

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.

**Réponse**

race.RaceSimulator	
<b>Responsabilités</b> <ul style="list-style-type: none"> <li>– Conserve la course en cours</li> <li>– Affiche l'état de la course</li> <li>– Interagit avec l'utilisateur</li> </ul>	<b>Collaborateurs</b> <ul style="list-style-type: none"> <li>– race.Race</li> <li>– race.Team</li> <li>– java.util.Scanner</li> </ul>
race.Clock	
<b>Responsabilités</b> <ul style="list-style-type: none"> <li>– Conserve une horloge (triplet &lt; heures, minutes, secondes &gt;)</li> <li>– Consulter chaque valeur du triplet</li> <li>– Convertir en secondes le temps indiqué par l'horloge</li> <li>– Incrémenter l'horloge (les secondes, les minutes ou les heures)</li> <li>– Calculer la différence entre deux horloges</li> <li>– Tester l'égalité entre deux horloges</li> <li>– Valeur sous forme d'une chaîne de caractères</li> </ul>	<b>Collaborateurs</b>
race.Racer	
<b>Responsabilités</b> <ul style="list-style-type: none"> <li>– Conserve le nom du coureur</li> <li>– Conserve le numéro de dossard</li> <li>– Conserve la vitesse de course du coureur</li> <li>– Valeur sous forme d'une chaîne de caractères</li> <li>– Enregistre l'heure de départ du coureur</li> <li>– Fait "avancer" le coureur</li> <li>– Indique la distance parcourue depuis le départ</li> <li>– Indique si le coureur est arrivé</li> </ul>	<b>Collaborateurs</b> <ul style="list-style-type: none"> <li>– race.Clock</li> <li>– race.Race</li> </ul>
race.Team	
<b>Responsabilités</b> <ul style="list-style-type: none"> <li>– Conserve le nom de l'équipe</li> <li>– Ajouter un coureur à l'équipe</li> <li>– Connaître le nombre de coureur dans l'équipe</li> <li>– Obtenir un coureur particulier de l'équipe</li> <li>– Donner le top départ à toute l'équipe</li> <li>– Donner le top départ à un coureur</li> <li>– Inscrire l'équipe à une course</li> <li>– Fait "avancer" chaque membre de l'équipe</li> <li>– Indique si l'équipe est arrivée</li> <li>– Valeur sous forme d'une chaîne de caractères</li> </ul>	<b>Collaborateurs</b> <ul style="list-style-type: none"> <li>– race.Race</li> <li>– race.Racer</li> <li>– race.Clock</li> </ul>
race.Race	
<b>Responsabilités</b> <ul style="list-style-type: none"> <li>– Générer un nouveau numéro de dossard</li> <li>– Inscrire une équipe à cette course</li> <li>– Obtenir l'heure actuelle</li> <li>– Obtenir la distance totale à parcourir</li> <li>– Connaître si la course est finie (tout le monde est arrivé)</li> <li>– Lancer le départ de la course</li> <li>– Faire "avancer" le temps</li> <li>– Consulter le podium</li> </ul>	<b>Collaborateurs</b> <ul style="list-style-type: none"> <li>– race.Team</li> <li>– race.Clock</li> </ul>

**Fin réponse**

▷ **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()`).

**Réponse**

L'utilisation de la syntaxe suivante risquerait de violer le contrat (notamment la symétrie) lors d'un éventuel héritage avec redéfinition de la classe `Clock`.

```
1 if (! (this instanceof Clock))
2     return false;
```

**Fin réponse**

## Réponse

```

1 package race;
2
3 import java.util.Scanner;
4
5 public class RaceSimulator {
6
7     private Race race;
8
9     public RaceSimulator() {
10    }
11
12    public static void main(String[] args) {
13        System.out.println("+++ RaceSimulator +++ \n");
14
15        RaceSimulator simulator = new RaceSimulator();
16        simulator.createRace();
17        simulator.registration();
18        simulator.main();
19
20        //RaceSimulator.demo();
21    }
22
23    public void main() {
24        System.out.println("Départ de la course :");
25        System.out.println("-----");
26        this.race.start();
27
28        while (!this.race.isFinished()) {
29            int secs = askInteger("Avancer l'horloge de (x sec.) : ");
30            this.race.run(secs);
31        }
32
33        System.out.println("La course est terminée");
34    }
35
36    public void createRace() {
37        int distance = askInteger("Longueur de la course :");
38        this.race = new RaceImpl(distance);
39    }
40
41    private void registration() {
42        System.out.println("Inscriptions :");
43        System.out.println("-----");
44
45        int teamCount = askInteger("Nombre d'équipes : ");
46
47        for (int i = 0; i < teamCount; i++) {
48            Team team = registerTeam();
49            race.registerTeam(team);
50            System.out.println("-----");
51        }
52    }
53
54    private static Team registerTeam() {
55        System.out.println("-----");
56        String teamName = askString("Nom de l'équipe : ");
57        Team team = new TeamImpl(teamName);
58        int teammateCount = askInteger("Nombre de coureurs : ");
59        System.out.println("-----");
60        for (int j = 0; j < teammateCount; j++) {
61            String name = askString("Nom du coureur : ");
62            int speed = askInteger("Vitesse de déplacement du coureur (m/s) : ");
63            Racer racer = new RacerImpl(name, speed);
64            team.joinTeam(racer);
65        }
66        return team;
67    }
68
69    private static int askInteger(String msg) {
70        Scanner scan = new Scanner(System.in);
71        System.out.print(msg);
72        while (!scan.hasNextInt()) {
73            if (scan.hasNext())
74                scan.next();
75            System.out.print(msg);
76        }
77        return scan.nextInt();
78    }
79
80    private static String askString(String msg) {
81        Scanner scan = new Scanner(System.in);

```

```

82         System.out.print(msg);
83         return scan.next();
84     }
85 }

```

Fin réponse

Réponse

```

1 package race;
2
3 public interface Clock {
4
5     /**
6      * reinitialise l'horloge à 00 h 00mn 00s.
7      */
8     public void reset();
9
10    /**
11     * Retourne l'heure de l'horloge.
12     *
13     * @return la valeur du compteur des heures de l'horloge.
14     */
15    public int getHours();
16
17    /**
18     * Retourne les minutes de l'horloge.
19     *
20     * @return la valeur du compteur des minutes de l'horloge.
21     */
22    public int getMinutes();
23
24    /**
25     * Retourne les secondes de l'horloge.
26     *
27     * @return la valeur du compteur des secondes de l'horloge.
28     */
29    public int getSeconds();
30
31    /**
32     * Retourne la valeur totale de l'horloge en secondes.
33     *
34     * @return la valeur de l'horloge en secondes
35     */
36    public int inSeconds();
37
38    /**
39     * incremente la valeur de l'horloge.
40     *
41     * @param nbh
42     *         le nombre d'heures à ajouter.
43     * @param nbm
44     *         le nombre de minutes à ajouter.
45     * @param nbs
46     *         le nombre de secondes à ajouter.
47     */
48    public void increment(int nbh, int nbm, int nbs);
49
50    /**
51     * incremente la valeur de l'horloge.
52     *
53     * @param nbs
54     *         le nombre de secondes à ajouter.
55     */
56    public void incrementSeconds(int nbs);
57
58    /**
59     * incremente la valeur de l'horloge.
60     *
61     * @param nbm
62     *         le nombre de minutes à ajouter.
63     */
64    public void incrementMinutes(int nbm);
65
66    /**
67     * incremente la valeur de l'horloge.
68     *
69     * @param nbh
70     *         le nombre d'heures à ajouter.
71     */
72    public void incrementHours(int nbh);
73
74    /**

```

```

75     * Calcule la différence de temps écoulée entre l'horloge courante et
76     * l'horloge passée en paramètre.
77     *
78     * @param other
79     *     représente l'horloge avec laquelle on souhaite calculer la
80     *     différence.
81     * @return une nouvelle horloge représentant la différence.
82     */
83     public Clock differenceBetween(Clock other);
84
85     /**
86     * Indique si un objet est égale à l'horloge actuelle.
87     *
88     * @param other
89     *     représente la référence vers l'objet avec lequel on souhaite
90     *     comparer.
91     * @return true si les deux objets sont égaux.
92     */
93     public boolean equals(Object other);
94
95     /**
96     * Calcule une valeur entière (nommée hashCode) représentant l'objet.
97     *
98     * @return la valeur entière correspondant au hashCode.
99     */
100    public int hashCode();
101
102    /**
103    * Retourne une représentation de l'horloge sous forme de chaîne de
104    * caractères.
105    *
106    * @return une chaîne de caractères de la forme 'Xh Ymn Zs'.
107    */
108    public String toString();
109
110 }

```

```

1 package race;
2
3 public class ClockImpl implements Clock {
4     /* BEGIN KILL */
5     private int hours;
6     private int minutes;
7     private int seconds;
8
9     public ClockImpl() {
10         this.hours = 0;
11         this.minutes = 0;
12         this.seconds = 0;
13     }
14
15     public ClockImpl(int hours, int minutes, int seconds) {
16         //assert 0 <= hours && hours < 24;
17         //assert 0 <= minutes && minutes < 60;
18         //assert 0 <= seconds && seconds < 60;
19         this(hours*3600+minutes*60+seconds);
20     }
21
22     public ClockImpl(int seconds) {
23         this();
24         this.incrementSeconds(seconds);
25     }
26
27     public ClockImpl(ClockImpl c) {
28         this.hours = c.hours;
29         this.minutes = c.minutes;
30         this.seconds = c.seconds;
31     }
32
33     public void reset() {
34         this.hours = 0;
35         this.minutes = 0;
36         this.seconds = 0;
37     }
38
39     public int getHours() {
40         return this.hours;
41     }
42
43     public int getMinutes() {
44         return this.minutes;
45     }
46
47     public int getSeconds() {
48         return this.seconds;

```

```

49     }
50
51     public int inSeconds() {
52         return this.hours*3600+this.minutes*60+this.seconds;
53     }
54
55     public void increment(int nbh, int nbm, int nbs) {
56         this.incrementSeconds(nbh * 3600 + nbm * 60 + nbs);
57     }
58
59     public void incrementSeconds(int nbs) {
60         int s = this.seconds + nbs;
61         int m = this.minutes + s / 60;
62         this.seconds = s % 60;
63         this.hours = this.hours + m / 60;
64         this.minutes = m % 60;
65     }
66
67     public void incrementMinutes(int nbm) {
68         int m = this.minutes + nbm;
69         this.hours = this.hours + m / 60;
70         this.minutes = m % 60;
71     }
72
73     public void incrementHours(int nbh) {
74         this.hours = this.hours + nbh;
75     }
76
77     public Clock differenceBetween(Clock other) {
78         ClockImpl res = new ClockImpl();
79
80         int otherTotalSeconds = other.inSeconds();
81         int totalSeconds = this.inSeconds();
82
83         int diffSeconds;
84         if (totalSeconds > otherTotalSeconds) {
85             diffSeconds = totalSeconds - otherTotalSeconds;
86         } else {
87             diffSeconds = otherTotalSeconds - totalSeconds;
88         }
89
90         res.hours = diffSeconds / 3600;
91         diffSeconds = diffSeconds % 3600;
92         res.minutes = diffSeconds / 60;
93         res.seconds = diffSeconds % 60;
94
95         return res;
96     }
97
98     @Override
99     public boolean equals(Object other) {
100         if (this == other)
101             return true;
102         if (other == null || this.getClass() != other.getClass())
103             return false;
104
105         ClockImpl otherClock = (ClockImpl) other;
106         return (this.hours == otherClock.hours
107             && this.minutes == otherClock.minutes
108             && this.seconds == otherClock.seconds);
109     }
110
111     @Override
112     public int hashCode() {
113         final int prime = 31;
114         int result = 1;
115         result = prime * result + hours;
116         result = prime * result + minutes;
117         result = prime * result + seconds;
118         return result;
119     }
120
121     @Override
122     public String toString() {
123         return this.hours + "h " + this.minutes + "mn " + this.seconds + "s";
124     }
125     /* ENDKILL */
126 }

```

Fin réponse

## Réponse

```

1 package race;
2
3 public interface Racer {
4
5     /**
6      * Affecte un numéro de dossard à un coureur.
7      *
8      * @param num
9      *         numéro du dossard.
10     */
11     public void setNumber(int num);
12
13     /**
14      * Indique le numéro de dossard du coureur.
15      * Si aucun numéro n'a été affecté alors la méthode retourne -1
16      *
17      * @return le numero du dossard associé au coureur (sinon -1).
18     */
19     public int getNumber();
20
21     /**
22      * Indique le nom du coureur.
23      *
24      * @return le nom du coureur.
25     */
26     public String getName();
27
28     /**
29      * Indique la distance parcourue par le coureur.
30      *
31      * @return la distance parcourue par le coureur.
32     */
33     public int getDistance();
34
35     /**
36      * Indique la vitesse de marche du coureur.
37      *
38      * @return la vitesse en m/s.
39     */
40     public int getSpeed();
41
42     /**
43      * Notifie le coureur qu'il prend le départ. Le coureur conserve alors
44      * l'heure de départ.
45      *
46      * @param race
47      *         la course a laquelle le coureur prend le départ.
48     */
49     public void start(Race race);
50
51     /**
52      * Notifier le coureur qu'il doit avancer. La distance parcourue est alors
53      * modifiée en conséquence.
54      *
55      * @param race
56     */
57     public void run(Race race);
58
59     /**
60      * Indique si le coureur est arrivé à la fin du parcours.
61      *
62      * @return true si le coureur est arrivé.
63     */
64     public boolean isArrived();
65
66     /**
67      * Indique le temps de course du coureur.
68      * N'est disponible qu'une fois le coureur arrivé.
69      * Sinon retourne null
70      *
71      * @return le temps de course du coureur.
72     */
73     public Clock getTime();
74
75     /**
76      * Retourne une représentation du coureur sous forme de chaîne de caractères
77      *
78      * @return une chaîne de caractères représentant le coureur
79     */
80     public String toString();
81 }
82

```



```

1 package race;
2
3 public class RacerImpl implements Racer {
4     /* BEGIN KILL */
5     private String name;
6     private int num;
7     private int speed;
8
9     private int distance;
10
11     private Clock departureTime;
12     private Clock arrivalTime;
13     private Clock lastTime;
14
15     public RacerImpl() {
16         this.name = "";
17         this.speed = 0;
18         this.distance = 0;
19         this.num = -1;
20     }
21
22     public RacerImpl(String name, int speed) {
23         this.name = name;
24         this.speed = speed;
25         this.distance = 0;
26         this.num = -1;
27     }
28
29     public void setNumber(int num) {
30         this.num = num;
31     }
32
33     public int getNumber() {
34         return this.num;
35     }
36
37     public String getName() {
38         return this.name;
39     }
40
41     public int getDistance() {
42         return this.distance;
43     }
44
45     public int getSpeed() {
46         return this.speed;
47     }
48
49     public void start(Race race) {
50         this.departureTime = new ClockImpl((ClockImpl) race.getCurrentTime());
51         this.lastTime = this.departureTime;
52     }
53
54     public void run(Race race) {
55         int time = lastTime.differenceBetween(race.getCurrentTime()).inSeconds();
56         this.distance = this.distance + this.speed * time;
57         this.lastTime = new ClockImpl((ClockImpl) race.getCurrentTime());
58
59         if (this.distance >= race.getDistance()) {
60             this.distance = race.getDistance();
61             int requiredTime = race.getDistance() / this.speed;
62             this.arrivalTime = new ClockImpl((ClockImpl) this.departureTime);
63             this.arrivalTime.incrementSeconds(requiredTime);
64         }
65     }
66
67     public boolean isArrived() {
68         return (this.arrivalTime != null);
69     }
70
71     public Clock getTime() {
72         if (this.departureTime != null && this.arrivalTime != null) {
73             return this.departureTime.differenceBetween(this.arrivalTime);
74         } else {
75             return null;
76         }
77     }
78
79     @Override
80     public String toString() {
81         String res = this.name + "[" + this.num + "]";
82         if (this.departureTime != null) {
83             res = res + "(" + this.departureTime;
84             if (this.arrivalTime != null)

```

```

85         res = res + " -> " + this.arrivalTime;
86         res = res + ")";
87     }
88     return res;
89 }
90 /* ENDKILL */
91 }

```

Fin réponse

Réponse

```

1 package race;
2
3 public interface Team {
4
5     /**
6      * Indique le nom de l'équipe.
7      *
8      * @return le nom de l'équipe.
9      */
10    public String getName();
11
12    /**
13     * Enregistre un nouveau coureur dans l'équipe.
14     *
15     * @param racer
16     *        le nouveau coureur à enregistrer.
17     */
18    public void joinTeam(Racer racer);
19
20    /**
21     * Indique de combien de coureurs l'équipe est constituée.
22     *
23     * @return le nombre de coureurs de l'équipe.
24     */
25    public int getTeammateCount();
26
27    /**
28     * Retourne un coureur spécifique
29     *
30     * @param number
31     *        le numéro de dossard du coureur recherché.
32     * @return le coureur si il fait parti de l'équipe, null si le coureur n'est
33     *         pas trouvé.
34     */
35    public Racer getRacer(int number);
36
37    /**
38     * Indique à un coureur qu'il prend le départ immédiatement.
39     *
40     * @param race
41     *        la course à laquelle le coureur prend le départ.
42     * @param number
43     *        le numéro de dossard du coureur concerné.
44     */
45    public void startRacer(Race race, int number);
46
47    /**
48     * Inscrit l'équipe à une course. Un numéro de dossard est demandé pour
49     * chaque coureur de l'équipe.
50     *
51     * @param race
52     *        la course à laquelle l'équipe s'inscrit.
53     */
54    public void registerRace(Race race);
55
56    /**
57     * Notifie tous les coureurs que la course a commencé.
58     *
59     * @param race
60     *        la course à laquelle l'équipe participe.
61     */
62    public void start(Race race);
63
64    /**
65     * "Fait avancer" les coureurs de l'équipe.
66     *
67     * @param race
68     *        la course à laquelle l'équipe participe.
69     */
70    public void run(Race race);
71 }

```

```

72     /**
73      * Indique si tous les membres de l'équipe ont terminé la course.
74      *
75      * @return true si toute l'équipe est arrivée.
76      */
77     public boolean isArrived();
78
79     /**
80      * Retourne une représentation de l'équipe sous forme de chaîne de
81      * caractères.
82      *
83      * @return une chaîne de caractères représentant l'équipe.
84      */
85     public String toString();
86
87 }

```

```

                                race.TeamImpl
1 package race;
2
3 import java.util.ArrayList;
4
5 public class TeamImpl implements Team {
6     /* BEGINKILL */
7     private ArrayList teammates;
8     private String name;
9
10    public TeamImpl(String name) {
11        this.teammates = new ArrayList();
12        this.name = name;
13    }
14
15    public String getName() {
16        return this.name;
17    }
18
19    public void joinTeam(Racer racer) {
20        this.teammates.add(racer);
21    }
22
23    public int getTeammateCount() {
24        return this.teammates.size();
25    }
26
27    public void registerRace(Race race) {
28        for (int i = 0; i < teammates.size(); i++) {
29            Racer racer = (Racer) this.teammates.get(i);
30            racer.setNumber(race.generateNumber());
31        }
32    }
33
34    public Racer getRacer(int number) {
35        Racer r = null;
36        for (int i=0; i < this.teammates.size(); i++) {
37            r = (Racer) this.teammates.get(i);
38            if (r.getNumber() == number) {
39                return r;
40            }
41        }
42        return null;
43    }
44
45    public void startRacer(Race race, int number) {
46        Racer r = getRacer(number);
47        if (r != null) {
48            r.start(race);
49        }
50    }
51
52    public void start(Race race) {
53        for (int i=0; i<this.teammates.size(); i++) {
54            Racer r = (Racer) this.teammates.get(i);
55            r.start(race);
56        }
57    }
58
59    public void run(Race race) {
60        for (int i=0; i<this.teammates.size(); i++) {
61            Racer r = (Racer) this.teammates.get(i);
62            if (! r.isArrived()) {
63                r.run(race);
64                if (r.isArrived()) {
65                    System.out.println("Arrivee de :"+r);
66                }
67            }
68        }
69    }

```

```

68     }
69 }
70
71 public boolean isArrived() {
72     for (int i=0; i<this.teammates.size(); i++) {
73         Racer r = (Racer) this.teammates.get(i);
74         if (! r.isArrived()) {
75             return false;
76         }
77     }
78     return true;
79 }
80
81 @Override
82 public String toString() {
83     String s = "Team (" + this.name + ") [";
84     for (int i = 0; i < this.teammates.size(); i++) {
85         Racer r = (Racer) this.teammates.get(i);
86         s = s + "\n\t" + r.toString();
87     }
88     s = s + "\n]";
89     return s;
90 }
91 /* ENDKILL */
92 }

```

Fin réponse

Réponse

```

1 package race;
2
3 public interface Race {
4     /**
5      * Inscrit une équipe.
6      *
7      * @param team
8      *      l'équipe à inscrire.
9      */
10    public void registerTeam(Team team);
11
12    /**
13     * Génère un nouveau numéro de dossard non attribué.
14     *
15     * @return le nouveau numéro de dossard.
16     */
17    public int generateNumber();
18
19    /**
20     * Indique l'heure courante de la course.
21     *
22     * @return l'heure actuelle
23     */
24    public Clock getCurrentTime();
25
26    /**
27     * @return la distance à parcourir.
28     */
29    public int getDistance();
30
31    /**
32     * Retourne une représentation de la course sous forme de chaîne de
33     * caractères
34     *
35     * @return une chaîne de caractères représentant la course
36     */
37    public String toString();
38
39    /**
40     * Indique si la course est terminée (si tous les coureurs inscrits sont
41     * arrivés).
42     *
43     * @return true si la course est terminée.
44     */
45    public boolean isFinished();
46
47    /**
48     * Lance le départ de la course (notifie les équipes et les coureurs que le
49     * départ est donné).
50     */
51    public void start();
52 }

```

```

53     /**
54      * Incrémente l'horloge et "fait avancer" les coureurs.
55      *
56      * @param nbSecs
57      *       le nombre de secondes écoulées.
58      */
59     public void run(int nbSecs);
60
61     /**
62      * @return une chaîne de caractères représentant le podium d'arrivée.
63      */
64     public String podium();
65 }

```

race.RaceImpl

```

1 package race;
2
3 import java.util.ArrayList;
4
5 public class RaceImpl implements Race {
6     /* BEGIN KILL */
7     private ArrayList teams;
8     private int registeredRacer = 0;
9     private ClockImpl currentTime;
10    private boolean finished;
11    private int distance;
12
13    public RaceImpl(int distance) {
14        this.teams = new ArrayList();
15        this.currentTime = new ClockImpl();
16        this.finished = false;
17        this.distance = distance;
18    }
19
20    public void registerTeam(Team t) {
21        t.registerRace(this);
22        this.teams.add(t);
23    }
24
25    public int generateNumber() {
26        return ++this.registeredRacer;
27    }
28
29    public ClockImpl getCurrentTime() {
30        return this.currentTime;
31    }
32
33    public int getDistance() {
34        return this.distance;
35    }
36
37    public String toString() {
38        String s = "Race: {";
39        for (int i = 0; i < teams.size(); i++) {
40            Team t = (Team) teams.get(i);
41            s = s + "\n" + t.toString();
42        }
43        s = s + "\n}";
44        return s;
45    }
46
47    public boolean isFinished() {
48        for (int i = 0; i < teams.size(); i++) {
49            Team team = (Team) teams.get(i);
50            if (!team.isArrived()) {
51                return false;
52            }
53        }
54        return true;
55    }
56
57    public void start() {
58        this.currentTime.reset();
59        for (int i = 0; i < this.teams.size(); i++) {
60            Team team = (Team) this.teams.get(i);
61            team.start(this);
62        }
63    }
64
65    public void run(int nbSecs) {
66        this.currentTime.incrementSeconds(nbSecs);
67        for (int i = 0; i < this.teams.size(); i++) {
68            Team team = (Team) this.teams.get(i);
69            if (!team.isArrived()) {
70                team.run(this);

```

```
71         if (team.isArrived()) {
72             System.out.println("Toute l'équipe " + team.getName() + " est arrivée")
73         }
74     }
75 }
76
77
78 public String podium() {
79     // TODO: not yet implemented ;-)
80     return "";
81 }
82 /* ENDKILL */
83 }
```

---

Fin réponse