

L'objectif de cette séance est de modéliser et de réaliser une application de simulation d'une course.

Les participants à une course sont organisés en équipe. Chaque équipe porte un nom et chaque coureur se voit attribué un numéro de dossard lors de son inscription.

Au départ, l'horloge est initialisée à 00h 00mn 00s, et les coureurs partent lorsque le départ est donné. Pendant la course, chaque coureur avance à son rythme. À chaque arrivée, l'heure d'arrivée et le numéro du coureur sont comptabilisés. À la fin de la course, les classements par coureur et par équipe sont affichés.

Voici un exemple d'interaction avec le système de simulation (le texte saisi par l'utilisateur est indiqué en gras).

+++ RaceSimulator +++

Longueur de la course :

Inscriptions :

Nombre d'équipes :

Nom de l'équipe :

Nombre de coureurs :

Nom du coureur :

Vitesse de déplacement du coureur (m/s) :

Nom du coureur :

Vitesse de déplacement du coureur (m/s) :

Nom de l'équipe :

Nombre de coureurs :

Nom du coureur :

Vitesse de déplacement du coureur (m/s) :

Nom du coureur :

Vitesse de déplacement du coureur (m/s) :

Nom du coureur :

Vitesse de déplacement du coureur (m/s) :

Départ de la course :

Avancer l'horloge de (x sec.) :

Avancer l'horloge de (x sec.) :

Avancer l'horloge de (x sec.) :

Avancer l'horloge de (x sec.) :

Avancer l'horloge de (x sec.) :

Arrivée de :Batman[1](0h 0mn 0s -> 0h 3mn 20s)

Arrivée de :Wolverine[2](0h 0mn 0s -> 0h 4mn 10s)

Toute l'équipe SuperHero est arrivée

Arrivée de :Yoshi[4](0h 0mn 0s -> 0h 4mn 10s)

Avancer l'horloge de (x sec.) :

Avancer l'horloge de (x sec.) :

Arrivée de :Mario[3](0h 0mn 0s -> 0h 5mn 33s)

Avancer l'horloge de (x sec.) :

Avancer l'horloge de (x sec.) :

Arrivée de :Princess[5](0h 0mn 0s -> 0h 8mn 20s)

Toute l'équipe Mario est arrivée

La course est terminée

▷ **Question 1.** En utilisant des cartes CRC, identifier les différentes classes, leurs responsabilités et leurs collaborations.

▷ **Question 2.** À partir des cartes CRC, écrire l'interface de chaque classe que vous avez identifiée.

Rappel sur le contrat à respecter par la méthode `equals()`

La méthode `public void equals(Object o)` implémente une relation d'équivalence.

- *reflexive* : pour toute référence `x`, `x.equals(x)` doit retourner `true`.
- *symétrique* : pour toutes références `x` et `y`, `x.equals(y)` doit retourner `true` si et seulement si `y.equals(x)` retourne `true`.
- *transitive* : pour toutes références `x`, `y` et `z`, si `x.equals(y)` retourne `true` et `y.equals(z)` retourne `true` alors `x.equals(z)` doit retourner `true`.
- *cohérente* : pour toutes références `x` et `y`, des invocations successives de l'appel `x.equals(y)` doit retourner la même valeur (aucune information de l'objet lors de l'exécution de la méthode).
- Pour tout référence `x` non `null`, `x.equals(null)` doit retourner `false`.

Il est important lorsque l'on redéfinit le comportement de la méthode `equals()` de redéfinir en conséquence le comportement de la méthode `int hashCode()`. Le contrat de cette méthode précise :

- La valeur entière retournée ne doit pas changer lors d'appels successifs à la méthode `hashCode()` si l'état de l'objet n'a pas changé au point d'avoir des impacts sur le résultat d'une comparaison pas la méthode `equals()`. Il n'est pas nécessaire que la valeur soit la même lors de différentes exécutions du programme.
- Si deux références sont égales en accord avec la méthode `equals()` alors la méthode `hashCode()` doit retourner la même valeur entière pour ces deux références.
- Il n'est pas requis que les valeurs entières soient différentes pour deux références "non égales" (au sens de `equals()`).