UNIVERSITAT POLITÈCNICA DE CATALUNYA

Pràctica de planificació

PLANIFICACIO

INTEL·LIGÈNCIA ARTIFICIAL

Autors

ANDRÉS GÜIZZO
MIQUEL FLORENSA
DAVID LATORRE

Supervisor

SERGIO ÁLVAREZ NAPAGAO

CONTINGUTS

1. El problema

En aquesta pràctica, el problema principal que se'ns planteja tracta sobre l'assignació d'habitacions a un hotel, a partir de les peticions de reserva que aquest rep. Aquestes assignacions seran en el rang d'un mes.

De cara a esbrinar quines habitacions haurem de assignar a una petició u altre, haurem de mirar les característiques de les habitacions i de les peticions.

El problema es divideix en diverses extensions, de forma que una extensió serà més completa i tindrà en compte més coses que l'anterior, de forma que aquestes característiques aniràn augmentant segons l'extensió del problema.

Principalment, però, pel que fa a nivell bàsic del problema, les dades que tindrem sobre una habitació seràn les següents:

- **Identificador:** cada habitació s'identifica amb una sèrie de símbols per tal de poder-la diferencia de la resta.
- **Nombre de persones que pot allotjar:** aquest nombre serà entre 1 i 4.

I pel que fa a les reserves:

- **Identificador:** té la mateixa funció que l'identificador de habitacions.
- **Nombre de persones a la reserva:** ens informa sobre el nombre de persones que han fet conjuntament la reserva, i que per tant voldran estar assignades a la mateixa habitació. Aquest nombre serà entre 1 i 4.
- **Dia d'inici:** dia d'inici de la reserva.
- **Dia final:** dia final de la reserva.

2. Aspectes a tenir en compte

Abans de començar a tractar la resolució del domini i dels problemes de cada nivell del problema que se'ns planteja, és important avaluar quins aspectes hem de tenir en compte per la creació d'un bon domini.

Això és important perquè de maneres de resoldre aquest problema en tenim moltes, però de forma eficient hi haurà millors maneres que altres:

- Hem d'intentar minimitzar el nombre d'operadors i el factor de ramificació d'aquests, de forma que l'exploració de quins operadors sigui aplicable en un determinat moment es redueixi.

- Intentarem evitar usuar operadors amb paràmetres similars, de forma que l'existència d'uns objectes o altres dirigeixi la instanciació d'operadors.

- Intentarem usar tipus en les variables per tal de reduir la quantitat d'objectes que el planificador comprovarà per cada paràmetre del operador.

- Intentarem evitar l'ús de exists i forall, a les precondicions i efectes, ja que es tracta d'operacions ineficients.

- Al principi de les precondicions haurem de posar aquelles que facin un filtratge més ràpid dels objectes. D'aquesta forma, si aquestes precondicions no es compleixen, no fa falta passar a les següents.

Cal destacar que, al cap i a la fi, la prioritat màxima és l'eficiència, per tant, si cal fer el planificador més complexe, afegint per exemple operacions, per tal que sigui eficient, ho farem.

A més, també s'ha de dir que, coses que hem de considerar com evitar l'ús de exists o forall, no sempre ho podrem fer, com per exemple en aquesta pràctica com veurem posteriorment.

3. Modelització de Domini i Extensions

3.1. Nivell bàsic

3.1.1. Domini

3.1.1.1. Variables

- **Habitacion**: representa una habitació de l'hotel
- **Reserva**: representa una reserva d'una habitació per uns dies concrets
- **Dia**: representa una unitat de temps (un dia)

3.1.1.2. Predicats

- **(assignada ?x - reserva):** indica si una reserva x ha estat assignada o no.
- **(assignacion ?x - reserva ?y - habitacion):** indica l'assignació x d'una habitació y per a una reserva.
- **(ocupada ?x - dia ?y - habitacion):** indica si una habitació y ja ha estat assignada/ocupada per el dia x.

3.1.1.3. Funcions

- **(maxPersonas ?x - habitacion):** número màxim que pot allotjar una habitació determinada.
- **(personas ?x - reserva):** número de persones que es volen allotjar en aquella reserva.
- **(inicio ?x - reserva):** dia d'inici de la reserva.
- **(final ?x - reserva):** dia final de la reserva.
- **(numero ?x - dia):** el numero el qual representa el dia.

3.1.1.4. Accions

- **assignar (x, y):** donada una reserva x i una habitació y, si la reserva encara no ha estat assignada i l'habitació compleix amb els requisits de la reserva, l'habitació y pasa a estar assignada.

En aquesta solució, per tant, distingim entre habitacions, reserves i dies. Tot i que hem posat que són reserves, tècnicament són peticions a reserves.

El fet pel qual hem de declarar un dia com a objecte, és perquè necessitem, donada una habitació, saber si per a cada dia del mes estarà disponible o no. En canvi, com que per una reserva només ens fa falta saber el dia quan comença i quan acaba, això ho podem enmagatzemar directament en dos funcions, inicio i final.

Pel que fa als operadors, només en tenim un, que assigna una reserva a una habitació, ja que de fet és la única operació que hem de realitzar.

Pel que fa a l'ordre de les precondicions a l'operador que hi ha, això ho hem fet colocant-los de forma que es minimitzin els objectes que el planificador intenta comprobar. És per això que hem posat el forall de la precondicio al final, ja que com que és una operació costosa, només es comprovarà si s'han complert les dos altres precondicions.

3.1.2. Modelització del Problema

Pel que fa a la creació dels problemes per a aquest apartat, això ho hem implementat amb el fitxer: generadorProblemBasic.java. A aquest arxiu, li direm quantes habitacions volem que tingui el problema, i també quantes reserves, i ell automàticament ens generarà un fitxer .pddl amb el problema.

Si observem aquest arxiu, veurem que primerament el que fem és crear els objectes del problema. El que es farà serà crear tantes habitacions i reserves com hem introduit que volem. L'identificador d'aquests objectes al cap i a la fi és indiferent.

També, el que farem, serà crear 30 objectes de tipus dia, els quals faran referència als 30 dies d'un mes.

A continuació, es definirà l'estat inicial del problema. Dins d'aquest no definirem cap predicat per a cap objecte, ja que principalment no hi haurà ni habitacions assignades, i, per tant, no hi haurà d'ocupades.

Però sí que haurem de definir totes les funcions en aquest cas, ja que representen les característiques tant de les habitacions, com dels dies (el número del dia al que es fa referència en aquest cas), i de reserves.

Finalment només haurem de definir l'objectiu del problema, el qual serà tenir totes les reserves assignades.

3.2. Extensió 1

3.2.1. Domini

Aquesta extensió és molt semblant al problema base. Si ens hi fixem, podem observar que el problema és el mateix, però si no es pot complir la funció objectiu, el que hem de fer és maximitzar el nombre de reserves assignades.

Per tant, l'únic canvi que haurem de fer serà afegir una funció, la qual s'anomeni **reservas_asignadas.** Aquesta funció enmagatzemarà el nombre de reserves que s'han assignat. Per tant, com a efecte de la funció assignar, aquesta funció haurà de incrementar-se.

3.2.2. Modelització del Problema

Pel que fa a la creació dels problemes per a aquest apartat, això ho hem implementat amb el fitxer: generadorProblemExtension1.java. A aquest arxiu, li direm quantes habitacions volem que tingui el problema, i també quantes reserves, i ell automàticament ens generarà un fitxer .pddl amb el problema.

En la modelització d'aquesta extensió passa el mateix que en el domini, és a dir, serà el mateix. No obstant, en el problema, a part de definir l'objectiu, també haurem de definir una metrica que maximitzi la funcio: reservas_asignadas.

3.3. Extensió 2

3.3.1. Domini

En aquest cas, el problema és el mateix que la extensió 1, però a més les peticions inclouran la orientació preferent a on es vol que estigui situada l'habitació.

Per tant, hem decidit crear un objecte que es digui **orientacion.** Llavors, com que necessitem saber on estarà orientada una habitació, hem creat el predicat: **orientadaA.** I com que també necessitem saber l'orientació que es desitja en una reserva, creem el predicat: **peticionOrientacion**.

Finalment, com que el que volem és també maximitzar l'assignació de reserves amb la orientació que vol la reserva, crearem una funció: **assignadas_con_orientacion.** El que aquesta funció farà és enmagatzemar el nombre de asignacions que compleixen amb la orientació que hi ha fins al moment.

3.3.2. Modelització del Problema

Pel que fa a la creació dels problemes per a aquest apartat, això ho hem implementat amb el fitxer: generadorProblemExtension2.java. A aquest arxiu, li direm quantes habitacions volem que tingui el problema, i també quantes reserves, i ell automàticament ens generarà un fitxer .pddl amb el problema.

Complint amb el que s'ha dit en el domini, en el problema, per tant, el que haurem de fer serà crear 4 instàncies de orientació, les quals facin referència a les quatre orientacions que hi ha al problema: nort, sud, est, i oest.

Un cop fet això, a l'estat inicial també haurem de saber a on està orientada una habitació, i a on vol la orientació una reserva. Per tant, per a cada habitació, haurem de definir el predicat **orientadaA**, i per cada reserva haurem de definir **peticionOrientacion**.

Finalment, com que el que volem és maximitzar conjuntament el nombre de reserves assignades i el nombre de reserves assignades amb orientacio, a la mètrica, farem una suma entre les dos funcions que enmagatzemen aquests valors.

I com que tenim preferència per maximitzar més les reserves assignades abans que les reserves assignades amb orientacio, el que farem serpa multiplicar per un natural superior a o reservas_asignadas.

3.4. Extensió 3

3.4.1. Domini

La extensió 3 és una ampliació de la primera extensió on també hem de minimitzar el malbaratament de places. Per tant, pel que fa el domini, només canvien un parell de coses. Primer de tot, hem creat una funció **plazas_sin_ocupar** on emmagatzarem el número de places de habitacions que quedin sense utilitzar, és a dir si una habitació és per a 4 persones i només li assignem aquesta habitació a una sola persona, augmentarem **plazas_sin_ocupar** en 3 unitats. Aquesta ultima part la implementarem dins l'acció de **assignar** amb la següent sentència: *(increase (plazas_sin_ocupar) (- (maxPersonas ?y) (personas ?x))).*

3.4.2. Modelització del Problema

Pel que fa a la creació dels problemes per a aquest apartat, això ho hem implementat amb el fitxer: generadorProblemExtension3.java. A aquest arxiu, li direm quantes habitacions volem que tingui el problema, i també quantes reserves, i ell automàticament ens generarà un fitxer .pddl amb el problema.

D'altra banda, les instàncies d'aquesta extensió seran les mateixes que les de la primera extensió. Tot i això, hem de canviar la mètrica que maximitza el nombre de reserves per tal que també minimitzi el nombre de places malbaratades. És per això que afegirem a la mètrica les **plazas_sin_ocupar** restant a **reservas_asignadas.** Això farà que es minimitzi el nombre de places sense ocupar. Cal afegir que aquest nombre

ja està ponderat ja que afecta més que s'assigni una habitació de 4 p. a una sola persona que no que s'asigni la mateixa habitació a 3 persones.

3.5. Extensió 4

3.5.1. Domini

La extensió 4 és una ampliació de la tercera extensió, on a més, volem minimitzar el nombre d'habitacions diferentes ocupades. És per això que necessitarem modificar el domini una mica. Primerament afegirem una funció que ens indiqui el nombre de habitacions diferents reservades: **habitaciones_reservadas.**

En segon lloc creearem una altre acció, que serà la que assigni habitacions anteriorment ja reservades (en diferents dies òbviament). Aquesta segona acció **assignar_habitacion_previamente_assignada** sumarà en una unitat la funció de **habitaciones_reservadas** cada cop que faci una assignació. Tanmateix, la funció que ja teníem de assignar pasa a ser **assignar_habitacion_previamente_sin_assignar**. Aquesta acció només assignarà en cas que no s'hagin fet assignacions a una habitació determinda.

3.5.2. Modelització del Problema

Pel que fa a la creació dels problemes per a aquest apartat, això ho hem implementat amb el fitxer: generadorProblemExtension4.java. A aquest arxiu, li direm quantes habitacions volem que tingui el problema, i també quantes reserves, i ell automàticament ens generarà un fitxer .pddl amb el problema.

D'altra banda, les instàncies d'aquesta extensió seran les mateixes que les de la tercera extensió. Tot i això, hem de canviar la mètrica que maximitza el nombre de reserves i minimitza el nombre de places malbaratades per tal que també tingui prioritat assignar habitacions que previament s'han utilitzat. És per això que afegirem a la

mètrica les **habitaciones_reservadas**. Aquestes seran sumades a les **plazas_sin_ocupar** i el resultat serà restat de les **reservas_asignadas**. D'aquesta forma maximitzarem el nombre de places assignades mentres que minimitzarem el nombre de places malbaretades i el nombre d'habitacions utilitzades en el mes.

4. Desenvolupament dels models

Pel que fa el desenvolupament dels diversos problemes, hem començat fent el problema bàsic i hem anat incrementant-lo i afegint-li coses a ell per poder completar la resta de extensions. A excepció de l'extensió 2, que compta amb preferències de orientació, les altres extensions es basen en l'anterior, d'aquesta forma la extensió 4 no és molt diferent a la 3. És per aquest motiu que no ens ha estat molt difícil fer les diferents extensions. A més del programes pddl, hem realitzat generadors de jocs de prova per a cada extensió.

5. Jocs de prova

5.1. Nivell bàsic

Tots els jocs de proves que hem creat en aquest document (i que es troben a la carpeta JocsDeProves) han estat creats pels seus corresponents generadors. A l'hora de crear els jocs de prova, als generadors li hem dit que volem 7 habitacions i 7 reserves, per tant, tots els jocs de proves tenen 7 habitacions i 7 reserves.

A continuació tenim el que seria l'output que extrauria el programa Metric-FF per al joc de proves que hem creat per al nivell bàsic. Aquest programa no inclou mètrica, i, el seu goal es que totes les reserves s'assignin. Per tant, es podria donar el cas que el planificador no pugui trobar cap pla. No obstant, ha trobat una solució:

```
checking for cyclic := effects --- OK.

ff: search configuration is EHC, if that fails then  best-first on 1*g(s) + 5*h(s) where
    metric is  plan length

Cueing down from goal distance:    7 into depth [1]
                                   6            [1]
                                   5            [1]
                                   4            [1]
                                   3            [1]
                                   2            [1]
                                   1            [1]
                                   0

ff: found legal plan as follows

step    0: ASSIGNAR RESERVA6 HABITACIO6
        1: ASSIGNAR RESERVA5 HABITACIO5
        2: ASSIGNAR RESERVA4 HABITACIO3
        3: ASSIGNAR RESERVA3 HABITACIO5
        4: ASSIGNAR RESERVA2 HABITACIO5
        5: ASSIGNAR RESERVA1 HABITACIO1
        6: ASSIGNAR RESERVA0 HABITACIO4


time spent:    0.00 seconds instantiating 0 easy, 34 hard action templates
               0.00 seconds reachability analysis, yielding 408 facts and 34 actions
               0.00 seconds creating final representation with 348 relevant facts, 0 relevant fluents
               0.00 seconds computing LNF
               0.00 seconds building connectivity graph
               0.00 seconds searching, evaluating 8 states, to a max depth of 1
               0.00 seconds total time
```

5.2. Extensió 1

A continuació ens trobem amb l'output de l'extensió 1. En aquesta extensió, a diferència del programa bàsic, incloem una mètrica, de forma que en el cas que no es pugui trobar cap pla de resolucio de l'objectiu, igualment s'intentaria maximitzar o minitzar el que s'introdueix a la mètrica. No obstant, per al nostre joc de proves, tot i això s'ha trobat una solució, de forma que la mètrica finalment en aquest cas no ens ha servit de gaire:

```
checking for cyclic := effects --- OK.

ff: search configuration is EHC, if that fails then  best-first on 1*g(s) + 5*h(s) where
    metric is  plan length

Cueing down from goal distance:    7 into depth [1]
                                   6            [1]
                                   5            [1]
                                   4            [1]
                                   3            [1]
                                   2            [1]
                                   1            [1]
                                   0

ff: found legal plan as follows

step    0: ASSIGNAR RESERVA6 HABITACIO6
        1: ASSIGNAR RESERVA5 HABITACIO5
        2: ASSIGNAR RESERVA4 HABITACIO3
        3: ASSIGNAR RESERVA3 HABITACIO5
        4: ASSIGNAR RESERVA2 HABITACIO3
        5: ASSIGNAR RESERVA1 HABITACIO5
        6: ASSIGNAR RESERVA0 HABITACIO2


time spent:    0.00 seconds instantiating 0 easy, 41 hard action templates
               0.00 seconds reachability analysis, yielding 440 facts and 41 actions
               0.00 seconds creating final representation with 405 relevant facts, 1 relevant fluents
               0.00 seconds computing LNF
               0.00 seconds building connectivity graph
               0.00 seconds searching, evaluating 8 states, to a max depth of 1
               0.00 seconds total time
```

5.3. Extensió 2

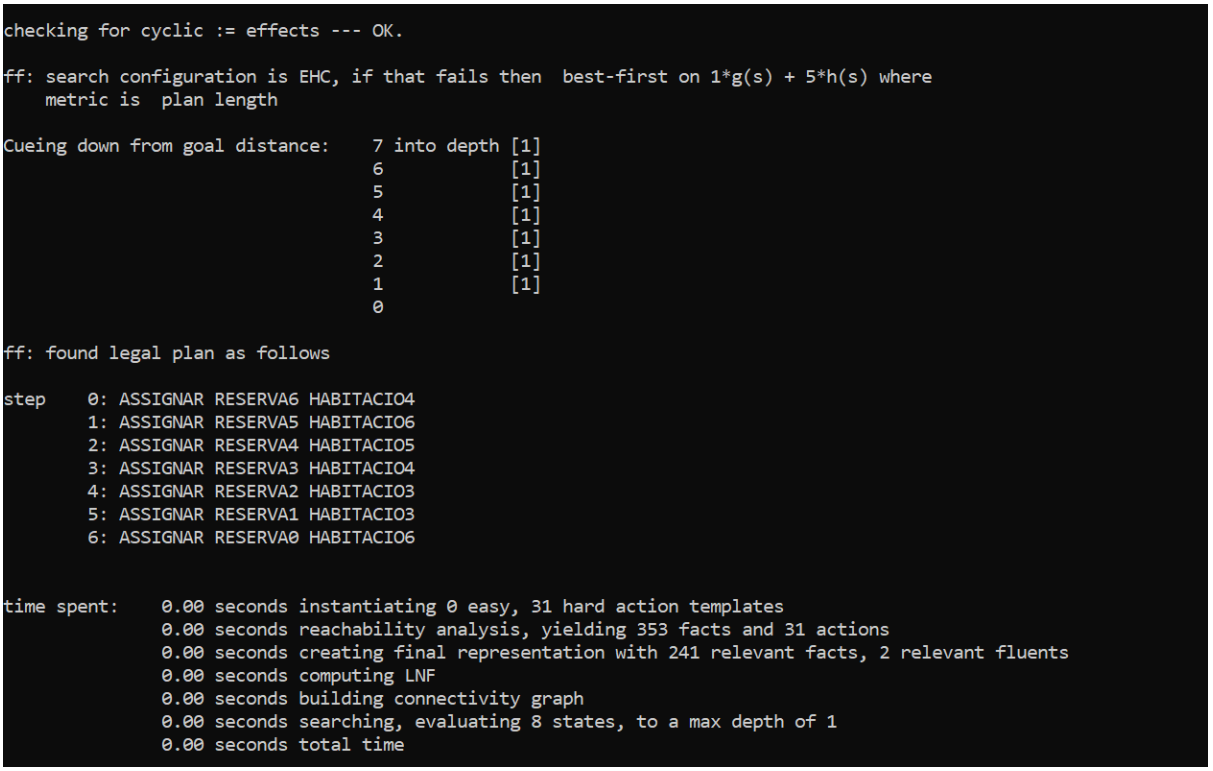
Posteriorment, aquí tenim el resultat que ens dona el joc de proves per a l'extensió 2. Si recordem, aquesta extensió afegeix l'objecte orientació, i per tant certs predicats relacionats amb aquest. Per tant, alguna cosa que seria interessant observar, seria observar el temps d'execució en comparació amb l'anterior extensió, però al tenir tantes poques instàncies de objectes, els dos temps són nuls.

Podem observar que en aquest cas també hem trobat un pla:

```
checking for cyclic := effects --- OK.

ff: search configuration is EHC, if that fails then  best-first on 1*g(s) + 5*h(s) where
    metric is  plan length

Cueing down from goal distance:    7 into depth [1]
                                   6            [1]
                                   5            [1]
                                   4            [1]
                                   3            [1]
                                   2            [1]
                                   1            [1]
                                   0

ff: found legal plan as follows

step    0: ASSIGNAR RESERVA6 HABITACIO6
        1: ASSIGNAR RESERVA5 HABITACIO6
        2: ASSIGNAR RESERVA4 HABITACIO5
        3: ASSIGNAR RESERVA3 HABITACIO4
        4: ASSIGNAR RESERVA2 HABITACIO3
        5: ASSIGNAR RESERVA1 HABITACIO2
        6: ASSIGNAR RESERVA0 HABITACIO1


time spent:    0.00 seconds instantiating 0 easy, 43 hard action templates
               0.00 seconds reachability analysis, yielding 445 facts and 43 actions
               0.00 seconds creating final representation with 413 relevant facts, 2 relevant fluents
               0.00 seconds computing LNF
               0.00 seconds building connectivity graph
               0.00 seconds searching, evaluating 8 states, to a max depth of 1
               0.00 seconds total time
```

5.4. Extensió 3

El input d'aquesta prova és el mateix que el de la primera extensió. I per tant a priori podria semblar que hauria de donar la mateixa solució. Però tal i com es pot veure en la següent imatge (i comparant-la amb la sortida del joc de proves de la primera extensió) els resultats son diferents. Això es deu a que el programa ara està prioritzant fer assignacions d'habitacions que s'apropin al nombre de persones que volen fer l'assignació, és a dir, que no es malbaratin places.



5.5. Extensió 4

De la mateixa manera que en l'apartat anterior, farem l'execució de la extensió 4 amb el mateix input que a la extensió 1 i 3. A la imatge següent podem observar com ara la sortida és diferent i també ho son les habitacions assignades. En primer lloc, hi ha dos tipus d'assignació, una assignació per a aquelles habitacions que encara no han estat reservades en el més, i una altre tipus d'assignació que només es fa quan la habitació en concret ja ha estat assignada anteriorment (per a uns altres clients). En segon lloc també podem observar canvis de assignacions d'habitacions respecte el joc de proves anterior, això es deu a que també influeix el nombre d'habitacions diferents utilitzades en el mes, on aquest fet es minimitza.

```
checking for cyclic := effects --- OK.

ff: search configuration is EHC, if that fails then  best-first on 1*g(s) + 5*h(s) where
    metric is  plan length

Cueing down from goal distance:    7 into depth [1]
                                   6            [1]
                                   5            [1]
                                   4            [1]
                                   3            [1]
                                   2            [1]
                                   1            [1]
                                   0

ff: found legal plan as follows

step    0: ASSIGNAR_HABITACION_PREVIAMENTE_SIN_ASSIGNAR RESERVA6 HABITACIO6
        1: ASSIGNAR_HABITACION_PREVIAMENTE_SIN_ASSIGNAR RESERVA5 HABITACIO4
        2: ASSIGNAR_HABITACION_PREVIAMENTE_SIN_ASSIGNAR RESERVA4 HABITACIO3
        3: ASSIGNAR_HABITACION_PREVIAMENTE_SIN_ASSIGNAR RESERVA3 HABITACIO2
        4: ASSIGNAR_HABITACION_PREVIAMENTE_ASSIGNADA RESERVA2 HABITACIO3
        5: ASSIGNAR_HABITACION_PREVIAMENTE_SIN_ASSIGNAR RESERVA1 HABITACIO1
        6: ASSIGNAR_HABITACION_PREVIAMENTE_ASSIGNADA RESERVA0 HABITACIO3


time spent:    0.02 seconds instantiating 49 easy, 1110 hard action templates
               0.00 seconds reachability analysis, yielding 483 facts and 1147 actions
               0.00 seconds creating final representation with 483 relevant facts, 3 relevant fluents
               0.00 seconds computing LNF
               0.01 seconds building connectivity graph
               0.00 seconds searching, evaluating 8 states, to a max depth of 1
               0.03 seconds total time
```

6. Conclusió

Podem concloure que hem resolt els problemes de planificació proposats satisfactòriament. En aquests problemes hem pogut veure com assignar diferentes habitacions d'hotel a diferents clients tot seguint unes optimitzacions i preferències dels clients. Per aconseguir-ho hem creat models de PDDL i hem utilitzat Fast Forward. A més també hem realitzat generadors de jocs de prova aleatoris per a cadascuna de les extensions en Java.