Plan de test

Hassani Yawoumihani - Rafii Ayoub - Randriamahefa Christelle – Besbes Alaaeddine

RESUME

Ce document est le plan de test du projet d'INFO832 sur le projet JDBFS

HISTORIQUE DU DOCUMENT

Version	Modification	Date	Auteur
1.0	Version initiale	28-jan-22	YH & AR & CR & AB
2.0	Version finale	17-avr-22	YH & AR & CR & AB

Table des matières

I. Introduction

1. But

Ce plan de test pour le projet de simulation de comportement qui a pour but de pallier les problèmes liés à l'acquisition de données réelles. Effectivement, ce simulateur se base sur les concepts des Simulateurs à Evènement Discret (SED). Celui-ci se décompose en trois niveaux. Il s'agira ici, de tester la gestion du temps, la gestion des actions, ainsi le simulateur même.

Pour ce faire, nous devrons tester les classes de chacun des packages :

- Package **Timer** qui permet la génération des laps de temps d'attente est composée des six classes suivantes :
  - ***DateTimer*** (générateur de temps basé à partir d'une liste de date)
  - **MergedTimer** (générateur de temps basé à partir de deux types de timers)
  - **OneShotTimer** (générateur de temps qui donnera qu'une seule valeur)
  - **PeriodicTimer** (générateur de temps basé sur une période)
  - **RandomTimer** (générateur de temps basé sur des lois probabilité)
  - **TimeBoundedTimer**

  Ainsi que d'une interface (***Timer***).

- Package Action qui permet de
- Package discreteBehaviorSimulator :

2. Portée

La première étape consiste à effectuer des tests unitaires afin de s'assurer du bon fonctionnement des méthodes de chaque package. Ainsi, nous pourrons isoler le comportement de la méthode à tester de tout facteur extérieur et vérifier la conformité par rapport à nos attendus. Les tests unitaires permettront de couvrir en priorité les cas nominaux, les cas d'erreurs et les cas aux limites.

Après avoir réalisé cette étape, nous mettrons en œuvre des tests fonctionnels qui nous permettent de nous assurer que simulateur sera fonctionnel une fois mis à disposition.

3. Documents

La table ci-dessous indique les documents disponibles pour l'activité de test.

Document	Version	Disponible	Notes
Sources	1.0.0	oui	Archive contenant le code source du projet
Conception	X	oui	Document expliquant le concept du projet
Travail à faire 2021-2022	X	oui	Enoncé donnant les directives du projet

Informations complémentaires projet	X	oui	Informations additionnelles
Sujet	X	oui	Enoncé du projet spécifiant les contraintes des outils à utiliser et les objectifs à réaliser

Table : Tableau des documents disponibles

II. Les exigences des tests

- Les tests seront de nature fonctionnelle et auront pour but de tester si le système est cohérent avec la réalité.
- Ne seront pas effectués de tests de charges.
- Les tests seront de type unitaire pour chacune de nos classes avec pour but de s'assurer de la cohérence des données attendues pour chacune unité.
    - Un seuil de couverture des tests de 80% est attendu.

III. La stratégie de test

- Pour chaque classe de chaque Package, on effectue une série de tests. Les plus utilisés sont les tests fonctionnels et les tests hors limite.

IV. Les tests

1. Package : timer

Classe : DateTimer

Tests fonctionnels

Id	Entrée	Attendu	Obtenu	Description
DT1	`[3,1,4] (objet Vector<Integer>`	+ hasNext() return [True,True,True] + next() return [3,1,4]	+ hasNext() return [True,True,True] + next() return [3,1,4]	Test d'un constructeur ayant comme paramètre un objet tree contennant des int en testant les valeurs retournées par hasNext() et next()

DT2	[3,1,4] (objet TreeSet)	+ hasNext() return True + next() return 3	+ hasNext() return [True,True,True] + next() return [3,1,4]	Test d'un constructeur ayant comme paramètre un objet tree contennant des int en testant les valeurs retournées par hasNext() et next()

Test hors limite

Id	Entrée	Attendu	Obtenu	Description
DT5	["2",1,4] Vector<Integer>	+ hasNext() return "exception" + next() return "Exception"	+ hasNext() return True + next() return "Exception"	Test d'un constructeur ayant comme paramètre un objet Vector <Integer> contenant un string et des int en testant les valeurs retournées par hasNext() et next()
DT6	["2",1,4] (objet TreeSet)	+ hasNext() return "exception" + next() return "Exception"	On peut pas initializer un DateTimer avec une TreeSet qui contient un String	Test d'un constructeur ayant comme paramètre un objet tree contenant un string et des int en testant les valeurs retournées par hasNext() et next()
DT7	[1.5,1,4] Vector<Integer>	+ hasNext() return "exception" + next() return "Exception"	+ hasNext() return True + next() return "Exception"	Test d'un constructeur ayant comme paramètre un objet Vector <Integer> contenant un string en testant les valeurs retournées par hasNext() et next()
DT8	[1.5,1.0,4.0] (objet TreeSet)	+ hasNext() return "exception" + next() return "Exception"	On peut pas initializer l'objet DateTimer avec une treeSet de double	Test d'un constructeur ayant comme paramètre un objet tree contenant un string en testant les valeurs retournées par hasNext() et next()

Test à la limite

Id	Entrée	Attendu	Obtenu	Description
DT3	[0,0,0]	+ hasNext() return True + next() return 0	+ hasNext() return True + next() return 0	Test d'un constructeur ayant comme paramètre un objet tree contenant des int en testant les valeurs retournées par hasNext() et next()
DT4	[0,1,4]	+ hasNext() return True + next() return 0	+ hasNext() return True + next() return 0	Test d'un constructeur ayant comme paramètre un objet tree contenant des int en testant les valeurs retournées par hasNext() et next()

Classe : MergedTimer

Tests fonctionnels

Id	Entrée	Attendu	Obtenu	Description
MT1	-Objet OneShotTimer avec 1 comme argument et un autre avec l'argument 2	+ hasNext() return True + next() return 3	+ hasNext() return True + next() return 3	Test du constructeur ayant comme paramètre deux objets de OneShotTimer en testant les valeurs retournées par hasNext() et next()
MT2	-Objet OneShotTimer avec 1 comme argument et un objet DateTimer avec [1,1,1] comme argument.	+ hasNext() return True + next() return "une exception"	+ hasNext() return True + next() return "une exception"	Test du constructeur ayant comme paramètre un objet OneShotTimer et un objet DateTimer en testant les valeurs retournées par hasNext() et next()
MT3	-Objet DateTimer avec [2,2,2] comme argument et un objet DateTimer avec [1,1,1] comme argument.	+ hasNext() return True + next() return 3	+ hasNext() return True + next() return 3	Test du constructeur ayant comme paramètre2 objets DateTimer en testant les valeurs retournées par hasNext() et next()
MT4	Objet MergedTimavec deux argument objet OneShotTimer les deux avec l'argument 1 OneShotTimer et un objet OneShotTimer avec 1 comme argument.	+ hasNext() return True + next() return 3	+ hasNext() return True + next() return 3	Test du constructeur ayant comme paramètre 2 objets OneShotTimer en testant les valeurs retournées par hasNext() et next()

Test hors limite

Id	Entrée	Attendu	Obtenu	Description
MT5	-Objet OneShotTimer avec 1 comme argument et un Objet null	+ hasNext() return "une exception" + next() return "une exception"	+ hasNext() return "une exception" + next() return "une exception"	Test du constructeur ayant comme paramètre un objetOneShotTimer et un objet null en testant les valeurs retournées par hasNext() et next()
MT6	-2 Objets null.	+ hasNext() return "une exception" + next() return "une exception"	+ hasNext() return "une exception" + next() return "une exception"	Test du constructeur ayant comme paramètre un objetOneShotTimer et un objet null en testant les valeurs retournées par hasNext() et next()

Classe : OneShotTimer

Tests fonctionnels

Id	Entrée	Attendu	Obtenu	Description
OST1	1 (entier positif)	+ HasNext() return True + next () return 1	+HasNext() return True + next () return 1	Test du constructeur ayant comme parametre un entier negatif

Test hors limite

Id	Entrée	Attendu	Obtenu	Description
OST3	« 1 » (string)	+HasNext() return "Une exception" + next() return "une exception"	Exception	Test du constructeur ayant comme parametre un string
OST4	-1	+HasNext() return "Une exception" + next() return "une exception"	HasNext() Return True Next return -1	Test du constructeur ayant comme parametre un entier negatif

Test à la limite

Id	Entrée	Attendu	Obtenu	Description
OST5	0 (entier)	+HasNext() return True + next() return 0	- HasNext() return True - next() return 0	Test du constructeur ayant comme parametre un entier null

Classe : PeriodicTimer

Tests fonctionnels

Id	Entrée	Attendu	Description
PT1	1	+getPeriod() retourne 1 +next() retourne 1 +hasNext() retourne True	Test du constructeur ayant comme parametre un entier
PT2	(1,2)	+getPeriod() retourne 1 +next() retourne 2 +hasNext() retourne True	Test du constructeur ayant comme parametre Deux entiers
PT3	(1, RT) 1: entier RT: objet RandomTimer sans entrées	+getPeriod() retourne 1 +next() retourne : 1+(RT.next()-RT.getMean()). +hasNext() retourne : True	Test du constructeur ayant comme parametre un entier et un objet random Timer

PT4	(1, 2, RT) 1: entier 2:entier RT: objet RandomTimer sans entrées	+getPeriod() retourne :1 +next() retourne : 1+(RT.next()-RT.getMean()). +hasNext() retourne : True	Test du constructeur ayant comme parametre deux entiers et un objet random Timer

Test à la limite :

Id	Entrée	Attendu	Description
PT5	0 : entier	+getPeriod() retourne 0 +next() retourne 0 +hasNext() retourne True	Test du constructeur ayant comme parametre un entier null
PT6	(0,0) 0:entier 0:entier	+getPeriod() retourne 0 +next() retourne 0 +hasNext() retourne True	Test du constructeur ayant comme parametre deux entier null
PT7	(0,RT) 0: entier RT: objet RandomTimer sans entrées	+getPeriod() retourne 0 +next() retourne 0 +hasNext() retourne True	Test du constructeur ayant comme parametre un entier null et un objet Random Timer sans entrées
PT8	(0,0,RT) 0: entier 0: entier RT: objet RandomTimer sans entrées	+getPeriod() retourne 0 +next() retourne 0 +hasNext() retourne True	Test du constructeur ayant comme parametre deux entiers null et un objet Random Timer sans entrées

Test hors limite :

Id	Entrée	Attendu	Description
PT9	-1 : entier	+getPeriod() retourne Exception +next() retourne Exception +hasNext() retourne Exception	Test du constructeur ayant comme parametre un entier négatif
PT10	(-1,-1) -1 : entier -1 : entier	+getPeriod() retourne Exception +next() retourne Exception +hasNext() retourne Exception	Test du constructeur ayant comme parametre deux entiers négatifs
PT11	(-1,Null) -1 : entier Null: null object	+getPeriod() retourne Exception +next() retourne Exception +hasNext() retourne Exception	Test du constructeur ayant comme parametre un entier négatif et un objet Null
PT12	(-1,-1,Null) -1 : entier -1 : entier Null: null object	+getPeriod() retourne Exception +next() retourne Exception +hasNext() retourne Exception	Test du constructeur ayant comme parametre deux entier négatifs et un objet Null

Classe : RandomTimer

Tests fonctionnels

Id	Entrée	Attendu	Description
RT1	(EXP,1.1) EXP:objet randomDistribution 1.1 : float	+ getDistribution() retourne “EXP” + getDistributionParam() retourne “ rate: 1.1” + getMean() retourne 1/1.1 +toString() retourne “EXP rate:1.1” +next() retourne nextTimeExp() +hasNext() retourne True	Tester si les différents méthodes et fonctions fonctionnent bien pour la loi exponentielle
RT2	(POISSON,2.3) POISSON:objet randomDistribution 2.3 :float	+ getDistribution() retourne “POISSON” + getDistributionParam() retourne “ rate: 2.3” + getMean() retourne 1/2.3 +toString() retourne “POISSON rate:2.3” +next() retourne (nextTimePoisson()) +hasNext() retourne True	Tester si les différents méthodes et fonctions fonctionnent bien pour la loi de poisson
RT3	(POSIBILIST,1,2) POSIBILIST:objet randomDistribution 1: entier 2: entier	+ getDistribution() retourne “POSIBILIST” + getDistributionParam() retourne "lolim:1 hilim:2 " + getMean() retourne 1.5 +toString() retourne “POSIBILIST LoLim: 1 HiLim: 2” +next() retourne (nextTimePosibilist()) +hasNext() retourne True	Tester si les différents méthodes et fonctions fonctionnent bien pour la loi possibiliste
RT4	(GAUSSIAN,1,2) GAUSSIAN:objet randomDistribution 1: entier 2: entier	+ getDistribution() retourne “GAUSSIAN” + getDistributionParam() retourne "lolim:1 hilim:2 " + getMean() retourne 1.5 +toString() retourne “GAUSSIAN LoLim: 1 HiLim: 2” +next() retourne (nextTimeGaussian()) +hasNext() retourne True	Tester si les différents méthodes et fonctions fonctionnent bien pour la loi gaussienne

Test hors limite

Id	Entrée	Attendu	Description
RT6	(GAUSSIAN,1.1) GAUSSIAN:objet randomDistribution 1.1: float	Throws Exception “Bad Timer constructor for selected distribution”	Tester si on aura des exceptions en mettant en paramètre float pour la loi gaussienne
RT7	(POSIBILIST,1.1) POSIBILIST:objet randomDistribution 1.1: float	Throws Exception “Bad Timer constructor for selected distribution”	Tester si on aura des exceptions en mettant en paramètre float pour la loi possibiliste
RT	(EXP,1,2) EXPobjet randomDistribution 1: entier 2: entier	Throws Exception “Bad Timer constructor for selected distribution”	Tester si on aura des exceptions en mettant en paramètre float pour la loi gaussienne

2. Package : Action

Classe : DiscretAction

Tests fonctionnels

Id	Entrée	Attendu	Description
DA1	**(clock, ‘’test avec getInstance’’, oneST)** On crée une instance clock de la classe Clock et une instance oneST de la classe OneShotTimer. Dans la classe DiscreteAction, on passe : (clock, ‘’test avec getInstance’’, oneST)	On s’attend à ce que integer et discreteA.getCurrentLapsTime() soient identiques.	On applique la méthode getInstance() sur l’instance clock puis on appelle la méthode assertEquals pour être sûr qu’il s’agit d’une instance unique.
DA2	**(clock1, ‘’test1 avec getInstance’’, oneST) (clock1, ‘’test1 avec getInstance’’, oneST)** On crée deux instances clock1 et clock2 de Clock, une instance oneST de OneShotTimer et deux instances discreteA1 et discreteA2 de DiscreteAction.	On s’attend à ce que discreteA1 et discreteA2 aient le même lapsTime qui est égal à 1.	On appelle la méthode assertEquals en prenants 1 et discreteA1.compareTo(discreteA2) en arguments.
DA3	**(clock,"test avec getInstance",oneST)**	Même instance	On applique la méthode getInstance() sur l’instance clock puis on appelle la méthode assertEquals(discreteA, discreteA.next())
DA4	**(clock,"test avec getInstance",oneST)**	+ hasNext() retourne True	On appelle la méthode assertTrue(discreteA.hasNext())
DA5	**(clock,"test avec getInstance",oneST)**	+ hasNext() retourne False	On appelle la méthode assertFalse(discreteA.hasNext())
DA6	**(clock,"test avec getInstance",oneST)**	Même instance	On applique la méthode assertEquals
DA7	**(clock,"test avec getInstance",oneST)**	Le currentLapsTime de discreteA est null.	On applique la méthode assertEquals en lui passant comme arguments (null, discreteA.getCurrentLapsTime())

DA8	**(clock,"test avec getInstance",oneST)**	Clock et discreteA.getObject() font référence au même objet.	On applique la méthode assertSame en lui passant comme arguments (clock, discreteA.getObject())

Classe : DiscretActionDependant

Tests fonctionnels

Id	Entrée	Attendu	Description
DAD1	**(clock,"test avec getInstance",oneST1)**	testAddDependence() retourne True	On utilise la méthode assertTrue() avec comme paramètre le résultat de la méthode hasNext() appliqué à l'instance discreteAD.
DAD2	**(clock,"test avec getInstance",oneST)**	testNextMethod() retourne True	On réalise trois assertEquals().
DAD3	**(clock,"test avec getInstance",oneST)** On passe 10 (un integer) en entrée	testSpendTime() retourne True	On appelle la méthode assertEquals() pour comparer l'integer et le résultat de discreteAD.getCurrentLapsTime().
DAD4	**(clock,"test avec getInstance",oneST)**	testUpdateTimeLaps() retourne True	On appelle la méthode assertEquals().
DAD5	**(clock1,"test avec getInstance",oneST)** **(clock2,"test avec getInstance",oneST)**	discreteAD1 et discreteAD2 ont la même valeur qui est 1 donc testCompareTo() retourne True	On appelle la méthode assertEquals pour vérifier si le résultat de discreteAD1.compareTo(discreteAD2) est bien 1.
DAD6	**(clock,"test avec getInstance",oneST1)**	DiscreteAD n'est pas vide donc testIsEmpty() retourne False.	On appelle la méthode assertFalse() pour vérifier que le résultat de discreteAD.isEmpty() est False.
DAD7	**(clock,"test avec getInstance",oneST1)**	discreteAD est vide donc testIsEmpty() retourne True	On appelle la méthode assertTrue() pour vérifier que le résultat de discreteAD.isEmpty() est True.
DAD8	**(clock,"test avec getInstance",oneST)**	testNext() retourne True	On appelle la méthode assertEquals() pour vérifier si discreteAD et result sont les mêmes.
DAD9	**(clock,"test avec getInstance",oneST)**	testHasNext() retourne True	On appelle la méthode assertTrue() pour vérifier que le résultat de discreteAD.hasNext() est bien True.
DAD10	**(clock,"test avec getInstance",oneST)**	testHasNext() retourne False	On appelle la méthode assertFalse() pour vérifier que le résultat de discreteAD.hasNext est bien False.

Classe : DiscretActionOnOffDependent

Tests fonctionnels

Id	Entrée	Attendu	Description
DAOD1	On crée une instance clock de la classe Clock, deux instances oneST_ON et oneST_OFF de la classe OneShotTimer et une instance discreteA_OnOff de la classe DiscreteActionOnOff Dependent	testNextAction() retourne True	On réalise deux assertEquals() successifs.
DAOD2	On passe 5 (un integer) en entrée.	testSpendTime() retourne True	On appelle la méthode assertEquals() pour vérifier que le résultat de discreteA_OnOff.getCurrentLapsTime() et integer sont égaux.
DAOD3	On crée des instances des classes Clock, OneShotTimer et DiscreteActionOnOff Dependent	testCompareTo() retourne True	On appelle la méthode assertEquals() pour vérifier que le résultat de discreteA_OnOff.compareTo() est bien égal à 1.
DAOD4	On crée des instances des classes Clock, OneShotTimer et DiscreteActionOnOff Dependent. La méthode gère également les exceptions NoSuchMethodException et SecurityException	testNext() retourne True	On appelle la méthode assertEquals().
DAOD5	On crée des instances des classes Clock, OneShotTimer et DiscreteActionOnOff Dependent.	testHasNext() retourne True	On appelle la méthode assertTrue() pour vérifier que le résultat de discreteA_OnOff.hasNext() est bien True.
DAOD6	On crée des instances des classes Clock, OneShotTimer et DiscreteActionOnOff Dependent.	testHasNext() retourne True	On appelle la méthode assertTrue() pour vérifier que le résultat de discreteA_OnOff.hasNext() est bien True.
DAOD7	On crée des instances des classes Clock, OneShotTimer et DiscreteActionOnOff Dependent.	testHasNext() retourne False	On appelle la méthode assertTrue() pour vérifier que le résultat de discreteA_OnOff.hasNext() est bien False.

3. Package : discreteBehaviorSimulator

Classe : Clock

Tests fonctionnels

Id	Entrée	Attendu	Description
C1	On crée deux instances d'une Clock c1	Nous nous attendons à ce qu'il s'agisse de la même instance.	On teste getInstance() pour s'assurer de la présence de l'unicité du singleton Clock.
C2	On passe la variable mo1 qui est de type MyObserver (voir Description pour détails)	On s'attend à ce que mo1 ait bien été ajouté en tant qu'observateur dans la Clock.	On teste la fonction AddObserver() pour s'assurer que l'ajout d'observateurs s'effectue correctement. On crée une classe MyObserver afin de pouvoir faire ce test indirectement car nous n'avons pas accès à la liste des observateurs.
C3	On passe la variable mo1 qui est de type MyObserver (voir Description pour détails)	On s'attend à ce que mo1 ait bien été supprimé de la liste des observateurs de la Clock.	On teste la fonction RemoveObserver() pour s'assurer que la suppression d'observateurs s'effectue correctement. On crée une classe MyObserver afin de pouvoir faire ce test indirectement car nous n'avons pas accès à la liste des observateurs.
C4	On passe FALSE en paramètre.	On s'attend à ce que la valeur de isVirtual soit FALSE.	On teste la méthode setVirtual() afin de s'assurer que la modification du paramètre s'est bien effectuée.
C5	On passe TRUE en paramètre de la méthode setVirtual()	On s'attend à ce que isVirtual() nous retourne TRUE.	On teste la méthode isVirtual() lorsque la valeur de virtual est vrai.
C6	On passe FALSE en paramètre de la méthode setVirtual().	On s'attend à ce que isVirtual() nous retourne FALSE.	On teste la méthode isVirtual() lorsque la valeur de virtual est false.
C7	On passe l'entier : 3	On s'attend à ce que tous les « observers » de la Clock aient leur nextClockChange = 3	On teste la méthode setNextJump() afin de s'assurer que la valeur nextClockChange de tous les « observers » de l'horloge est bien celle passé en paramètre.
C8	On passe l'entier : 5	On s'attend à ce que le « time » de la Clock soit incrémenté de 5 et que le temps d'attente de ces « observers » soit égale à ce nouveau temps.	On teste la méthode increase() pour s'assurer que l'incrémentation du temps de l'horloge a bien lieu et que le temps d'attente des « observers » est bien modifié en conséquence.

C9	On passe la valeur booléenne : True à la méthode setVirtual()	On s'attend à ce que la méthode getTime() nous retourne l'entier 0.	On teste l'accesseur getTime() afin de s'assurer qu'il retourne la bonne valeur dans le cas où « virtual » est true.
C10	On passe la valeur booléenne : False	On s'attend à ce que la méthode nous retourne le nombre de millisecondes depuis le 1er janvier 1970.	On teste l'accesseur getTime() afin de s'assurer qu'il retourne la bonne valeur dans le cas où « virtual » est false.
C11	/	On s'attend à ce que l'accès en lecture soit bloqué.	On teste la méthode lockReadAccess pour s'assurer que le verrou est bien bloqué lorsqu'on fait appel à cette méthode
C13	/	On s'attend à ce que l'accès en lecture soit débloqué.	On teste la méthode UnlockReadAccess pour s'assurer que le verrou est bien débloqué lorsqu'on fait appel à cette méthode
C14	/	On s'attend à ce que l'accès en écriture soit bloqué.	On teste la méthode lockWriteAccess pour s'assurer que le verrou est bien bloqué lorsqu'on fait appel à cette méthode
C15	/	On s'attend à ce que l'accès en écriture soit débloqué.	On teste la méthode lockWriteAccess pour s'assurer que le verrou est bien débloqué lorsqu'on fait appel à cette méthode
C16	/	toString() doit nous retourner « 0 ».	On teste la méthode toString().

Classe : ClockObserver (interface → pas de test)

Classe : DiscretActionSimulator

Tests fonctionnels

Id	Entrée	Attendu	Description
DAS1	On passe la variable a1 qui est de type	On s'attend à ce que a1 ait bien été ajouté en tant	On teste la fonction AddAction() pour s'assurer que

	MyAction (voir Description pour détails)	qu'action dans la liste des actions.	l'ajout d'actions s'effectue correctement. On crée une classe MyAction afin de pouvoir faire ce test indirectement car nous n'avons pas accès à la liste des actions.
DAS2	On passe une valeur positive 2 dans setNbLoop()	On s'attend à ce que la méthode affiche « DAS : 2 actions done ! »	On teste la méthode run() pour s'assurer que la simulation tourne bien.
DAS3	On passe une valeur positive -2 dans setNbLoop()	On s'attend à ce que le programme tourne sans fin.	On teste la méthode run() pour s'assurer que la simulation tourne bien.
DAS4	/	On s'attend à ce que la valeur de running soit passé true.	On teste la méthode start().
DAS5	/	On s'attend à ce que la valeur de running soit passé false.	On teste la méthode stop().
DAS6	/ (Sans avoir ajouté d'actions à la liste d'actions)	toString doit nous retourner la chaine de caractère suivante : "------------------ TestAuto :0 "-------------------- \n"	On teste la méthode toString().
DAS7	/	getRunning() doit nous renvoyer false.	On teste la méthode getRunning()

Classe : LogFormatter

Tests fonctionnels

Id	Entrée	Attendu	Description
LF1	On passe l'objet rec qui est de type LogRecord initialisé comme suit : LogRecord(null,null)		On teste la méthode format() pour s'assurer qu'elle nous renvoie bien la bonne chaîne de caractère.
LF2	On passe l'objet h qui est de type Handler	getHead() doit retourner « »	On teste la méthode getHead()
LF3	On passe l'objet h qui est de type Handler	getTail() doit retourner « »	On teste la méthode geTail()

V. Les outils

Outil	Version	Commentaire

Junit	5	
Sonar Lint		
Eclemma		
UML Lab plugin for Eclipse		

VI. Les ressources

1. Les ressources humaines

Ressources humaines		
Personne	**Rôle(s)**	**Responsabilité(s)**
Besbes Alaaeddine	Ecriture des tests des classes du package **Timer.**	Vérifier les tests et lancer les rapports SonarLint, JUnit du package **Timer**.
Rafii Ayoub	Ecriture des tests des classes du package **Timer.**	Vérifier les tests et lancer les rapports SonarLint, JUnit du package **Timer**.
Hassani Yawoumihani	Ecriture des tests des classes du package **discreteBehaviorSimulator.**	Vérifier les tests et lancer les rapports SonarLint, JUnit du package **action**.
Randriamahefa Christelle	Ecriture des tests des classes du package **action.**	Vérifier les tests et lancer les rapports SonarLint, JUnit du package **discreteBehaviorSimulator**.

Table : Tableau des ressources humaines

VII. Résultats

Lien GitHub : https://github.com/yawoumi/SimulatorTest

Les résultats de SonarLint sont disponibles dans le dossier SonarLintRapports du lien GitHub.

Les résultats de Junit sont trouvables dans le dossier JUnitRapports du lien GitHub.

Les résultats de Couverture de test sont dans le dossier RapportECLEMMA.