



2-4 Développement des classes métiers Point, PointDeTrace, Trace, Utilisateur (en travail individuel)

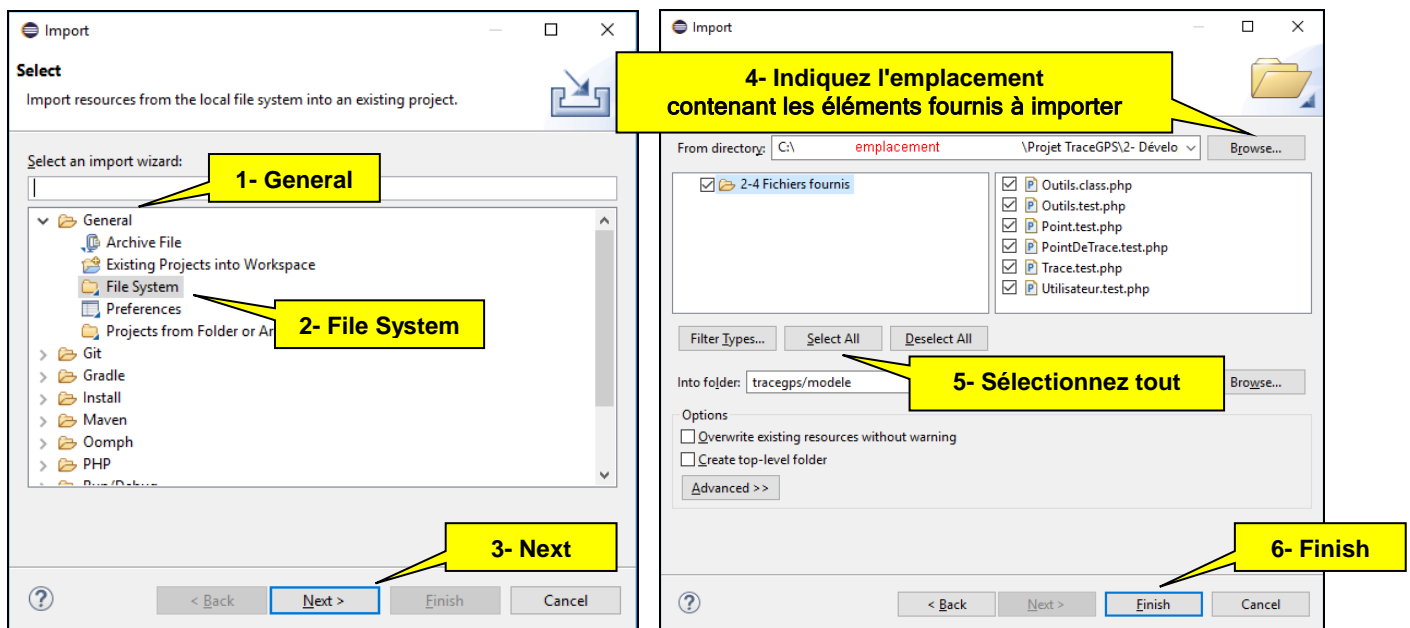
- 1- Importations des pages de test
- 2- Développement de la classe Point
- 3- Développement de la classe PointDeTrace
- 4- Développement de la classe Trace
- 5- Développement de la classe Utilisateur

1- Importations des pages de test

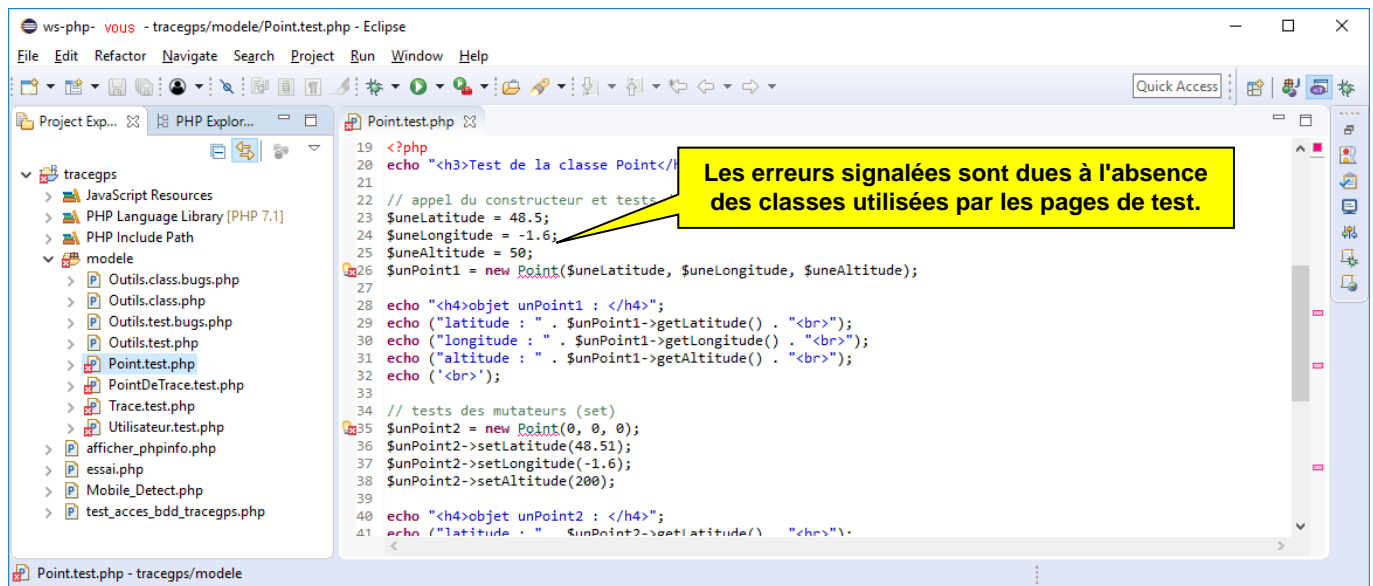
Nous allons importer dans le dossier **modele** les fichiers fournis du dossier "2-4 Fichiers fournis" :

- **Outils.class.php** : la classe Outils (sans bug cette fois)
- **Outils.test.php** : la page de test de la classe **Outils** (déjà utilisée)
- **Point.test.php** : la page de test de la classe **Point**
- **PointDeTrace.test.php** : la page de test de la classe **PointDeTrace**
- **Trace.test.php** : la page de test de la classe **Trace**
- **Utilisateur.test.php** : la page de test de la classe **Utilisateur**

Faire un clic droit **sur le dossier modele** (dans le **PHP Explorer**) et choisir la commande **Import...** :



On constate dans cet écran que les fichiers fournis sont bien importés :



2- Développement de la classe Point

Voici le diagramme UML de la classe **Point** à créer (les **getters** sont indiqués mais pas les **setters** afin de ne pas encombrer le diagramme) :

Point	
- \$latitude	: double
- \$longitude	: double
- \$altitude	: double
- getDistanceBetween ()	: double
+ __construct ()	: void
+ getLatitude ()	: double
+ getLongitude ()	: double
+ getAltitude ()	: double
+ getDistance(point1, point2) ()	: double
+ toString ()	: String

Création du fichier :

Pour créer la classe **Point** dans le dossier **modele**, faire un clic droit sur le dossier **modele**, choisir la commande **New / PHP File** et donner le nom du fichier (**Point.class.php**).

Codage :

Commencer par créer la structure de la classe et un entête documentaire :

```
<?php
// Projet TraceGPS
// fichier : modele/Point.class.php
// Rôle : la classe Point représente un point géographique
// Dernière mise à jour : 9/7/2019 par dP

class Point
{
} // fin de la classe Point

// ATTENTION : on ne met pas de balise de fin de script pour ne pas prendre le risque
// d'enregistrer d'espaces après la balise de fin de script !!!!!!!!!!!!!
```

Ajouter les 3 **attributs** avec le mode d'accès **protected** pour les rendre accessibles à la classe fille :

```
class Point
{
    // -----
    // ----- Attributs protégés de la classe -----
    // -----
    // protected au lieu de private car cette classe fera l'objet d'un héritage
    protected $latitude;           // latitude
    protected $longitude;          // longitude
    protected $altitude;           // altitude
}
```

A la suite des attributs, ajouter le **constructeur** (son nom commence par 2 underscores) :

```
// -----
// ----- Constructeur -----
// -----
public function __construct($uneLatitude, $uneLongitude, $uneAltitude) {
    $this->latitude = $uneLatitude;
    $this->longitude = $uneLongitude;
    $this->altitude = $uneAltitude;
}
```

Remarquez la notation `$this` qui représente l'instance créée par le constructeur et la flèche qui correspond au point du C# ou du



On aurait pu également nommer le constructeur en lui donnant le nom de la classe (comme en C# ou en Java) ; mais les dernières versions de PHP préconisent d'utiliser **__construct**.



Remarquez également qu'on ne peut pas écrire plusieurs constructeurs (comme en C# ou Java) car la surcharge de méthodes n'existe pas en PHP.

A la suite du constructeur, ajouter les **getters** et les **setters** :

```
// -----
// ----- Getters et Setters -----
// -----
public function getLatitude()      {return $this->latitude;}
public function setLatitude($uneLatitude) {$this->latitude = $uneLatitude;}

public function getLongitude()     {return $this->longitude;}
public function setLongitude($uneLongitude) {$this->longitude = $uneLongitude;}

public function getAltitude()      {return $this->altitude;}
public function setAltitude($uneAltitude) {$this->altitude = $uneAltitude;}
}
```

A la suite des getters et des setters, ajouter la méthode d'instance **toString** (que nous trouverons dans toutes les classes) qui sera surtout utile pour les tests de la classe :

```
// -----
// ----- Méthodes d'instances -----
// -----
// Fournit une chaine contenant toutes les données de l'objet
public function toString() {
    $msg = "latitude : " . $this->latitude . "<br>";
    $msg .= "longitude : " . $this->longitude . "<br>";
    $msg .= "altitude : " . $this->altitude . "<br>";
    return $msg;
}
```

A la suite de la méthode d'instance **toString**, ajouter les 2 méthodes statiques (ou méthodes de classes) permettant de calculer la distance entre 2 points :

```
// ----- Méthodes statiques -----
// -----
// Méthode statique privée
// calcule la distance (en Km) entre 2 points géographiques passés avec 4 paramètres :
// $latitude1 : latitude point 1 (en degrés décimaux)
// $longitude1 : longitude point 1 (en degrés décimaux)
// $latitude2 : latitude point 2 (en degrés décimaux)
// $longitude2 : longitude point 2 (en degrés décimaux)
// fournit : la distance (en Km) entre les 2 points
private static function getDistanceBetween($latitude1, $longitude1, $latitude2, $longitude2) {
    if (abs($latitude1 - $latitude2) < 0.000001 && abs($longitude1 - $longitude2) < 0.000001) return 0;
    try
    {
        $a = pi() / 180;
        $latitude1 = $latitude1 * $a;
        $latitude2 = $latitude2 * $a;
        $longitude1 = $longitude1 * $a;
        $longitude2 = $longitude2 * $a;
        $t1 = sin($latitude1) * sin($latitude2);
        $t2 = cos($latitude1) * cos($latitude2);
        $t3 = cos($longitude1 - $longitude2);
        $t4 = $t2 * $t3;
        $t5 = $t1 + $t4;
        $rad_dist = atan(-$t5 / sqrt(-$t5 * $t5 + 1)) + 2 * atan(1);
        return ($rad_dist * 3437.74677 * 1.1508) * 1.6093470878864446;
    }
    catch (Exception $ex)
    {
        return 0;
    }
}

// Méthode statique publique
// calcule la distance (en Km) entre 2 points géographiques passés en paramètres :
// point1 : le premier point
// point2 : le second point
// fournit : la distance (en Km) entre les 2 points
public static function getDistance(Point $point1, Point $point2) {
    // A VOUS DE TROUVER LE CODE MANQUANT
    // (il faut appeler la méthode précédente)
}
```

Test de la classe :

Analyser le contenu de la page de test (**Point.test.php**) et afficher cette page dans un navigateur.

Analyser le résultat affiché pour voir si la classe **Point** est correcte.

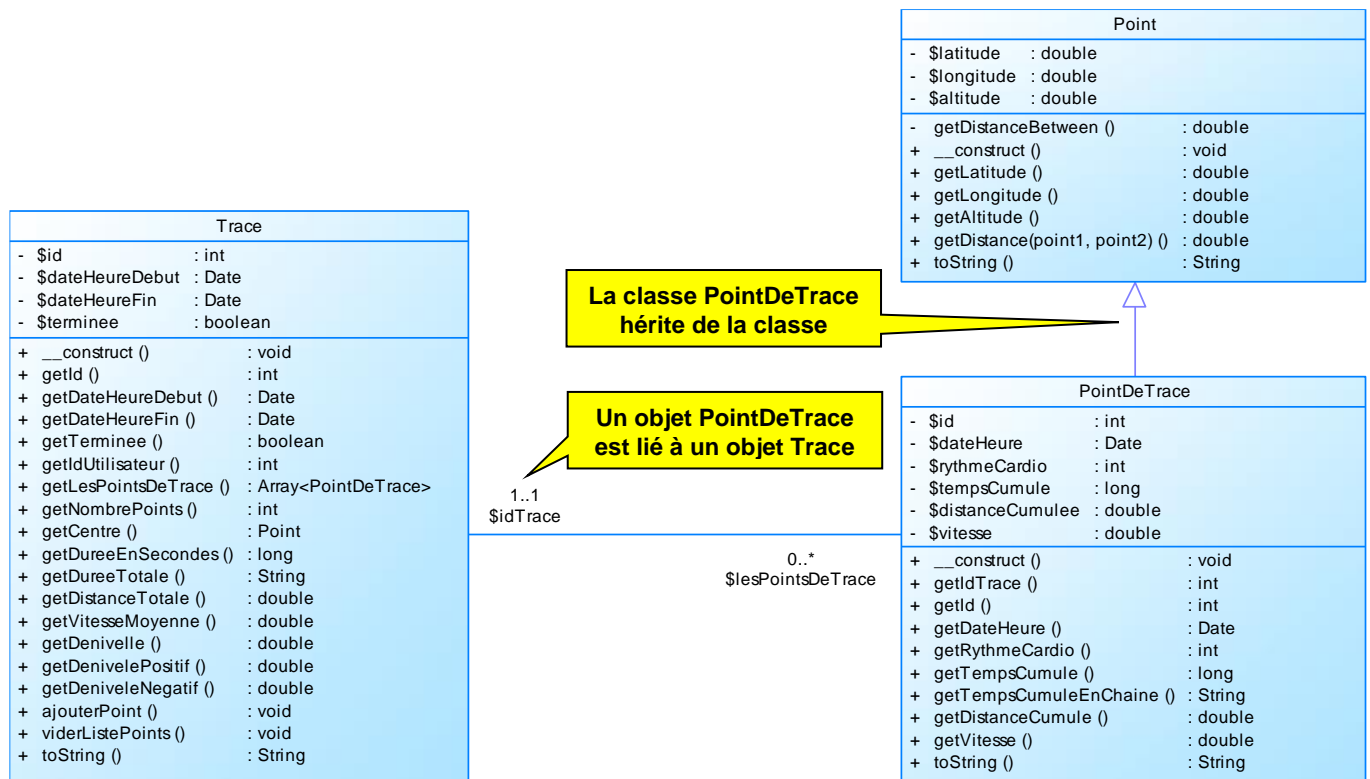
Pour connaître la distance entre 2 points géographiques, on peut utiliser le site suivant :

http://www.lexilogos.com/calcul_distances.htm

En cas d'erreur, corriger le code et retester.

3- Développement de la classe PointDeTrace

Voici le diagramme UML de la classe **PointDeTrace** à créer (les **getters** sont indiqués mais pas les **setters** afin de ne pas encombrer le diagramme) :



Création du fichier :

Pour créer la classe **PointDeTrace** dans le dossier **modele**, faire un clic droit sur le dossier **modele**, choisir la commande **New / PHP File** et donner le nom du fichier (**PointDeTrace.class.php**).

Codage :

Commencer par créer la structure de la classe et un entête documentaire :

```

<?php
// Projet TraceGPS
// fichier : modele/PointDeTrace.class.php
// Rôle : la classe PointDeTrace représente un point de passage sur un parcours
// Dernière mise à jour : 9/7/2021 par dP

include_once ('Point.class.php');

class PointDeTrace extends Point
{
} // fin de la classe PointDeTrace

// ATTENTION : on ne met pas de balise de fin de script pour ne pas prendre le risque
// d'enregistrer d'espaces après la balise de fin de script !!!!!!!!!!!!!
  
```

Inclure la classe Point

La classe PointDeTrace hérite de la classe

Ajouter les **attributs** avec le mode d'accès **private** :

```
class PointDeTrace extends Point
{
    // -----
    // ----- Attributs privés de la classe -----
    // -----

    private $idTrace;           // identifiant de la trace
    private $id;                // identifiant relatif du point dans la trace
    private $dateHeure;         // date et heure du passage au point
    private $rythmeCardio;      // rythme cardiaque (en bpm : battements par minute)
    private $tempsCumule;       // temps cumulé depuis le départ (en secondes)
    private $distanceCumulee;    // distance cumulée depuis le départ (en Km)
    private $vitesse;           // vitesse instantanée au point de passage (en Km/h)
}
```

A la suite des attributs, ajouter le **constructeur** :

```
// -----
// ----- Constructeur -----
// -----

// Constructeur avec 10 paramètres :
// $unIdTrace : identifiant de la trace
// $unId : identifiant relatif du point dans la trace
// $uneLatitude : latitude du point (en degrés décimaux)
// $uneLongitude : longitude du point (en degrés décimaux)
// $uneAltitude : altitude du point (en mètres)
// $uneDateHeure : heure de passage au point
// $unRythmeCardio : rythme cardiaque au passage au point
// $unTempsCumule : temps cumulé depuis le départ(en secondes)
// $uneDistanceCumulee : distance cumulée depuis le départ (en Km)
// $uneVitesse : vitesse instantanée, calculée entre le point précédent et le point suivant (en Km/h)
public function __construct($unIdTrace, $unID, $uneLatitude, $uneLongitude, $uneAltitude,
    $uneDateHeure, $unRythmeCardio, $unTempsCumule, $uneDistanceCumulee, $uneVitesse) {
    // appelle le constructeur de la classe mère avec 3 paramètres
    parent::__construct($uneLatitude, $uneLongitude, $uneAltitude);
    // initialise les nouveaux attributs
    // A VOUS DE TROUVER LE CODE MANQUANT
}
```

A la suite du constructeur, ajouter les **getters** et les **setters** :

```
// -----
// ----- Getters et Setters -----
// -----

public function getIdTrace() {return $this->idTrace;}
public function setIdTrace($unIdTrace) {$this->idTrace = $unIdTrace;}

public function getId() {return $this->id;}
public function setId($unId) {$this->id = $unId;}

public function getDateHeure() {return $this->dateHeure;}
public function setDateHeure($uneDateHeure) {$this->dateHeure = $uneDateHeure;}

public function getRythmeCardio() {return $this->rythmeCardio;}
public function setRythmeCardio($unRythmeCardio) {$this->rythmeCardio = $unRythmeCardio;}

public function getTempsCumule() {return $this->tempsCumule;}
public function setTempsCumule($unTempsCumule) {$this->tempsCumule = $unTempsCumule;}
```

```

public function getDistanceCumulee() {return $this->distanceCumulee;}
public function setDistanceCumulee($uneDistanceCumulee) {$this->distanceCumulee = $uneDistanceCumulee;}

public function getVitesse() {return $this->vitesse;}
public function setVitesse($uneVitesse) {$this->vitesse = $uneVitesse;}

```

A la suite des getters et des setters, ajouter les méthodes d'instance **toString** et **getTempsCumuleEnChaine** :

```

// ----- Méthodes d'instances -----
// Fournit une chaine contenant toutes les données de l'objet
public function toString() {
    $msg = "IdTrace : " . $this->getIdTrace() . "<br>";
    $msg .= "Id : " . $this->getId() . "<br>";
    $msg .= parent::toString();
    if ($this->dateHeure != null) {
        $msg .= "Heure de passage : " . $this->dateHeure . "<br>";
    }
    $msg .= "Rythme cardiaque : " . $this->rythmeCardio . "<br>";
    $msg .= "Temps cumule (s) : " . $this->tempsCumule . "<br>";
    $msg .= "Temps cumule (hh:mm:ss) : " . $this->getTempsCumuleEnChaine() . "<br>";
    $msg .= "Distance cumulée (Km) : " . $this->distanceCumulee . "<br>";
    $msg .= "Vitesse (Km/h) : " . $this->vitesse . "<br>";
    return $msg;
}

// Méthode fournissant le temps cumulé depuis le départ (sous la forme d'une chaine "hh:mm:ss")
public function getTempsCumuleEnChaine()
{
    // A VOUS DE TROUVER LE CODE MANQUANT

    return sprintf("%02d",$heures) . ":" . sprintf("%02d",$minutes) . ":" . sprintf("%02d",$secondes);
}

```

Test de la classe :

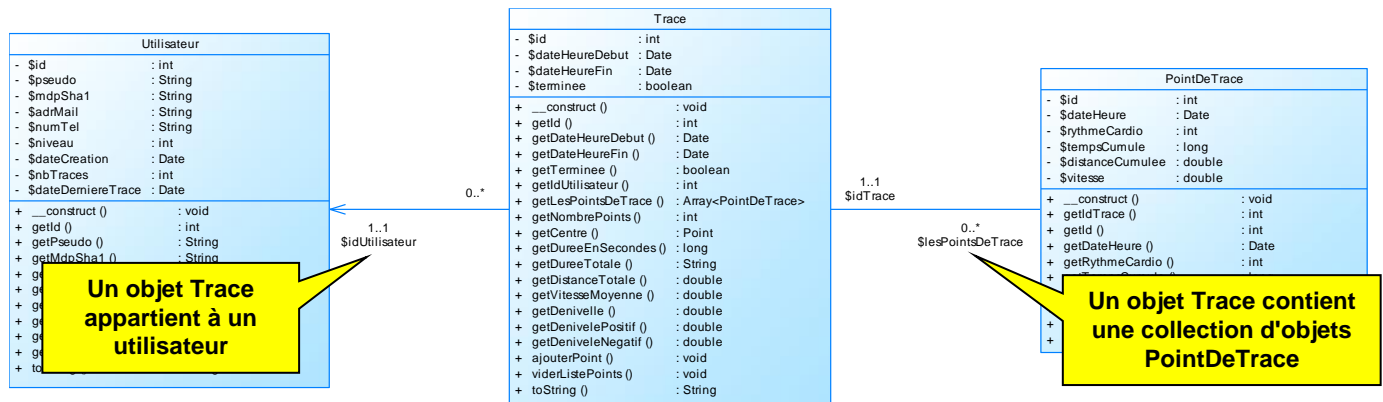
Analyser le contenu de la page de test (**PointDeTrace.test.php**) et afficher cette page dans un navigateur.

Analyser le résultat affiché pour voir si la classe **PointDeTrace** est correcte.

En cas d'erreur, corriger le code et retester.

4- Développement de la classe Trace

Voici le diagramme UML de la classe **Trace** à créer (les **getters** sont indiqués mais pas les **setters** afin de ne pas encombrer le diagramme) :



Informations sur les collections en PHP :

La déclaration de l'attribut représentant la collection :

```
private $lesPointsDeTrace; // la collection (array) des objets PointDeTrace formant la trace
```

La création (ou instanciation) de la collection :

```
$this->lesPointsDeTrace = array();
```

La taille de la collection est obtenue avec la fonction **sizeof()** :

```
$nbrePoints = sizeof($this->lesPointsDeTrace);
```

L'ajout d'un objet dans la collection se fait comme pour un tableau, mais sans indiquer de position :

```
$this->lesPointsDeTrace[] = $unPoint;
```

Le parcours des objets de la collection se fait avec une boucle **for** :

```
for ($i = 0; $i < sizeof($this->lesPointsDeTrace) ; $i++) {
    $lePoint = $this->lesPointsDeTrace[$i];
    .....
}
```

Création du fichier :

Pour créer la classe **Trace** dans le dossier **modele**, faire un clic droit sur le dossier **modele**, choisir la commande **New / PHP File** et donner le nom du fichier (**Trace.class.php**).

Codage de la classe :

Commencer par créer la structure de la classe et un entête documentaire :

```
<?php
// Projet TraceGPS
// fichier : modele/Trace.class.php
// Rôle : la classe Trace représente une trace ou un parcours
// Dernière mise à jour : 9/7/2021 par dP

include_once ('PointDeTrace.class.php');
class Trace
{
} // fin de la classe Trace

// ATTENTION : on ne met pas de balise de fin de script pour ne pas prendre le risque
// d'enregistrer d'espaces après la balise de fin de script !!!!!!!!!!!
```

Inclure la classe PointDeTrace

Ajouter les **attributs** avec le mode d'accès **private** :

```
class Trace
{
    // -----
    // ----- Attributs privés de la classe -----
    // -----

    private $id;                // identifiant de la trace
    private $dateHeureDebut;     // date et heure de début
    private $dateHeureFin;      // date et heure de fin
    private $terminee;          // true si la trace est terminée, false sinon
    private $idUser;            // identifiant de l'utilisateur ayant créé la trace
    private $lesPointsDeTrace;  // la collection (array) des objets PointDeTrace formant la trace
}
```

A la suite des attributs, ajouter le **constructeur** (ne pas oublier de créer la collection) :

```
// -----
// ----- Constructeur -----
// -----

public function __construct($unId, $uneDateHeureDebut, $uneDateHeureFin, $terminee, $unIdUtilisateur) {
    // A VOUS DE TROUVER LE CODE MANQUANT
}
```

A la suite du constructeur, ajouter les **getters** et les **setters** :

```
// -----
// ----- Getters et Setters -----
// -----

public function getId() {return $this->id;}
public function setId($unId) {$this->id = $unId;}

public function getDateHeureDebut() {return $this->dateHeureDebut;}
public function setDateHeureDebut($uneDateHeureDebut) {$this->dateHeureDebut = $uneDateHeureDebut;}

public function getDateHeureFin() {return $this->dateHeureFin;}
public function setDateHeureFin($uneDateHeureFin) {$this->dateHeureFin = $uneDateHeureFin;}

public function getTerminee() {return $this->terminee;}
public function setTerminee($terminee) {$this->terminee = $terminee;}

public function getIdUtilisateur() {return $this->idUser;}
}
```

```

public function setIdUtilisateur($unIdUtilisateur) {$this->idUtilisateur = $unIdUtilisateur;}
public function getLesPointsDeTrace() {return $this->lesPointsDeTrace;}
public function setLesPointsDeTrace($lesPointsDeTrace) {$this->lesPointsDeTrace = $lesPointsDeTrace;}

```

A la suite des getters et des setters, ajouter la méthode d'instance **toString** :

```

// Fournit une chaine contenant toutes les données de l'objet
public function toString() {
    $msg = "Id : " . $this->getId() . "<br>";
    $msg .= "Utilisateur : " . $this->getIdUtilisateur() . "<br>";
    if ($this->getDateHeureDebut() != null) {
        $msg .= "Heure de début : " . $this->getDateHeureDebut() . "<br>";
    }
    if ($this->getTerminee()) {
        $msg .= "Terminée : Oui <br>";
    }
    else {
        $msg .= "Terminée : Non <br>";
    }
    $msg .= "Nombre de points : " . $this->getNombrePoints() . "<br>";
    if ($this->getNombrePoints() > 0) {
        if ($this->getDateHeureFin() != null) {
            $msg .= "Heure de fin : " . $this->getDateHeureFin() . "<br>";
        }
        $msg .= "Durée en secondes : " . $this->getDureeEnSecondes() . "<br>";
        $msg .= "Durée totale : " . $this->getDureeTotale() . "<br>";
        $msg .= "Distance totale en Km : " . $this->getDistanceTotale() . "<br>";
        $msg .= "Dénivelé en m : " . $this->getDenivele() . "<br>";
        $msg .= "Dénivelé positif en m : " . $this->getDenivelePositif() . "<br>";
        $msg .= "Dénivelé négatif en m : " . $this->getDeniveleNegatif() . "<br>";
        $msg .= "Vitesse moyenne en Km/h : " . $this->getVitesseMoyenne() . "<br>";
        $msg .= "Centre du parcours : " . "<br>";
        $msg .= " - Latitude : " . $this->getCentre()->getLatitude() . "<br>";
        $msg .= " - Longitude : " . $this->getCentre()->getLongitude() . "<br>";
        $msg .= " - Altitude : " . $this->getCentre()->getAltitude() . "<br>";
    }
    return $msg;
}

```



Pour coder les méthodes demandées, vous pouvez bien sûr reprendre le code C# de la classe **Trace** que vous avez créée lors du TP 2.2.

Il faudra bien sûr transformer le code C# en code PHP !

A la suite de la méthode **toString**, coder les méthodes suivantes :

1. la méthode d'instance publique **getNombrePoints** qui fournit le nombre de points de la collection (ce nombre est fourni par la fonction **sizeof()**)
2. la méthode d'instance publique **getCentre** qui fournit un objet **Point** correspondant au centre du parcours ; le calcul nécessite de parcourir tous les points pour déterminer les latitudes mini et maxi, ainsi que les longitudes mini et maxi :
 - la latitude du centre sera la moyenne de la latitude mini et la latitude maxi
 - la longitude du centre sera la moyenne de la longitude mini et la longitude maxi
3. la méthode d'instance publique **getDenivele** qui fournit l'écart d'altitude entre le point le plus bas et le point le plus haut du parcours ; le calcul nécessite de parcourir tous les points pour déterminer les altitudes mini et maxi.
4. la méthode d'instance publique **getDureeEnSecondes** qui fournit le temps cumulé au dernier point (ou la valeur 0 si la collection est vide).
5. la méthode d'instance publique **getDureeTotale** qui fournit l'écart de temps (sous forme d'une chaîne au format "**hh:mm:ss**") entre le passage au premier point et le passage au dernier point. Cette chaîne est calculée en décomposant le résultat fourni par **getDureeEnSecondes**.
6. la méthode d'instance publique **getDistanceTotale** qui fournit la distance cumulée (en **km**) au dernier point (ou la valeur 0 si la collection est vide).
7. la méthode d'instance publique **getDenivelePositif** qui fournit le cumul des écarts d'altitude (en **m**) de tous les couples de points successifs et montants.
8. la méthode d'instance publique **getDeniveleNegatif** qui fournit le cumul des écarts d'altitude (en **m**) de tous les couples de points successifs et descendants.
9. la méthode d'instance publique **getVitesseMoyenne** qui fournit la vitesse moyenne (en **km/h**) sur la totalité du parcours. Cette vitesse est calculée à partir des résultats fournis par **getDistanceTotale** (en km) et **getDureeEnSecondes** (en secondes). Attention à bien éviter une division par zéro...
10. la méthode d'instance publique **ajouterPoint** qui permet d'ajouter un objet **PointDeTrace** (passé en paramètre) à la collection.
Avant d'ajouter le point, on calculera ses attributs **\$distanceCumulee**, **\$tempsCumule** et **\$vitesse** à partir du point précédent (qui est actuellement le dernier point de la collection). Si le point à ajouter est le premier du parcours, il n'y a pas de point précédent, et ces 3 attributs sont initialisés à 0.
11. la méthode d'instance publique **viderListePoints** qui permet de vider la collection. Il suffit de créer une nouvelle collection.

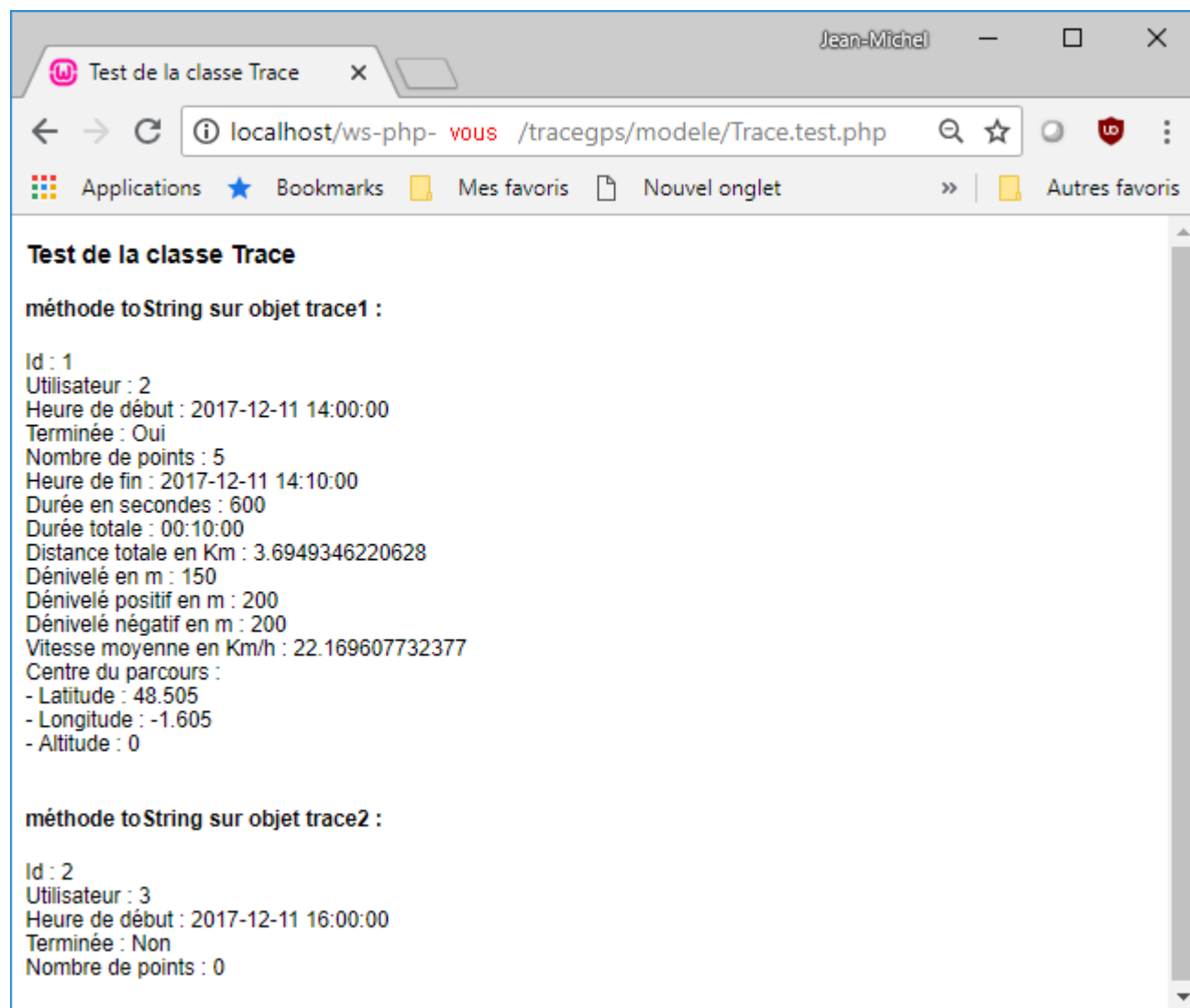
Test de la classe :

Analyser le contenu de la page de test (**Trace.test.php**) et afficher cette page dans un navigateur.

Analyser le résultat affiché pour voir si la classe **Trace** est correcte.

En cas d'erreur, corriger le code et retester.

Le résultat attendu :



5- Développement de la classe Utilisateur

Voici le diagramme UML de la classe **Utilisateur** à créer (les **getters** sont indiqués mais pas les **setters** afin de ne pas encombrer le diagramme) :

Utilisateur	
- \$id	: int
- \$pseudo	: String
- \$mdpSha1	: String
- \$adrMail	: String
- \$numTel	: String
- \$niveau	: int
- \$dateCreation	: Date
- \$nbTraces	: int
- \$dateDerniereTrace	: Date
+ __construct ()	: void
+ getId ()	: int
+ getPseudo ()	: String
+ getMdpSha1 ()	: String
+ getAdrMail ()	: String
+ getNumTel ()	: String
+ getNiveau ()	: int
+ getDateCreation ()	: Date
+ getNbTraces ()	: int
+ getDateDerniereTrace ()	: Date
+ toString ()	: String

Création du fichier :

Pour créer la classe **Utilisateur** dans le dossier **modele**, faire un clic droit sur le dossier **modele**, choisir la commande **New / PHP File** et donner le nom du fichier (**Utilisateur.class.php**).

Codage de la classe :

Commencer par créer la structure de la classe et un entête documentaire :

```
<?php
// Projet TraceGPS
// fichier : modele/Utilisateur.class.php
// Rôle : la classe Utilisateur représente les utilisateurs de l'application
// Dernière mise à jour : 9/7/2021 par dP

include_once ('Outils.class.php');
class Utilisateur
{
} // fin de la classe Utilisateur

// ATTENTION : on ne met pas de balise de fin de script pour ne pas prendre le risque
// d'enregistrer d'espaces après la balise de fin de script !!!!!!!!!!!!!
```

Inclure la classe Outils

Ajouter les **attributs** avec le mode d'accès **private** :

```
class Utilisateur
{
    // -----
    // ----- Attributs privés de la classe -----
    // -----

    private $id;           // identifiant de l'utilisateur (numéro automatique dans la BDD)
    private $pseudo;       // pseudo de l'utilisateur
    private $mdpSha1;      // mot de passe de l'utilisateur (hashé en SHA1)
    private $adrMail;      // adresse mail de l'utilisateur
    private $numTel;       // numéro de téléphone de l'utilisateur
    private $niveau;       // niveau d'accès : 1 = utilisateur (pratiquant ou proche) 2 = administrateur
    private $dateCreation; // date de création du compte
    private $nbTraces;     // nombre de traces stockées actuellement
    private $dateDerniereTrace; // date de début de la dernière trace
}
```

A la suite des attributs, ajouter le **constructeur** :

```
// -----
// ----- Constructeur -----
// -----

public function __construct($unId, $unPseudo, $unMdpSha1, $uneAdrMail, $unNumTel, $unNiveau,
    $uneDateCreation, $unNbTraces, $uneDateDerniereTrace) {
    // A VOUS DE TROUVER LE CODE MANQUANT
    // Utilisez la classe Outils pour améliorer la forme du numéro de téléphone
}
```

A la suite du constructeur, ajouter les **getters** et les **setters** :

```
// -----
// ----- Getters et Setters -----
// -----

// A VOUS DE TROUVER LE CODE MANQUANT
// Utilisez la page de test fournie pour retrouver les noms des getters et des setters
```

A la suite des getters et des setters, ajouter la méthode d'instance **toString** :

```
// -----
// ----- Méthodes d'instances -----
// -----

public function toString() {
    $msg = 'id : ' . $this->id . '<br>';
    $msg .= 'pseudo : ' . $this->pseudo . '<br>';
    $msg .= 'mdpSha1 : ' . $this->mdpSha1 . '<br>';
    $msg .= 'adrMail : ' . $this->adrMail . '<br>';
    $msg .= 'numTel : ' . $this->numTel . '<br>';
    $msg .= 'niveau : ' . $this->niveau . '<br>';
    $msg .= 'dateCreation : ' . $this->dateCreation . '<br>';
    $msg .= 'nbTraces : ' . $this->nbTraces . '<br>';
    $msg .= 'dateDerniereTrace : ' . $this->dateDerniereTrace . '<br>';
    return $msg;
}
```

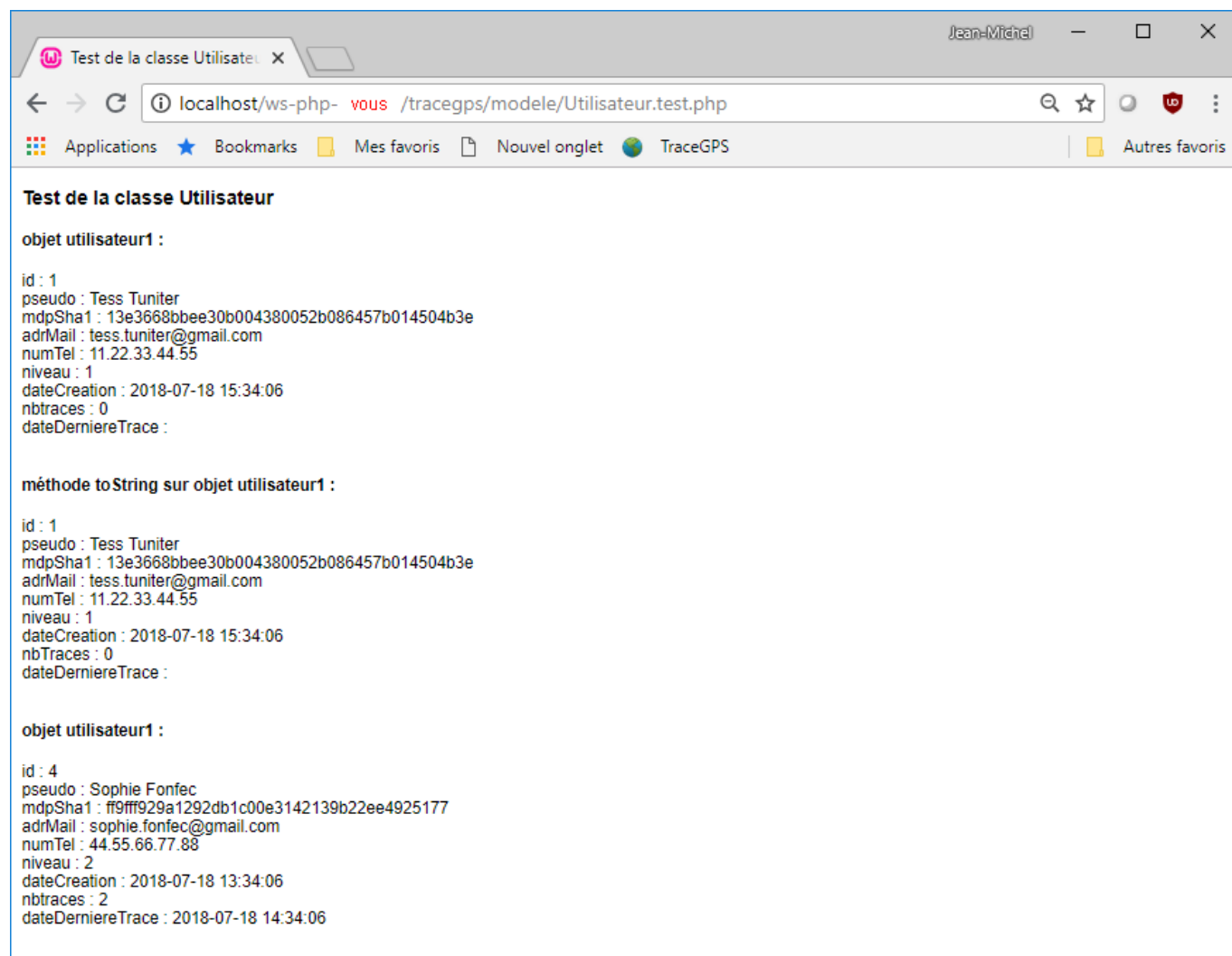
Test de la classe :

Analyser le contenu de la page de test (**Utilisateur.test.php**) et afficher cette page dans un navigateur.

Analyser le résultat affiché pour voir si la classe **Utilisateur** est correct.

En cas d'erreur, corriger le code et retester.

Le résultat attendu (en dehors des dates qui ne correspondront pas le jour du test) :



Test de la classe Utilisateur

objet utilisateur1 :

id : 1
pseudo : Tess Tuniter
mdpSha1 : 13e3668bbe30b004380052b086457b014504b3e
adrMail : tess.tuniter@gmail.com
numTel : 11.22.33.44.55
niveau : 1
dateCreation : 2018-07-18 15:34:06
nbtraces : 0
dateDerniereTrace :

méthode toString sur objet utilisateur1 :

id : 1
pseudo : Tess Tuniter
mdpSha1 : 13e3668bbe30b004380052b086457b014504b3e
adrMail : tess.tuniter@gmail.com
numTel : 11.22.33.44.55
niveau : 1
dateCreation : 2018-07-18 15:34:06
nbTraces : 0
dateDerniereTrace :

objet utilisateur1 :

id : 4
pseudo : Sophie Fonfec
mdpSha1 : ff9fff929a1292db1c00e3142139b22ee4925177
adrMail : sophie.fonfec@gmail.com
numTel : 44.55.66.77.88
niveau : 2
dateCreation : 2018-07-18 13:34:06
nbtraces : 2
dateDerniereTrace : 2018-07-18 14:34:06