EXERCICI SOBRE POO EN JAVA

JOC DE ROL

FASE 1: HERÈNCIA

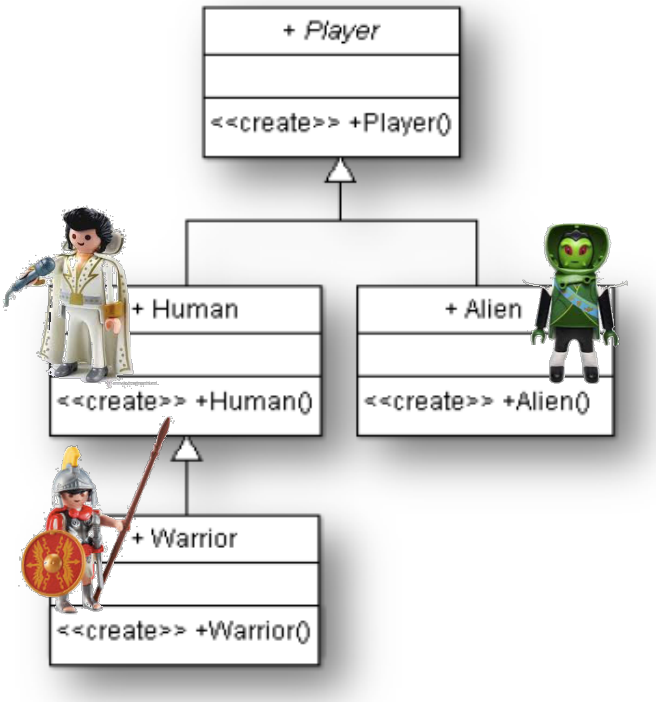
Volem programar un joc de rol on cada usuari ha de crear un personatge jugador (Player). Es pot triar entre 3 tipus diferents de jugador: humà (Human), guerrer (Warrior) o alienígena (Alien). Cada personatge té unes característiques privades, que són: els punts d'atac, els punts de defensa i els punts de vida.

El joc consistix en atacar els altres jugadors. Cada vegada que el jugador A ataca el jugador B, passen 2 coses:

1) B és colpejat per A
2) A és colpejat per B

Quan B és colpejat per A 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. 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.

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 atac i penalitzacions en defensa.
- ✓ Warrior: Poden aguantar més ferides que la resta de jugadors.

Exercicis:

1. Crea l'aplicació ***JocDeRol*** amb els següents paquets:
   - ✓ teclat: tindrà la classe Llegir (amb els mètodes llegirString(), llegirInt(), etc que vages necessitant.
   - ✓ personatges: tindrà les classes Player, Human, Alien, Warrior
   - ✓ inici: tindrà la classe JocDeRol, amb el programa principal (main).

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

"Sóc el constructor de <**nomClasse**>. Estic creant un objecte <**nomClasseInstanciada**>

On tenim que:

<**nomClasse**> és el nom de la classe on està el constructor

<**nomClasseInstanciada**> és el nom de la classe amb què s'ha fet el new que ha provocat la crida al constructor. Caldrà usar el getSimpleName().

3. En la classe JocDeRol del paquet inici crea la funció provaFase1(), 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 ("Vaig a crear un Warrior"). En executar-ho, comprovarem que quan es crida al constructor d'una classe, es crida automàticament, i primer que res, al constructor de la classe de la qual hereta (i així successivament).

Per exemple, si en programa principal creem un objecte de la classe Warrior ("... = new Warrior()") farà que es mostre per pantalla:

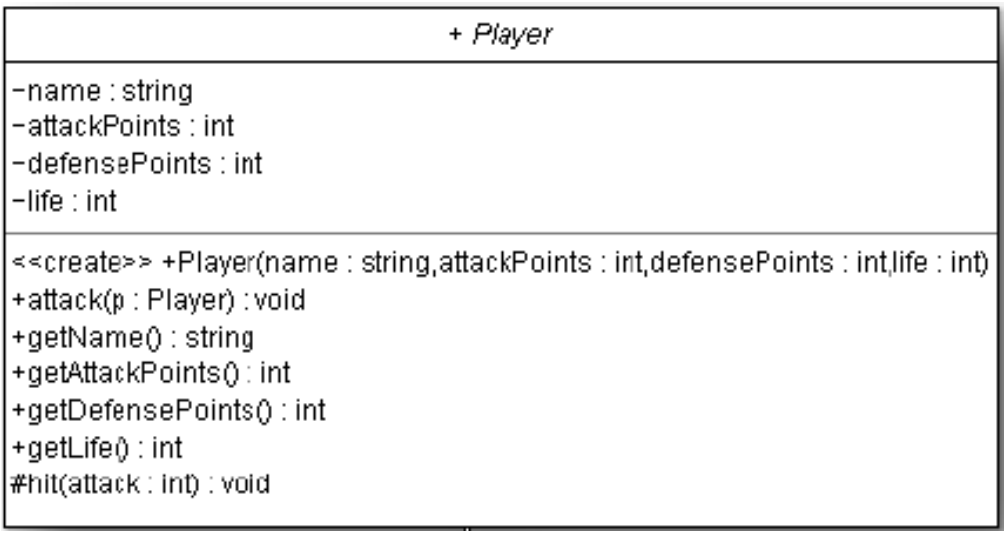
Sóc el constructor de Player. Estic creant un objecte Warrior

Sóc el constructor de Human. Estic creant un objecte Warrior

Sóc el constructor de Warrior. Estic creant un objecte Warrior

FASE 2: DONEM COS ALS JUGADORS

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:



Exercicis:

Modifica la classe *Player*:

4. Afig 4 atributs: name, attackPoints, defensePoints, life

5. Modifica el constructor per a passar-li els 4 atributs. Caldrà modificar també els constructors de les altres classes.

6. Afix els 4 mètodes *get* corresponents per a consultar cada atribut.

7. Sobreescriu el mètode toString() de la classe Object en la classe Playr. Retornarà totes les dades del jugador amb el següent format:

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

8. 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 A siga colpejat per B (una crida al mètode *hit*) i que B siga colpejat per 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: Plk Dfw PA:27 / PD:2 / PV:32
ATAC:
Plk Dfw 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: Plk Dfw PA:27 / PD:2 / PV:21

> *Nota*: les línies de l'atac les mostra el mètode *hit* en les dos crides corresponents a este mètode.

9. Afig el mètode *void hit(int attackPoints)*

a. Serà *protected* (només accessible des de la classe i classes hereves), ja que només l'ha de cridar el mètode *attack*.

b. 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.

c. Mostrarà un missatge dient qui és atacat, amb quants punts l'ataquen, amb quants punts es defén, quantes vides tenia, quantes li'n lleven i quantes en tindrà finalment. *Per exemple:*

John Smith és atacat amb 27 punts i es defén amb 8. Vides: 39 - 19 = 20

FASE 3: POLIMORFISME

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.
- ✓ 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 però també disminueixen en 3 els seus punts de defensa. Si estan greument ferits (puts de vida iguals o inferiors a 20) es comporten de manera normal. Nota: Els punts d'atac i defensa queden modificats també després de.
- ✓ Els jugadors de tipus Warrior, degut al seu entrenament, tenen una gran agilitat. Si la ferida no és superior a 5 punts, esta queda reduïda a 0. La ferida és la diferència entre l'atac que sofrix i la defensa del jugador.

Exercicis:

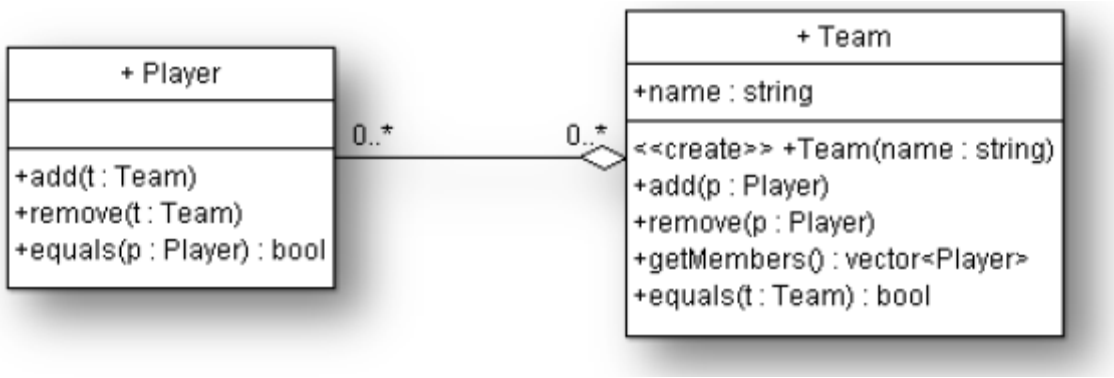
10. Codifiqueu les classes Human, Alien i Warrior, sobreescrivint els mètodes necessaris per tal que tinguen el comportament descrit anteriorment.

11. En la classe JocDeRol del paquet inici crea la funció provaFase3() 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.

FASE 4: RELACIONS ENTRE CLASSES

Volem crear equips de jugadors. 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. Per a la realització de les comparacions s'ha d'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'afegix a un equip, l'equip s'afegix 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*.

Exercicis:

12. Crea la classe Team en el paquet *inici*, amb l'atribut *name*.
    a. Crea el constructor, passant-li el nom de l'equip
    b. 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.
    c. 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)*

13. Modifica la classe *Player*:
    a. 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). També per a llevar un jugador d'un grup.

    b. 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)

14. En la classe *JocDeRol* del paquet *inici* crea la funció *provaFase4()* 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.

FASE 5: MÉS RELACIONS

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

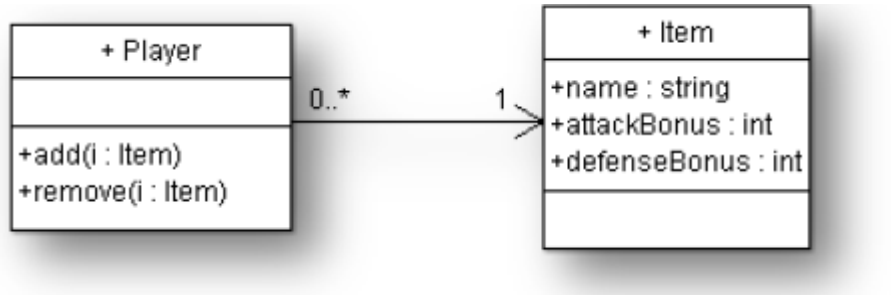
Cada arma té un nom, un bonus d'atac i un bonus de defensa. Quan un jugador es dispose a atacar, ho farà amb la suma dels seus punts d'atac habituals més la suma de tots els bonus d'atac de les seues armes.

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.

La relació que volem implementar quedaria de la següent manera (la classe Item representa la classe Arma):



Exercicis:

15. Crea la classe *Item* en el paquet *inici*, amb els 3 atributs corresponents i un constructor al qual li passes els 3 paràmetres.

16. Fes els canvis necessaris a la classe *Player* per a implementar el funcionament anterior. És a dir:

a. Crea un ArrayList d'ítems anomenat *items*.

b. Implementa els mètodes *add* i *remove* per a afegir i llevar un ítem a un jugador.

c. 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.

d. 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

17. En la classe *JocDeRol* del paquet *inici* crea la funció *provaFase5()* per a comprovar el funcionament. Pots utilitzar el fitxer *test4.java*, però fes tu les teues proves.

FASE 6: PROGRAMA PRINCIPAL

Exercicis:

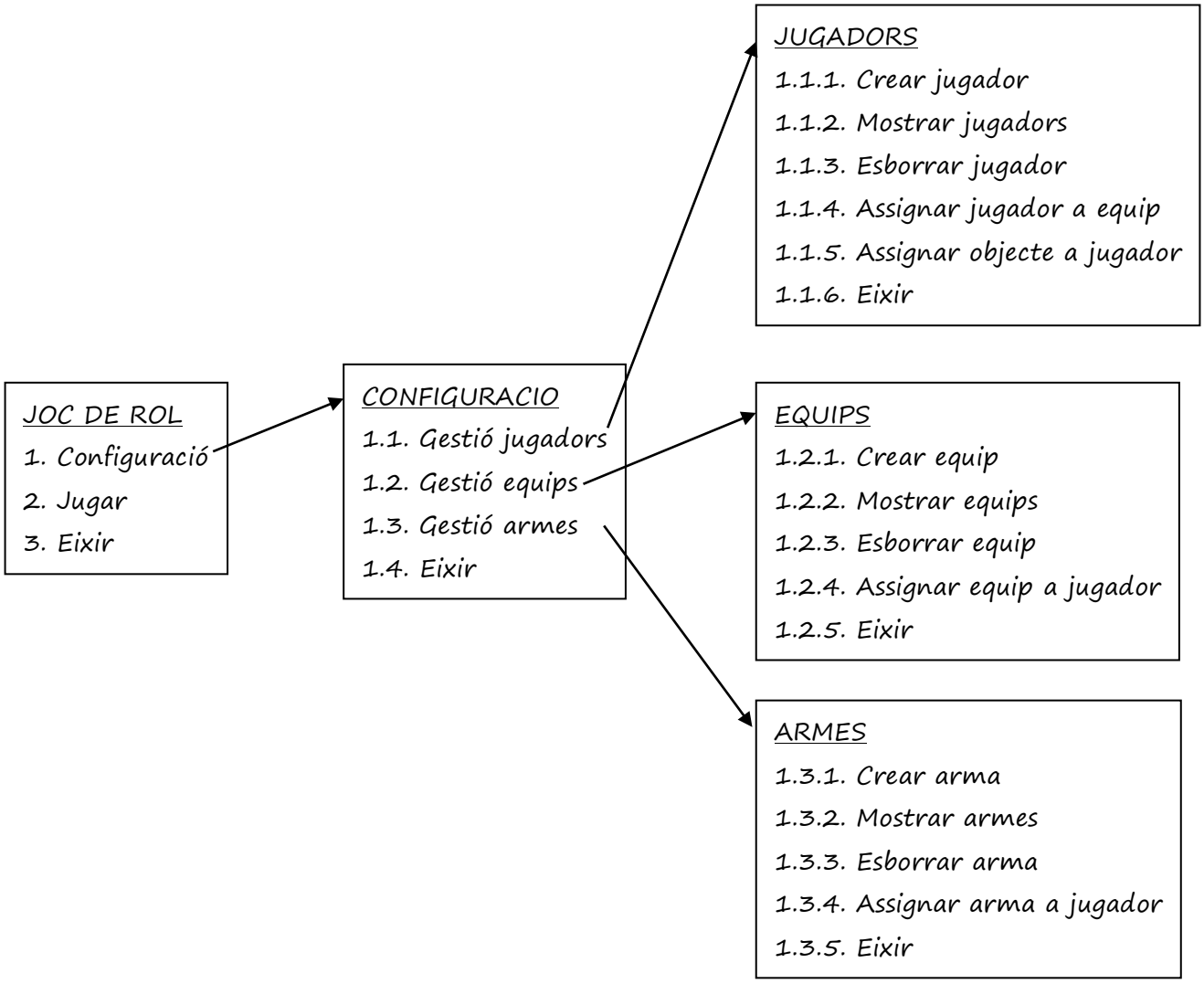
18. En el programa principal (en la classe JocDeRol del paquet inici) definirem un ArrayList per a guardar els jugadors, altre per als grups i altre per a les armes. I farem un bucle amb el menú inicial:

MENÚ INICIAL
1. Configuració
2. Jugar
3. Eixir

19. Opció de configuració. En esta opció crearem tots els objectes de l'aplicació (jugadors, equips i armes). Serà un bucle amb el menú:

CONFIGURACIÓ
1. Gestió jugadors
2. Gestió equips
3. Gestió armes
4. Eixir

És a dir, l'estructura de menús podria ser esta:



Per exemple, l'opció de crear jugador (1.1.1) podria fer el següent:

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

- Assignar els punts de defensa (sabent que PA + PD = 100), i els punts de vida inicials. Este valor serà igual per a tots els jugadors i, per tant hauria de posar-se en la classe corresponent com a estàtica. Esta variable (que no té res a vore amb els punts de vida que cada jugador té en un moment determinat) es podria modificar en l'apartat de configuració. Si no es posa res, serà 100.

- 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.

20. Opció de jugar. Consistirà en un bucle on li toque el torn cada vegada a un jugador de l'ArrayList de jugadors. Este triarà a quin jugador vol atacar. Així fins que només quede un jugador viu, que seria el guanyador. Tira-li imaginació i crea les teues normes!

FASE 7: EXCEPCIONS PERSONALITZADES (PART 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 salte una excepció quan intentem atacar un personatge que ja està mort.

Veiem quins passos caldria seguir:

1.- Creem una classe filla d'Exception, que es dirà AtacAMortException i que tinga el tractament de l'excepció en algun/s constructor/s:

```
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 attack, en compte de fer-ho en el main. (En este cas hem optat per tirar el marró fora però s'haguera pogut tractar dins amb un try-catch)

```
    public void attack(Player p) throws AtacAMortException{
        ... // mostrar dades de l'atacant i de l'atacat
        if (  atacat no té punts de vida ){
                throw new AtacAMortException()
        }
        p.hit(...)
        this.hit(...)
        ...
}
```

3. On es crida al mètode attack (main, segurament), ens obligarà a capturar l'excepció amb el try-catch (ja que el mètode attack té un throws):

```
public class JocDeRol {
    public static void main(String[] args) {
        ...
        try{
           jug1.attack(jug2);
        }catch(AtacAMortException e){
           System.out.println(e.getMessage())
           ...
        }
        ...
    }
}
```

Exercicis:

21. Crea les següents excepcions i fes el tractament corresponent:

- Un mort no pot atacar ni ser atacat

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

- No podem llevar d'un equip a un jugador que no li pertany.

- Un equip no pot tindre jugadors repetits i viceversa.