EXERCICI SOBRE POO EN JAVA

JOC DE ROL

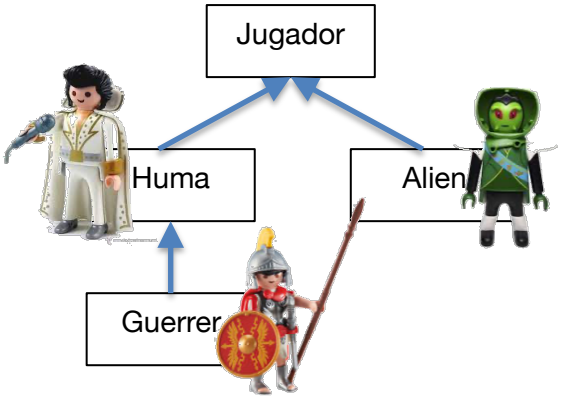
El joc consisteix en tindre una sèrie de jugadors, de forma que s'ataquen els uns als altres i vagen perdent vides, etc., fins que finalment només hi haja un viu.

El programa consta de diverses fases. En cada fase hi ha una explicació prèvia i després estan els exercicis corresponents que hauràs de fer.

FASE 1: INTRODUCCIÓ. HERÈNCIA I CONSTRUCTORS

Volem programar un joc de rol on hi haurà molts personatges o jugadors (objectes de la classe *Jugador*). Hi haurà 3 tipus diferents de jugador: humà (classe *Huma*), guerrer (classe *Guerrer*) o alienígena (classe *Alien*).

La relació dels diferents tipus de jugadors es mostra en el este esquema:



Veiem que un *Guerrer* és una especialització d'un *Huma*. I *Alien* i *Huma* són també una especialització de *Jugador*.

Cada personatge (*Jugador*) té unes característiques privades: els punts d'atac, els punts de defensa i les vides que té.

Ja vorem que cada tipus de jugador tindrà un comportament diferent:

Huma: No tenen bonificacions en defensa ni en atac.

Alien: Tenen bonificacions en atac i penalitzacions en defensa.

Guerrer: Poden aguantar més ferides que la resta de jugadors.

Exercicis:

1. Crea l'aplicació ***JocDeRolNomTeu*** amb els següents paquets i classes:
   - ✓ personatges: crea les 4 classes (*Jugador, Huma, Alien, Guerrer*), un fitxer per a cada classe, i amb les relacions d'herència corresponent. No els poses encara cap atribut.
   - ✓ inici: tindrà la classe *JocDeRol*, amb el programa principal (*main*).
   - ✓ teclat: tindrà la classe *Teclat* amb els mètodes per a llegir de teclat. En compte de paquet pots usar-ho com a llibreria.

2. En la classe *JocDeRol* del paquet *inici* crea el procediment *provaFase1()*, que ens ajudarà a entendre com funciona el mecanisme de l'herència. Eixe procediment ha de crear un objecte de cada tipus (Huma, Guerrer i Alien). Abans de crear cada objecte avisarà per pantalla el tipus d'objecte que va a crear ("Vaig a crear un Guerrer").

3. Recorda que quan es crida al constructor d'una classe, este crida automàticament, i primer que res, al constructor de la classe de la qual hereta (i així successivament). Per a comprovar-ho, crea els constructors de les 4 classes, sense paràmetres de forma que cadascun mostre un text per pantalla indicant la classe corresponent. Per exemple: "Sóc el constructor de Huma". Comprova que:

   Si fem un ***new Guerrer()*** es mostrarà:

   Sóc el constructor de Jugador

   Sóc el constructor de Huma

   Sóc el constructor de Guerrer

   Si fem un ***new Huma()*** es mostrarà:

   Sóc el constructor de Jugador

   Sóc el constructor de Huma

Ara volem que els constructors mostren també el nom de la classe de la qual s'està fent el *new*. Modifica els constructors per a que:

Si fem un ***new Guerrer()***, es mostre:

Sóc el constructor de Jugador però estic creant un **Guerrer**.

Sóc el constructor de Huma però estic creant un **Guerrer**.

Sóc el constructor de Guerrer però estic creant un **Guerrer**.

Si fem un ***new Huma()*** es mostre:

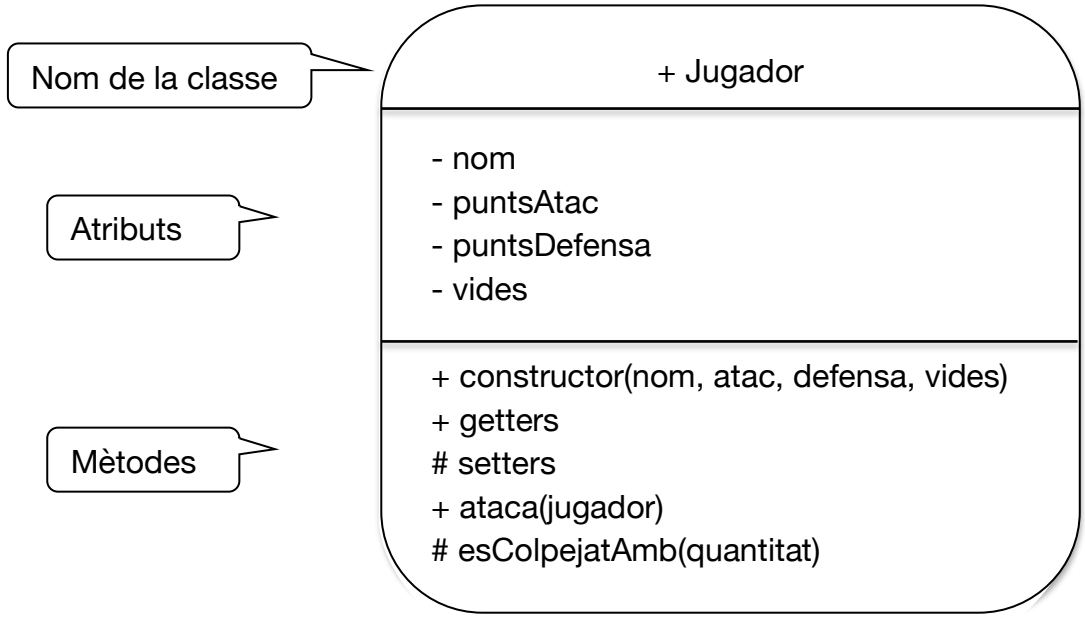
Sóc el constructor de Jugador però estic creant un **Huma**.

Sóc el constructor de Huma però estic creant un **Huma**.

És a dir: una cosa és el nom de la classe on està el constructor i altra el nom de la classe amb què s'ha fet el new. Per a obtindre el nom d'esta classe, cal usar *this.getClass().getSimpleName()*.

FASE 2: DONEM COS ALS JUGADORS

Cada jugador tindrà un nom, uns punts d'atac, punts de defensa, i vides. I tindrà mètodes per a atacar un altre jugador. Este seria el diagrama de la classe *Jugador*:



Els atributs seran privats. Els únics mètodes públics seran el constructor, els *getters* i el mètode *ataca*. Els altres mètodes seran *protected* (com si foren privats però sí que els poden usar les classes filles).

Exercicis:

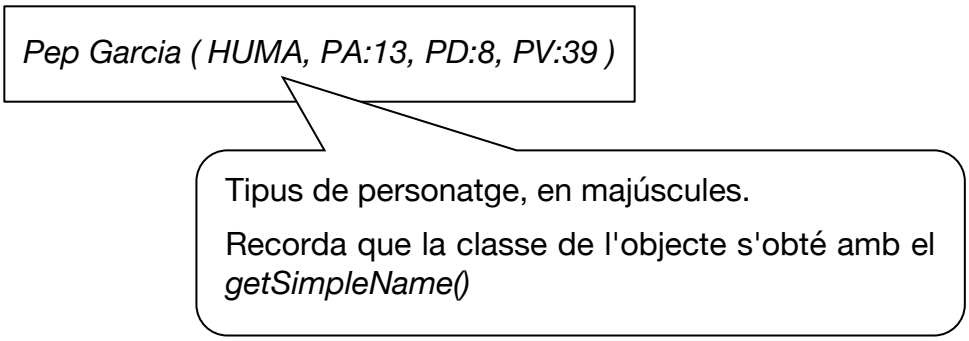
Modifica la classe *Jugador*:

4. Afig 4 atributs privats: *nom, puntsAtac, puntsDefensa, vides*

5. Modifica el constructor per a passar-li els 4 paràmetres i que inicialitze els atributs corresponents. No ha d'existir un constructor sense paràmetres. Fes el mateix en els constructors de les altres classes.

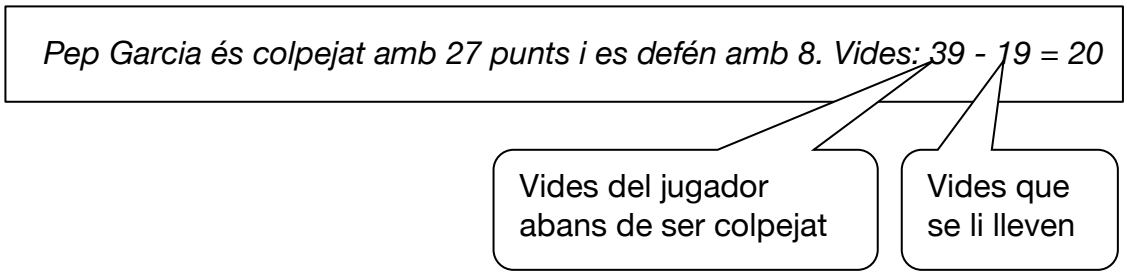
6. Crea els *getters* i *setters* per als 4 atributs. Els *getters* han de ser públics però no els *setters*. Estos només volem que puguen ser accedits des de la pròpia classe i per les filles. Per tant, hauran de ser *protected*.

7. Crea el mètode *toString()* (realment estaràs sobreescrivint eixe mètode de la classe *Object*). Retornarà totes les dades del jugador, amb el següent format:



8. Crea el mètode *esColpejatAmb()*, que rep la quantitat de punts d'atac amb els quals estan colpejant al jugador.

- El mètode serà *protected* (només accessible des de la classe *Jugador* i classes filles), ja que només l'ha de cridar el mètode *ataca* que farem després.
- El mètode ha de restar vides al jugador. Tantes vides com la diferència entre els punts amb què l'ataquen i els punts de defensa que té. Per exemple, si és colpejat amb 27 punts d'atac, i el jugador té 8 punts de defensa, al jugador se li restaran 19 vides. Però tin en compte que un jugador no pot tindre vides negatives, ni pot augmentar les vides que tenia.
- Mostrarà un missatge dient qui és colpejat, amb quants punts el colpegen, amb quants punts es defén (punts de defensa del jugador), quantes vides tenia, quantes li'n lleven i quantes en tindrà finalment. Per exemple:



9. Crea el mètode públic *ataca()*, que rep un jugador com a paràmetre (no un nom de jugador, sinó un objecte de la classe *Jugador*).

- Ja vorem que serà invocat en el programa principal, quan un jugador vullga atacar un altre. És a dir: quan un jugador *jug1* vol atacar un altre jugador *jug2*, es farà la crida *jug1.ataca(jug2)*.
- Suponent que es crida a la funció amb *jug1.ataca(jug2)*, este mètode consisteix en que *jug2* siga colpejat amb els punts d'atac de *jug1* (una crida al mètode *esColpejatAmb*) i que *jug1* siga colpejat amb els punts d'atac de *jug2* (altra crida al mètode *esColpejatAmb*). Abans i després de fer estes 2 accions, caldrà mostrar les dades de cadascun dels 2 jugadors. Per exemple:

Usa el *toString()* per a mostrar les dades de cada jugador.

ABANS DE L'ATAC:
Atacant: Pep Garcia (HUMA, PA:13 / PD:8 / PV:39)
Atacat: Superpep (GUERRER, PA:27 / PD:2 / PV:32)

ATAC:
Superpep és colpejat amb 13 punts i es defén amb 2. Vides: 32 - 11 = 21
Pep Garcia és colpejat amb 27 punts i es defén amb 8. Vides: 39 - 19 = 20

DESPRÉS DE L'ATAC:
Atacant: Pep Garcia (HUMA, PA:13 / PD:8 / PV:20)
Atacat: Superpep (GUERRER, PA:27 / PD:2 / PV:21)

Estes 2 línies de l'atac les mostra el mètode *esColpejatAmb* en les 2 crides a este mètode.

10. En la classe *JocDeRol* del paquet *inici* crea la funció *provaFase2()* per a comprovar el funcionament. Crea alguns jugadors, mostra les seues dades i fes atacs entre ells per vore si el resultat és l'esperat.

FASE 3: POLIMORFISME

Com hem dit abans, les classes hereves de *Jugador* són especialitzacions seues. És a dir: *Jugador* 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 filla.

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

- ✓ Els jugadors de tipus *Huma* no podran tindre més de 100 punts de vida. Per tant, modifica el seu constructor per a que si s'intenta crear amb més de 100 vides, que li'n pose 100 directament.
- ✓ Els jugadors de tipus *Alien* embogixen quan ataquen, però obliden la seua defensa. Quan un *Alien* ataca, si no està greument ferit (més de 20 vides) els seus punts d'atac augmenten en 3 però també disminueixen en 3 els seus punts de defensa. Si estan greument ferits (20 vides o menys) es comporten de manera normal. Nota: Els punts d'atac i defensa queden modificats també després de l'atac.
- ✓ Els jugadors de tipus *Guerrer*, degut al seu entrenament, si la ferida és lleu, no li afecta. Concretament, si li van a llevar menys de 5 vides, no se li'n lleva cap. Com ja sabem, la ferida és la diferència entre l'atac que sofrix i la defensa del jugador.

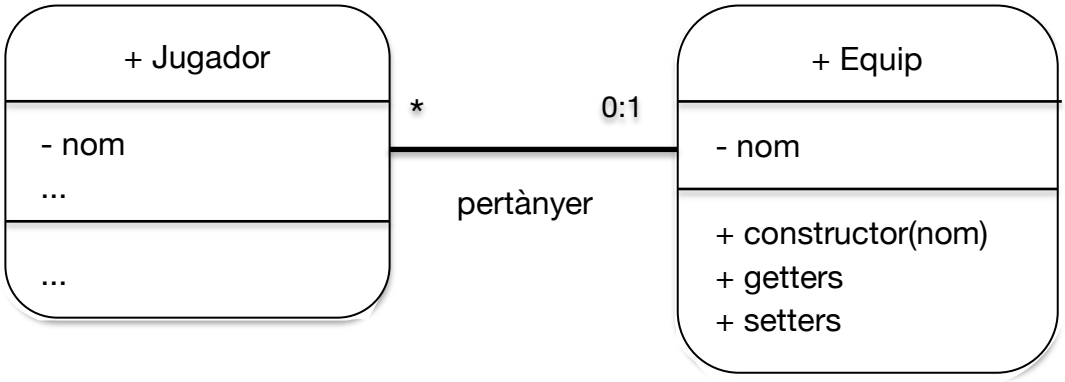
Exercicis:

11. Sobreescriviu els mètodes necessaris en les classes *Huma*, *Alien* i *Guerrer*, per tal que tinguen el comportament descrit anteriorment.

12. En la classe *JocDeRol* del paquet *inici* crea la funció *provaFase3()* per a comprovar el funcionament. Crea jugadors de distintes classes i fes atacs entre ells de forma que pugues comprovar els canvis de comportament fets en cada classe.

FASE 4: RELACIONS ENTRE CLASSES

Volem crear equips de jugadors. La idea és tindre en el programa principal una llista de jugadors i una llista d'equips, i poder associar jugadors a equips. Un equip podrà tindre molts jugadors però un jugador només pot pertànyer a un equip. Diguem que la relació entre jugadors i equips és 1:M (recorda les cardinalitats de bases de dades). En UML es representaria així:



Possibles solucions per a associar jugadors a equips:

a) Unidireccional:

- Cada jugador tindrà una referència de l'equip al qual pertany.

b) Unidireccional:

- Cada equip tindrà una llista amb els jugadors que li pertanyen.

c) Bidireccional:

- Cada jugador tindrà una referència de l'equip al qual pertany.

- Cada equip tindrà una llista amb els jugadors que li pertanyen.

Segons els usos que pretenem fer serà millor una solució o altra. En la solució bidireccional té redundància però ens serà més fàcil accedir a totes les dades. Per tant, ho farem d'esta forma.

Per a aconseguir-ho, en la classe *Jugador* posarem altre atribut per a indicar l'equip al qual pertany. I en la classe *Equip* posarem un altre atribut que serà una llista de jugadors. I els mètodes corresponents per a assignar un jugador a un equip o per a llevar-lo. Està detallat en els exercicis següents.

Exercicis:

13. Crea un altre paquet, anomenat *altres*. Crea en ell la classe *Equip*, i allí:
    a. Crea l'atribut privat *nom*, amb *getter* i *setter*.
    b. Crea un constructor que reba el nom de l'equip com a paràmetre.
    c. Crea l'atribut *jugadors*, que serà un *ArrayList* de jugadors (no de noms de jugadors sinó d'objectes *Jugador*) per a guardar els jugadors que pertanyen a l'equip.
    d. Crea el mètode *posa()*, per a posar un jugador en l'equip. Se li passa com a paràmetre un jugador (no el nom, sinó l'objecte). El mètode ha de posar eixe objecte en la llista de jugadors de l'equip.
    e. Evita que en un equip hi haja 2 jugadors iguals. Per a fer això:
        - Implementa el mètode *equals()* en la classe *Jugador* (un jugador serà igual a un altre si té el mateix nom).
        - Modifica el mètode *posa()* per a no posar el jugador si ja estava en la llista. En compte de recórrer la llista per a vore si ja estava, usa el *contains* de l'*ArrayList* (ja que este usa l'*equals()* que has definit en la classe *Jugador*).
    f. Crea el mètode *lleva()*, per a llevar un jugador de l'equip. Se li passa com a paràmetre un nom de jugador. En compte de recórrer l'*ArrayList* per a trobar l'element a esborrar, usa el *remove()* de l'*ArrayList*. Per a això caldrà passar-li com a paràmetre al *remove()* un jugador amb eixe nom, per a que n'esborre un "igual" que eixe (internament el *remove()* també crida a l'*equals()* de *Jugador*).
    g. Fes el *toString()* que retorne una cadena amb el nom de l'equip i les dades dels seus jugadors, com en este exemple:

```
Equip Els peps:
   - Pep Garcia ( PA:13 / PD:8 / PV:39 )
   - Superpep ( PA:9 / PD:4 / PV:100 )
```

Recorda que eixa informació ja la tenim amb el *toString()* de *Jugador*.

14. Bidireccionalitat: la classe *Jugador* també tindrà a quin equip pertany:

a. En la classe *Equip* ja hem posat els jugadors que té cada equip. Però com volíem fer l'associació bidireccional, ara en cada jugador posarem a quin equip pertany. Per tant, crea en la classe *Jugador* l'atribut *equip*, però no serà el nom de l'equip sinó un objecte de la classe Equip. Fes-lo privat i amb *getter* i *setter*.

b. Modifica el toString de Jugador per a que també mostre el nom de l'equip al qual pertany (entre claudàtors). Per exemple:

> *Pep Garcia* ***[Els peps]*** *(PA:13, PD:8, PV:39)*

c. Modifica el mètode *posa()* (de la classe *Equip*) per a que pose també a eixe jugador l'equip al qual pertany (posar l'equip en l'atribut equip del jugador).

d. Modifica el mètode *lleva()* (de la classe *Equip*) per a que lleve també a eixe jugador l'equip al qual pertany (posar un *null* a l'atribut equip del jugador).

e. El mètode *setEquip()* (de la classe *Jugador*) està assignant un equip al jugador però no té en compte la bidireccionalitat. Per tant:

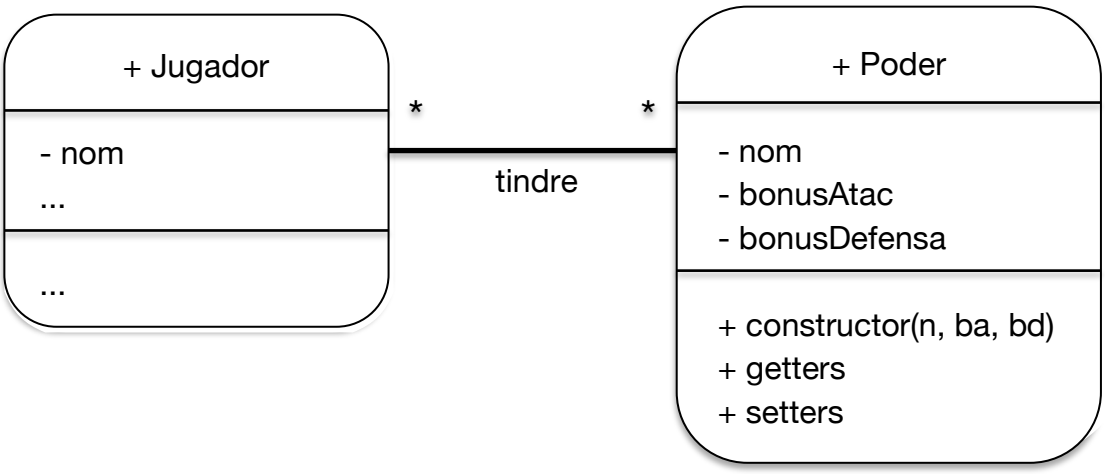
- Fes que *setEquip()* invoque al mètode *posa()* de la classe *Equip* per a que també el pose allí. Però compte! Cal evitar recursió infinita indirecta ja que *setEquip()* invoca a *posa()*, i *posa()* invoca *setEquip()*.
- A més, si a *setEquip()* se li passa un null, vol dir que estem llevant el jugador a l'equip. Per tant, caldria cridar a *lleva()* de la classe *Equip* (i no a *posa()*). Tin en compte també esta possible recursió indirecta.

15. En la classe *JocDeRol* del paquet *inici* crea el procediment *provaFase4()* que cree alguns jugadors i equips, i prova a assignar diverses vegades un mateix jugador a un equip, llevar un jugador, etc.

FASE 5: MÉS RELACIONS

Esta fase és opcional (però recomanable).
Sí que cal fer la fase 6.

Ara volem que els jugadors tinguen poders. L'objectiu és que en el programa principal tingam una llista de poders disponibles, per a poder assignar-los als jugadors, de forma que un jugador puga tindre molts poders. Un mateix poder el podran tindre molts jugadors. Cada poder té un nom, un bonus d'atac i un bonus de defensa



Possibles solucions per a associar jugadors a poders:

a) Unidireccional:

- Cada jugador tindrà una llista amb els poders que té.

b) Unidireccional:

- Cada poder tindrà una llista amb els jugadors que tenen eixe poder.

c) Bidireccional:

- Cada jugador tindrà una llista amb els poders que té.
- Cada poder tindrà una llista amb els jugadors que tenen eixe poder.

Com la relació entre jugadors-equips l'havíem feta bidireccional, la relació jugadors-poders la farem unidireccional. Concretament, l'opció a), ja que és la més adequada perquè després voldrem saber quins poders té cada jugador (no al revés).

Quan un jugador es dispose a atacar, ho farà amb la suma dels seus punts d'atac habituals més la suma dels bonus d'atac de tots els seus poders.

I 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 tots els seus poders.

Exercicis:

16. Crea la classe *Poder* en el paquet *altres*, amb:

- 3 atributs encapsulats: *nom*, *bonusAtac*, *bonusDefensa*.
- Constructor amb els 3 paràmetres.
- El *toString()*, amb este format d'exemple:

> *Invisibilitat (BA:10, BD:6)*

17. En la classe *Jugador*:

- Crea la llista *poders* (no de noms de poders sinó d'objectes de la classe *Poder*). Serà un *ArrayList* privat.
- Crea el mètode *posa()* per a posar un poder a un jugador. Se li passa com a paràmetre un poder i l'ha de posar en la seua llista de poders.
- Crea el mètode *lleva()* per a llevar un poder a un jugador. Se li passa com a paràmetre un poder i l'ha de llevar de la seua llista de poders.
- Fins ara un jugador colpejava amb els seus punts d'atac. Modifica el mètode que cregues per a que quan un jugador colpege, ho faça amb la suma dels seus punts d'atac més la suma dels bonus d'atac dels seus poders.
- I fins ara un jugador es defensava d'un colp amb els seus punts de defensa. Modifica el mètode que cregues per a que quan un jugador siga colpejat es defenga amb la suma dels seus punts de defensa més la suma dels bonus de defensa dels seus poders.
- El mètode *toString()* s'haurà de modificar per a retornar també els noms dels seus poders amb els respectiux bonus. Per exemple:

> *Pep Garcia (PA:10, PD:19, PV:39) té els poders:*
> *- Invisibilitat (BA:10, BD:6)*
> *- Volar (BA: 10, BD: 20)*
> *- Megaforça (BA: 20 / BD:10)*

Recorda que la informació de cada poder ja la tenim amb el *toString()* de *Poder*.

18. En la classe *JocDeRol* del paquet *inici* crea el procediment *provaFase5()* per a comprovar el funcionament (crear poders, afegir-los a jugadors, atacar i comprovar que es contemplen tots els punts d'atac i de defensa...)

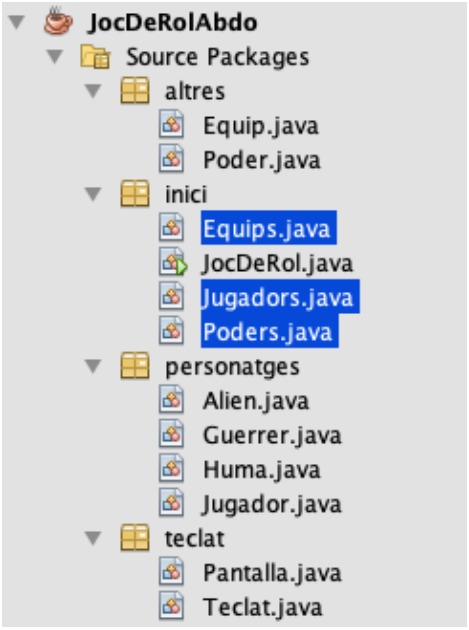
FASE 6: PROGRAMA PRINCIPAL

Ja tenim feta l'estructura: les classes necessàries. Ara farem la part principal del programa, on crearem les llistes d'objectes (una per a posar tots els jugadors del joc, altra per al tots els equips i altra per als poders) i començarem la partida.

Nota: si no has fet la part dels poders, no faces tampoc res d'esta fase que afecte als poders.

Exercicis:

19. En el paquet *inici* crearem tres classes: en cadascuna d'elles posarem la llista corresponent:



a. Crea en eixe paquet el fitxer *Jugadors.java* (en plural), on estarà la classe pública *Jugadors* (en plural), on estarà la llista de tots els jugadors del joc. Per a això, crea eixa llista (de nom *llista*) com un *ArrayList* d'objectes de la classe *Jugador*. Eixa classe només està per a guardar la llista de jugadors (i els mètodes que usen eixa llista). Per tant, des del *main* no crearem cap objecte d'eixa classe, sinó que accedirem a eixa llista posant el nom de la classe davant. És per això que la llista haurà de tindre el modificador *static* (com també l'hauran de tindre els mètodes que crees en eixa classe). No poses la llista com a *private*, però tampoc com com a *public* (és a dir, cap modificador d'accés). Així, la llista de jugadors podrà ser accedida directament des de qualsevol classe del paquet *inici*. Com deus saber, la forma d'accedir a ella des del paquet serà amb *Jugadors.llista*

b. Fes el mateix per als equips: una classe que es diga *Equips* (en un fitxer *Equips.java*) on tinga la llista d'equips (que també es dirà *llista*).

c. I fes el mateix per als poders.

20. Fes un bucle amb el menú inicial:

JOC DE ROL

1. Configuració
2. Jugar
0. Eixir

Recorda que en la classe *Teclat* tens la funció *lligOpcio* que t'ajuda en la gestió dels menús.

Depenent de l'opció triada, cridarem al procediment corresponent: *menuConfiguracio()* o *jugar()*.

21. Crea el procediment *menuConfiguracio()*. Des d'ací es crearan tots els objectes de l'aplicació (jugadors, equips i poders). Serà un bucle amb el menú:

CONFIGURACIÓ

1. Jugadors
2. Equips
3. Poders
0. Eixir

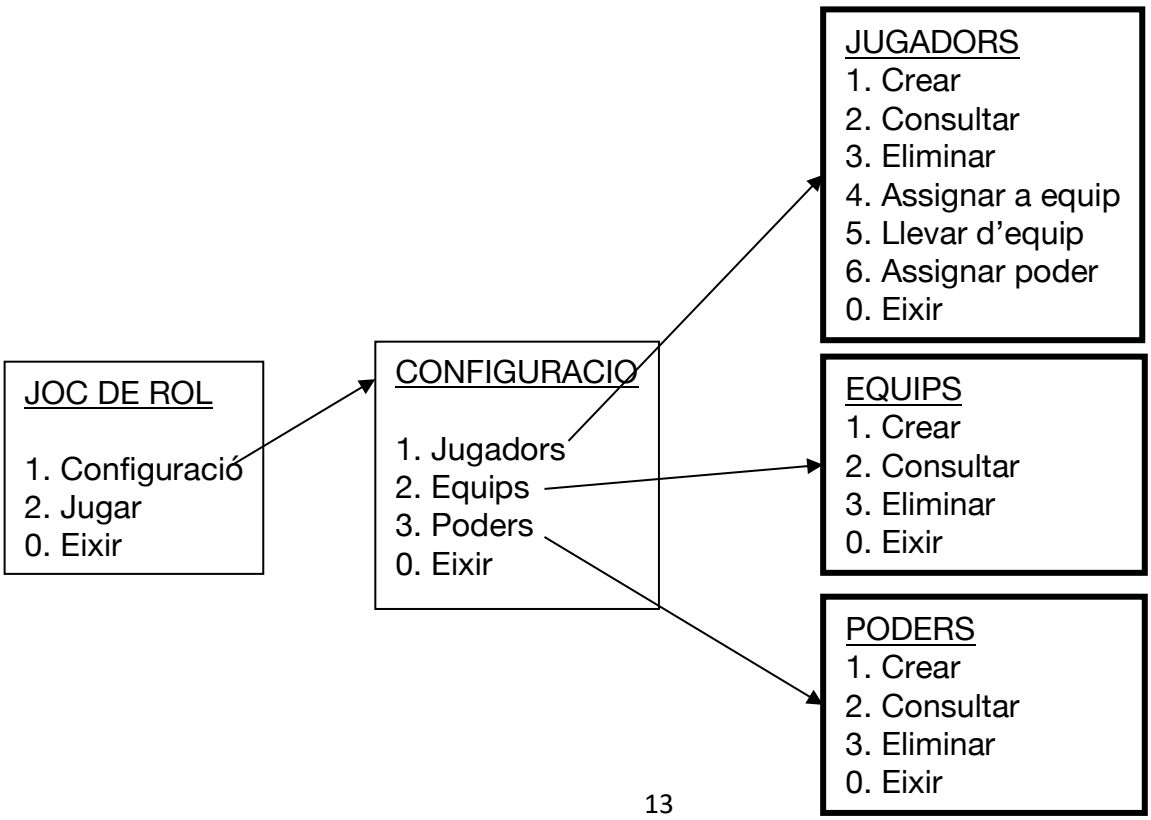
Depenent de l'opció triada, cridarem al procediment corresponent:

Jugadors.menu(), Equips.menu(), Poders.menu().

És a dir, crearem el procediment *menu()* en cadascuna de les 3 classes.

22. Completa l'estructura dels menús:

Ja has fet el menú del joc de rol i de la configuració. Fes ara (amb altres procediments) el menú de jugadors, el d'equips i el de poders.



23. Implementa cadascuna de les opcions del menú de **jugadors** (llevat de la d'eixir, clar), amb un procediment per a cada opció. Implementa eixes funcions en la **classe *Jugadors***. Recorda que hauran de tindre l'*static*:

a) *crear()*:

- Pregunta per teclat:
  - El tipus de jugador (H, G o A)
  - El nom del jugador
  - Els punts d'atac (entre 1 i 100)

Recorda que en la classe *Teclat* tens funcions que fan el control sobre els possibles valors que s'han d'introduir.

- Els punts de defensa del jugador no es demanen per teclat sinó que es calcularan tenint en compte que els punts d'atac més els de defensa han de sumar 100. Per tant, si per a un jugador hem triat que tinga 80 punts d'atac, tindrà 20 punts de defensa.

- Les vides del jugador no es demanen per teclat sinó que serà la mateixa quantitat de vides per a tots els jugadors que es creen. Este valor estarà guardat en una variable en la classe *Jugador* però estàtica (ja que tindrà el mateix valor per a tots els jugadors). Dis-li a eixa variable *videsInicials*, que tindrà el valor de 200 (si vols, que es puga modificar en l'apartat de configuració).

- Crea el jugador i posa'l en la llista de jugadors, comprovant prèviament que no existia ja un jugador "igual.

No recórregues la llista per vore si ja existeix, sinó que has d'usar el *contains*

b) *consultar()*

Mostrarà llista dels jugadors de la partida. Simplement recorre la llista i fes un *sout* de cada jugador (el *sout* ja crida ell al *toString()* del jugador).

c) *eliminar()*

Demana el nom del jugador i esborra de la llista un jugador "igual" que eixe.

Fes-ho sense haver de recórrer la llista de jugadors. Si no saps, pregunta'm.

d) *assignarEquip()*

- Demana el nom del jugador i el nom de l'equip.
- Obtín l'objecte jugador de la llista de jugadors.
- Obtín l'objecte equip de la llista d'equips.
- Posa el jugador en eixe equip cridant al mètode *posa()*.

No recórregues les llistes. Usa l'*indexOf()* i el *get()* de l'*ArrayList().*

e) *assignarPoder()*

Fes el mateix que has fet en l'apartat anterior però per als poders.

24. Implementa cadascuna de les opcions del menú d'**equips**, amb un procediment per a cada opció (que també es diguen *crear(), consultar()* i *eliminar()*). Implementa eixes funcions en la **classe *Equips***. Per a crear un equip, tin en compte que no s'han de permetre que en la llista d'equips hi haja equips amb el mateix nom.

25. Implementa cadascuna de les opcions del menú de **poders**, amb un procediment per a cada opció (que també es diguen *crear(), consultar()* i *eliminar()*). Implementa eixes funcions en la **classe *Poders***. No podrà haver 2 poders amb el mateix nom.

26. Procediment *jugar()*

Tria quines de les 2 opcions vols fer:

a) Automatitzat

Bucle fins que només quede 1 jugador viu:

- Tria aleatòriament el jugador de la llista que atacarà
- Tria aleatòriament el jugador de la llista que serà atacat
- Es farà l'atac invocant el mètode *atacar()*.

b) Manual

Bucle fins que només quede 1 viu:

- A cada vegada li tocarà atacar a un jugador de la llista. Després de l'últim tornarà a atacar el primer.
- Es preguntarà per teclat a quin jugador de la llista es vol atacar.
- Es farà l'atac

Tant en una opció de joc com en l'altra, finalment es mostrarà el guanyador.

FASE 7: LLISTES ORDENADES

Esta fase és opcional

FASE 8: GUARDEM LA PARTIDA

Esta fase és opcional

FASE 9: EXCEPCIONS PERSONALITZADES

Esta fase és opcional

A voltes en els nostres programes necessitem crear-nos les nostres pròpies excepcions, llençar-les quan toque i capturar-les. Per exemple, suposem que volem que bote una excepció quan intentem atacar un personatge que ja està mort.

Veiem quins passos caldria seguir (però només fes l'exercici que es detallarà després):

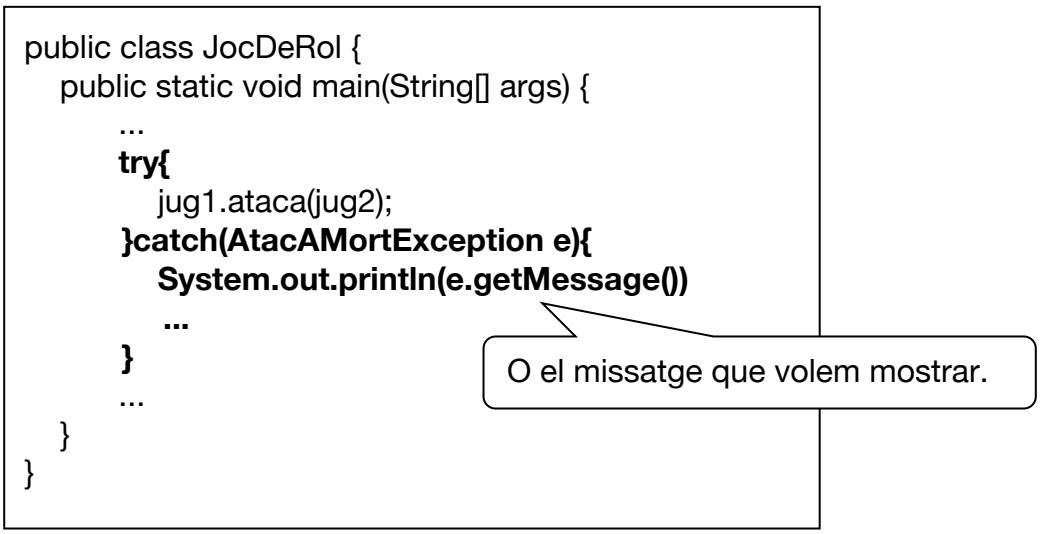
1. Creem una classe filla d'*Exception*, que es dirà *AtacAMortException* i que tinga el tractament de l'excepció (en, almenys, un constructor):

```
public class AtacAMortException extends Exception{

        // Constructor passant-li el text de l'error
        public AtacAMortException(String msg){
                super(msg);
        }

        // Constructor sense passar-li el text de l'error
        public AtacAMortException(){
                super("No es pot atacar a un mort");
        }
}
```

2. Llençar la nostra excepció en el mètode més concret que es puga. És a dir, en este cas, en el mètode *ataca()*, en compte de fer-ho en el *main()*. (En este cas hem optat per tirar el marró fora però s'haguera pogut tractar l'execepció dins, amb un *try-catch*)

```
    public void ataca(Jugador p) throws AtacAMortException{
        ... /
        if (  atacat no té vides ){
                throw new AtacAMortException()
        }
        p.esColpejatAmb(...)
        this.esColpejatAmb(...)
        ...
}
```

3. On es crida al mètode *ataca()* (en el programa principal), ens obligarà a capturar l'excepció amb el *try-catch* (ja que el mètode *ataca()* té un *throws*):



Exercicis:

27. Crea el paquet *excepcions*. Crea en ell les següents excepcions i fes el tractament corresponent:

- No es pot atacar a un mort (és l'excepció de l'exemple que hem vist)

- Un mort no pot ser atacat

- Un jugador no pot atacar-se a ell mateix.