Projet Qualité Logiciel

« Java Discrete Behavior Framework Simulator »

Historique du document :

Version	Modification	Auteur	Date
0.1	Version initiale	Gabriel, Romain, Anas, Ahmed et Quentin	28/01/2022
0.2	Mise en place des plans de tests	Gabriel, Romain, Anas, Ahmed et Quentin	11/02/2022
1	Rendu final	Gabriel, Romain, Anas, Ahmed et Quentin	15/04/2022

Table des matières

I- Introduction

1) But

Le projet porte sur un simulateur de comportement. Ce dernier permet d'invoquer des méthodes d'objet à des instants définis par des lois de probabilité.

Les objectifs de projet sont les suivants :

1. Pise en main du projet – reverse Engineering
2. Plan de test et compte rendu de test
3. Reengineering

2) Portée

Les plans de tests comprendront les tests fonctionnels, aux limites et hors limites. Le timer, la gestion des actions et le simulateur en général sont à tester.

3) Documents

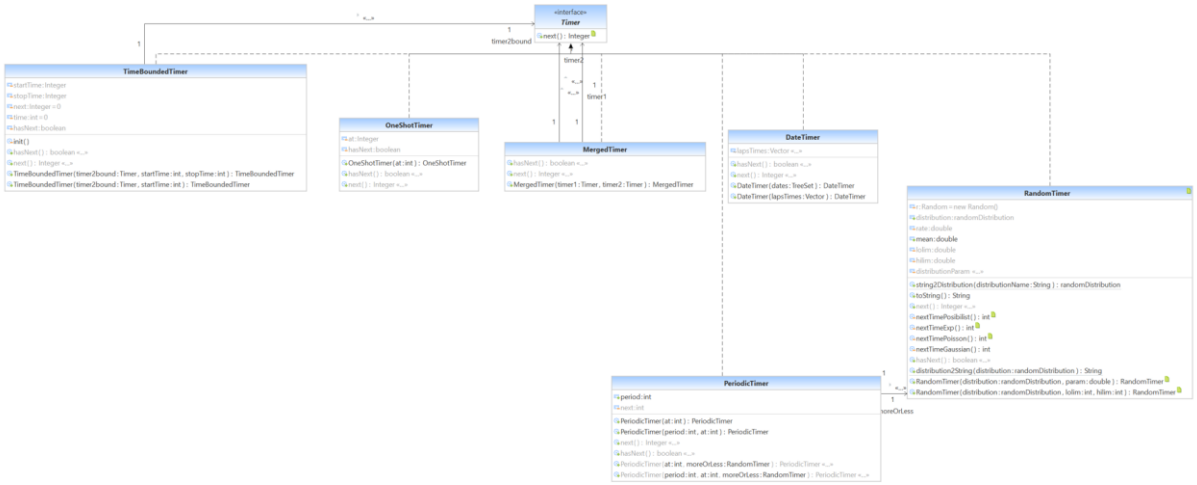
Liste des documents mis à disposition pour le projet :

Document	Version	Notes
Sujet	0.4	
Conception		Ne pas prendre en compte le dernier paragraphe
Questions et informations additionelles		
Sources		Code du projet

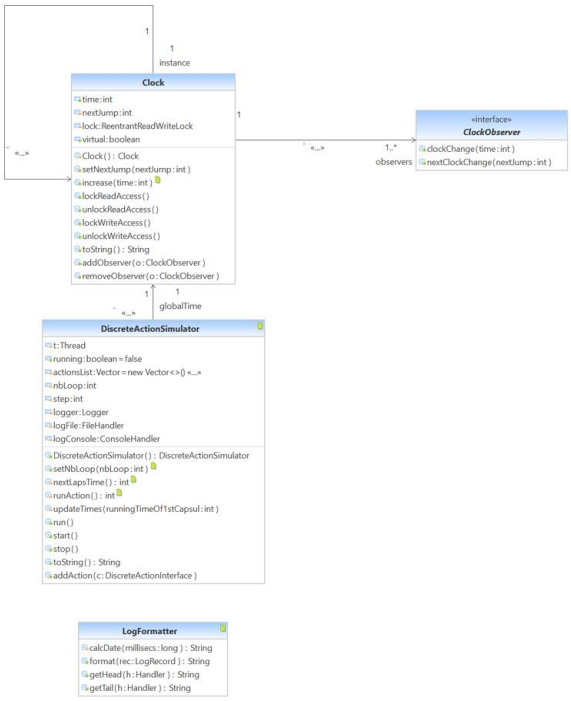
II- Prise en main du projet – Reverse Engineering

1) Diagramme de classe avant refactoring

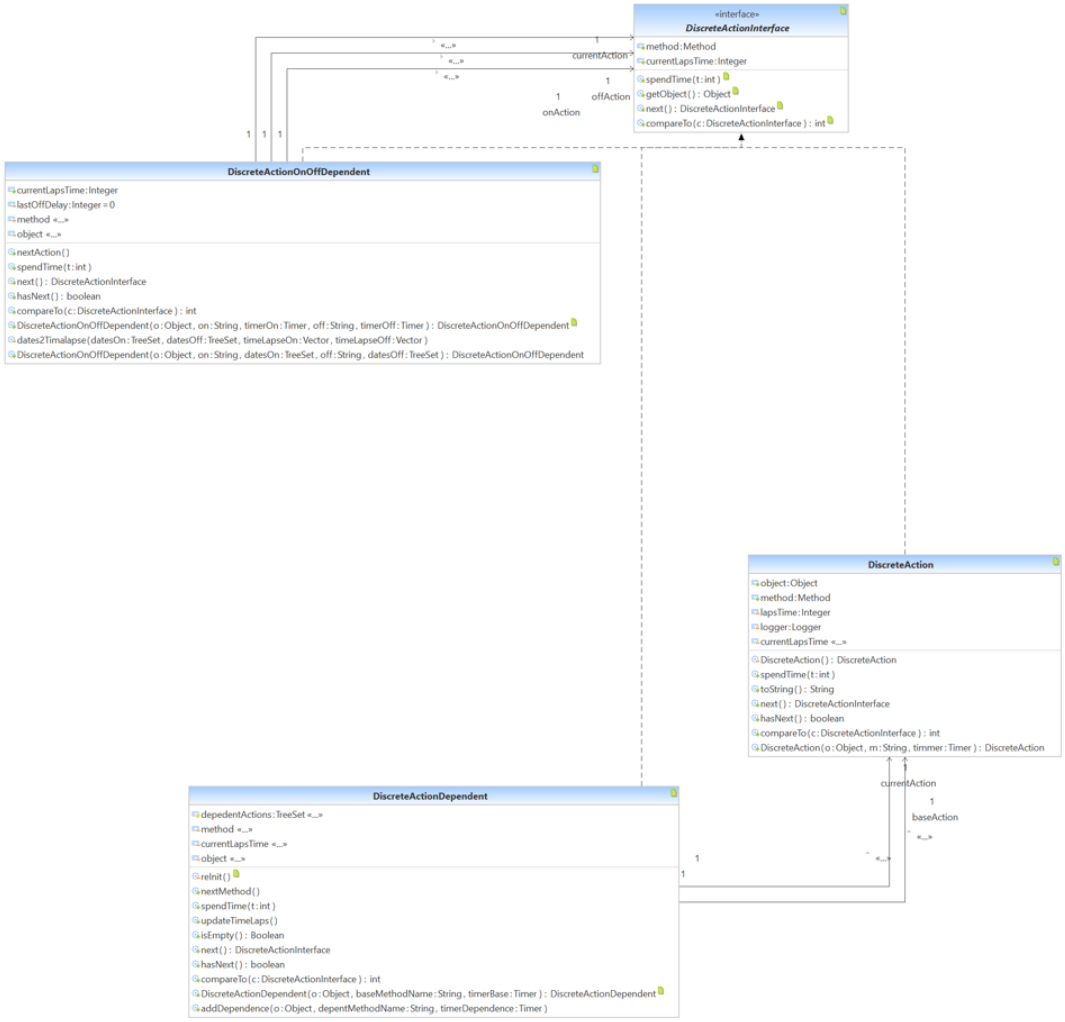
A) Diagramme de classe de « timer »



B) Diagramme de classe de « discreteBehaviorSimulator »



C) Diagramme de classe de « action »



III- Les exigences des tests

Liste des éléments qui doivent passer les tests :

Package action : tests fonctionnels et non fonctionnels

- DiscreteAction
  - DiscreteAction(Object, String, Timer)
  - compareTo(DiscreteActionInterface)
  - hasNext()
  - next()
  - spendTime(int)
- DiscreteActionDependent
  - DiscreteActionDependent(Object, String, Timer)
  - compareTo(DiscreteActionInterface)
  - flemme de continuer à détailler toutes les fonctions
- DiscreteActionOnOffDependent

Package timer : tests fonctionnels et non fonctionnels

- DateTimer
- MergedTimer
- OneShotTimer
- PeriodicTimer
- RandomTimer
- TimeBoundedTimer

Package discreteBehaviorSimulator : tests fonctionnels et non fonctionnels

- Clock
- DiscreteActionSimulator
- LogFormatter

IV- La stratégie de test

1) Les types de test

Donner les spécifications des tests (valeur seuil de coverage)

Plan de tests en cours :

Ahmed B	DiscreteAction, DiscreteActionDependant
Quentin P	LogFormater, DistrictAction Simulator
Anas E	DateTimer, OneShotTimer, PeriodicTimer, RandomTimer
Gabriel F	MergedTimer, TimeBoundedTimer
Romain S	Clock

A) Les tests unitaires

Classe DiscreteAction.java

ID	Entrée	Sortie attendue	Sortie obtenue	Description
t1	reflexionFields[0].get(discreteAction)	ost	ost	Création de l'objet dans le constructeur
t1	eflexionFields[2].get(discreteAction)	timmer	timmer	Creation du timmer dans le constructeur
t2	reflexionFields[3].get(discreteAction)	null	null	Attribution du lapstime à old si differente de 0
t2	reflexionFields[3].get(discreteAction)	2	2	Attribution du lapstime à old si egale à 0
t3	discreteAction.getMethod()	reflexionFields[1].get(discreteAction)	reflexionFields[1].get(discreteAction)	Récupération de la méthode du string
t3	ost.getClass().getDeclaredMethod("next", new Class<?>[0])	reflexionFields[1].get(discreteAction)	reflexionFields[1].get(discreteAction)	Récupération de la methode du string
t4	discreteAction.getCurrentLapsTime()	reflexionFields[0].get(discreteAction)	reflexionFields[0].get(discreteAction)	Récupération du nombre de lapstime
t5	,discreteAction.getObject()	reflexionFields[0].get(discreteAction)	reflexionFields[0].get(discreteAction)	Recupération de l'objet
t6	discreteAction1.compareTo(discreteAction2)	1	Erreur de constructeur	Comparaison du nombre de lapstime au nombre de lapstime courant
t7	discreteAction.toString()	"Object : "+ discreteAction.getObject().getClass().getName()+"\n" + " Method : "+discreteAction.getMethod().getName()+"\n" + " Stat. : "+ost+"\n" + " delay: "+ discreteAction.getCurrentLapsTime()	"Object : "+ discreteAction.getObject().getClass().getName()+"\n" + " Method : "+discreteAction.getMethod().getName()+"\n" + " Stat. : "+ost+"\n" + " delay: "+ discreteAction.getCurrentLapsTime()	Transforme l'objet en string
t8	discreteAction.next()	discreteAction	discreteAction	Associe au lapstime et au logger les attributs de l'objet
t9	discreteAction.hasNext()	True	True	Retourne vrai ou faux et dépend de de la valeur du timer
t9	discreteAction.hasNext()	True	True	Retourne vrai ou faux et depend de la valeur du timer

Classe DiscreteActionDependant.java

ID	Entrée	Attendu	Obtenu	Description
T1	clock.getClass().getDeclaredMethod("getInstance"	new Class<?>[0]),actionOnOff.getMethod()	new Class<?>[0]),actionOnOff.getMethod()	Fait une mise à jour du temps de lapstime
T1	clock.getClass().getDeclaredMethod("getTime"	new Class<?>[0]),actionOnOff.getMethod()	new Class<?>[0]),actionOnOff.getMethod()	Fait une mise à jour du temps de lapstime
T2	actionOnOff.getCurrentLapsTime()	null	null	Attribution du lapstime à old
T3	actionOnOff.getMethod()	clock.getClass().getDeclaredMethod("getTime", new Class<?>[0])	clock.getClass().getDeclaredMethod("getTime", new Class<?>[0])	Recupération de la methode du string
T4	actionOnOff.getCurrentLapsTime()	null	null	Recupération du nombre de lapstime
T5	actionOnOff.getObject()	clock	clock	Recupération de l'objet
T6	actionOnOff1.compareTo(actionOnOff2)	1	1	Comparaison du nombre de lapstime au nombre de lapstime courant
T7	result.getMethod()	clock.getClass().getDeclaredMethod("getInstance", new Class<?>[0])	clock.getClass().getDeclaredMethod("getInstance", new Class<?>[0])	Applique hasNext() si le lapstime est vide
T8	actionOnOff.hasNext()	True	True	Associe au lapstime et au logger les attributs de l'objet
T9	actionOnOff1.getObject()	clock	clock	Retourne vrai ou faux et dépend de la valeur du timer
T9	actionOnOff2.getObject()	clock	clock	Retourne vrai ou faux et dépend de la valeur du timer

Classe Clock.java

ID	Entrée	Résultat attendu	Résultat obtenu	Description	Rapport	Responsable
Cgi1	clock1 = clock2	True	True	Création de deux Clock et vérification que ce sont les mêmes		Romain S
Civ1	clock.setVirtual(true); clock.isVirtual() == true	True	True	Test de la fonction isVirtual dans les bonnes conditions lorsque this.virtual = True		Romain S
Civ2	clock.setVirtual(false); clock.isVirtual() == false	True	True	Test de la fonction isVirtual dans les bonnes conditions lorsque this.virtual = True		Romain S
Csv1	setVirtual(True)	True	True	Test de la fonction setVirtual avec comme argument True		Romain S
Csv2	setVirtual(False)	True	True	Test de la fonction setVirtual avec comme argument False		Romain S
Csnj1	setNextJump(0)	if not exception true	True	Test de la fonction setNextJump avec un entier		Romain S
Csnj2	setNextJump(-1)	if not exception true	True	Test de la fonction setNextJump avec une valeur égale à -1		Romain S
Csnj3	setNextJump(999999)	if not exception true	True	Test de la fonction setNextJump avec une valeur tendant vers ∞		Romain S
Cgt1	clock1.setVirtual(true); clock.getTime() == 0	True	True	Test de la fonction avec virtual a true		Romain S
Cgt2	clock1.setVirtual(false); clock1.getTime() == clock2.getTime()	True	True	Test de la fonction avec virtual a false		Romain S
Cura1	unlockReadAccess()	if not exception true	Fail	Test de la fonction unlockReadAccess	Problème avec le unlockRead Access	Romain S
Cts1	clock.toString() == "0"	True	True	Test de la fonction toString		Romain S

Classe DiscreteActionSimulator.java

ID	Entrée	Résultat attendu (this.nbLoop, this.step)	Résultat obtenu	Description	Responsable	Rapport
DASsnl1	5	(5, 1)	(5, 1)	Test fonctionnel d'attribution de nbLoop	Quentin P	Nécessite l'implémentation de getters, getNbLoop(self), getStep(self)
DASsnl2	0	(0, -1)	(0, -1)	Test aux limites d'attribution de nbLoop égal à 0	Quentin P	
DASsnl3	-5	Exception	(0,-1)	Test hors limites d'attribution de nbLoop négative	Quentin P	Ne devrait pas être possible
addAction(DiscreteActionInterface c)						

ID	Entrée	Résultat attendu (this.actionList)	Résultat obtenu	Description	Responsable	Rapport
DASaa1	act1	act1	Act1	Test fonctionnel de l'ajout d'une action à la liste des actions	Quentin P	
DASaa2	null	Exception		Test hors limites de l'ajout d'une action null à la liste des actions	Quentin P	
nextLapsTime()		actionList = [act1]	act1.getCurrentLapsTime() = 10			

ID	Entrée	Résultat attendu	Résultat obtenu	Description	Responsable	Rapport
DASnlt1		10		Test fonctionnel de l'affichage du lapsTime de la prochaine action	Quentin P	Fonction en privée, on ne peut pas tester directement
DASnlt2		Exception		Test fonctionnel de l'affichage du lapsTime alors qu'il n'y a pas de prochaine action	Quentin P	Fonction en privée, on ne peut pas tester directement
getRunning()						

ID	Entrée	Résultat attendu (this.actionList)	Résultat obtenu	Description	Responsable	Rapport
DASgr1		true		Test fonctionnel d'affichage de l'état de l'objet si il est en marche	Quentin P	Fonction en privée, on ne peut pas tester directement
DASgr2		false		Test fonctionnel d'affichage de l'état de l'objet si il est à l'arrêt	Quentin P	Fonction en privée, on ne peut pas tester directement
runAction()		actionList = [act1]	act1.getCurrentLapsTime() = 10			

ID	Entrée	Résultat attendu	Résultat obtenu	Description	Responsable	Rapport
DASra1		10		Test fonctionnel de l'affichage du temps d'exécution d'une action	Quentin P	Fonction en privée, on ne peut pas tester directement
DASra2		Exception		Test foncitonnel de l'affichage du temps d'exécution alors qu'il n'y a pas de prochaine action	Quentin P	Fonction en privée, on ne peut pas tester directement
updateTimes(int runningTimeOf1stCapsul)						

ID	Entrée	Résultat attendu	Résultat obtenu	Description	Responsable	Rapport
DASut1	10			Test fonctionnel de la mise à jour des time laps off de toutes les actions	Quentin P	Fonction en privée, on ne peut pas tester directement
DASut2	0			Test aux limites de la mise à jour des time laps off de toutes les actions avec une valeur égal à 0	Quentin P	Fonction en privée, on ne peut pas tester directement
DASut3	-4	Exception		Test hors limites de la mise à jour des time laps off de toutes les actions avec une valeur négative	Quentin P	Fonction en privée, on ne peut pas tester directement
DASut4	null	Exception		Test hors limites de la mise à jour des time laps off de toutes les actions avec une valeur null	Quentin P	Fonction en privée, on ne peut pas tester directement

DASta1	getRunning()	True	True	Test fonctionnel du démarrage	Quentin P	
DASto1	getRunning()	False	False	Test fonctionnel de l'arrêt	Quentin P	
DAStt1	setNbLoop(5)	0	0	Test aux limites, si on ajoute pas l'action au simulateur	Quentin P	

Classe LogFormatter.java

format (LogRecord rec)					
ID	Entrée	Résultat attendu	Résultat obtenu	Description	Responsable
LFf1	LogRecord(WARNING, "msg")	"02/02/2022: WARNING /n msg /n"		Test fonctionnel de l'affichage d'un message LogRecord	Quentin P
LFf2	LogRecord(WARNING, "")	02/02/2022: WARNING /n /n		Test aux limites de l'affichage d'un message vide LogRecord	Quentin P
calcDate(long millisecs)					
ID	Entrée	Résultat attendu	Résultat obtenu	Description	Responsable
LFcd1	1,64381E+12	2022.02.02 14:31:24.34		Test fonctionnel du calcul de la date actuelle	Quentin P
LFcd2	0	1970.01.01 00:00:00.00		Test aux limites du calcul de la date d'origine	Quentin P
LFcd3	-50	Exception		Test hors limites du calcul de la date avec une valeur négative	Quentin P
LFcd4	null	Exception		Test hors limites du calcul de la date avec une valeur null	Quentin P
getHead(Handler h)	null	« »	« »	Test fonctionnel de la récupération de la tête	Quentin P
getTail(Handler h)	null	« »	« »	Test fonctionnel de la récupération de la queue	Quentin P

Classe DateTimer.java

Testé par Anas.El

Classe	ID	Entrée	Attend	Obten	Description
DateTimer	ts 1	this.dateti me1.lapsTi mes	vec1	vec1	On vérifie la methode DateTimerVectorOfInteger() en spécifiant le vecteur en entrée
	ts2	L'iterator de vec1 "it"	True	True	On vérifie la méthode HasNext() qui retourne False si L'iterator "it" de vec1 a d'autres éléments à Itérer
	ts3	TreeSet "ts"	1	1	On vérifie la méthode DateTimerTreeSetOfInteger en spécifiant un TreeSet en entrée avec 3 élements [1,2,3] et on vérifie que le premier élement de l'iterator crée est "1" ce qui a été vrai.
	ts4	Iterator de vec1	7	7	On vérifie la méthode Next() qui retourne le premier élement de l'itérator it du vec1

Classe MergedTimer.java

Testée par Gabriel F.

ID	Entrée	Attendu	Obtenu	Description
mt1	mgTimer.has_next()	True	True	On test si la fonction has_next fonctionne pour un timer où elle est senser renvoyer true
mt2	mgTimer.has_next()	False	False	On test si la fonction has_next fonctionne pour un timer dont un des deux n'a pas de next
mt3	mgTimer.has_next()	False	False	On test si la fonction has_next fonctionne pour un timer dont les deux n'ont pas de next
mt4	mgTimer.next()	30	30	On test la récupération de prochain nombre, on choisis des timers périodiques de valeurs 10 et 20 est senser avoir la somme des deux
mt5	mgTimer.next()	null	null	On test la récupération de prochain nombre, on choisis des timers dont un n'a pas de nombre suivant
mt6	mgTimer.next()	null	null	On test la récupération de prochain nombre, on choisis des timers dont les deux n'ont pas de nombre suivant

Tous les tests de cette classe retournent le résultat attendu.

Classe OneShotTimer.java

Testé par Anas.El

OneShotTimer	ts5	oneShotTimer.hasNext(),reflexionFields[0].get(oneShotTimer)	at1=1,hasnext=true	True	Un vérifie la méthode OneShotTimerforint() qui attribu les valeurs à OneShotTimer.at et OneShotTimer.hasNext,en premier temps on remarque que les attributs sont en privé ainsi la nécessité de les rendre accessibles via la méthode reflexion sans changer le code d'origine pour vérifier que les résultats attendus et obtenus sont les memes ce qui a été vrai .
	ts6	hasNeoneShotTimer.hasNext()	True	True	On vérifie la méthode hasNext() qui retourne l'attribut OneShotTimer.hasNext ,la valeur de l'attribut attendu est la meme ainsi c'est vrai.
	ts7	oneShotTimer.next()	(1,null,false)	(1,null,false)	On vérifie la méthode next() qui retourne l'attribut OneShotTimer.at et attribut null à OneShotTimer.at et false à OneShotTimer.hasNext

Classe PeriodicTimer.java

Testé par Anas.El

ID	Entrée	Attendu	Obtenu	Description
ts8	ptI.getPeriod()	1	1	On test le constructeur PeriodicTimer qui prend en argument un entier,test est fonctionnel
ts9	ptIE.getPeriod() ptIP.getPeriod() ptIPOS.getPeriod() ptIG.getPeriod()	1	1	On test le constructeur PeriodicTimer qui prend en argument un entier et un RandomTimer ,test est fonctionnel
ts10	ptII.getPeriod()	1	1	On test le constructeur PeriodicTimer qui prend en argument deux entiers ,test est fonctionnel
ts11	ptIIE.getPeriod() ptIIP.getPeriod() ptIIPOS.getPeriod() ptIIG.getPeriod()	1	1	On test le constructeur PeriodicTimer qui prend en argument deux entiers et un RandomTimer,test est fonctionnel
ts12	ptIE.getPeriod() ptIP.getPeriod() ptIPOS.getPeriod() ptIG.getPeriod() ptIIE.getPeriod() ptIIP.getPeriod() ptIIPOS.getPeriod() ptIIG.getPeriod()	1	1	On teste le getter de PeriodicTimer sur differents instanciations de l'objet PeriodicTimer,la méthode getter est fonctionnel
ts13	ptI.next() ptIP.next()	1	1	On test la méthode next() sur différents instanciations de l'objet PeriodicTimer ,le test est fonctionnel
ts14	PtIE.hasNext() ptIP. hasNext () ptIPOS. hasNext () ptIG. hasNext () ptIIE. hasNext () ptIIP. hasNext () ptIIPOS. hasNext () ptIIG. hasNext ()	true	True	On test la méthode hasnext() sur différent instanciations de l'objet PeriodicTimer ,la réponse boolenne True prévu est réalisé,le test est fonctionnel

Classe RandomTimer.java

Testé par Anas.El

ID	Entrée	Prévu	Obtenu	Description
ts1511	randomTimer.string2Distribution("EXP")	randomDistribution.EXP	randomDistribution.EXP	On teste la méthode string2Distribution qui prend en paramétre une chaine de caractere contenant une loi et l'applique sur l'objet RandomTimer ,la loi appliqué et mis en paramétre est la meme , le test est fonctionnel.
ts1512	RandomTimer.distribution2String(randomDistribution.EXP)	"EXP"	"EXP"	On teste la méthode distribution2String qui retourne la loi du RandomTimer en chaine de caractére,on a initialisé un randomtimer avec la loi EXP et on a eu "EXP" en retour ,le test est fonctionnel.
ts1213	RandomTimer(randomDistribution.POISSON, 1.1)	RandomDistribution.POISSON 1.0 0.0 Double.NaN Double.POSITIVE_INFINITY	RandomDistribution.POISSON 1.0 0.0 Double.NaN Double.POSITIVE_INFINITY	On test le constructeur RandomTimer prenant en argument une loi de distribution et un parametre,après avoir comparé les variables de l'objet on a trouver que c'est les memes prévu,le test est fonctionnel
ts1514	RandomTimer(randomDistribution.POSIBILIST,1,1)	RandomDistribution.POSIBILIST Double.NaN 1.0 1.0 1.0	RandomDistribution.POSIBILIST Double.NaN 1.0 1.0 1.0	On test le constructeur RandomTimer prenant en argument une loi de distribution et deux entiers,après avoir comparé les variables de l'objet on a trouver que c'est les memes prévu,le test est fonctionnel
ts1516	randomTimer2.getDistribution()	POSIBILIST	POSIBILIST	On test le getter de la loi du RandomTimer,le test est fonctionnel
ts1517	RandomTimer randomTimer5 = new RandomTimer(randomDistribution.EXP,1.1); RandomTimer randomTimer6 = new RandomTimer(randomDistribution.POISSON, 1.1); RandomTimer randomTimer7 = new RandomTimer(randomDistribution.GAUSSIAN ,1,1);	"rate: 1.1" "mean: 1.1" lolim: 1.0 hilim: 1.0	"rate: 1.1" "mean: 1.1" lolim: 1.0 hilim: 1.0	On test la méthode getdistributionParameter qui retourne les paramétres du Random Timer selon la loi appliqué on a appliqué le test sur 3 différents RandomTimer avec des lois différentes ,et les paramétres étaient correctes comme prévus ,le test est fonctionnel
ts1518	RandomTimer(randomDistribution.POISSON, 1.1)	1.1	1.1	On test le getter de la variable mean sur un radomtimer qui applique la loi de POISSON , le résultat est correcte ,test fonctionnel
ts1519	RandomTimer randomTimer9 = new	"EXP rate:1.0"	"EXP rate:1.0"	On test la méthode toString qui retourne une chaine de caractére décrivant les paramétres de du

	RandomTimer(random Distribution.EXP,1); RandomTimer randomTimer10 = new RandomTimer(random Distribution.POISSON, 1.1); RandomTimer randomTimer11 = new RandomTimer(random Distribution.GAUSSIAN ,1,1); RandomTimer randomTimer12 = new RandomTimer(random Distribution.POSIBILIS T,1,1);	"POISSON mean:1.1" "GAUSSIAN LoLim:1.0 HiLim:1.0" "POSIBILIST LoLim:1.0 HiLim:1.0"	"POISSON mean:1.1" "GAUSSIAN LoLim:1.0 HiLim:1.0" "POSIBILIST LoLim:1.0 HiLim:1.0"	RandomTimer seon la loi appliqué,on a pris 4 RandomTimer avec 4 lois différentes et le résultat est correcte avec celui prévu,le test est fonctionnel

Classe TimeBoundedTimer.java

Testée par Gabriel F.

ID	Entrée	Attendu	Obtenu	Descritpion
tbt1	tbTimer.has_next()	True	True	On test le fonctionnement classique du timer, pour un timer périodique qui permet d'etre dans l'interval
tbt2	tbTimer.has_next()	False	False	On test le fonctionnement classique du timer, pour un timer périodique dont le deuxième valeure est égale à celle de fin
tbt3	tbTimer.has_next()	False	False	On test le fonctionnement classique du timer, pour un timer périodique qui permet d'etre dans l'interval, quand il en sors
tbt4	tbTimer.has_next()	False	Erreur dans le constructeur	On test le timer avec un one shot qui est avant l'interval
tbt5	tbTimer.has_next()	False	False	On test le timer avec un one shot qui est après l'interval
tbt6	tbTimer.next()	3,1,1,null	3,1,1,null	On test les resultats d'un timer périodique de période 1, pour un intervalle de 3 à 5
tbt7	tbTimer.next()	null	Erreur dans le constructeur	On test le timer avec un one shot qui est avant l'interval
tbt8	tbTimer.next()	null	Null	On test le timer avec un one shot qui est après l'interval
tbt9	new TimeBoundedTimer(timer, 2, 1)	Erreur	Rien	On test si le timer revoie bien une erreur quand on choisit un temps de fin plus petit que le temps de départ

2) Les outils

Outils	Version	Commentaire
Java	8	
Eclipse IDE	4.22.0	
JUnit5		
EclEmma Java Code Coverage	3.1.3	
SonarLint	7.2.1	

3) Comptes-rendus de tests

Après exécution des tests, nous obtenons le coverage ECLMA suivant :

Element	Coverage	Covered Instru...	Missed Instruct...	Total Instructio...
INFO832-Projet-Qualite	83,4 %	3 815	760	4 575
src	83,4 %	3 815	760	4 575
discreteBehaviorSimulator	59,1 %	502	347	849
DiscreteActionSimulator.java	52,6 %	337	304	641
Clock.java	70,9 %	105	43	148
LogFormatter.java	100,0 %	60	0	60
action	57,5 %	328	242	570
DiscreteActionDependent.jav	0,0 %	0	150	150
DiscreteActionOnOffDepende	71,7 %	152	60	212
DiscreteAction.java	84,6 %	176	32	208
tests	96,5 %	2 343	86	2 429
ClockTest.java	75,0 %	138	46	184
TimeBoundedTimerTest.java	79,1 %	129	34	163
PeriodicTimerTest.java	98,4 %	300	5	305
DiscreteActionSimulatorTest.j	99,8 %	537	1	538
DateTimerTest.java	100,0 %	82	0	82
DiscreteActionDependentTes	100,0 %	351	0	351
DiscreteActionTest.java	100,0 %	295	0	295
LogFormatterTest.java	100,0 %	29	0	29
MergedTimerTest.java	100,0 %	142	0	142
OneShotTimerTest.java	100,0 %	79	0	79
RandomTimerTest.java	100,0 %	261	0	261
timer	88,3 %	642	85	727
RandomTimer.java	77,4 %	291	85	376
DateTimer.java	100,0 %	62	0	62
MergedTimer.java	100,0 %	37	0	37
OneShotTimer.java	100,0 %	24	0	24
PeriodicTimer.java	100,0 %	110	0	110
TimeBoundedTimer.java	100,0 %	118	0	118

On peut voir que nos tests font appels à une majorité du code (83,4% fichiers de tests inclus).
On peut voir que les classes des Timers ont quasiment toutes étés testée à 100%.

Le fichier complet et interactif est disponible sur notre GitHub (le lien est à la fin de ce document).

4) Rapport de bugs

La documentation de conception parle d'un CycleTimer qui n'a pas été implémenté. Une autre fonctionnalité qui n'a pas été implémentée est "DiscretActionCycle" qui est mentionnée dans la documentation de conception.

TimeBoundedTimer :

Lors des tests de cette classe, nous avons relevés plusieurs bugs et erreurs.

Tout d'abords, nous obtenons des erreurs sur les tests tbt4 et tbt7 qui portent tous deux sur le test d'un timer qui se finis avant d'avoir commencé. Pour ceux deux tests, nous nous retrouvons avec une erreur lors de la création du timer. Or cette erreur est une erreur Java et non pas une erreur retournée par la classe elle-même.

Pour les tests tbt2 et tbt3, qui test les hasNext sur et après la limite du timer, on se retrouve avec un échec de l'assertion. En effet, la classe devrait renvoyer false mais renvoie True. Ceci est un bug à corriger lors du refactoring.

Le test tbt9 regarde si le constructeur renvoie une erreur lorsque le temps de départ est inférieur au temps de fin, ce qu'il ne fait pas.

DiscreteActionSimulator :

Lors des tests de cette classe, nous avons aussi relevés plusieurs bugs et choix qui ne nous semblent pas logiques.

Nous pensons qu'il n'est pas logique de pouvoir initialiser nbLoop avec un entier négatif. Or, si on le fait, le programme va juste s'exécuter comme si nbLoop était égal à 0. Il faudrait que la condition sur nbLoop encadre aussi les valeurs négatives en testant s'il est négatif en premier.

Plusieurs fonctions sont en privée, pour les tester il faut passer par l'utilisation classique du simulateur et de sa fonction run() se qui complique les tests.

LogFormatter :

Au même titre que pour la classe DiscreteActionSimulator, la fonction calcDate() est en privée. Heureusement cette fois ci, la classe étant moins complexe, les tests fonctionnels permettent de tester l'entièreté de la classe.

V- Les ressources

1) Les ressources humaines

Ressources humaines		
Personne	Rôle(s)	Responsabilité(s)
Ahmed Bekkari		
Anas El Abedelalaoui		Test des classes Timer
Gabriel Forray		Ecrire la Javadoc Nettoyage du code avec SonarLint
Quentin Pascal		Mise en forme du compte rendu Test de LogFormatter et de DiscreteActionSimulator
Romain Souchon		Retro Engineering des diagrammes UML Test de Clock

2) GitHub

Le lien vers notre GitHub est le suivant :

https://github.com/ForrayGabriel/INFO832-Projet-Qualite

Il contient le projet de base, ce rapport, les fichiers de tests et le rapport Eclma.