PRÀCTICA Java. JOC DE ROL.

Joan Gerard Camarena Estruch

Continguts

Generalitats **3**

Fase 1. Herència **4**
Explicació 4
Tasques a efectuar: 5

Fase 2. Donant cos als jugadors **6**
Explicació 6
Es demana 6

Fase 3. Sobrecàrrega i Polimorfisme **8**
Explicació 8
Es demana 8

Fase 4. Relacions entre classes **9**
Explicació 9
Es demana 9

Fase 5. Més relacions **11**
Explicació 11
Es demana 11

Fase 6. Configuració del programa principal i Excepcions **13**
Explicació 13
Es demana 13
Configuració 13
Jugar 14
Excepcions 14

Generalitats

En aquesta pràctica anem a fer-la durant 3 entregues, de manera que anirem afegint-li funcionalitat a la mateixa. Per aixòp farem un poquet i acabem, farem altre poquet més i parem, i així fins a completar les 7 fases que consta la totalitat de la pràctica.

Per a desenvolupar-ho i no perdre res ho podem fer de dos maneres:

1. 7 projectes distints en 7 carpetes distintes. Quan acabem la primera copiem i apeguem i a partir d'aquesta còpia fem la segona. D'aquesta manera quan acabem la fase *n* és l'inici de la fase *n+1*
2. Fer servir **GIT**. De manera que ens posem a programar. Quan acabem cada fase fem un *commit* donant-li com a nom la fase en la que estem. Això ens permetrà tornar a cadascuna de les fases anteriors.

Fase 1. Herència

Explicació

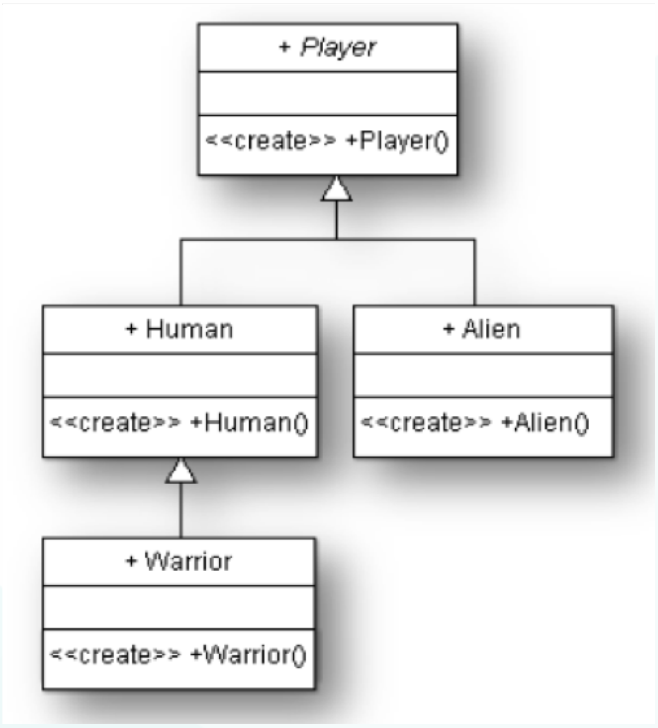
Volem programar un joc de rol on es van a crear diversos jugadors. A aquests jugadors els anomenarem de manera genèrica `Player`. Al moment de jugar, es pot triar entre 3 tipus diferents de jugadors: humà (`Human`), guerrer (`Warrior`) o alienígena (`Alien`). Cada personatge té unes característiques pròpies, que són: els punts d'atac, els punts de defensa i els punts de vida i el comportament de cada jugador dependrà de la seua raça.

El joc consistirà en que els jugadors s'ataquen entre ells. Cada vegada que el jugador `A` ataca el jugador `B`, passen 2 coses:

- A colpeja B
- B colpeja A (condicional)

Quan el jugador `A` colpeja a `B` es fa la resta entre els punts d'atac del jugador A i els punts de defensa del jugador B. Si el resultat és positiu, el jugador B perd esta quantitat de punts de vida. Si al jugador `B` encara li queden punts de vida (no està mort) es podrà defendre i colpejar a `A`. Quan els punts de vida d'un jugador arriben a zero, diem que és mort.

La relació dels diferents tipus de jugadors es mostra en el següent diagrama:



Com podem vore, un jugador `Warrior` és una especialització d'un jugador `Human`. I `Alien` i `Human` són també una especialització de `Player`. Això ho podem aconseguir mitjançant l'*herència*.

La classe `Player` serà abstracta. És a dir, no podrem crear jugadors (objectes) d'eixa classe, sinó de les especialitzades, les quals es diferencien en:

- `Human` → No tenen bonificacions en defensa ni en atac.
- `Alien` → Tenen bonificacions en cada atac però també penalitzacions en defensa.
- `Warrior` → Poden aguantar més ferides que la resta de jugadors.

Tasques a efectuar:

1. Crea l'aplicació `JocDeRol_V1` amb els següents paquets (cas de Java).

- `io`: Per a les classes d'utilitats i de lectura escriptura de teclat i pantalla. Pots fer servir les subministrades al llarg del curs pel professor.
- `joc`: Per a les classes `Player`, `Human`, `Alien` i `Warrior`.
- `inici`: Per a la classe `JocDeRol`, amb el programa principal.

> Nota: en Python pot fer-se tot en el mateix fitxer. Ja estudiarem com separar-ho en distints fitxers.

2. Implementa en el paquet joc les 4 classes (`Player`, `Human`, `Warrior`, `Alien`), amb les herències que pertoquen i amb els mètodes constructors per defecte corresponents, sense paràmetres. Els constructors, de moment, han de traure per pantalla el següent text:

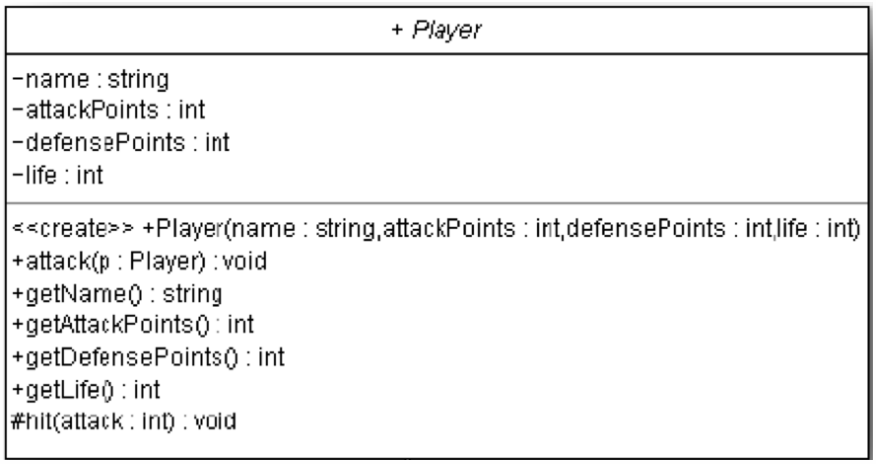
```
CONSTRUCTOR -> He creat un <nom_de_la_classe_corresponent>
```

3. En la classe `JocDeRol` del paquet inici crea la funció `provaFase()`, que ens ajudarà a entendre com funciona el mecanisme de l'herència. Eixa funció ha de crear un objecte de cada tipus (`Human`, `Warrior` i `Alien`). Abans de crear cada objecte avisarà per pantalla el tipus d'objecte que va a crear. En executar-ho, comprovarem que quan es crida al constructor d'una classe, es crida automàticament (i per ordre) a tots els constructors de les classes de les quals hereta.
4. Si fas servir GIT, ara és el moment de fer el `commit rol_v1`

Fase 2. Donant cos als jugadors

Explicació

Afegirem els atributs per a guardar els punts d'atac, defensa i vida, així com els mètodes necessaris per a atacar un altre jugador. El diagrama de classes quedarà de la manera següent:



> Nota: Aquesta representació és del diagrama de classes de UML. A la qual, els camps i/o mètodes s'han de posar com:
>
> - Es posa un − quan l'atribut o mètode és private
> - Es posa un + quan l'atribut o mètode és public
> - Es posa un # quan l'atribut o mètode és protected

Es demana

1. Clona el projecte a `JocDeRol_V2`, amb *copia i refactorització* (aquest pas no és necessari si fas servir git).
2. Modifica la classe `Player`:

- Afig 4 atributs: `name`, `attackPoints`, `defensePoints`, `life`
- Modifica el constructor per a passar-li els 4 atributs. Caldrà modificar també els constructors de les altres classes hereves.
- Afix els 4 mètodes *getters* corresponents per a consultar cada atribut.

3. Sobreescriu el mètode `toString()` (o `__str__`) de la classe `Object` en la classe `Player`. Retornarà totes les dades del jugador amb el següent format:

```
    John Smith PA:13 / PD:8 / PV:39
```

- Afig el mètode void `attack(Player p)`, que s'utilitza quan un jugador vol atacar un altre. Quan un jugador A vol atacar un altre jugador B, es fa la crida `A.attack(B)`. Este mètode consisteix en que B siga colpejat per A (una crida al mètode `hit()` de B) i que, en cas de que B encara estigui amb vida li podrà tornar el colp, i A siga colpejat per B (una crida al mètode `hit()` de A). Abans i després de fer estes 2 accions, caldrà mostrar les dades de cadascun dels 2 jugadors. Per exemple:

```
    //    ABANS DE L'ATAC:
    Atacant: John Smith PA:13 / PD:8 / PV:39
    Atacat:  Martian PK PA:27 / PD:2 / PV:32
    //   ATAC:
    Martian PK és colpejat amb 13 punts i es defén amb 2. Vides: 32 - 11 = 21
    John Smith és colpejat amb 27 punts i es defén amb 8. Vides: 39 - 19 = 20
    //    DESPRÉS DE L'ATAC:
    Atacant: John Smith PA:13 / PD:8 / PV:20
    Atacat: Martian PK PA:27 / PD:2 / PV:21
```

> Nota: les línies de l'atac les mostra el mètode `hit` en les dos crides corresponents que es fan a aquest mètode.

- Afig el mètode void `hit(int attackPoints)`. Serà **`protected`** (només accessible des de la classe i classes hereves), ja que només ha de cridar-se des del mètode `attack`. És cridat des del mètode `attack` (dos vegades) per a dir que un jugador és colpejat per un altre amb tants punts d'atac. El mètode ha de restar tants punts de vida com la diferència entre els punts amb què l'ataquen i els punts de defensa que té. Els punts de vida mai podran ser augmentats ni ser menors que 0. Mostrarà un missatge dient qui és atacat, amb quants punts l'ataquen, amb quants punts es defén, quanta vides tenia, quantes li'n lleven i quantes en tindrà finalment.

4. En la classe JocDeRol del paquet inici modifica la funció `provaFase()`, que ens mostre el funcionament de la funció dels atacs.

- Eixa funció ha de crear un objecte de cada tipus (Human, Warrior i Alien).
- Fer alguns atacs entre ells
- Comprovar en l'eixida que les puntuacions estan ben efectuades

1. Si estàs programant-ho en Python, ara deuries de fer un nou commit amb el nom `rol_v2`

Fase 3. Sobrecàrrega i Polimorfisme

Explicació

Com hem dit abans, les classes hereves de `Player` són especialitzacions seues. És a dir: `Player` implementa un comportament genèric que cadascuna de les classes hereves pot modificar. Este canvi de comportament es pot realitzar bé afegint atributs i mètodes propis a la classe hereva o bé tornant a codificar algun dels mètodes de la classe heretada.

El *polimorfime* consistix en utilitzar el mecanisme de redefinició (overriding en anglés). Consistix en redefinir. És a dir: tornar a codificar el comportament heretat d'acord a les necessitats d'especialitació de la classe hereva.

En el nostre cas, volem tornar a codificar els mètodes necessaris a cada classse hereva per a aconseguir el següent comportament:

- Els jugadors de tipus `Human` no podran tindre més de 100 punts de vida i, per tant, s'ha de limitar esta característica al moment de la seua creació.
- Els jugadors de tipus `Alien` embogixen quan ataquen, però obliden la seua defensa. Quan un `Alien` ataca, si no està greument ferit (punts de vida superiors a 20) augmenten en 3 els seus punts d'atac i disminueixen en 3 els seus punts de defensa. Si estan greument ferits (punts de vida iguals o inferiors a 20) es comporten de manera normal.

> Nota: Els punts d'atac queden augmentats també després de l'atac (i també els de la defensa després de defendre), és a dir aquestes modificacions es mantenen.

- Els jugadors de tipus `Warrior`, degut al seu entrenament, tenen una gran agilitat. Si el colp no és superior a 5 punts, aquest queda reduït a 0. El colp (*hit*) és la diferència entre l'atac que sofrix i la defensa del jugador.

Es demana

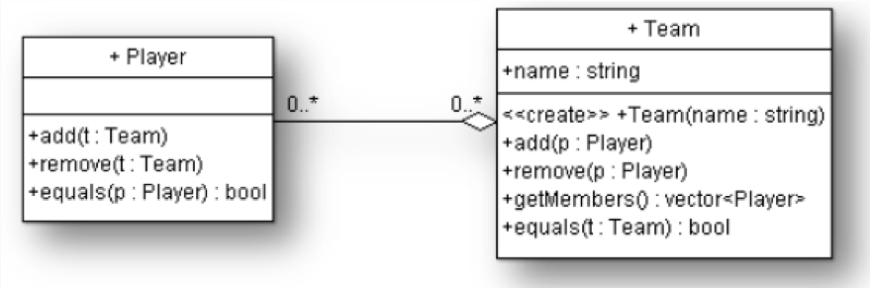
1. Clona el projecte a `JocDeRol_V3` (amb copia i refactorització).
2. Codifiqueu les classes `Human`, `Alien` i `Warrior`, sobreescrivint els mètodes necessaris per tal que tinguen el comportament descrit anteriorment.
3. En la classe JocDeRol del paquet inici adapta la funció `provaFase()` per a comprovar el funcionament. Eixa funció ha de crear, almenys, 3 jugadors de diferent tipus i mostrarà les seues dades. També ha d'incloure diversos atacs d'uns a altres.
4. Fer un `commit` a `JocDeRol_V4`, si estàs fent servir.

Fase 4. Relacions entre classes

Explicació

Volem crear equips de jugadors amb la següent característica. Qualsevol jugador pot afegir-se a un equip. Un equip, per tant, és un conjunt de jugadors, i cada jugador pot pertànyer a més d'un equip.

El diagrama UML que representa esta associació entre les classes Player i Team és el següent:



És a dir: podrem `afegir`, `esborrar` i `llistar` els jugadors que pertanyen a un equip.

Ademès implementarem un nou mètode per a la realització de les comparacions : implementar el mètode `equals`.

La classe `Team` també tindrà el mètode `toString()` , que servirà per a retornar en forma de cadena els jugadors que formen l'equip.

Fixeu-vos que, quan un jugador s'afegeix a un equip, l'equip s'afegeix alhora a la llista d'equips del jugador. La relació `Team-Player` és *bidireccional* i, per tant, `Player` coneix els seus `Team` i `Team` coneix els seus `Player`.

Es demana

1. Clona el projecte a `JocDeRol_V4` (amb copia i refactorització).
2. Crea la classe `Team` en el paquet joc, amb l'atribut `name`:

- Crea el constructor, passant-li el nom de l'equip
- Per a implementar la relació entre classes definida al diagrama anterior, afegeix un atribut privat de tipus `ArrayList`, anomenat `players`, i els mètodes que apareixen al diagrama per a afegir membres, llevar-los o bé consultar-los. També caldrà incloure el mètode `toString()` que retorne el nom de l'equip i les dades dels seus jugadors entre parèntesi. Per exemple:

```
    Equip Els Guais:
        John Smith PA:13 / PD:8 / PV:39 (pertany a 2 equips)
        Geronimo PA:9 / PD:4 / PV:100 (pertany a 1 equip)
```

- Afegir el mètode `equals`. El criteri d'igualtat és el contingut de tot l'objecte.

1. Modifica la classe `Player`:

- Per tal que puguen afegir-se els equips d'un jugador caldrà afegir un atribut privat de tipus `ArrayList`, anomenat `teams`, i els mètodes corresponents (per a afegir un jugador a un grup o per a llevar-lo). Tin en compte que si assignes un jugador a un grup, automàticament s'ha d'assignar també el grup al jugador, i viceversa (vés en compte: no provoques *recursió infinita*). També per a llevar un jugador d'un grup.
- El mètode `toString()` s'haurà de modificar per a retornar entre parèntesi la quantitat d'equips als quals pertany el jugador. Per exemple:

```
    John Smith PA:13 / PD:8 / PV:39 (pertany a 2 equips)
```

- Afegir el mètode equals. El criteri d'igualtat és el nom del jugador i els punts d'atac i defensa i la vida.

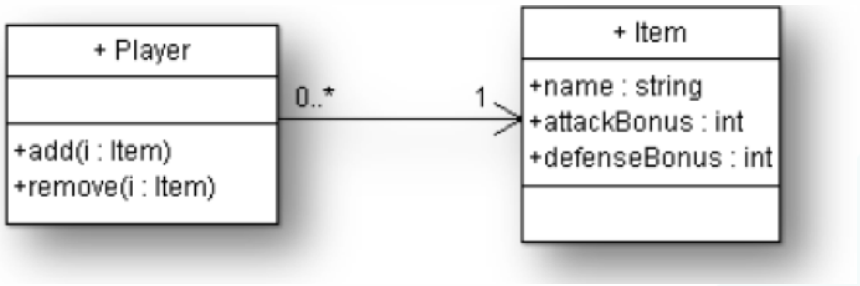
1. En la classe `JocDeRol` del paquet inici adapta la funció `provaFase()` per a comprovar el funcionament de les últimes modificacions fetes. Eixa funció ha de crear alguns jugadors i equips. També ha d'assignar jugadors a equips, desassignar-los, etc. També ha de comprovar la igualtat entre alguns `Team` i `Player`.

Fase 5. Més relacions

Explicació

Per tal d'afegir-hi interès, volem implementar en el nostre joc la possibilitat que els jugadors porten armes o escuts que modifiquen la seua capacitat d'atac i de defensa. Cada jugador pot portar múltiples armes, però cada arma pertany a un únic jugador. A dites armes i escuts les anomenarem `Item`

- Cada arma té un nom, un bonus d'atac i un bonus de defensa. Quan un jugador es dispose a atacar, ho farà amb la suma dels seus punts d'atac habituals més la suma de tots els bonus d'atac de les seues armes.
- De la mateixa manera, quan un jugador es dispose a defendre, ho farà amb la suma dels seus punts de defensa habitual més la suma dels bonus de defensa de totes les seues armes.
- Els bonus d'atac i de defensa poden ser negatius i, per tant, penalitzar al jugador.
- Les característiques individuals de cada tipus de jugador (Human, Alien i Warrior) no es modifiquen, sinó que es deixen igual que estaven.



Es demana

1. Clona el projecte a `JocDeRol_V5` (amb copia i refactorització).
2. Crea la classe `Item` en el paquet joc, amb els 3 atributs corresponents i un constructor al qual li passes els 3 paràmetres.
3. Fes els canvis necessaris a la classe Player per a implementar el funcionament anterior. És a dir:

- Crea un vector d'ítems anomenat items.
- Implementa els mètodes `add` i `remove` per a afegir i llevar un ítem a un jugador.
- A l'hora d'atacar o defendre, augmenta els punts d'atac o de defensa amb la suma dels punts dels items d'un jugador.

4. El mètode `toString()` s'haurà de modificar per a retornar també els noms dels seus ítems amb els respectius bonus. Per exemple:

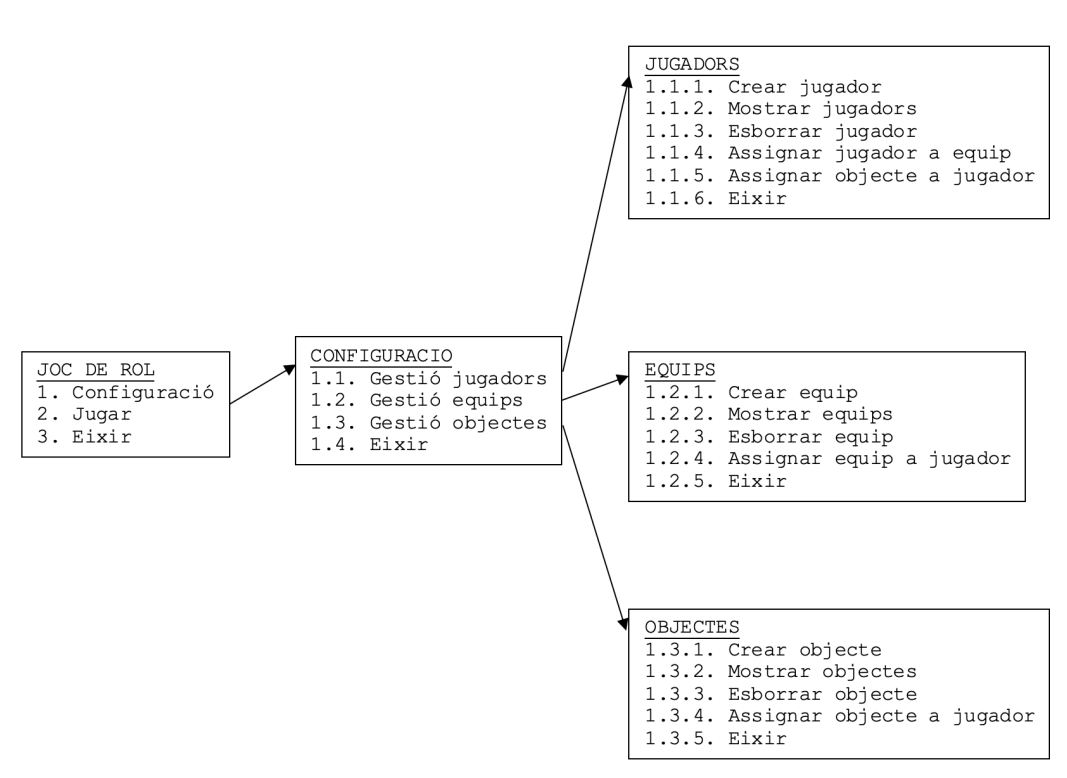
```
John Smith PA:10 / PD:19 / PV:39 (pertany a 1 equip) té els ítems:

- Sunglasses BA:-1 / BD:-1
- False Nails BA:5 / BD:2
```

1. En la classe `JocDeRol` del paquet inici adapta la funció `provaFase()` per a comprovar el funcionament.

Fase 6. Configuració del programa principal i Excepcions

Explicació

En el programa principal (en la classe JocDeRol del paquet inici) definirem un ArrayList per a guardar els jugadors, altre per als grups i altre per a les armes. El que volem fer és tota la configuració de donar d'alta els jugadors, items i equips.



Es demana

1. Clona el projecte a `JocDeRol_V6` (amb copia i refactorització) i implementar:

Configuració

En esta opció crearem tots els objectes de l'aplicació. Per exemple, l'opció de crear jugador (1.1.1) podria fer el següent:

1. Preguntar el tipus de jugador (A, W, P), el nom i els punts d'atac (entre 1 i 100)

2. Assignar els punts de defensa (PA + PD = 100) i els punts de vida inicials. Este valor serà igual per a tots els jugadors. Per tant, hauria d'anar com a `static` en la classe corresponent i es podria modificar en l'apartat de configuració. Per defecte, 100.
3. Crear el jugador i posar-lo en l'`ArrayList` de jugadors, comprovant prèviament que no existia ja un jugador *igual*. 2 jugadors seran iguals si tenen el mateix nom. Caldrà implementar el mètode `equals` de la classe `Player`.

Jugar

Bucle on li toque el torn cada vegada a un jugador de l'ArrayList de jugadors. Este triarà a quin jugador vol atacar de manera aleatòria entre tots els altres. Així fins que només quede un jugador viu, que seria el guanyador.

Tira-li imaginació i crea les teues normes!

Excepcions

Crear i llançar (i capturar) les següents situacions anòmales al llarg del joc:

Exercici: Crea les següents excepcions al projecte

1. Un jugador mort no pot atacar ni ser atacat
2. Un jugador no pot atacar-se a ell mateix.
3. No podem llevar d'un equip a un jugador que no li pertany.
4. Un equip no pot tindre jugadors repetits i viceversa.