Exemple: Pacman

```
(defclass participant ()
  ((position :initarg :position :accessor participant-position))
  (:documentation "a generic game participant"))

(defclass mobile-participant (participant)
  ()
  (:documentation "base class for all mobile participants"))

(defclass pacman (mobile-participant)
  ((force :initform 0 :accessor pacman-force)
   (invincible :initform nil :accessor pacman-invincible))
  (:documentation "the hero of the game"))
```

Création d'instances

```
(defclass position ()
  ((x :initarg :x :reader pos-x)
   (y :initarg :y :reader pos-y)))

(defun make-position (x y)
  (make-instance 'position :x x :y y))

(defun make-pacman (&optional (x 0) (y 0))
  (make-instance 'pacman :position (make-position x y)))
```

Fonctions génériques

```
(defgeneric move-participant (mobile-participant x y)
  (:documentation "change the location of a participant on the board"))

(defgeneric collision (participant1 participant2)
  (:documentation "handle a collision between two participants"))
```

Les méthodes

```
(defmethod move-participant ((p mobile-participant) x y)
  (setf (participant-position p)
        (make-position x y)))

(defmethod collision ((p1 participant) (p2 (eql nil)))
  p1)

(defmethod collision ((p1 (eql nil)) (p2 participant))
  p2)

(defmethod collision ((pac pacman) (tab tablet))
  (incf (slot-value pac 'force))
  pac)
```

Les méthodes (suite)

La liste des argument (lambda list) contient des paramètres spécialisés par des classes.

Une méthode est applicable uniquement si la restriction sur les classes est vérifiée.

Les méthodes (suite)

Utilisation de méthodes *:after*

```
(defmethod collision :after ((pac pacman) (tab super-tablet))
  (setf (slot-value pac 'invincible) t))

(defmethod initialize-instance :after ((p participant)
                                       &rest initargs
                                       &key &allow-other-keys)
  (declare (ignore initargs))
  (let* ((pos (participant-position p))
         (x (pos-x pos))
         (y (pos-y pos)))
    (setf (aref *board* x y) p)))
```

Les méthodes (suite)

Utilisation de méthodes *:before* et *:after* sur les *setters* :

```
(defmethod (setf participant-position) :before ((pos position)
                                                 (p participant))
  (let ((old-pos (participant-position p)))
    (setf (aref *board* (pos-x old-pos) (pos-y old-pos))
          nil)))

(defmethod (setf participant-position) :after ((pos position)
                                                (p participant))
  (let ((x (pos-x pos))
        (y (pos-y pos)))
    (setf (aref *board* x y)
          (collision p (aref *board* x y)))))
```