

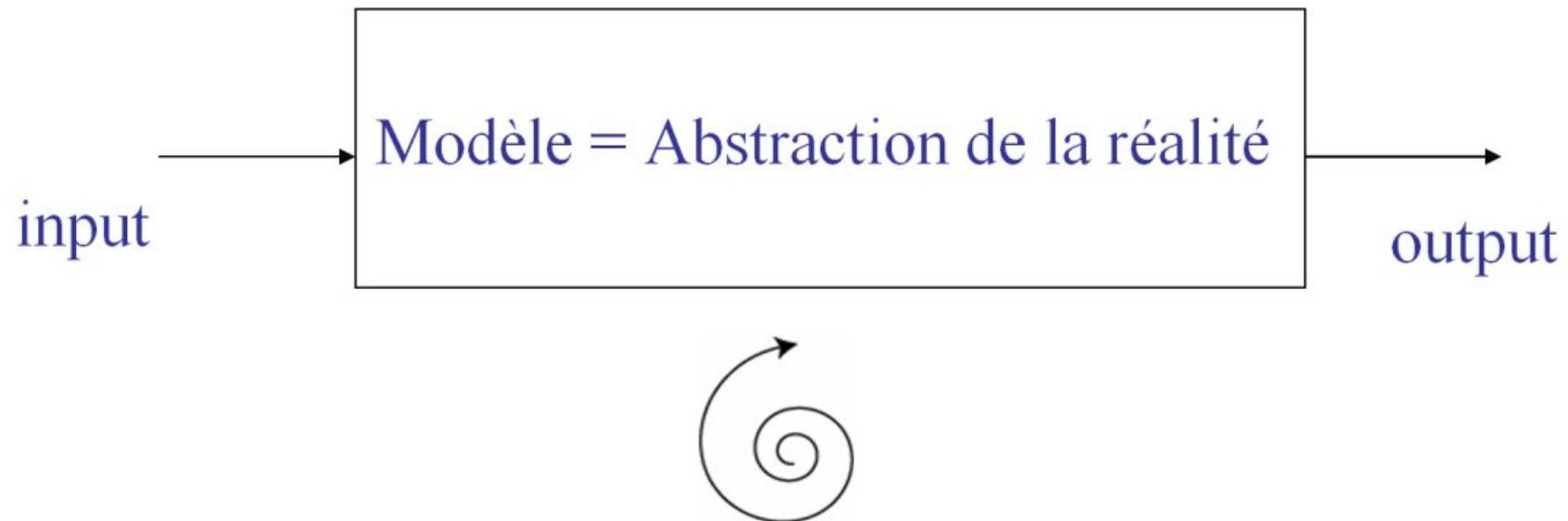
# Plan du cours

- Génie Logiciel
- Le langage de modélisation UML
- Design Patterns
- SimJ
- Sujet choisi : processus de développement
- Conclusion

# Plan du chapitre

- SimJ
  - ^ Introduction
  - ^ Concepts généraux
  - ^ Jeux de la simulation
  - ^ Elaboration du framework SimJ
  - ^ Applications: Supermarché
  - ^ Améliorations possibles

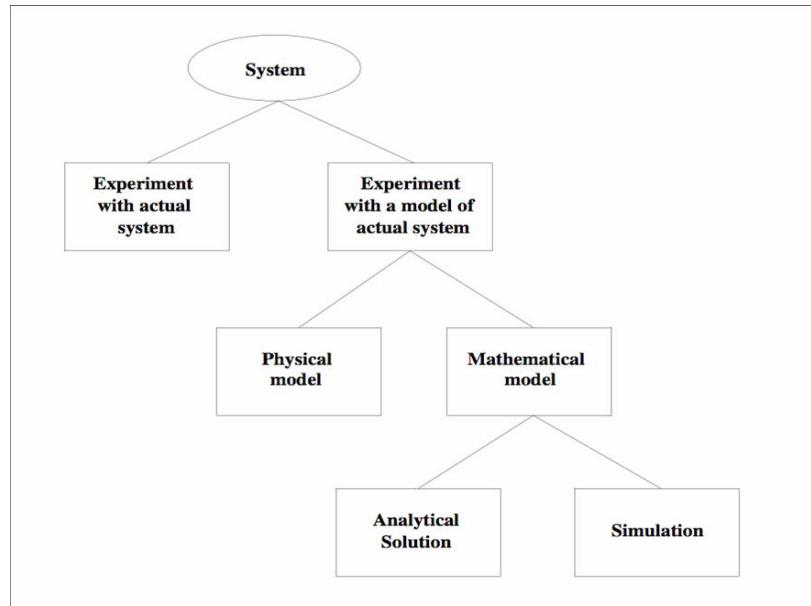
# Définition



On fait tourner dans le temps

# Motivation

## Manières d'étudier un système



- La **simulation** permet de:
  - ^ Prévoir l'évolution d'un système.
  - ^ Mieux comprendre un système.
  - ^ Accélérer ou ralentir le temps.
  - ^ Contrôler les paramètres de variation.
  - ^ Reproduire plusieurs fois.
  - ^ Contrôler le niveau de détail.
  - ^ Maîtriser les coûts.

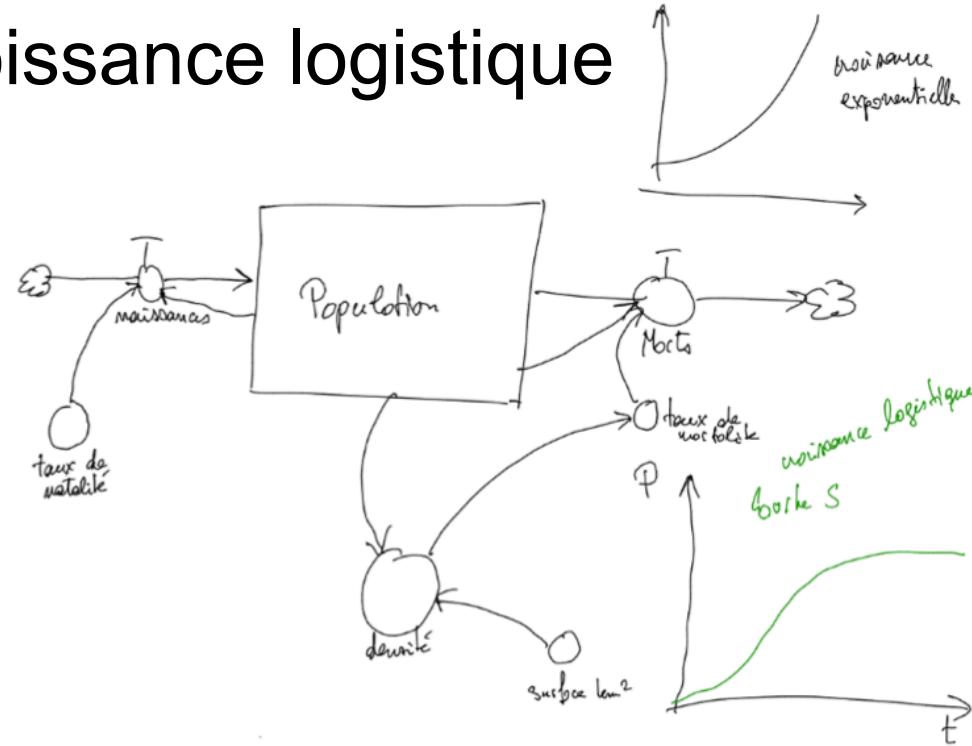
# Simulation continue (1)

- On envisage un système comme un ensemble de réservoirs qui peuvent se remplir et/ou se vider selon des taux qui peuvent être interdépendants.
- Idéalement, le temps doit être envisagé de manière continue et le système modélisé avec des équations différentielles.
- Concrètement, le temps est discrétisé pour résoudre le système numériquement.

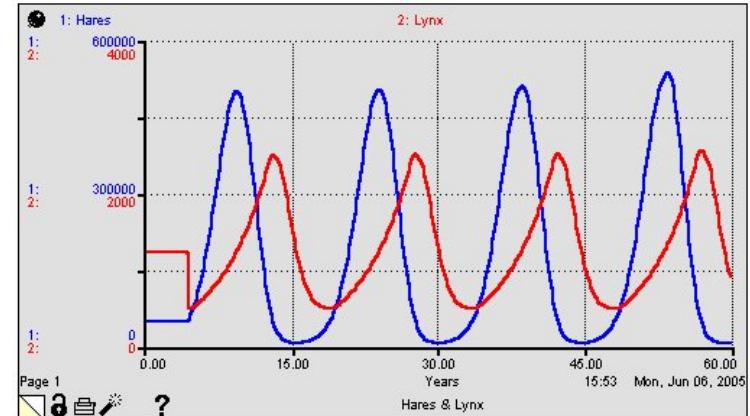
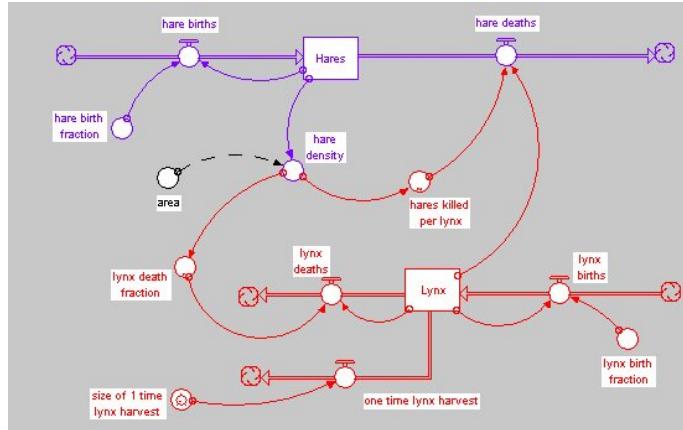


# Simulation continue (2)

## ■ Croissance logistique



# Simulation continue (3)

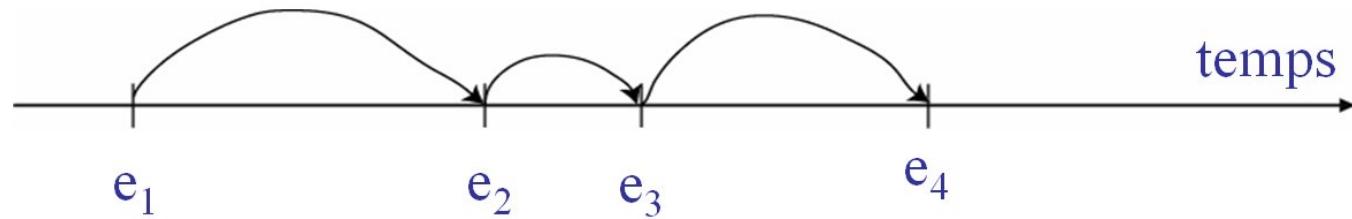


Modèle de proies et prédateurs (logiciel Stella)  
(modèle de Lotka-Volterra)

Output observé sur l'évolution des populations

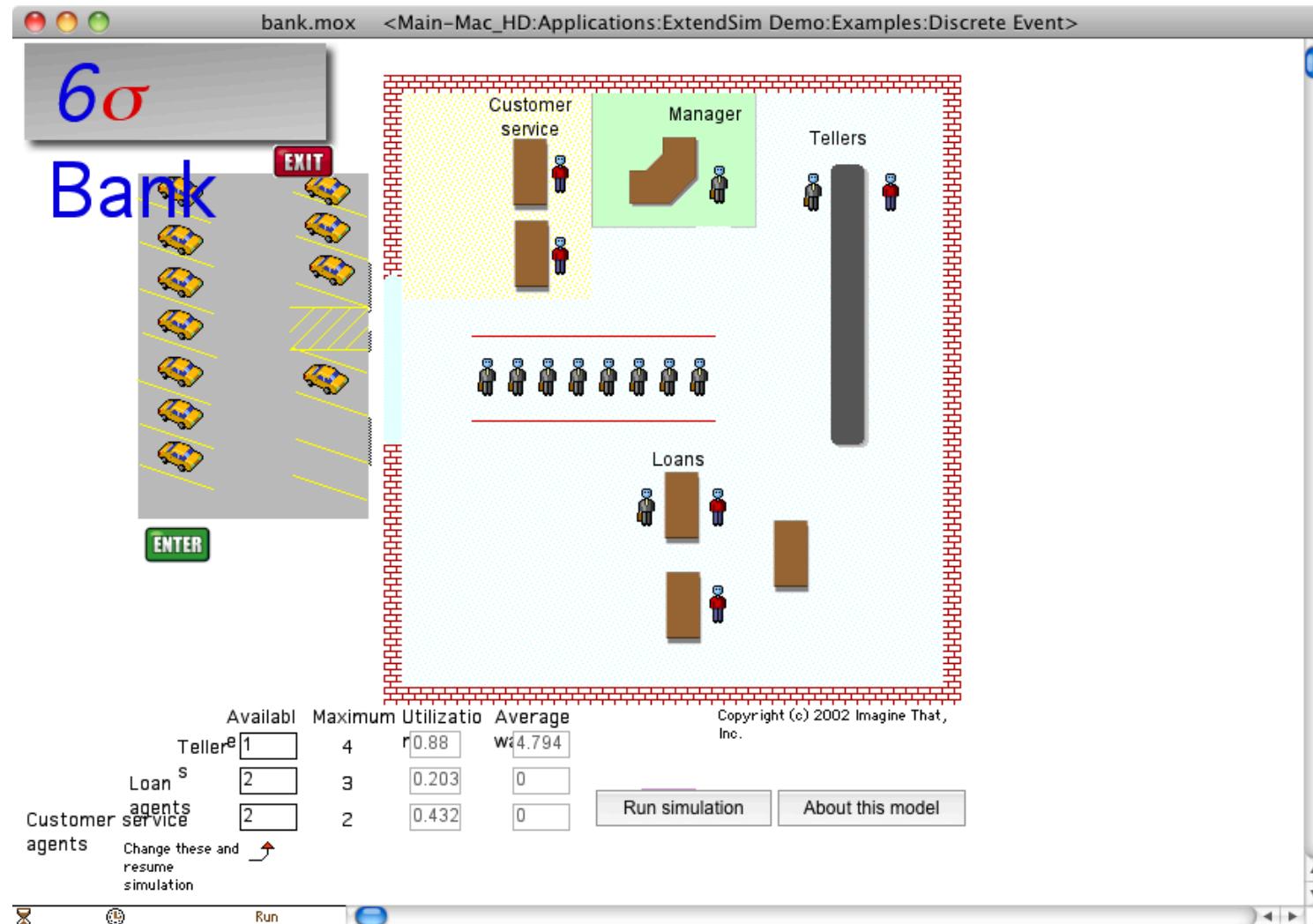
# Simulation par événements discrets (1)

- On considère que l'état du système ne change que lorsque certains **événements** se produisent, par exemple arrivée ou départ d'un client.
- On ne s'intéresse qu'aux instants du temps où un événement/changement se produit.

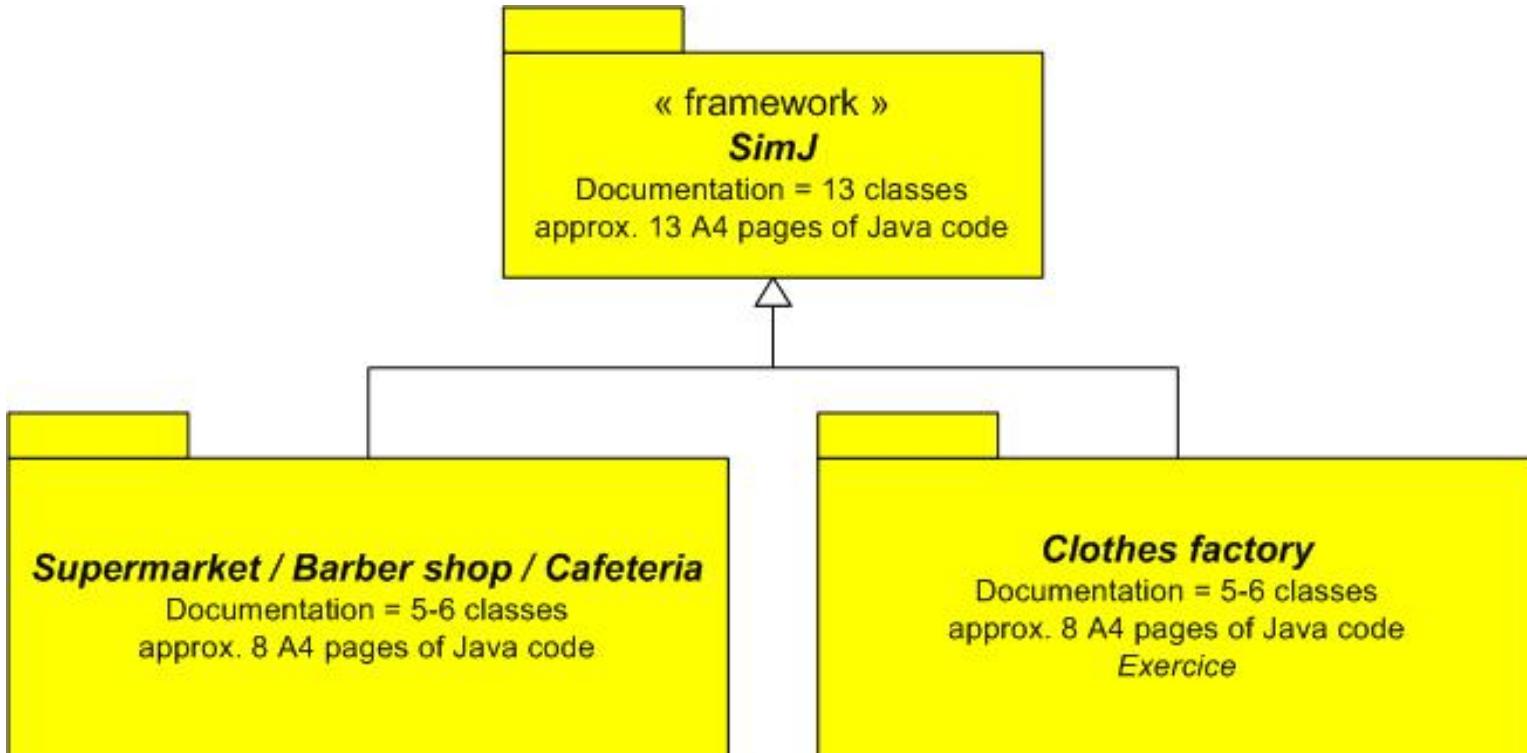


- Une variable (appelée "**horloge/clock**" de simulation) saute chronologiquement d'un événement à l'autre et maintient le temps simulé.
- Plusieurs événements peuvent se produire au même instant.

# Simulation par événements discrets (2)



# But final



# Concepts généraux : La notion d'état (une première définition)

- Première définition: Une fois un système modélisé avec un certain niveau d'abstraction, l'état du modèle à un instant  $t$  doit pouvoir décrire complètement ce dernier au temps  $t$ .
- Par exemple, si on considère un parking pour voitures, l'état du modèle peut se résumer:
  - ^ au nombre de places occupées;
  - ^ au nombre de places vacantes;
  - ^ à la file d'attente devant le parking.

# Ressources et Entités (1)

- Avec la terminologie employée dans le monde de la simulation, il est possible de modéliser un système à l'aide des concepts suivants:
  - ▲ **Entités** (temporaires). Au plus haut niveau une entité (p.ex. une voiture) peut n'avoir aucune caractéristique particulière si ce n'est un **numéro d'identification**. En principe de telles entités peuvent être créées à tout instant de façon exogène au système. Elles peuvent aussi disparaître du système.
  - ▲ **Ressources** (permanentes). Au plus haut niveau, une ressource (p.ex. un parking) est caractérisée par:
    - sa capacité  $1$  à  $n$ ;
    - les  $m$  entités temporaires en train de l'utiliser;
    - les  $p$  entités temporaires en attente de service;
    - divers paramètres (nom, temps de service,...).

# La notion d'état (2<sup>ème</sup> définition)

- Si "Système = Ressources + Entités" Alors l'**état du système** à l'instant  $t$  est donné par les caractéristiques à cet instant de toutes les ressources.
- En effet, les caractéristiques de toutes les ressources indiquent:
  - △ les entités qui se trouvent dans les files d'attente;
  - △ les entités qui sont actuellement en service;
- De plus, les ressources peuvent avoir différents paramètres (calcul du temps de service, son nom, interruption, etc...).

# Resources et entités (2)

- Une entité temporaire ne peut être associée à une ressource que de 2 manières:
  - ^ celles qui sont **en attente** devant la ressource, qui est occupée à 100% et
  - ^ celles qui sont **en train d'être servies**.

# La notion d'événement discret

- Un **événement discret** représente un **changement de l'état** du système à un instant  $t$ .
- Un événement n'a donc pas de durée!
- Exemples:
  - ▲ l'arrivée d'une voiture (qui se met dans la file d'attente ou qui rentre dans le parking);
  - ▲ l'entrée d'un véhicule dans le parking.
  - ▲ la sortie du parking d'un véhicule.

# Notion de hasard

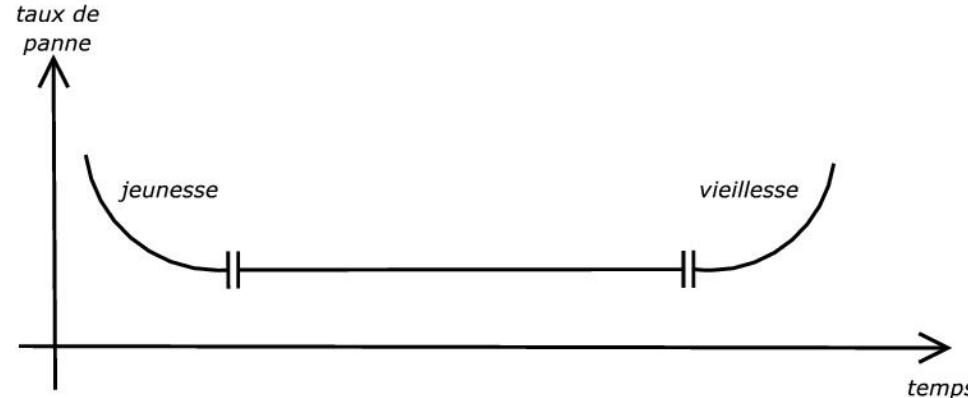
- Pour bien représenter les fluctuations inhérentes à un vrai système, il faut introduire dans le modèle de simulation des aspects aléatoires.
- Concrètement, on utilise un **générateur de nombres aléatoires** (la façon de générer ces nombres est en dehors du sujet de ce cours).
- Un nombre aléatoire est caractérisé par une loi de probabilité. Deux lois parmi les plus utilisées en simulation sont:
  - ^ la loi **uniforme** entre  $a$  et  $b$ .
  - ^ la loi **exponentielle** de moyenne  $\mu$ .

# Usages de la loi exponentielle (1)

- Modélisation du temps entre des arrivées: Si on considère des arrivées ayant une certaine fréquence  $f$  et qui sont indépendantes entre elles, alors le nombre d'arrivées dans un laps de temps  $t$  suit une loi de Poisson de paramètre  $f \cdot t$  et le **temps entre deux arrivées** suit une loi exponentielle de moyenne  $1/f = \mu$ .
- Exemple: Si on a en moyenne 3 clients par minute et que l'unité de temps de la simulation est la seconde, alors le temps entre deux arrivées suit une loi exponentielle de moyenne 20 secondes.

# Usages de la loi exponentielle (2)

- Modélisation du temps entre deux pannes:
  - △ Soit la notion de taux de panne  $\lambda$  d'une machine.
  - △ Le taux de panne évolue généralement ainsi:



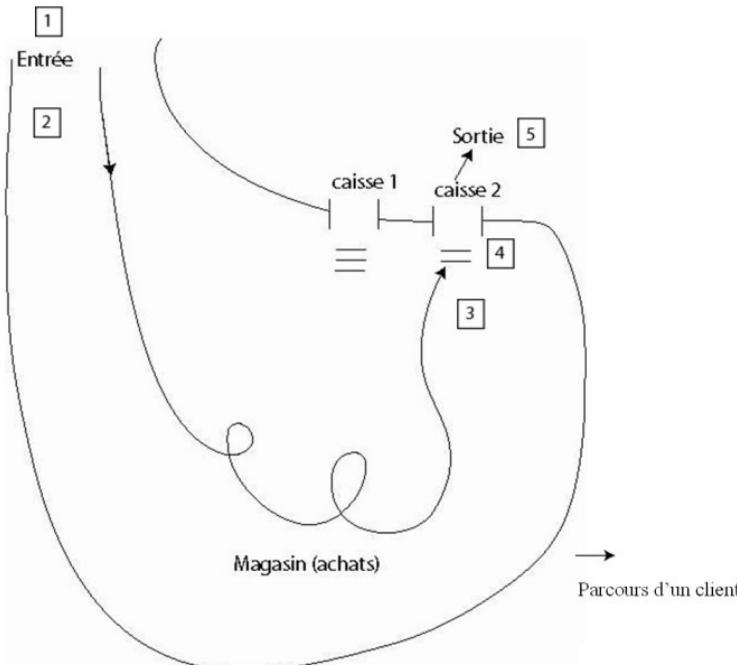
- △ Si on s'intéresse à la partie où le taux est stable de valeur  $\lambda$ , alors le **temps entre deux pannes** suit une loi exponentielle de moyenne  $\mu=1/\lambda$  .

# Exécution d'un modèle de simulation

- On a une **horloge virtuelle** qui indique le temps "actuel" et qui saute de l'événement actuel au prochain événement qui est lui-même prévu, soit au même instant, soit après une durée aléatoire.
- On a l'idée d'un **échéancier** ("future event chain") des événements ordonnés par leurs instants d'activation.
- Par exemple: Arrivée d'une voiture à l'instant  $t$ 
  - ▲ Mettre le temps de l'horloge à  $t$ .
  - ▲ Entrée dans la file d'attente du parking et si le parking n'est pas plein, alors prévoir une entrée dans le parking à l'instant  $t$ .
  - ▲ Prévoir une autre arrivée de voiture à  $t+?$  (mise à jour de l'échéancier).

# Jeu de la simulation: Supermarché

- Faisons le jeu de la simulation en prenant l'exemple d'un supermarché. Voici une vue globale du modèle:



- Le système est composé de:
  1. entités (clients)
  2. ressources (magasin, caisses)
- Evénements:
  1. Création du client
  2. Entrée du client
  3. Fin des achats
  4. Début de service
  5. Fin de service

# Modèle

- L'état du supermarché à un instant  $t$  est caractérisé par:
  - △ les clients associés au **magasin**, à savoir:
    - ceux qui sont à l'intérieur du magasin.
  - △ les clients associés à la **caisse 1**, à savoir:
    - ceux qui sont dans la file d'attente et
    - celui qui est en train d'être servi.
  - △ les clients associés à la **caisse 2**.
    - ceux qui sont dans la file d'attente et
    - celui qui est en train d'être servi.

# Ressources et entités

- Rappelons que le **magasin**, la **caisse 1** et la **caisse 2** sont des **ressources** et qu'un client (entité temporaire) ne peut être associé à une ressource que de 2 manières, à savoir:
  - ▲ celles qui sont en attente devant la ressource, qui est occupée à 100% et
  - ▲ celles qui sont en train d'être servies.
- Dans l'exemple du supermarché, le magasin est toujours libre et, par conséquent, n'a pas de file d'attente, alors que les caisses 1 et 2 ne peuvent servir qu'un seul client à la fois.

# Evénements

- Cinq types d'événements sont susceptibles de modifier l'état du système:
  1. **la création d'un client  $c$**  et son arrivée devant le magasin (création de  $c$  qui demande la ressource magasin);
  2. **l'entrée d'un client  $c$  dans le magasin** (début de service de  $c$  par la ressource magasin);
  3. **la fin des achats pour un client  $c$**  et le choix de la caisse avec la plus petite file d'attente (fin de service de  $c$  par la ressource magasin et demande par  $c$  de la ressource de type caisse ayant la plus petite file d'attente);
  4. **le début de service d'un client  $c$**  (début de service de  $c$  par une ressource de type caisse);
  5. **la fin de service pour un client  $c$**  et son départ du supermarché (fin de service de  $c$  par une ressource de type caisse et "disparition" de  $c$  en dehors du système).

# Temps et durées

Pour "jouer le jeu de la simulation", les temps (instants) d'arrivée, les durées d'achat dans le magasin et les durées du service aux caisses des quatre premiers clients sont résumés dans le tableau ci-dessous:

Numéro	Temps d'arrivée	Durée d'achat	Durée du service
1	0	110	15
2	5	100	15
3	10	105	20
4	999	-	-

Remarques:

- Ces chiffres sont normalement des nombres générés aléatoirement.
- Nous arrêtons la simulation au temps 800.

# Le tableau d'exécution: Explication (1)

- Chacune de ses lignes correspond à l'exécution d'un événement entraînant un changement de l'état du système.
- Verticalement, le tableau est décomposé en trois parties de la manière suivante:
  1. La première colonne indique le **temps actuel**  $t$  de la simulation. Le paramètre  $t$  représente donc l'instant où se produit un événement et où l'on considère l'état du système.
  2. Le deuxième groupe de colonnes (de couleur jaune) illustre la gestion des événements dans la boucle de simulation en montrant l'évolution du contenu de l'**échéancier**.
  3. Le troisième groupe de colonnes (de couleur bleue) représente l'**état** du supermarché à l'aide des trois paramètres présentés au transparent "Modèle".

# Le tableau d'exécution: Explication (2)

- Le deuxième groupe de colonnes (de couleur jaune) illustre la gestion des événements dans la boucle de simulation en montrant l'évolution du contenu de l'échéancier. Chaque ligne représente un événement et ses colonnes sont organisées comme suit:
  1.  $e_{t,n}^{c,r}$  représente un événement de type  $t$ , dont le numéro d'ordre de création est  $n$  et qui est associé au client  $c$  et à la ressource  $r$ ;
  2.  $T$ : représente l'instant de création de l'événement. Il est à noter que ce ne sont pas nécessairement les événements créés le plus tôt qui sont exécutés en premier;
  3. Brève explication du groupe d'actions associées à un événement.

# Le tableau d'exécution: Explication (3)

- Le troisième groupe de colonnes (de couleur bleue) représente l'état du supermarché à l'aide des trois paramètres présentés au transparent "Modèle", à savoir:
  1. les clients associés au magasin (ressource 1);
  2. les clients associés à la caisse 1 (ressource 2);
  3. les clients associés à la caisse 2 (ressource 3).

Un client  $c$  est toujours identifié par son indice de création  $k$ .
- Pour chaque ressource il y a deux sous-colonnes. Elles représentent de gauche à droite et conformément au transparent "Ressources et entités":
  - ▲ les entités en attente de service
  - ▲ les entités en train d'être servies

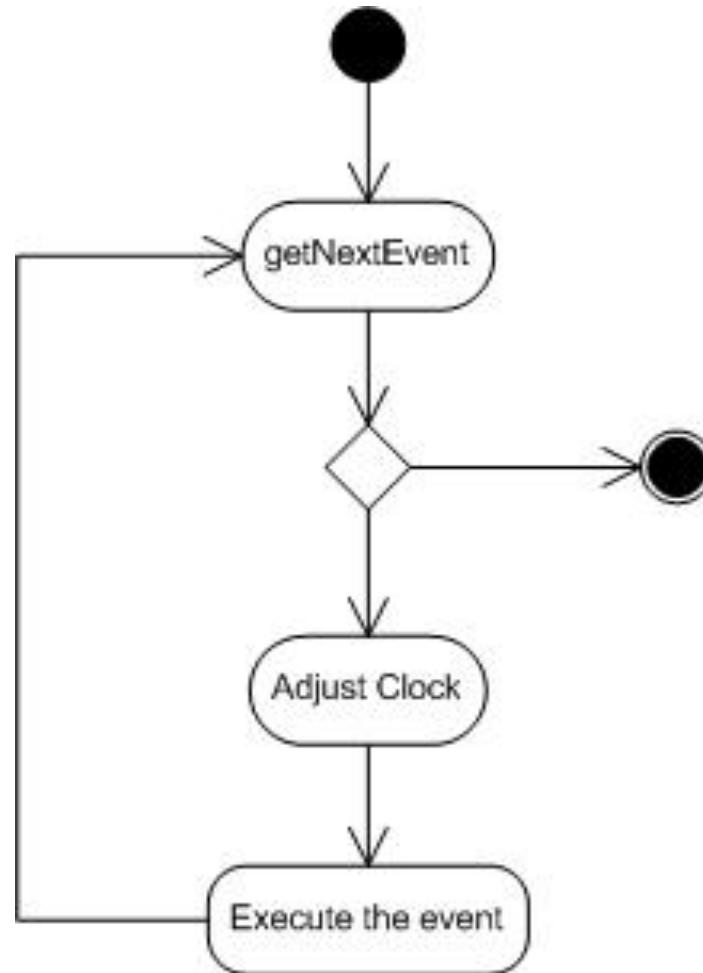
# Le tableau d'exécution (partie 1)

t	Événements			Etat du supermarché		
	$e_{t,n}^{c,r}$	T	Actions	Magasin	Caisse 1	Caisse 2
0	$e_{1,1}^{1,\cdot}$	0	crée $c_1$ et l'envoie au magasin crée $e_{2,2}^{1,1}$ et l'ordonnance à $t + 0$ crée $e_{1,3}^{2,\cdot}$ et l'ordonnance à $t + 5$	$c_1$		
0	$e_{2,2}^{1,1}$	0	fait entrer $c_1$ dans le magasin crée $e_{3,4}^{1,1}$ et l'ordonnance à $t + 110$		$c_1$	
5	$e_{1,3}^{2,\cdot}$	0	crée $c_2$ et l'envoie au magasin crée $e_{2,5}^{2,1}$ et l'ordonnance à $t + 0$ crée $e_{1,6}^{3,\cdot}$ et l'ordonnance à $t + 5$	$c_2$	$c_1$	
5	$e_{2,5}^{2,1}$	5	fait entrer $c_2$ dans le magasin crée $e_{3,7}^{2,1}$ et l'ordonnance à $t + 100$		$c_2, c_1$	
10	$e_{1,6}^{3,\cdot}$	5	crée $c_3$ et l'envoie au magasin crée $e_{2,8}^{3,1}$ et l'ordonnance à $t + 0$ crée $e_{1,9}^{4,1}$ et l'ordonnance à $t + 989$	$c_3$	$c_2, c_1$	
10	$e_{2,8}^{3,1}$	10	fait entrer $c_3$ dans le magasin crée $e_{3,10}^{3,1}$ et l'ordonnance à $t + 105$		$c_3, c_2, c_1$	
105	$e_{3,7}^{2,1}$	5	sort $c_2$ du magasin et l'envoie à la caisse 1 crée $e_{4,11}^{2,2}$ et l'ordonnance à $t + 0$		$c_1, c_3$	$c_2$

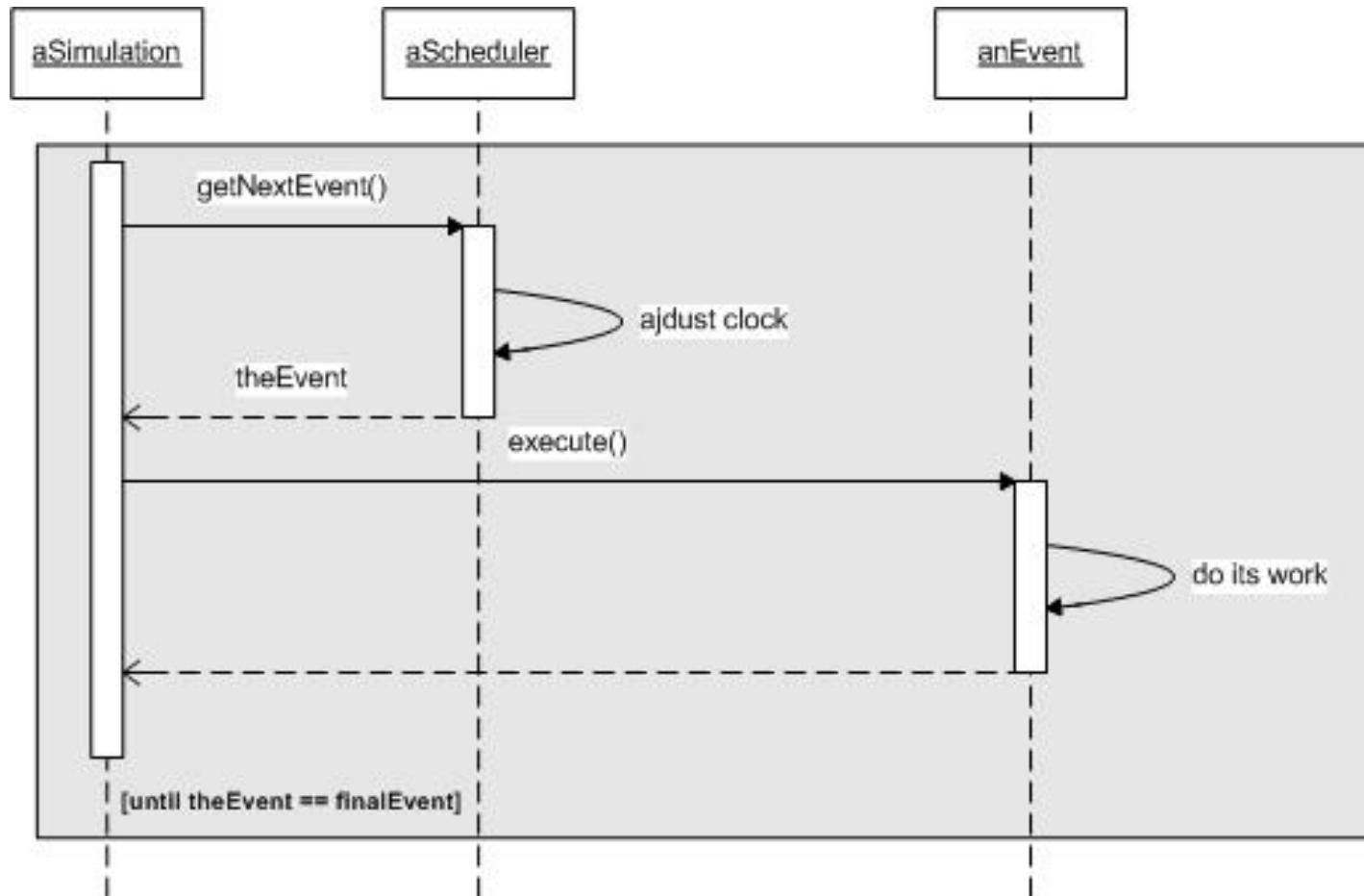
# Le tableau d'exécution (partie 2)

t	Événements			Etat du supermarché			
	$e_{t,n}^{c,r}$	T	Actions	Magasin	Caisse 1	Caisse 1	Caisse 2
105	$e_{4,11}^{2,2}$	105	débute le service de $c_2$ par la caisse 1 crée $e_{5,12}^{2,2}$ et l'ordonnance à $t + 15$		$c_1, c_3$	$c_2$	
110	$e_{3,4}^{1,1}$	0	sort $c_1$ du magasin et l'envoie à la caisse 2 crée $e_{4,13}^{1,3}$ et l'ordonnance à $t + 0$		$c_3$	$c_2$	$c_1$
110	$e_{4,13}^{1,3}$	110	débute le service de $c_1$ par la caisse 2 crée $e_{5,14}^{1,3}$ et l'ordonnance à $t + 15$		$c_3$	$c_2$	$c_1$
115	$e_{3,10}^{3,1}$	10	sort $c_3$ du magasin et l'envoie à la caisse 1		$c_3$	$c_2$	$c_1$
120	$e_{5,12}^{2,2}$	105	sort $c_2$ de la caisse 1 crée $e_{4,15}^{3,2}$ et l'ordonnance à $t + 0$		$c_3$		$c_1$
120	$e_{4,15}^{3,2}$	120	débute le service de $c_3$ par la caisse 1 crée $e_{5,16}^{3,2}$ et l'ordonnance à $t + 20$			$c_3$	$c_1$
125	$e_{5,14}^{1,3}$	120	sort $c_1$ de la caisse 2			$c_3$	
140	$e_{5,16}^{3,2}$	120	sort $c_3$ de la caisse 1				
999	$e_{1,9}^{4,1}$	10	ne s'exécute jamais, fin à $t = 800$				

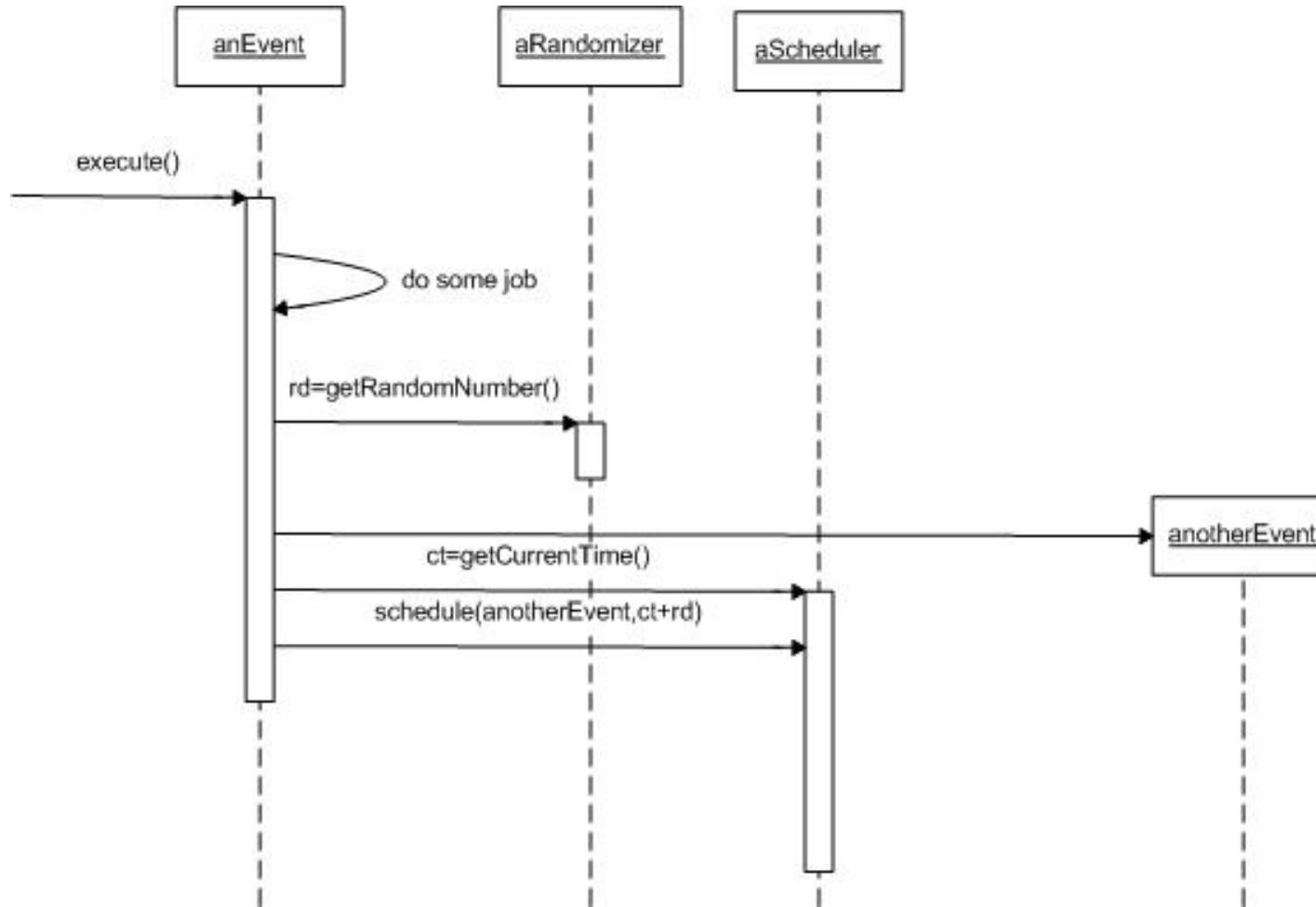
# La boucle de simulation



# Diagramme de séquence: boucle de simulation



# Diagramme de séquence: un événement



# Framework SimJ : Avant-propos

- Nous avons vu que nos modèles sont constitués de:
  - ^ d'**entités** (clients, machines,...)
  - ^ de **ressources** (magasin, caisses, ateliers,...)
  - ^ d'**événements** qui changent l'état du système
- Nous avons également mentionné les notions:
  - ^ d'**échéancier** ("scheduler") des événements
  - ^ d'un générateur de **nombres aléatoires**
- Ces substantifs doivent servir de base pour définir les classes de SimJ. Les actions (verbes) qui leur sont associées peuvent servir pour trouver les méthodes contenues dans ces classes.

# Entités

- Au plus haut niveau, il est juste nécessaire de pouvoir identifier les entités individuellement. Il faut donc leur attribuer un **numéro d'identification unique** lors de leur création.
- La classe **SimEntity** sera donc très simple. Il sera toujours possible de la sous-classer pour ajouter des comportements plus précis tels que le nombre d'articles achetés par un client, la taille du réservoir d'essence pour une voiture,...
- Afin de garantir un numéro unique pour chaque entité et vu que beaucoup d'entités sont susceptibles d'être créées en cours d'exécution de la simulation, nous allons prévoir une **SimEntityFactory**.

# Ressources (1)

- Une classe **SimResource** sera certainement nécessaire. Cette dernière sera abstraite, car s'il est possible de totalement préprogrammer certains comportements standards, d'autres dépendent de la ressource concrète que l'on envisage.
- Plus précisément, nous allons voir:
  - ^ ce qu'il est possible d'offrir au niveau du framework (niveau général et abstrait, commun à toutes les applications).
  - ^ ce qu'on ne pourra définir que dans les sous-classes concrètes propres à chaque application.

# Ressources (2)

- Ce qu'il est possible d'offrir **au niveau du framework**:
  - ^ des variables initialisées à la création pour le nom, le nombre d'entités pouvant être servies simultanément (capacité).
  - ^ une file d'attente d'entités en attente d'être servies.
  - ^ un ensemble d'entités en train d'être servies.
  - ^ les importantes méthodes **request** et **endServing**.
  - ^ diverses méthodes annexes permettant de faire des statistiques:
    - `numberOfEntitiesBeingServed`,
    - `numberOfEntitiesWaitingToBeServed`,
    - `hasEntitiesWaitingToBeServed`.

# Ressources (3)

- Ce qu'on ne pourra définir **que dans les sous-classes propres à l'application**:
  - ^ la méthode `getServiceTime` qui n'est pas simplement un paramètre mais qui peut être n'importe quelle fonction (y compris dépendante du nombre d'articles, ou de la taille du réservoir d'essence,...)
  - ^ la méthode `afterEndService` qui offre une grande souplesse pour continuer à s'occuper d'une entité une fois qu'elle a quitté une ressource. Cette méthode sera systématiquement appellée lors d'une fin de service, et par défaut elle ne fera rien (corps de la méthode vide, ce n'est donc pas une méthode abstraite).

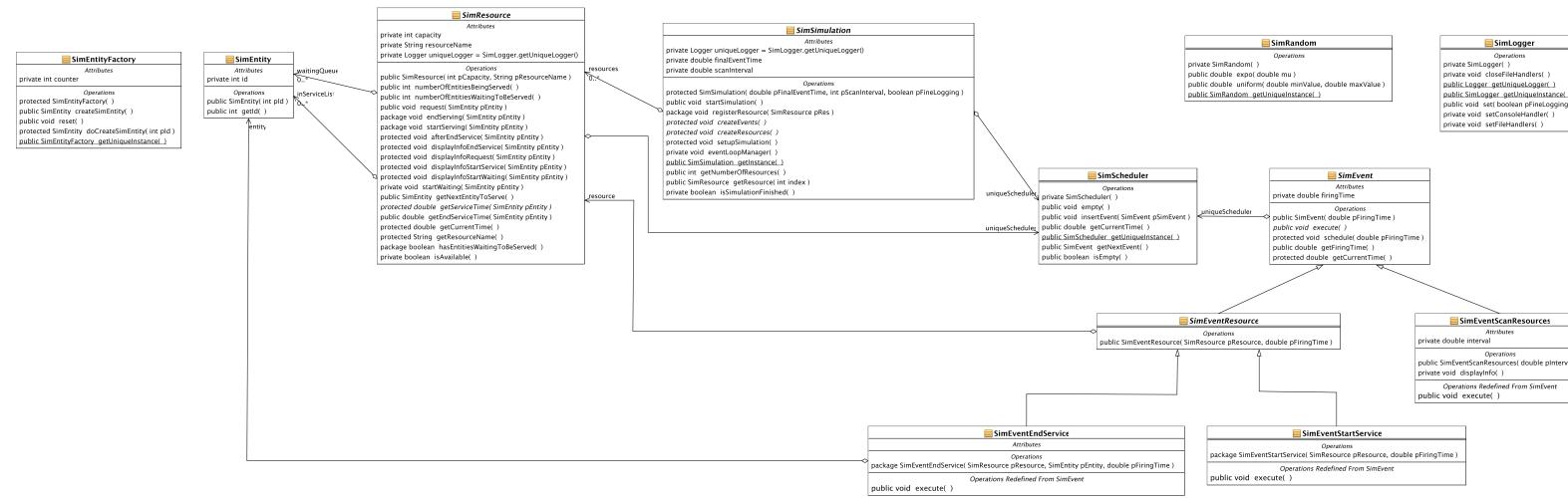
# Evénements

- Au plus haut niveau, la classe `SimEvent` est abstraite:
  - ▲ Chaque événement a un temps d'exécution (`firingTime`) permettant de l'ordonnancer/de le prévoir à un moment donné.
  - ▲ Son exécution particulière (méthode `execute`) dépend de son type.
- Le framework doit offrir les deux événements génériques liés aux ressources, marquant respectivement le début et la fin de service d'une entité dans une resource: le `SimEventStartService` et le `SimEventEndService`. Ces deux événements sont essentiels au fonctionnement de la simulation et sont donc déclarés `final`.
- On peut prévoir quelques événements particuliers: le `SimEventScanResources` qui à intervalles réguliers interroge et affiche l'état de toutes les resources.

# Autres

- Associée aux événements il y a la classe `SimScheduler` qui gère la liste des événements futurs et offre les méthodes `insertEvent` et `getNextEvent`. Implémentée comme un Singleton elle maintient également l'heure courante de la simulation et offre ainsi la méthode `getCurrentTime`.
- Associée aux temps aléatoires, il y a un autre Singleton: la classe `SimRandom` qui offre des méthodes pour obtenir des nombres aléatoires suivant les lois uniforme et exponentielle.
- Enfin pour lier le tout, il y a la classe abstraite `SimSimulation` qui offre la gestion de la boucle de simulation (`eventLoopManager`). Elle doit être sous-classée pour:
  - ▲ obtenir tous les paramètres d'input propres au modèle,
  - ▲ initialiser la simulation en créant les ressources, et en générant les événements de départ.

# SimJ: Diagramme de classes



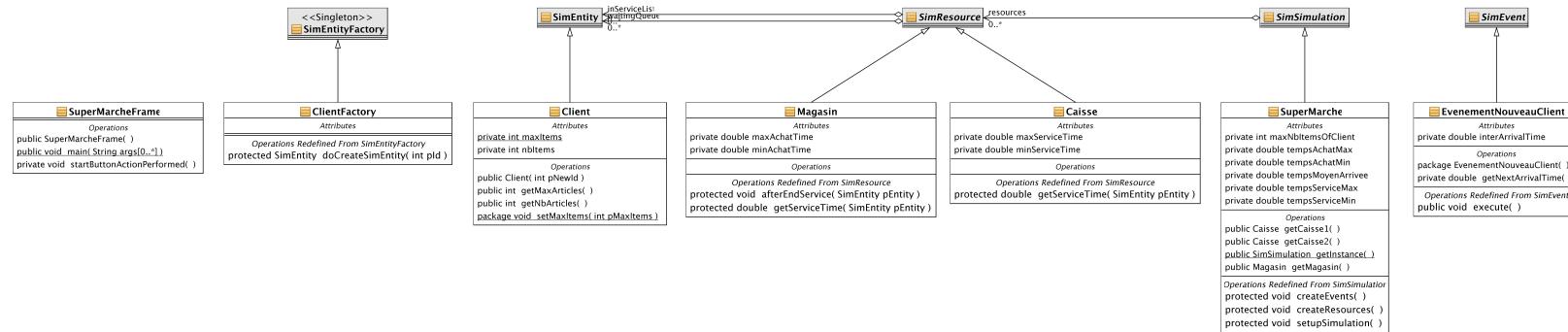
# Applications: Supermarché

Pour le supermarché il y a:

- deux types de ressources: le magasin et les caisses
- un type d'entités: les clients
- un événement particulier: l'arrivée d'un nouveau client devant le magasin.

Il faut donc offrir ces classes et implémenter les méthodes nécessaires.

# Diagramme de classes: Supermarket



# Améliorations possibles

- Dans la version actuelle du framework, la logique:
  - ▲ des files d'attente est codée directement dans la ressource.
    - en encapsulant la logique de la file d'attente dans une hiérarchie de classes séparée il serait possible d'offrir plusieurs types de file d'attente à choix. Par exemple: FIFO, LIFO, Priority Queues, Random,...
    - on appliquerait alors le pattern Strategy.
  - ▲ qui permet de suivre le déroulement d'une simulation est codée directement dans la ressource.
    - en encapsulant cette logique dans des "observeurs" qui seraient informés d'un changement d'état dans une ressource il serait possible d'offrir plusieurs types d'observeurs. Par exemple: output textuel, output graphique/graphes, animations, calcul de statistiques d'utilisation,...
    - on appliquerait, évidemment, le pattern Observer.
- Le framework SimJ gagnerait ainsi en maturité et serait plus extensible.