

**Exemple de Refresh :**

[ Hero ]	Ynovator	LIFE :	1	STAMINA :	0	PROTECTION :	0.00	BUFF:	0.00	(ALIVE)
ARMOR	1 : Dragon Slayer Leggings(10.2)			2:empty		3:empty				TOTAL:10.2
RINGS	1:[Ring of Death, 0.00]			2:[Dragon Slayer Ring, 14.00]						
CONSUMABLE : Uncle Greg's spicy Maroilles burger [40 life point(s)]										
WEAPON : Basic Sword (min:5 max:10 stam:20 dur:100)										
BAG : SmallBag [ 0 items   0/10 kg ]										
<ul style="list-style-type: none"> <li>(empty)</li> </ul>										
[ Lycanthrope ]	Lycanthrope	LIFE :	10	STAMINA :	10	PROTECTION:	30.00	BUFF:	0.00	(ALIVE)
WEAPON : Bloody Claw (min:50 max:150 stam:5 dur:100)										

## 2. LearningSoulsGame : tests exceptions

Nous allons maintenant créer une méthode qui va nous permettre de travailler/tester les exceptions du jeu.

2.1. Créez la méthode **testExceptions()** qui :

- ✎ Remplace l'arme du héros par **null**
- ✎ Lance le **fight1v1**

2.2. Dans le **main** :

- ✎ Appelez **testExceptions()** juste après **init()**
- ✎ Dans le jeu, lancez une attaque du héros : action (1)
- ✎ Que se passe-t-il ?
- ✎ Remontez la pile d'appels pour découvrir la ligne qui a généré l'erreur. Demandez au professeur si vous ne comprenez pas exactement ce qui s'est passé.

Notre jeu plante donc lorsqu'un personnage attaque alors qu'il n'a pas d'arme.

Nous allons mettre à profit la détection de ce problème pour faire 2 choses :

- ✎ éviter que le programme plante en faisant en sorte qu'il continue de s'exécuter de manière cohérente (dans ce cas précis, on peut considérer qu'une attaque sans arme produira un coup à 0)
- ✎ mettre à profit cette erreur pour que le programmeur et, *in fine*, l'utilisateur soit prévenu qu'un personnage est en train de se battre sans arme !

En Java, cela peut être réalisé en s'appuyant sur le mécanisme d'exceptions.

## 3. WeaponNullException

3.1. Créez la classe **lsg.exceptions.WeaponNullException**

- ✎ en tant que simple sous classe d'**Exception**
- ✎ et dont le message sera « **No Weapon !** » (utilisation du **super** dans le constructeur)

## 4. Character

4.1. Au tout début de **attackWith**, vérifiez que l'arme du personnage n'est pas nulle, et levez une **WeaponNullException** si elle l'est.

**ATTENTION** : pour toutes les méthodes modifiées, y compris dans la suite de cette partie, ne pas oublier de mettre à jour la javadoc en y ajoutant les exceptions potentiellement levées.

4.2. Répercutez l'exception au niveau de **attack()** pour qu'elle remonte la pile d'appel sans avoir été gérée.

#### 4.3. Gérez l'exception au niveau de **LearningSoulsGame : createExhaustedHero**

Étant donné que la méthode fournit une arme au héros juste avant l'appel à **attack**, il est peu probable que l'exception arrive réellement jusque là. Si tel est le cas, cela signifie qu'il y a vraiment un problème dans le programme... Entourez donc simplement l'appel à **attack** par un try/catch : la partie catch contenant simplement un **printStackTrace()** de l'exception (pour débogage).

#### 4.4. Gérez l'exception au niveau de **LearningSoulsGame : fight1V1**

Entourez le **attack** avec un try/catch en faisant en sorte qu'en cas d'exception :

- ✍ L'exécution continue avec une valeur de l'attaque égale à 0 (pour ce coup)
- ✍ Un message apparaît dans la console pour avertir l'utilisateur du fait qu'il frappe sans arme équipée.

**ATTENTION** : dans les lignes suivantes il y a un appel à **agressor.getWeapon().getName()** qui générera un **NullPointerException** si l'arme est **null**. Remplacez cet appel par un simple **agressor.getWeapon()**.

Appel à **testExceptions()** après les modifications :

<

#### 4.5. Dans **repairWeaponWith**, remplacez le return (en cas de **weapon == null**) par une levée de **WeaponNullException**.

Cette modification impacte la méthode **use**. Cependant **use** est plus lié à l'utilisation d'un consommable que d'une arme, et si une exception doit être levée par **use**, nous souhaitons qu'elle soit plus précise que **WeaponNullException**.

Dans un premier temps, nous allons donc nous contenter d'entourer l'appel à **repairWeaponWith** par un try/catch qui ne fait rien d'autre qu'un **printStackTrace** : nous y reviendrons plus tard afin de transformer cette **WeaponNullException** en un nouveau type d'exception plus adapté.

## 5. WeaponBrokenException

### 5.1. Créer la sous-classe **lsg.exceptions. WeaponBrokenException**

- ✍ Avec un constructeur **WeaponBrokenException(Weapon weapon)** (weapon représentant l'arme liée à cette exception)
- ✍ Le message de l'exception sera « **nomArme is broken !** »
- ✍ Implémentez un getter permettant de récupérer l'arme liée à l'exception.

5.2. Mettez l'exception en œuvre dans **Character : attackWith**.

5.3. Laissez **WeaponBrokenException** se propager au travers de **Character : attack**.

5.4. Gérez l'exception dans **learningSoulsGame : fight1v1** :

Entourez le **attack** avec un try/catch en faisant en sorte qu'en cas d'exception :

- ✎ L'exécution continue avec une valeur de l'attaque égale à 0 (pour ce coup)
- ✎ Un message apparait dans la console pour avertir l'utilisateur du fait qu'il frappe avec une arme cassée.

NB : il y aura donc maintenant plusieurs clauses **catch** qui permettent de différencier les exceptions levées par l'attaque, et d'agir de manière adéquate selon le cas (message adapté).

Ex. de fight1V1 en ayant équipé le héros avec une pelle cassée :

[ Hero ]	Ynovator	LIFE :	100	STAMINA :	50	PROTECTION :	10.20	BUFF:	14.00	(ALIVE)
ARMOR	1 : Dragon Slayer Leggings(10.2)		2:empty		3:empty		TOTAL:10.2			
RINGS	1:[Ring of Death, 0.00]		2:[Dragon Slayer Ring, 14.00]							
CONSUMABLE : Uncle Greg's spicy Maroilles burger [40 life point(s)]										
WEAPON : Pelle casse (min:0 max:100 stam:2 dur:0)										
BAG : SmallBag [ 0 items   0/10 kg ]										
• (empty)										
[ Lycanthrope ]	Lycanthrope	LIFE :	10	STAMINA :	10	PROTECTION:	30.00	BUFF:	0.00	(ALIVE)
WEAPON : Bloody Claw (min:50 max:150 stam:5 dur:100)										
Hero's action for next move : (1) attack   (2) consume > 1										
WARNING : Pelle cassee is broken !										
Ynovator attacks Lvcanthrope with Pelle casse (min:0 max:100 stam:2 dur:0) (ATTACK : 0   DMG : 0)										

## 6. StaminaEmptyException

Créer la sous-classe **lsg.exceptions. StaminaEmptyException**

- ✎ Avec un constructeur **StaminaEmptyException ()**
- ✎ Le message de l'exception sera « **no stamina !** »

6.1. Mettez l'exception en œuvre dans **Character : attackWith**.

6.2. Laissez **StaminaEmptyException** se propager au travers de **Character : attack**.

6.3. Gérez l'exception dans **learningSoulsGame : fight1v1** :

Entourez le **attack** avec un try/catch en faisant en sorte qu'en cas d'exception :

- ✎ L'exécution continue avec une valeur de l'attaque égale à 0 (pour ce coup)
- ✎ Un message apparait dans la console pour avertir l'utilisateur du fait qu'il n'a plus de stamina.

Hero's action for next move : (1) attack   (2) consume > 1 ACTION HAS NO EFFECT: no more stamina !!!
---

## 7. ConsumeException





Nous allons maintenant mettre en place le mécanisme d'exception au niveau des consommables.

Les erreurs liées à l'utilisation des consommables sont plus complexes que ce que nous avons vu précédemment : le consommable peut être **null**, avoir une **capacity** à 0, ou, dans le cas d'un **repairKit**, être utilisé sur une arme qui est **null**. Pour distinguer ces différents cas, nous allons implémenter un jeu d'exceptions spécialisées, mais qui sont toutes liées à l'utilisation d'un consommable.

Nous allons donc créer la classe **ConsumeException**, qui représente une erreur liée à un consommable, et des sous-classes précisant cette erreur.


Par choix de conception, et pour en faciliter le traitement, nous allons faire en sorte que les méthodes ne puissent pas lever une **ConsumeException** qui ne soit pas spécialisée. Pour ce faire, nous allons déclarer **ConsumeException** en tant que classe abstraite (même si toutes ses méthodes sont opérationnelles !), ce qui empêchera de l'instancier sans passer par une sous-classe.

### 7.1. Créez la classe abstraite **lsg.exceptions.ConsumeException** :





-  Sous classe abstraite d'**Exception**
-  Un constructeur :  
**public ConsumeException(String message, Consumable consumable)**
-  Un getter permettant de récupérer le consommable associé à cette exception
-  (NB : vous pouvez vérifier qu'il est impossible d'instancier cette classe)

## 8. ConsumeNullException

### 8.1. Créer la sous-classe **lsg.exceptions.ConsumeNullException**

-  Le message de l'exception sera « **Consumable is null !** »

### 8.2. Levez l'exception quand le consommable en paramètre est **null** dans :


-  **Character : use**
-  **Character : drink**
-  **Character : eat**
-  **Weapon : repairWith**

NB : dans **use**, si le consommable est **null**, les tests **instanceof** seront faux, et l'exception ne sera pas générée par **drink**, **eat**, etc... Il fallait donc bien aussi faire la vérification au début de **use**.

### 8.3. Laissez **ConsumeNullException** se propager au travers de toutes les méthodes impactées de **Character** (**repairWeaponWith**, **consume**, etc.)

**ATTENTION** : dans **fastUseFirst**, il faut retirer le test « **if(found != null)** » qui ne sert plus à rien et empêche l'exception d'être levée...

#### 8.4. Gérez l'exception dans **learningSoulsGame : fight1v1** :


-  On affichera un message dans la console pour prévenir l'utilisateur du fait qu'il n'a pas équipé de consommable :

Exemple avec **hero.setConsumable(null)** dans **testExceptions()** :

[ Hero ]	Ynovator	LIFE :	100	STAMINA :	50	PROTECTION :	10.20	BUFF:	14.00	(ALIVE)
ARMOR	1 : Dragon Slayer Leggings(10.2)			2:empty		3:empty			TOTAL:10.2	
RINGS	1:[Ring of Death, 0.00]			2:[Dragon Slayer Ring, 14.00]						
CONSUMABLE	: null									
WEAPON	: Basic Sword (min:5 max:10 stam:20 dur:100)									
BAG	: SmallBag [ 0 items   0/10 kg ]									
	• (empty)									
[ Lycanthrope ]	Lycanthrope	LIFE :	10	STAMINA :	10	PROTECTION:	30.00	BUFF:	0.00	(ALIVE)
WEAPON	: Bloody Claw (min:50 max:150 stam:5 dur:100)									
Hero's action for next move : (1) attack   (2) consume > 2										
IMPOSSIBLE ACTION : no consumable has been equipped !										

### 9. ConsumeEmptyException


#### 9.1. Créer la sous classe **lsg.exceptions.ConsumeEmptyException**

-  Le message de l'exception sera « **Consumable is empty !** »

#### 9.2. Dans **Consumable**, au tout début de **use**, vérifiez si la capacité du consommable vaut 0, et levez **ConsumeEmptyException** si tel est le cas.

#### 9.3. Laissez **ConsumeEmptyException** se propager au travers de toutes les méthodes impactées de **Character** (**consume**, etc.)

#### 9.4. Gérez l'exception dans **learningSoulsGame : fight1v1** :

-  On affichera un message dans la console pour prévenir l'utilisateur du fait qu'il veut utiliser un consommable vide.

NB : il y aura donc maintenant plusieurs clauses **catch** qui permettent de différencier les sous-types de **ConsumeException**, et d'agir de manière adéquate selon le cas (message adapté).

Test en jeu avec **play** et en appuyant 2 fois en suivant sur l'action (2) :

[ Hero ]	Ynovator	LIFE :	100	STAMINA :	50	PROTECTION :	10.20	BUFF:	14.00	(ALIVE)
ARMOR	1 : Dragon Slayer Leggings(10.2)			2:empty		3:empty			TOTAL:10.2	
RINGS	1:[Ring of Death, 0.00]			2:[Dragon Slayer Ring, 14.00]						
CONSUMABLE	: Uncle Greg's spicy Maroilles burger [0 life point(s)]									
WEAPON	: Basic Sword (min:5 max:10 stam:20 dur:100)									
BAG	: SmallBag [ 0 items   0/10 kg ]									
	• (empty)									
[ Lycanthrope ]	Lycanthrope	LIFE :	10	STAMINA :	10	PROTECTION:	30.00	BUFF:	0.00	(ALIVE)
WEAPON	: Bloody Claw (min:50 max:150 stam:5 dur:100)									
Hero's action for next move : (1) attack   (2) consume > 2										
Ynovator eats Uncle Greg's spicy Maroilles burger [0 life point(s)]										
ACTION HAS NO EFFECT : Uncle Greg's spicy Maroilles burger is empty !										

#### 9.5. Optionnel : dans **testException**, vous pouvez remplir le sac avec divers consommables (vides) et tester les méthodes **FastDrink**, **FastEat**, **FastRepair**.

## 10. ConsumeRepairNullWeaponException

Dans **Character** : **use**, nous avons laissé de côté la gestion de **WeaponNullException**. Le bloc catch autour de **repairWeaponWith** est atteint lorsque l'on tente d'utiliser un **RepairKit** sur une arme **null** : nous allons maintenant lever l'exception appropriée.

10.1. Créer la sous- classe

**lsg.exceptions.ConsumeRepairNullWeaponException**

✎ Le message de l'exception sera « **Trying to repair null weapon !** »

10.2. **Character** : **use**

✎ remplacer le **printStackTrace** que nous avons mis par la levée de l'exception appropriée : une instance de **ConsumeRepairNullWeaponException**.

Laissez **ConsumeRepairNullWeaponException** se propager au travers de toutes les méthodes impactées de **Character...** **SAUF** dans **fastEat** et **fastDrink** : en effet, cette exception n'a aucun sens dans ce contexte. Il s'agira donc de la gérer « syntaxiquement » avec des try/catch ne faisant rien, cette exception ne pouvant en réalité jamais être déclenchée dans ces méthodes.

10.3. Gérez l'exception dans **learningSoulsGame** : **fight1v1** :

✎ On affichera un message dans la console pour prévenir l'utilisateur du fait qu'il veut utiliser un kit sans avoir équipé d'arme.

Ex. **testException** avec un consommable de type **RepairKit** et une arme mise à **null** :

[ Hero ]	Ynovator	LIFE :	100	STAMINA :	50	PROTECTION :	10.20	BUFF:	14.00	(ALIVE)
ARMOR	1 : Dragon Slayer Leggings(10.2)			2:empty		3:empty			TOTAL:10.2	
RINGS	1:[Ring of Death, 0.00]			2:[Dragon Slayer Ring, 14.00]						
CONSUMABLE : Repair Kit [10 durability point(s)]										
WEAPON : null										
BAG : SmallBag [ 0 items   0/10 kg ]										
• (empty)										
[ Lycanthrope ]	Lycanthrope	LIFE :	10	STAMINA :	10	PROTECTION:	30.00	BUFF:	0.00	(ALIVE)
WEAPON : Bloody Claw (min:50 max:150 stam:5 dur:100)										
Hero's action for next move : (1) attack   (2) consume > 2										
IMPOSSIBLE ACTION : no weapon has been equipped !										

## 11. Problèmes de sac

11.1. On pourra remarquer que la méthode **Bag.transfer** plante (**NullPointerException**) si un des 2 sacs passés en paramètre est **null**.

Corrigez le problème en faisant un simple **return** en tout début de méthode (on ne fait rien) si c'est le cas.

11.2. La méthode **Character** : **setBag** a le même problème.

Gérez ce problème au mieux sans lever d'exception.

NB : on affichera « **null** » à la place du nom du sac **null**.

## Tests dans **testExceptions** :


[ Hero ]	Ynovator	LIFE :	100	STAMINA :	50	PROTECTION :	10.20	BUFF:	14.00	(ALIVE)
ARMOR	1 : Dragon Slayer Leggings(10.2)			2:empty		3:empty				TOTAL:10.2
RINGS	1:[Ring of Death, 0.00]			2:[Dragon Slayer Ring, 14.00]						
CONSUMABLE	: : Uncle Greg's spicy Maroilles burger [40 life point(s)]									
WEAPON	: Basic Sword (min:5 max:10 stam:20 dur:100)									
BAG : SmallBag [ 0 items   0/10 kg ]										
• (empty)										
[ Lycanthrope ]	Lycanthrope	LIFE :	10	STAMINA :	10	PROTECTION:	30.00	BUFF:	0.00	(ALIVE)
WEAPON	: Bloody Claw (min:50 max:150 stam:5 dur:100)									
Ynovator changes SmallBag for null										
[ Hero ]	Ynovator	LIFE :	100	STAMINA :	50	PROTECTION :	10.20	BUFF:	14.00	(ALIVE)
ARMOR	1 : Dragon Slayer Leggings(10.2)			2:empty		3:empty				TOTAL:10.2
RINGS	1:[Ring of Death, 0.00]			2:[Dragon Slayer Ring, 14.00]						
CONSUMABLE	: : Uncle Greg's spicy Maroilles burger [40 life point(s)]									
WEAPON	: Basic Sword (min:5 max:10 stam:20 dur:100)									
BAG : null										

## 12. NoBagException

### 12.1. Créer la sous classe **lsg.exceptions.NoBagException**

 Le message de l'exception sera « **No Bag !** »

### 12.2. Mettez l'exception en œuvre dans :


 **Character** : **pickUp**, **pullOut**, **getBagItems**, **getBagCapacity**, **getBagWeight**, **equip(Consumable)**, **equip(Weapon)**


 **Hero** : **equip(ArmorItem)** , **equip(Ring)**


### 12.3. Laissez **NoBagException** se propager au travers de toutes les méthodes impactées de **Character**.

## 13. BagFullException

### 13.1. Créer la sous-classe **lsg.exceptions.BagFullException**

 Avec un constructeur **BagFullException(Bag bag)**  
(bag représente le sac lié à l'exception).

 Le message de l'exception sera : « **nomDeLaClasseDuSac is full !** »

 Ajoutez un getter pour récupérer le bag concerné.

### 13.2. Levez l'exception dans **Bag** : **push**

### 13.3. Modifiez **Bag.transfer** en remplaçant le test sur la taille du sac par une gestion de l'exception.

NB : la méthode doit toujours avoir le même comportement !

### 13.4. Laissez **BagFullException** se propager au travers de toutes les méthodes impactées de **Character**.