{
	function sayHello()
```

```
	{
		echo "hau hau";
	}
}

class Bird implements Helloable
{
	function sayHello()
	{
		echo "tsvits tsvits";
	}
}

$cat = new Cat();
$cat->sayHello();

echo "<br>";

$dog = new Dog();
$dog->sayHello();

echo "<br>";

$bird = new Bird();
$bird->sayHello();

?>
```

הפלט שנקבל הוא:

```
miau miau
hau hau
tsvits tsvits
```

האופרטור instanceof

באמצעות האופרטור instanceof ניתן לבדוק אם אובייקט נתון מטיפוס מסויים. משתמשים בו באופן הבא:

reference לאובייקט `instanceof` שם של class או שם של interface

אופרטור זה מחזיר true בכל אחד משלושת המקרים הבאים:

1. האובייקט שאנו בודקים נוצר מהמחלקה ששמה מצויין מימין לאופרטור.
2. האובייקט שאנו בודקים נוצר ממחלקה שיורשת ממחלקה ששמה מצויין מימין לאופרטור.
3. האובייקט שאנו בודקים נוצר ממחלקה שמיישמת את ה-interface ששמו מצויין.

הגדרת פרמטרים מטיפוס מסויים

החל מגירסה 5.0 של שפת התיכנות PHP ניתן להגדיר פרמטרים בפונקציה (גם בפונקציה רגילה וגם במתודה) כפרמטרים מטיפוס מסויים (שם של מחלקה או שם של interface) ובכך לחייב כי בכל הפעלה של הפונקציה הערך שיישלח אל אותו פרמטר יהיה reference לאובייקט מאותו טיפוס (זה יכול להיות אובייקט שנוצר מהמחלקה ששמה צויין כטיפוס הפרמטר או אובייקט שנוצר ממחלקה שיורשת ממנה או אובייקט שנוצר ממחלקה שמיישמת את ה-interface, במקרה שהטיפוס שצויין הוא שמו של interface).

בדוגמת הקוד הבאה מוגדרת המחלקה Line תוך שהיא כוללת הגדרה של פונקציות שהפרמטרים שלהם הם מטיפוס מחלקה אחרת.

```
<?php
class Line
{
	var $p1, $p2;
	function Line(Point $ob_1, Point $ob_2)
	{
		$this->setP1($ob_1);
		$this->setP2($ob_2);
	}

	function setP1(Point $ob)
	{
		$this->p1 = $ob;
	}
	function setP2(Point $ob)
	{
		$this->p2 = $ob;
	}
	function length()
	{
		return sqrt(pow($this->p1->y-$this->p2->y,2)
			+pow($this->p1->x-$this->p2->x,2));
	}
}
```

```
class Point
{
	var $x,$y;

	function Point($x_val,$y_val)
	{
		$this->x = $x_val;
		$this->y = $y_val;
	}
}

$line_1 = new Line(new Point(3,3),new Point(7,6));
echo $line_1->length();
?>
```

החל מגירסה 5.1 של שפת התיכנות PHP ניתן אף להגדיר פרמטר של מתודה כפרמטר שהערך שהוא מקבל לתוכו חייב להיות reference למערך (ניתן להתייחס לכל מערך כאל אובייקט ובדרך זו לדמיין שגם למערך יש reference). הדוגמא הבאה כוללת הגדרה של המחלקה Line אשר כוללת constructor עם פרמטר אחד מטיפוס array. כאשר יש קריאה להפעלתו יש לשלוח reference לאובייקט שהוא מערך.

```
<?php
class Line
{
	var $p1, $p2;
	function Line(array $vec)
	{
		$this->setP1(new Point($vec[0],$vec[1]));
		$this->setP2(new Point($vec[2],$vec[3]));
	}
	function setP1(Point $ob)
	{
		$this->p1 = $ob;
	}
	function setP2(Point $ob)
	{
		$this->p2 = $ob;
	}
```

```
	function length()
	{
		return sqrt(pow($this->p1->y-$this->p2->y,2)
			+pow($this->p1->x-$this->p2->x,2));
	}
}

class Point
{
	var $x,$y;

	function Point($x_val,$y_val)
	{
		$this->x = $x_val;
		$this->y = $y_val;
	}
}

$line_1 = new Line(array(3,3,7,6));
echo $line_1->length();
?>
```