12/12/2018

Balise et satelites

Table des matières

[1. Introdiction 2](#_Toc532393754)

[2. Implementation 2](#_Toc532393755)

[A. Synchronisation des objets 2](#_Toc532393756)

[B. Le pattern dans la simulation 3](#_Toc532393757)

# Introdiction

Le but de ce mini projet est de créé une simulation. Cette simulation représente des balises sous-marines qui communique avec des satellites. Les balises se déplacent dans la mer puis remonte à la surface pour communiquer avec les satellites disponibles puis retournent dans la mer.

# Implementation

## Synchronisation des objets

### Présentation du pattern

Les balises ont besoins de communiquer avec les satellites. Pour cela j’ai utilisé le pattern « Observer/Observable » qui permet la synchronisation entre deux entités.

|  |
| --- |
| <<Abstract>>  Evenement  <<Observer>>  TestObserveur  <<Observable>>  TestObservable  EvenementTest |

Digramme de classe du pattern

|  |
| --- |
| // Creation des classes  TestObserveur observeur = new TestObserveur();  TestObservable observable = new TestObservable();  // enregistrement  observable.record(TestEvent.class, observeur);  // signal  TestEvent testEvent = new TestEvent(observable);  observable.signal(testEvent);  // signale reçus ?  assertTrue(testEvent.log.size() == 1);  assertTrue(testEvent.log.get(0).equals("class test.TestObservable -> class test.TestObserveur")); |

Cet exemple de code extrait de la classe de test « test.AnnoncerTest ». Présente mon implémentation du pattern.

1. Tout d’abord je crée les deux objets observer et observable
2. Puis l’observer s’abonne l’observable, pour cela je dois dire qu’elle est l’évènement qui intéresse mon observer. L’observer sera alors enregistré dans l’observable via une Map

|  |
| --- |
| Map<Class<? extends Evenement>, List<Observeur>> evenemens |

1. Puis l’observable vas créer un événement et lancer la méthode « signal »

|  |
| --- |
| public void signal(Evenement evt) {  List<Observeur> observeurs = this.evenemens.get(evt.getClass());  if (observeurs != null) {  Observeur[] arr = new Observeur[observeurs.size()];  arr = observeurs.toArray(arr);  for (Observeur observeur : arr) {  evt.traitement(observeur);  }  }  } |

1. Puis la méthode lancer la méthode « traitement » va être lancer en fonction pour les observers inscrits à l’événements.

|  |
| --- |
| abstract public void traitement(Cible cible);  @Override  public void traitement(TestObserveur testObserveur) {  this.log.add(this.getSource().getClass() + " -> " + testObserveur.getClass());  } |

### Le pattern dans la simulation

Dans l’application on utilise deux types d’évènement :

1. Le premier pour la communication entre les balises et les satellites car la balise ne connait pas le satellite et inversement

|  |
| --- |
| <<Abstract>>  Entite  Satellite  Balise |

1. Et entre la classe « Entite » et sa représentation graphique « EntiteVue » pour mettre à jour la position de la vue et ainsi redessiner l’entité avec la bonne position.

|  |
| --- |
| <<Observable>>  Entite  <<Observer>>  EntiteVue  SatelliteVue  BaliseVue |

## Stratégie de déplacement

Chaque entités (balises et satellites) se déplacés. Pour cela Elles doivent être capable de savoir si elles doivent capable de calculer leur prochaine position.

Mais la position suivante dépend du type de l’entité par exemple un satellite va juste se déplacer vers la droite alors que la balise va moter a la surface puis stagner et pour finir plonger dans la mer.