TP4
Multicast et broadcast

Ce TP est à résoudre en utilisant exclusivement les fonctions de moniteur des objets Java (`wait`, `notify`, `notifyAll`).

1. Multicast

Écrire une classe `Multicast` qui permet à un *thread* d'envoyer une donnée de type T quelconque à plusieurs autres *threads*. L'interface que la classe doit respecter est la suivante:

```
public interface IMulticast <T> {
     public void Send(T data);
     public T Receive() throws InterruptedException ;
}
```

Les *threads* récepteurs doivent indiquer leur disponibilité à recevoir la donnée en invoquant la méthode `Receive`. Ils sont *bloqués* par cet appel. Le *thread* émetteur envoie la donnée en invoquant la méthode `Send`. Toutes les tâches bloqués à ce moment là dans un `Receive` seront réveillées et la donnée passée dans `Send` est envoyée comme valeur de retour par les `Receive`.

Une fois le `Send` terminé, tout nouvel appel à `Receive` sera bloquant jusqu'au prochain envoi de donnée par un nouvel appel `Send`.

Utilisation :

Reprendre la version de l'application `Ball World` de la fin du TP1 et ajouter la fonctionnalité suivante : chaque fois qu'une balle se trouve dans la zone diagonale, elle s'arrête pour changer de couleur. Une nouvelle couleur (aléatoire) est diffusée périodiquement environ toutes les 3 secondés par un *thread* appelé `ColorMaster`, par l'intermédiaire d'un objet de type `Multicast`.

2. Broadcast

Un *broadcast* est similaire à un *multicast*, à la différence que ***tous*** les récepteurs doivent recevoir la donnée. Pour être connus, les récepteurs s'enregistrent au préalable auprès de l'objet *broadcast*. Écrire une classe Broadcast qui respecte l'interface suivante:

```
public interface IBroadcast <T> {
  public void Subscribe();
     // utilisé pour l'enregistrement d'un thread recepteur
  public void Send(T i) throws InterruptedException;
  public T Receive() throws InterruptedException;
}
```

Fonctionnement : Prenons l'exemple d'un *broadcast* avec 10 récepteurs enregistrés. Les *threads* récepteurs appellent la méthode `Receive` lorsqu'ils sont prêts à recevoir la donnée, et ils sont bloqués par cet appel. Un *thread* émetteur indique qu'il veut envoyer une donnée en invoquant la méthode `Send`. La méthode `Send` attend jusqu'à ce que TOUTES les 10 tâches sont prêtes à recevoir la donnée, avant de les débloquer et de passer la donnée.

N.B.: On suppose qu'un seul *thread* fait appel à `Send`.

Utilisation :

Reprendre le `ColorMaster` de la section précédente en remplaçant l'objet `Multicast` par un objet de type `Broadcast`. Que constatez-vous ?