Mise en contexte

Vous devez concevoir un système de gestion d'un parc d'attraction en utilisant les bons principes de programmation orientée objet. En plus de devoir programmer chaque fonctionnalité souhaitée, vous devrez choisir les structures de données qui permettent de résoudre le problème efficacement.

Consignes

Pour chaque structure de données choisies, vous devez expliquer les raisons de votre choix en commentaire directement dans le code.

Classe Attraction

Cette classe permet de conserver les données pour une attraction précise. On doit pouvoir y avoir un ID, un nom, une capacité et un type d'attraction. Cette classe ne devrait pas avoir d'information sur un parc ou sur son emplacement dans un parc.

Les types d'attractions sont les suivants : Sensation forte (S), Intermédiaire (I), Famille (F), Toilette (T), Magasin (M) ou Restaurant (R).

Classe Parc

Cette classe doit charger la liste des attractions du parc.

Les données doivent être conservées dans une structure qui respecte ces besoins :

- Pas de modification fréquente.
- On ne connaît pas la quantité d'attractions à l'avance.
- On souhaite avoir accès à une attraction précise par son ID facilement.

Le chargement se fera automatiquement à partir d'un fichier texte nommé « attractions.txt ».

Des données de tests sont fournies avec l'énoncé.

Elles sont présentées sous le format suivant :

```
ID;TYPE;NOM;CAPACITÉ
```

Classe Map

Cette classe doit charger la carte du parc d'attractions. Encore ici, les données seront chargées automatiquement à partir d'un fichier à l'instanciation d'un objet.

Un fichier de tests nommé « map.txt » est fourni avec l'énoncé.

La première ligne du fichier correspond à la taille de la grille (hauteur x largeur) qui représente la carte.

Les autres cellules représentent l'emplacement des attractions dans le parc. On y retrouve les IDs des attractions ou « ----- » pour les emplacements vides.

Les données doivent être conservées dans une structure qui respecte ces besoins :

- On connaît la taille de la carte à l'avance.
- Aucune modification après l'exécution.

Classe Visiteur

Cette classe permet de conserver les données pour un visiteur. On doit pouvoir y avoir le nom du visiteur. Cette classe doit également avoir un historique qui conserve les actions que le visiteur a effectuées dans le parc. Les entrées de l'historique peuvent être des chaînes de caractère.

Les données de l'historique doivent être conservées dans une structure qui respecte ces besoins :

- On ne connaît pas la quantité d'éléments à l'avance.
- Pas de modification mais des ajouts en fin de liste.

Classe GestionVisiteurs

Cette classe doit permettre de conserver l'état actuel du parc d'attractions.

Premièrement, on y trouvera la liste des visiteurs actuels.

Pour les visiteurs, les données doivent être conservées dans une structure qui respecte ces besoins :

- Les modifications sont fréquentes.
- Pas d'accès direct aux données.

Ensuite, on souhaite conserver l'état des files d'attente de chaque manège.

Pour les files d'attente, les données doivent être conservées dans une structure qui respecte ces besoins :

- On veut gérer facilement la liste des visiteurs présents pour chaque attraction.

Les actions suivantes doivent être supportées :

```
public void EntrerVisiteurDansFileAttente(string attractionId, Visiteur visiteur)
public void EntrerVisiteurDansAttraction(string attractionId)
public void EntrerVisiteurDansParc(Visiteur visiteur)
public void SortirVisiteurDuParc(Visiteur visiteur)
```

Ces 4 méthodes ajouteront une ligne dans l'historique du visiteur concerné.

Classe AffichageConsole

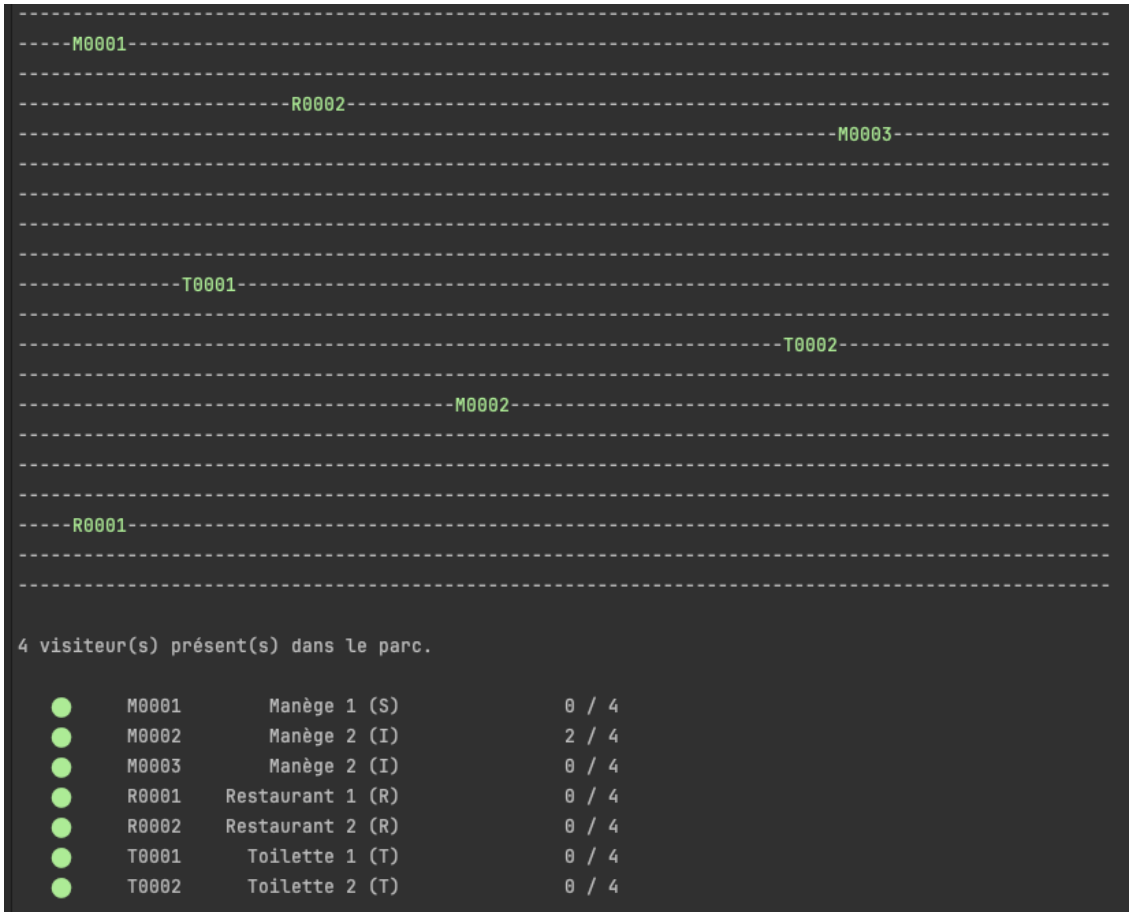
Cette classe doit permettre l'affichage en console de l'état de notre parc d'attractions.

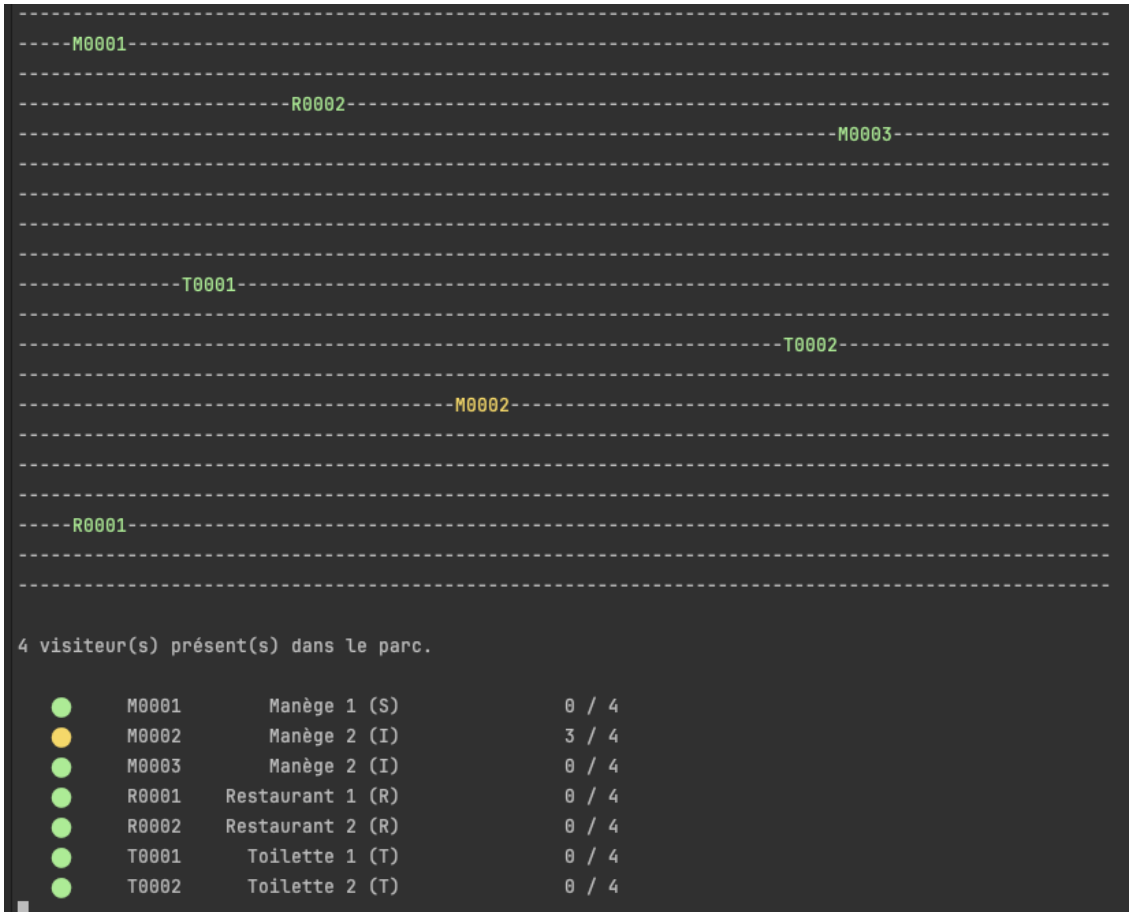
Les actions suivantes doivent être supportées :

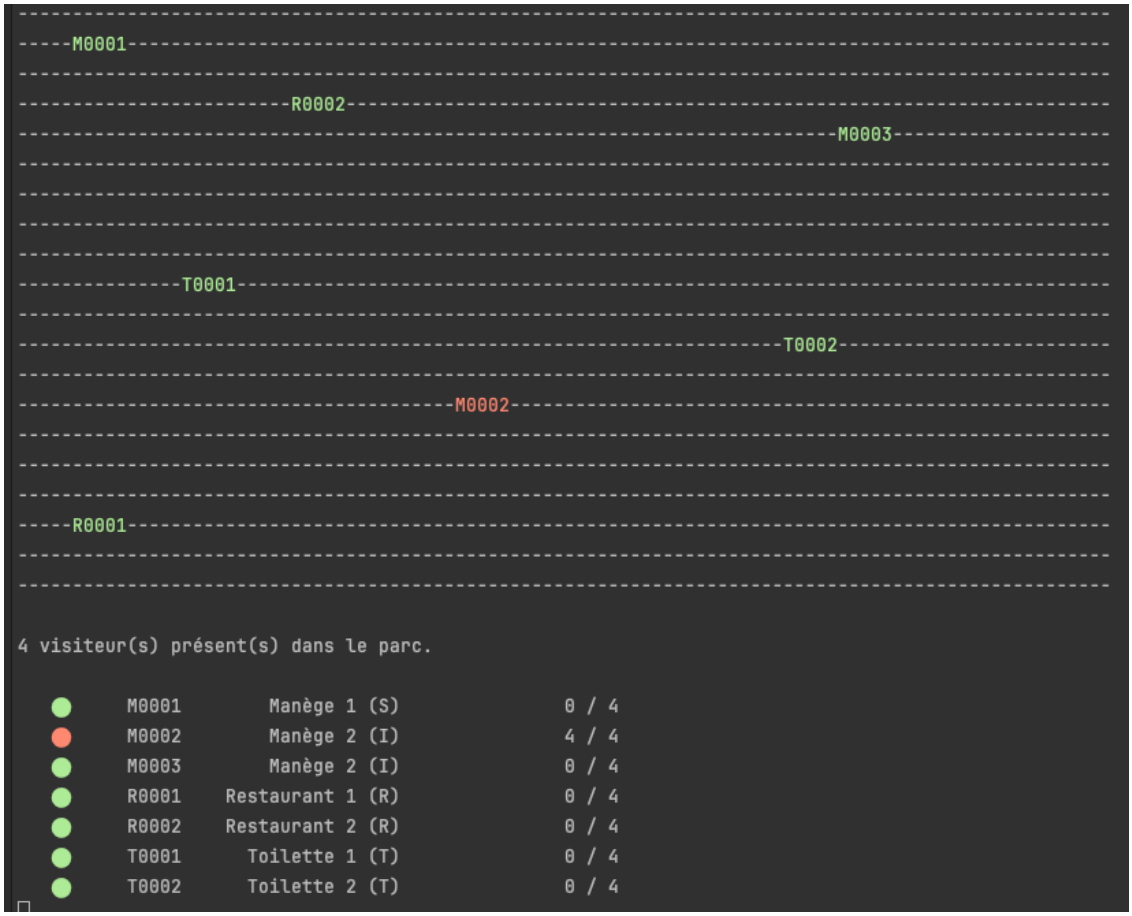
```
public static void Afficher(Parc parc, Map map, GestionVisiteurs gestionVisiteurs)
public static void AfficherHistoriqueVisiteur(Visiteur visiteur)
```

Pour la première méthode, vous devez afficher la carte, le nombre de visiteurs dans le parc ainsi que l'état de chaque attraction selon le format présenté plus bas.

Afin de bien représenter l'état du parc, la couleur de certains éléments doit être ajustée en fonction du nombre de visiteurs. Une attraction qui est à plus de 75% de sa capacité doit apparaître en jaune. Si elle est a plus de 100% de sa capacité, elle doit apparaître en rouge. Les autres attractions doivent apparaître en vert.




M0001
R0002
M0003
T0001
T0002
M0002
R0001
4 visiteur(s) présent(s) dans le parc.
M0001 Manège 1 (S) 0 / 4
M0002 Manège 2 (I) 3 / 4
M0003 Manège 2 (I) 0 / 4
R0001 Restaurant 1 (R) 0 / 4
R0002 Restaurant 2 (R) 0 / 4
T0001 Toilette 1 (T) 0 / 4
T0002 Toilette 2 (T) 0 / 4



La deuxième méthode affichera l'historique d'un visiteur sous ce format :

```
### Nom 1 ###
- Entrer dans le parc.
- Entrer dans la file d'attente de l'attraction M0002.
- Entrer dans l'attraction M0002.
- Sortir du parc.
```

Finalement, pour aider à valider votre solution et pour la correction, vous devez avoir ce code dans le `Main` de votre projet (n'oubliez pas de le commenter avant la remise) :

```
public static class Program
{
    private static readonly Parc Parc = new();
    private static readonly Map Map = new();
    private static readonly GestionVisiteurs GestionVisiteurs = new(Parc);

    private static void Afficher()
    {
        Thread.Sleep(1000);
        AffichageConsole.Afficher(Parc, Map, GestionVisiteurs);
    }

    private static void TestEntrerVisiteur(Visiteur visiteur)
    {
        GestionVisiteurs.EntrerVisiteurDansParc(visiteur);
        GestionVisiteurs.EntrerVisiteurDansFilAttente("M0002", visiteur);
        Afficher();
    }

    private static void TestSortirVisiteur(Visiteur visiteur)
    {
        GestionVisiteurs.SortirVisiteurDuParc(visiteur);
        Afficher();
    }

    public static void Main()
    {
        AffichageConsole.Afficher(Parc, Map, GestionVisiteurs);

        var visiteur1 = new Visiteur("Nom 1");
        TestEntrerVisiteur(visiteur1);

        var visiteur2 = new Visiteur("Nom 2");
        TestEntrerVisiteur(visiteur2);

        var visiteur3 = new Visiteur("Nom 3");
        TestEntrerVisiteur(visiteur3);

        var visiteur4 = new Visiteur("Nom 4");
        TestEntrerVisiteur(visiteur4);

        for (var i = 1; i <= 4; i++)
        {
            GestionVisiteurs.EntrerVisiteurDansAttraction("M0002");
            Afficher();
        }

        TestSortirVisiteur(visiteur3);
        TestSortirVisiteur(visiteur4);
        TestSortirVisiteur(visiteur2);
        TestSortirVisiteur(visiteur1);

        AffichageConsole.AfficherHistoriqueVisiteur(visiteur1);
    }
}
```

Le projet doit être fait dans *Visual Studio* ou *Rider* avec les outils d'analyse statique vus en classe.

Attention au plagiat et à la présentation de votre code. Ne pas oublier les commentaires et les bonnes pratiques.

Remise

- Le travail doit se faire en équipe de 2, sauf sur approbation préalable du professeur.
- La remise doit être sur Léa à l'endroit approprié.
- Vous devez envoyer votre projet complet dans un fichier zip.
- La remise doit se faire avant le 23 octobre 2025 à 21:59.