Présentation

Le service Emplacements de la Mairie de P. est responsable de la gestion de l'occupation du domaine public par les établissements et les commerces de la ville. Cette occupation du domaine public se rapporte notamment à l'installation :

- de terrasses, permanentes ou non, pour les bars et les restaurants,
- d'étalages pour les commerces désirant exposer des produits à la vente.

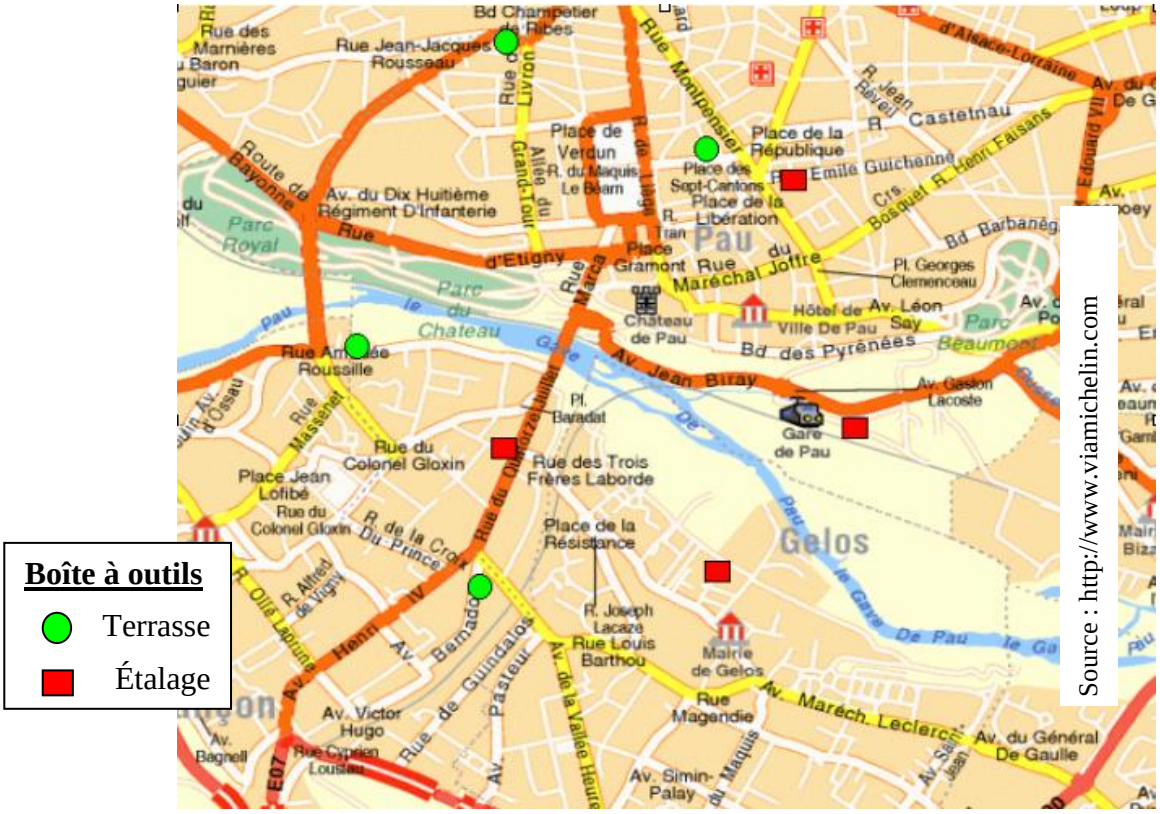
Qu'il s'agisse d'une terrasse ou d'un étalage, toute installation est soumise à une réglementation stricte définie par la loi et engendre une perception de droits de la part de la Mairie.

Le service Emplacements est dirigé par Monsieur Thalo.

Les dossiers qui suivent se rapportent à l'informatisation de certaines activités liées à la gestion des terrasses et des étalages. Pour des raisons de simplification, le contexte d'étude a dû être restreint et ne traduit donc pas fidèlement la réalité de la gestion de l'occupation du domaine public.

Dossier : Visualisation des emplacements occupant le domaine public

La Mairie souhaite pouvoir visualiser sur écran les plans des quartiers avec les différents emplacements loués, selon le modèle de la figure ci-dessous. Une application qui sera développée à l'aide d'un langage à objets est en cours de conception.



Les plans des quartiers sont stockés dans des fichiers images de type matriciel (bitmap). On distingue une terrasse par un rond vert et un étalage par un carré rouge. Ces deux symboles sont de dimension fixe.

Pour créer un nouvel emplacement, l'utilisateur clique sur l'objet de la boîte à outils souhaité (Terrasse ou Étalage) et le fait glisser sur le plan : une nouvelle instance de cet objet est alors créée, ses coordonnées sont mémorisées et un écran de saisie s'affiche.

Vous trouverez une partie des classes définies pour réaliser cette application en **annexe**. On s'intéresse à l'enregistrement d'un nouvel emplacement sur le plan.

TRAVAIL À FAIRE	
1	Expliquer pourquoi l'attribut `dimension` de la classe `Emplacement` est un attribut à portée classe.
2	Nommer et expliquer le (ou les) mécanisme(s) mis en œuvre pour la déclaration de la méthode `affiche()` dans les classes `Terrasse` et `Etalage`.
3	Écrire la méthode `supprimeEmplacement` de la classe `Plan`.

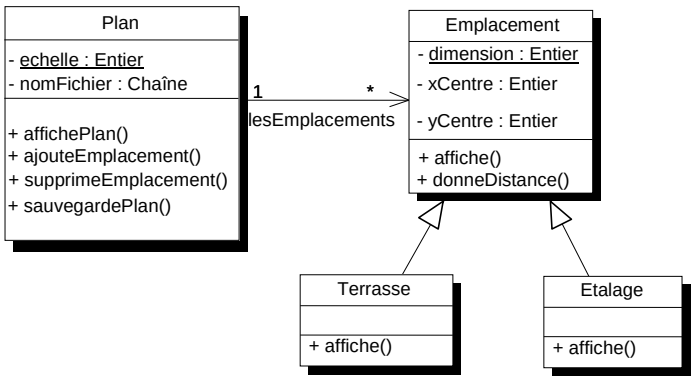
La méthode `ajouteEmplacement` a pour rôle d'ajouter un nouvel emplacement sur le plan. Cet ajout ne sera toutefois pas possible si l'emplacement est une terrasse située à moins de 50 mètres d'une autre terrasse. On suppose que l'emplacement à ajouter n'existe pas déjà dans la collection des emplacements rattachés au plan.

4	Écrire la méthode `ajouteEmplacement` de la classe `Plan`.

ANNEXE : Représentation des classes

Diagramme de classes :

Afin de ne pas alourdir le schéma, seuls figurent les membres (données et méthodes) significatifs. Les paramètres des méthodes ne sont pas présentés.



Commentaires :

Accessibilité :

Symbole – (moins) : membre privé

Symbole + (plus) : membre public

Un membre souligné est un membre à portée classe, appelé également membre de classe ou *static*.

Héritage :

Les classes *Terrasse* et *Etalage* héritent de la classe *Emplacement.*

Association entre Plan et Emplacement :

Représente l'ensemble des emplacements rattachés au plan.

Déclaration des classes correspondant au diagramme de classes

*Dans les fonctions ou procédures, les paramètres sont précédés de **e** pour « entrée »,*

classe Plan
// Attribut privé à portée classe

`echelle : Entier`
// Attributs privés

`nomFichier : Chaîne`

`lesEmplacements : Collection de Emplacement`
// Mémorise tous les emplacements placés sur le plan (terrasse ou étalage)

// Méthodes publiques

fonction `ajouteEmplacement(e unEmplacement : Emplacement) : Booléen`
// Ajoute le nouvel emplacement (terrasse ou étalage) dans la collection lesEmplacements. Elle renvoie une valeur booléenne permettant de savoir si l'ajout a été possible.

procédure `supprimeEmplacement(e unEmplacement : Emplacement)`
// supprime l'emplacement de la collection lesEmplacements. Elle vérifie au préalable l'existence de l'emplacement dans la collection.

procédure `affichePlan()`
// Affiche l'image bitmap du plan avec les emplacements (terrasse et étalage). On distingue une terrasse par un rond vert et un étalage par un carré rouge.

procédure `sauvegardePlan()`
// Sauvegarde le plan et ses emplacements dans le fichier

Fin classe Plan

classe Emplacement
// Cette classe hérite d'une méthode `estType(<nomClasse>) : Booléen` de la super-classe `Objet` permettant de savoir si l'objet invoquant cette méthode appartient à la classe dont le nom est passé en paramètre. Exemple d'utilisation : L'appel `monObjet.estType(maClasse)` renvoie `vrai` si `monObjet` est une instance de `maClasse, faux` sinon.

// Attribut privé à portée classe

`dimension : Entier` *// **diamètre** pour le cercle, largeur pour le carré*
// Attributs privés

`couleur : Entier`

`xCentre : Entier` *// abscisse du centre de l'emplacement sur le plan, cercle ou carré*

`yCentre : Entier` *// ordonnée du centre de l'emplacement sur le plan, cercle ou carré*
// Méthodes publiques

procédure `affiche()`

fonction `donneDistance(e unEmplacement : Emplacement) : Entier`

// Renvoie la distance en mètres séparant l'emplacement courant d'un autre emplacement passé en paramètre.

fin classe Emplacement

classe Terrasse hérite de Emplacement
// Méthode publique

procédure `affiche()` *// Affiche un cercle*

fin classe Terrasse

```
classe Etalage hérite de Emplacement
// Méthode publique
    procédure affiche() // Affiche un carré
fin classe Etalage
```

L'implémentation de la classe **Plan** utilise une classe technique **Collection** qui offre des services pour la gestion d'un ensemble d'objets.

```
classe Collection  // Méthodes publiques
    Fonction cardinal() : Entier  // Renvoie le nombre d' éléments de la collection
    Fonction existe(e unObjet : Objet) : Booléen
  // Teste si unObjet  existe dans la collection
    Fonction index(e unObjet : Objet) : Entier
  // Renvoie l'index de unObjet, le premier objet de la collection a pour index 1
    Fonction extraireObjet(e unIndex : Entier) : Objet
  // Retourne l'objet d'index unIndex
    Procédure ajouter(e unObjet : Objet)  // Ajoute un objet à la collection
    Procédure enlever(e unIndex : Entier) // Supprime l'objet d'index unIndex de la collection
    Procédure vider()      // Vide le contenu de la collection
Fin classe Collection
```