

Projet CSC4102 : Gestion des clefs dans un hôtel

LE GLAUNEC Alexis & CANTO Guillem
Année 2019–2020 — 17 janvier 2020

Table des matières

1	Spécification.....	3
1.1	Diagrammes de cas d'utilisation.....	3
1.2	Priorités, préconditions et postconditions des cas d'utilisation.....	4
2	Préparation des tests de validation.....	5
2.1	Tables de décision des tests de validation.....	5
3	Conception.....	7
3.1	Liste des classes.....	7
3.2	Diagramme de classes.....	8
3.3	Diagrammes de séquence.....	9
4	Fiche des classes.....	12
4.1	Classe GestionClefsHotel.....	12
4.2	Classe Badge.....	12
5	Diagrammes de machine à états et invariants.....	13
6	Préparation des tests unitaires.....	14
6.1	Classe Chambre.....	14
6.2	Classe Badge.....	14
	Annexe : Algorithmes Badge.....	16
1	ALGO1C1 : constructeur.....	16
2	ALGO2C1 : estDonnéAUnClient.....	16

1 Spécification

1.1 Diagrammes de cas d'utilisation

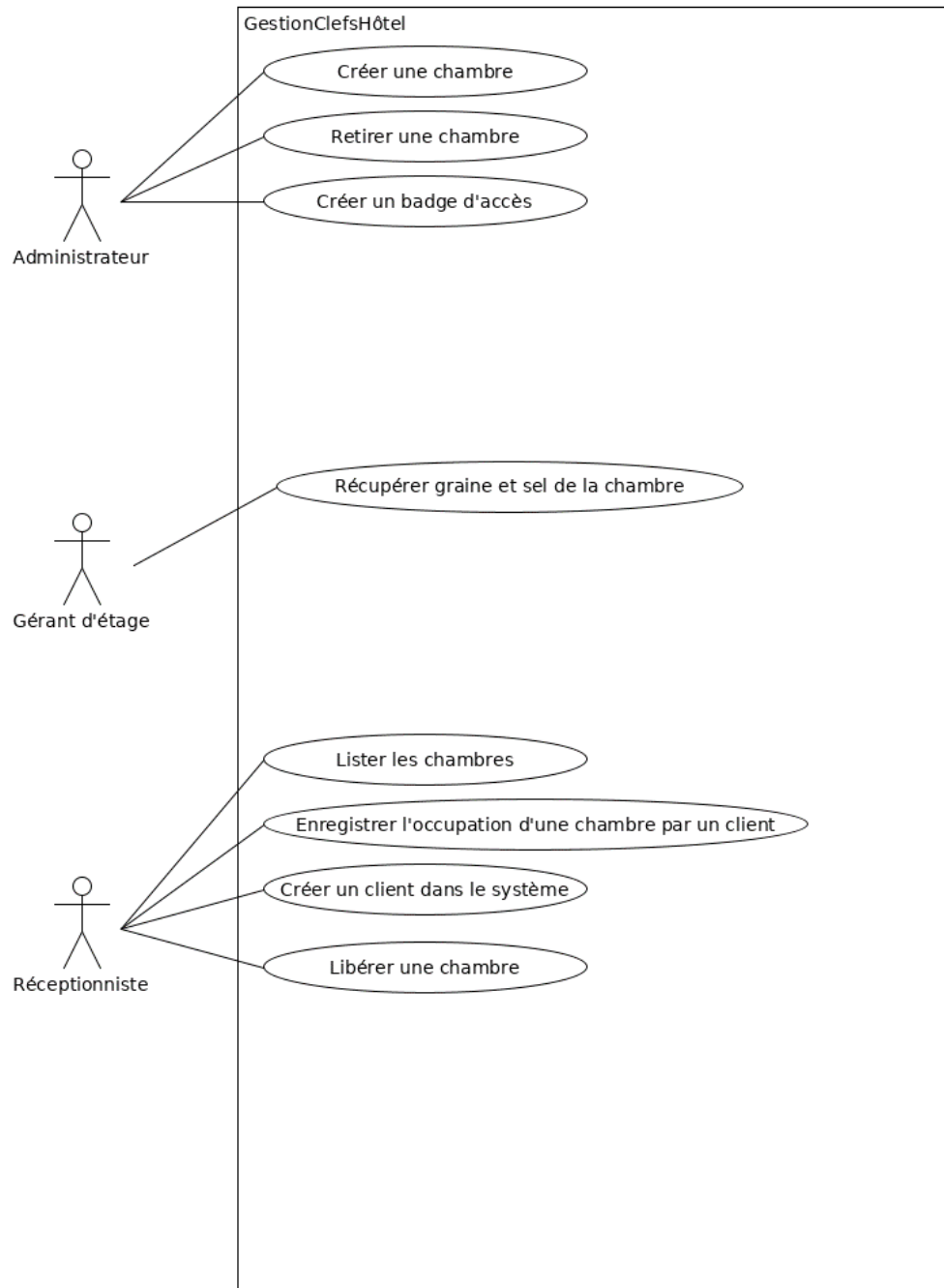


FIG. 1: Diagramme de cas d'utilisation

1.2 Priorités, préconditions et postconditions des cas d'utilisation

- Créer une chambre : **priorité haute**
 - précondition : identifiant/code de la chambre bien formé (non null et non vide) \wedge chambre avec ce code inexistante \wedge graine pour la génération des clefs bien formée (non null et non vide)
 - postcondition : chambre avec ce code existant
- Libérer une chambre : **priorité haute**
 - précondition : identifiant/code de la chambre bien formé (non null et non vide) \wedge chambre avec ce code existante \wedge chambre précédemment non libre
 - postcondition : clés sur le badge associé effacées \wedge badge à la réception \wedge client n'occupe plus de chambre \wedge chambre libre
- Enregistrer l'occupation d'une chambre par un client : **priorité haute**
 - précondition : identifiant/code du client bien formé (non null et non vide) \wedge identifiant/code de la chambre bien formé (non null et non vide) \wedge identifiant du client dans le système \wedge chambre avec ce code existante \wedge chambre avec ce code libre \wedge client n'occupe pas de chambre \wedge badge à la réception \wedge identifiant/code du badge bien formé (non null et non vide)
 - postcondition: chambre avec ce code occupée \wedge badge prêté au client \wedge client occupe une chambre
- Créer un badge d'accès : **priorité haute**
 - précondition : identifiant/code du badge bien formé (non null et non vide) \wedge badge avec cet identifiant inexistant
 - postcondition: badge avec cet identifiant existant
- Créer un client dans le système : **priorité haute**
- Récupération d'une graine et d'un sel : **priorité haute**
- Retirer une chambre : **priorité basse**
- Lister les chambres : **priorité moyenne**

2 Préparation des tests de validation

2.1 Tables de décision des tests de validation

Numéro de test	1	2	3	4
Identifiant/code de la chambre bien formé (\neq null \wedge \neq vide)	F	T	T	T
Graine pour la génération des clefs bien formée (\neq null \wedge \neq vide)		F	T	T
Chambre inexistante avec ce code			F	T
Création acceptée	F	F	F	T
Chambre avec ce code existant	F	F	F	T
Nombre de jeux de test	2	2	1	2

TAB. 1: Cas d'utilisation « créer une chambre »

Numéro de test	1	2	3	4	5	6	7	8	9
Identifiant/code du client bien formé (non null et non vide)	F	T	T	T	T	T	T	T	T
Identifiant/code de la chambre bien formé (non null et non vide)		F	T	T	T	T	T	T	T
Identifiant du client dans le système			F	T	T	T	T	T	T
Chambre avec ce code existante				F	T	T	T	T	T
Chambre avec ce code libre					F	T	T	T	T
Client n'occupe pas de chambre						F	T	T	T
Badge à la réception							F	T	T
Identifiant/code du badge bien formé (non null et non vide)								F	T
Enregistrement réussi (occupation créée)	F	F	F	F	F	F	F	F	T
Chambre avec ce badge occupée (occupation de chambre non null)									T
Badge prêté au client (occupation de badge non null)									T
Client occupe une chambre (occupation de client non null)									T
Nombre de jeux de test	2	2	2	1	1	1	1	2	4

TAB. 2: Cas d'utilisation «enregistrer l'occupation d'une chambre par un client»

Numéro de test	1	2	3	4	5
identifiant/code du client bien formé (non null et non vide)	F	T	T	T	T
chambre avec ce code existante		F	T	T	T
chambre précédemment non libre			F	T	T
badge rendu au réceptionniste				F	T
Libération acceptée	F	F	F	F	T
clés sur le badge associé effacées					T
badge à la réception					T
client n'occupe plus de chambre					T
chambre libre					T
Nombre de jeux de test	2	1	1	1	5

TAB. 3: Cas d'utilisation «libérer une chambre»

3 Conception

3.1 Liste des classes

À la suite d'un parcours des diagrammes de cas d'utilisation et d'une relecture de l'étude de cas, voici une première liste de classes avec quelques attributs :

- GestionClefsHotel (la façade),
- Chambre — id, graine, sel, Occupation
- Util (classe déjà programmée) — attribut de classe TAILLE_CLEF, et méthodes de classes genererUneNouvelleClef et clefToString.
- Badge — id, clef1 (byte[]), clef2 (byte[]), Occupation
- Occupation — id, , dateDebut, dateFin,
- Client — id, Nom, Prénom, Occupation

3.2 Diagramme de classes

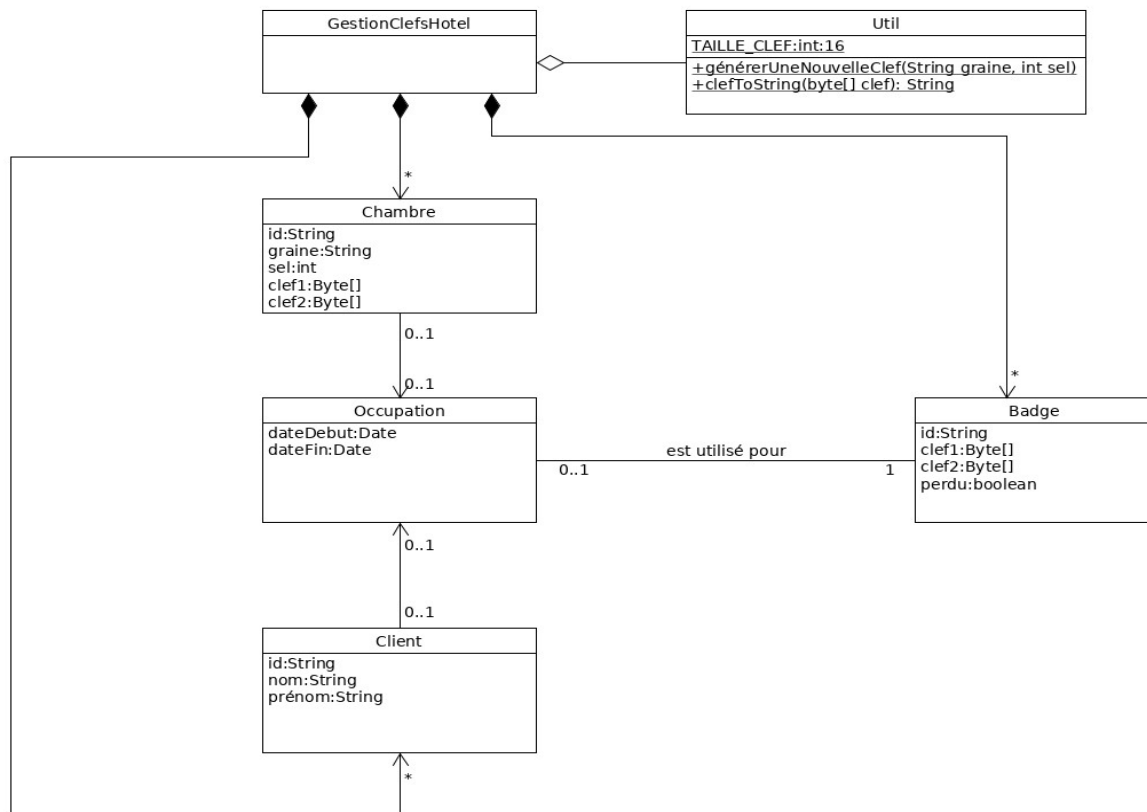


FIG. 1: Diagramme de classes

3. Diagrammes de séquence

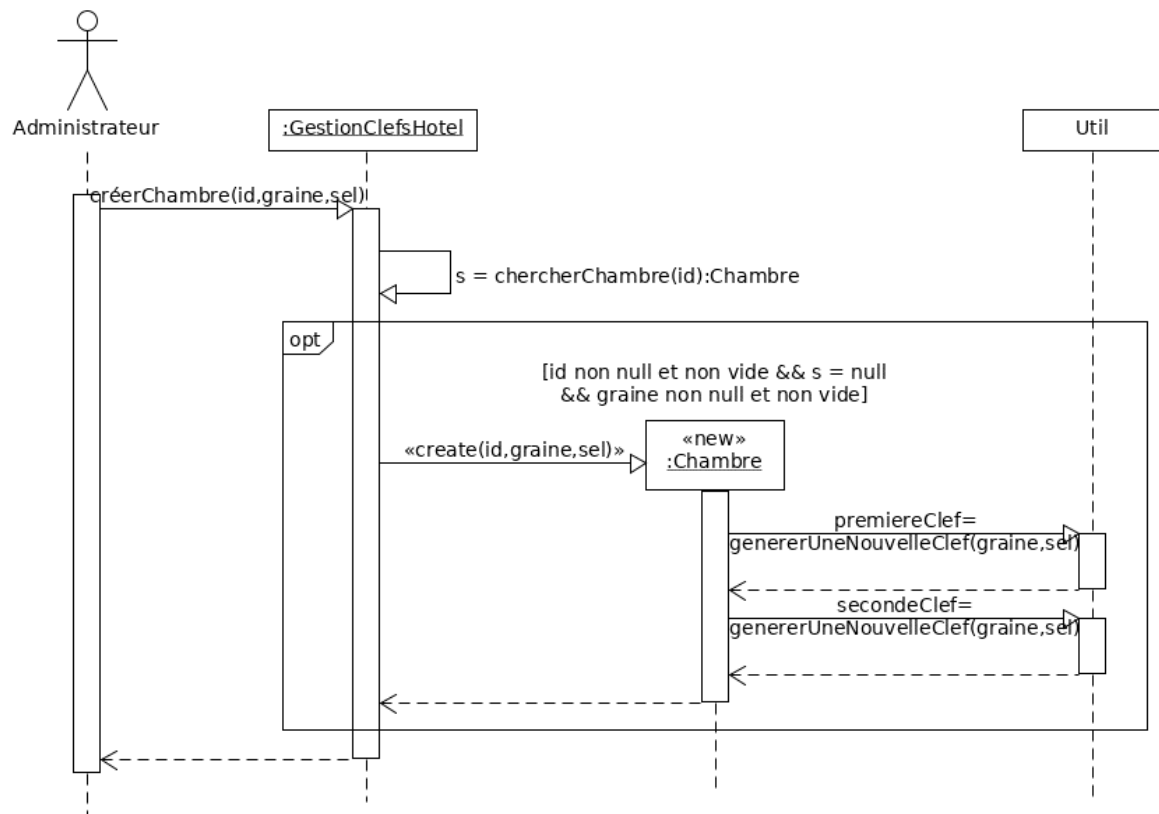


FIG. 1: Cas d'utilisation «créer une chambre»

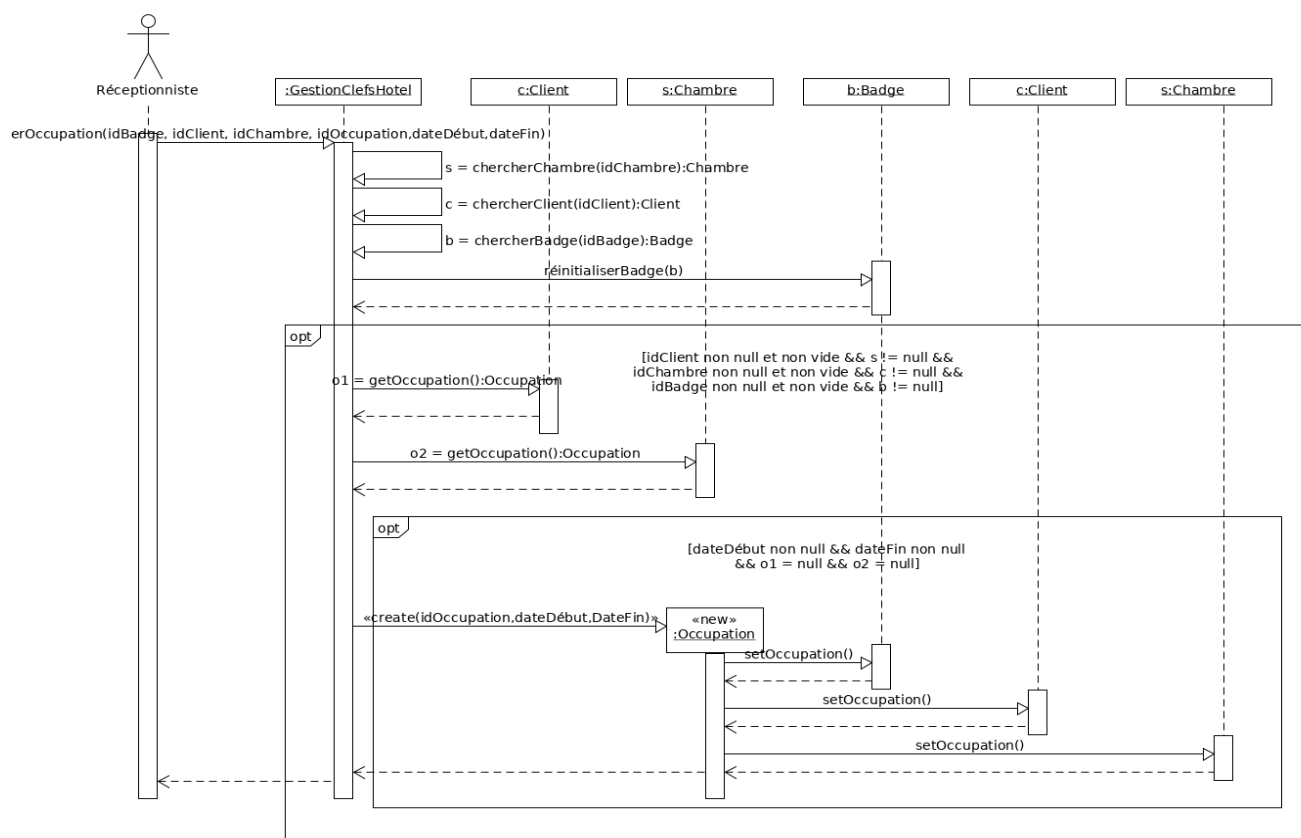


FIG. 2: Cas d'utilisation «enregistrer l'occupation d'une chambre par un client»

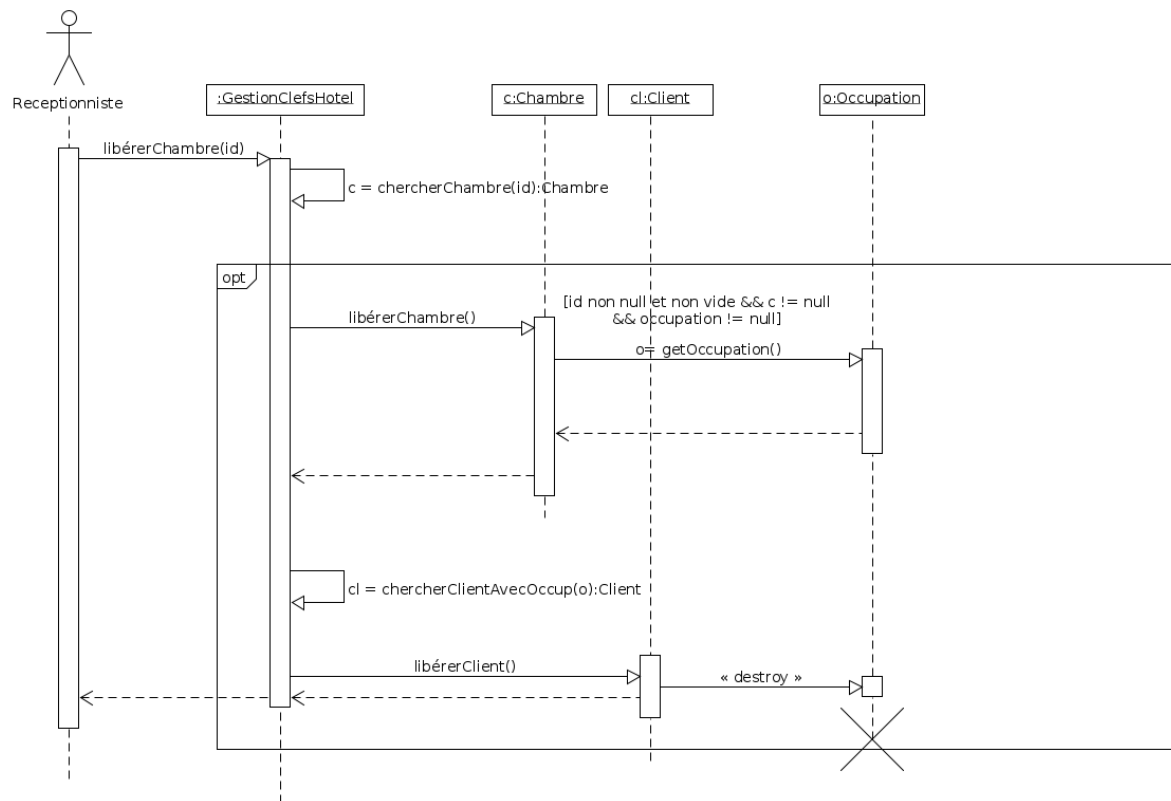


FIG. 3: Cas d'utilisation «libérer une chambre»

4 Fiche des classes

4.1 Classe GestionClefsHotel

GestionClefsHotel
<- attributs « association » -> - chambres : collection de @Chambre - clients : collection de @Client - badge : collection de @Badge
<- constructeur -> + GestionClefsHotel()
<- operations « cas d'utilisation » -> + créerChambre(String code, String graine, int sel) + retirerChambre(String code)
+ créerClient(String Nom, String Prenom) + listerChambre() + créerOccupation(String idBadge, String idChambre, String idClient, DateTime dateDébut, DateTime dateFin) + récupérerGraineSelChambre(String idChambre) + créerBadge(String code) + libérerChambre(String idChambre) + getChambre(String idChambre) + getClient(String idClient) + getOccupation(String idChambre) + getOccupation(String idClient) + getBadge(String idBadge)

4.2 Classe Badge

Badge
<- (private) attributs « association » -> - id : String - clef1 : tableau de Bytes - clef2 : tableau de Bytes - occupation : @Occupation
<- (public) constructeur -> + Badge(String identifiant)
<- operations « cas d'utilisation » -> + getOccupation() : Occupation + estDonnéAUnClient(String idOccupation) + rendreBadge(String idBadge) + setClef1(String valeur) + setClef2(String valeur) + getClef2() + invariant() : Booléen

5 Diagrammes de machine à états et invariants

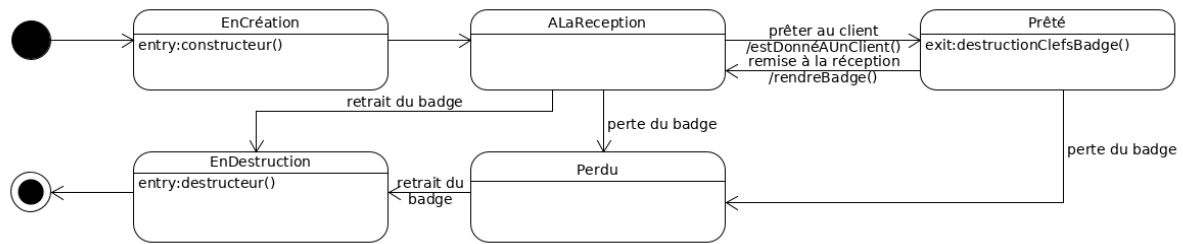


FIG. 1: Diagramme de machine à états «Badge d'accès»

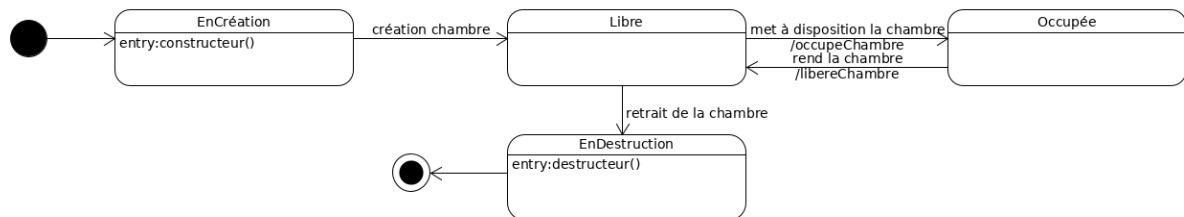


FIG. 2: Diagramme de machine à états «Chambre»

6 Préparation des tests unitaires

6.1 Classe Chambre

Invariant : $(\text{occupation} == \text{null}) \vee (\text{occupation} != \text{null})$

6.2 Classe Badge

Invariant : $((\text{occupation} != \text{null}) \wedge \neg \text{perdu}) \vee ((\text{occupation} == \text{null}) \wedge \neg \text{perdu}) \vee ((\text{occupation} == \text{null}) \wedge \text{perdu}) \wedge (\text{clef1} != \text{null}) \wedge (\text{clef1.length} == \text{Util.TAILLE_CLEF}) \wedge (\text{clef2} != \text{null}) \wedge (\text{clef2.length} == \text{Util.TAILLE_CLEF})$

Numéro de test	1	2	3	4	5
chambre != null	F	T	T	T	T
clef1' = clef2	F	F	T	T	T
clef2' ≠ clef2	F	F	F	T	T
ch.getSel() ' = ch.getSel() + 1	F	F	F	F	T
invariant	T	T	T	T	T
Levée exception	OUI	NON	NON	NON	NON
Objet crée	T	T	T	T	T
Nb jeux de test	1	1	1	1	1

Tableau 1 : Méthode « estDonnéAUnClient » de la classe Badge

Numéro de test	1	2	3
identifiant \neq null \wedge \neg vide	F	T	T
occupation == null	F	F	T
invariant		T	T
Levée exception	OUI	OUI	NON
Objet crée	F	F	T
Nb jeux de test	2	1	1

Tableau 2 : Méthode « constructeur» de la classe Badge

Annexe : Algorithmes Badge

1 ALGO1C1 : constructeur

badge(String identifiant)

si identifiant == null ou identifiant = "" alors lever une exception

si occupation != null alors lever une exception

id := identifiant

perdu := false

assert invariant()

2 ALGO2C1 : estDonnéAUnClient

estDonnéAUnClient(Chambre ch)

si chambre == null alors lever une exception

clef1 := Clef2

clef2:= générerUneNouvelleClef(ch.graine, ch.sel + 1)

assert invariant()