**Exercici 1 (2.5 punts)**

Abans de començar has de:

Llegir tots els apartats del 3.6 al 3.9 de la Guia de Java.

Mira el diagrama de classes anomenat PAC4Ex1.png que trobaràs en el zip de l'enunciat.

**Obre el projecte PAC4Ex1 amb IntelliJ**. Al *package* edu.uoc.pac4 del directori /src/test/java està l'arxiu amb els tests unitaris que et proporcionem. Al paquet edu.uoc.pac4 del directori /src/main/java has de codificar les classes GameMap i Position (juntament amb les corresponents classes per gestionar les excepcions), així com l'enumeracío GameMapType. Les seves representacions en diagrama de classes UML les tens disponibles a la imatge PAC4Ex1.png adjunta amb l'arxiu .zip d'aquest enunciat.

Com pots observar en el diagrama UML, modelarem el comportament d'un mapa (GameMap) d'un joc en línia juntament amb la seva associació amb una posició (Position) d'aquest mapa. **A més, la relació entre aquestes dues classes és una associació unidireccional d'agregació, sent la classe GameMap l'agregada a la classe Position**. Aquesta agregació és necessària per poder verificar si la posició que es vol crear no excedeix els límits del mapa (valors width, height o depth de la classe GameMap).

Per fer aquest exercici i comprendre el que estàs fent **codifica les classes** dins del *package* edu.uoc.pac4. Per codificar el diagrama de classes i la gestió d'excepcions, tingues en compte les següents especificacions:

**Enum GameMapType**● Els valors dels literals de l’enumeració són: CITY, VALLEY, DESERT, FOREST,

MOUNTAIN, OCEAN, PLAINS, ICE, VOLCANO y TEMPLE.

● **toString**: Si llegim la documentació oficial de Java, aquesta diu que toString retorna la "*representació textual de l'objecte*". Què vol dir això? Doncs que retorna una cadena de text (és a dir, un String) amb informació de l'objecte. Aquesta informació ha de ser concisa, però informativa/representativa de l'objecte perquè una persona pugui llegir-la. La recomanació és que totes les classes sobrescriguin aquest mètode, ja que la seva codificació per defecte és el nom de la classe a la qual pertany l'objecte, el símbol @ i la representació hexadecimal del codi hash de l'objecte (que sol ser l'adreça de memòria que ocupa l'objecte).

Per això, s'ha de sobreescriure aquest mètode (heretat de la classe Object de Java) i ha de retornar el nom del tipus de mapa amb la primera lletra en majúscula i la resta en minúscules. Per exemple, si s'invoquen amb el valor CITY, ha de retornar:

City

**Class GameMapException**

Utilitzar la classe Exception de Java per llançar excepcions quan es produeix un cas anòmal o indesitjat és correcte. No obstant això, quan en un programa es donen

excepcions que són particulars del problema/context que tractem, és millor crear les nostres pròpies excepcions (també anomenades excepcions personalitzades; en anglès, *custom exceptions* o *user-defined exceptions*) que siguin adequades per a les necessitats del nostre programa.

En aquest moment segurament et preguntaràs: *com faig la meva pròpia excepció?* Et sona un mecanisme que permet crear una classe a partir d'una altra indicant els canvis? Efectivament, el mecanisme de l'herència. Doncs així (de senzill) és com crearem les nostres pròpies excepcions, creant una classe que hereti de la classe Exception, la qual és pròpia de l'API de Java.

En el diagrama de classes adjunt, per representar que una classe llança una excepció hem utilitzat un tipus de relació, que no hem vist en els apunts, anomenada **dependència**. Com pots veure, es tracta d'una línia discontinua que va de la classe GameMap cap a la classe GameMapException. Per enfatitzar que es tracta d'una excepció algunes persones posen la paraula/rol/etiqueta throws sobre la línia. La relació/associació de dependència indica que una classe A (p.ex. GameMap) utilitza el servei d'una altra classe B (p.ex. GameMapException) per poder funcionar en la seva totalitat. És la classe A (p.ex. GameMap) la que coneix l'existència de la classe B (p.ex. GameMapException). Es sembla bastant a una associació binària unidireccional,

però a nivell de codificació, com que la relació de dependència indica l'ús d'un servei de la classe B per part de la classe A, es tradueix com que la classe A instància un objecte de la classe B però no el guarda en un atribut de la instància (és a dir, no guarda la referència). Això és precisament el que succeeix amb les excepcions, creem la instància de tipus Exception (o derivada d'Exception, com GameMapException) i la usem cridant a la instrucció throw, però no guardem la instància de tipus Exception (o classe derivada d'Exception) en un atribut de la instància.

Et comentem l'existència d'aquest tipus d'associació anomenada dependència perquè et donis compte de que en aquesta assignatura tractem una petita part (la més important i utilitzada) dels diagrames de classes, però que hi ha molt més. De fet, els diagrames de classes són un dels tipus de representació del llenguatge UML, però hi ha més: diagrama de casos d'ús, diagrama de seqüència, etc.

La classe GameMapException ubicada en el *package* edu.uoc.pac4.exception ha de:

* ●  Heredar de la classe de Java anomenada Exception. Moltes vegades, si el diagrama de classes és específic d'un llenguatge (en el nostre cas, Java), no s'indica que hereta de classes pròpies del llenguatge (p.ex. Exception) per evitar que el diagrama sigui de mida inmanejable. Si el diagrama de classes és independent del llenguatge de programació (com hauria de ser), llavors ja té sentit no indicar aquesta herència perquè depèn del llenguatge (en Java hi ha herència, en un altre llenguatge potser no, o heretaria d'una classe anomenada diferent).
* ●  Tenir els atributs mostrats en el diagrama de classes adjunt que contenen els missatges d'error que s'han de mostrar quan succeeixen certes anomalies i es llança una excepció d'aquest tipus.
* ●  Tenir un constructor parametritzat que rep un String. Aquest constructor només ha de cridar al constructor de la classe pare (és a dir, superclasse) la firma del qual és idèntica.

**Class PositionException**Codifica la classe PositionException seguint les mateixes instruccions que per a la

classe GameMapException. Això implica:

● La classe ha d’estar ubicada en el *package* edu.uoc.pac4.exception.

●  Aquesta classe ha d’heretar de la classe de Java Exception.

●  Definir tots els atributs tal com apareixen en el diagrama UML.

●  Implementar un constructor amb un paràmetre de tipus String que cridi al constructor de la classe pare.

**Class GameMap**

●  Defineix tots els atributs declarats en el diagrama UML d'aquesta classe amb el seu

corresponent modificador d'accés (private, public, ...).

●  **Sempre que hi hagi conflicte de noms**, sobretot en els mètodes *setter* (és a dir,

setXXX), **has d'utilitzar la paraula reservada this.**

●  **setGameMapId**: Si l'identificador rebut per paràmetre (gameMapId) és inferior o igual a 0, aquest mètode ha de llançar una excepció de tipus GameMapException amb el missatge "Game map ID cannot be negative or zero". En cas contrari, ha d'assignar l'identificador del mapa.

●  **setName**: Si el nom rebut per paràmetre (name) és null o no conté cap caràcter (una String buida), aquest mètode ha de llançar una excepció de tipus GameMapException amb el missatge "Name cannot be null or empty". En cas contrari, ha d'assignar el nom del mapa.

●  **setWidth**: Si l'amplada del mapa rebut per paràmetre (width) és inferior o igual a 0, aquest mètode ha de llançar una excepció de tipus GameMapException amb el missatge "Width cannot be negative or zero". En cas contrari, ha d'assignar l'amplada del mapa.

●  **setHeight**: Si l'altura del mapa rebuda per paràmetre (height) és inferior o igual a 0, aquest mètode ha de llançar una excepció de tipus GameMapException amb el missatge "Height cannot be negative or zero". En cas contrari, ha d'assignar l'altura del mapa.

●  **setDepth**: Si la profunditat del mapa rebuda per paràmetre (depth) és inferior o igual a 0, aquest mètode ha de llançar una excepció de tipus GameMapException amb el missatge "Depth cannot be negative or zero". En cas contrari, ha d'assignar la profunditat del mapa.

●  **setGameMapType**: Si el tipus de mapa rebut per paràmetre (gameMapType) és null, aquest mètode ha de llançar una excepció de tipus GameMapException amb el missatge "Game map type cannot be null". En cas contrari, ha d'assignar el tipus del mapa.

● **equals**: aquest mètode indica si l'objecte passat com a argument és igual (true) que l'objecte que invoca al mètode. En cas de ser diferents, retorna false.

Donada la següent instrucció:

x.equals(y);

La codificació per defecte retorna true si x == y, és a dir, si x i y apunten exactament al mateix objecte, a la mateixa adreça de memòria (és a dir, són la mateixa referència). En cas contrari, retorna false. Nosaltres volem afegir a aquest comportament alguna cosa més específica del nostre problema/context/escenari.

Així, doncs, a més del comportament descrit, s'ha de sobreescriure aquest mètode (heretat de la classe Object de Java) i **retornar true si l'identificador del mapa (gameMapId) del paràmetre rebut (obj) és el mateix que el de la classe que està invocant aquest mètode**. En cas contrari, ha de retornar false. Així, encara que dues instàncies d'aquesta mateixa classe tinguin valors diferents en tots els seus atributs a excepció de l'identificador (gameMapId), seran considerades com iguals, ja que la seva manera única d'identificar-los és mitjançant aquest atribut. Per aquells que ja hàgiu vist alguns conceptes de bases de dades, es podria dir que és la clau primària de la classe.

● **toString**: A l'igual que en l'enumeració anterior, aquest mètode ha de sobreescriure el mètode toString de classe Object. El missatge que haurà de mostrar ha de tenir el següent format:

**gameMapId** | **name** | (**width**x**height**x**depth**) | **gameMapType**Per exemple, en un cas real d’invocació a aquest mètode podria retorna la String

següent:

3 | Doyyumhwan (1536x1536x1024) | Volcano

**Pista**: fixeu-vos que el nom del tipus del mapa apareix amb la primera lletra en majúscula i la resta en minúscula. Investiga la manera d'obtenir aquesta part de la String sense haver de tornar a convertir el nom del tipus del mapa.

**Class Position**● Defineix tots els atributs declarats en el diagrama UML d'aquesta classe amb el seu

corresponent modificador d'accés (private, public, ...).

●  **Sempre que hi hagi conflicte de noms**, sobretot en els mètodes *setter* (és a dir, setXXX), **has d'utilitzar la paraula reservada this**.

●  **setGameMap**: Si la referència al mapa (això significa l'adreça de memòria a la qual apunta un objecte, tots els objectes en Java són tractats com a referències) rebut per paràmetre (gameMap) és null, aquest mètode ha de llançar una excepció del tipus PositionException amb el missatge "The map cannot be null". En cas contrari, ha d'assignar la referència del mapa al qual correspon aquesta coordenada.

●  **setX**: Si el valor de la coordenada x rebut per paràmetre (x) és inferior a 0, aquest mètode ha de llançar una excepció del tipus PositionException amb el missatge "The x coordinate must be greater than or equal to 0". A més, si aquest valor és superior o igual a l'amplada del mapa al qual pertany (width), ha de llançar una excepció de tipus PositionException amb el missatge "The x coordinate is out of bounds". En cas contrari, ha d'assignar el valor de la coordenada x.

●  **setY**: Si el valor de la coordenada y rebut per paràmetre (y) és inferior a 0, aquest mètode ha de llançar una excepció del tipus PositionException amb el missatge "The y coordinate must be greater than or equal to 0". A més, si aquest valor és superior o igual a l'altura del mapa al qual pertany (height), ha de llançar una excepció de tipus PositionException amb el missatge "The y coordinate is out of bounds". En cas contrari, ha d'assignar el valor de la coordenada y.

●  **setZ**: Si el valor de la coordenada z rebut per paràmetre (z) és inferior a 0, aquest mètode ha de llançar una excepció del tipus PositionException amb el missatge "The z coordinate must be greater than or equal to 0". A més, si aquest valor és superior o igual a la profunditat del mapa al qual pertany (depth), ha de llançar una excepció de tipus PositionException amb el missatge "The z coordinate is out of bounds". En cas contrari, ha d'assignar el valor de la coordenada z.

●  **euclideanDistance**: Ha de retornar la distància euclidiana entre la posició del mapa de la instància que invoca aquest mètode i la posició rebuda per paràmetre (position). És a dir, si aquest mètode s'invoquen de la següent manera:

coord1.euclideanDistance(coord2);

Aquest mètode ha de retornar la distància euclidiana entre coord1 i coord2. Recordeu que **una posició d'un mapa té 3 dimensions (x, y, z)**, que s'hauran de tenir en compte per calcular la distància euclidiana en l'espai.

● **equals**: A l'igual que es va fer amb la classe GameMap, s'ha de sobreescriure aquest mètode de la classe Object, quan el paràmetre rebut (obj) sigui exactament el mateix quant a direcció de memòria o quan el mapa al qual pertany (gameMap) i les seves coordenades (x, y, z) siguin les mateixes. És a dir, dues instàncies que apunten a dues direccions de memòria diferents poden ser considerades com iguals si compleixen la condició anterior. Per això, en cas de ser iguals, el mètode ha de retornar true. En cas contrari, false.

**Pista**: recordeu que un mapa es considera igual a un altre quan l'identificador (gameMapId) és el mateix.

● **toString**:Haderetornarunarepresentacióperescritdelaposicióquerepresenta. Com que una posició depèn del mapa, serà necessari mostrar la informació textual del mapa. Per això, el text que s'espera que retorni aquest mètode és el següent:

**gameMapId** | **name** | (**width**x**height**x**depth**) | **gameMapType** | Position (**x**, **y**, **z**) Per exemple, una posició podria retornar una String com la següent:

3 | Doyyumhwan (1536x1536x1024) | Volcano | Position (548, 1022, 29)

**Requisito mínimo para evaluar este ejercicio:** el programa debe pasar los tests de GameMapExceptionTest, PositionException, GameMapTypeTest,GameMapTest yPositionTest.

**Nota:** el estudiante puede recibir una penalización de hasta **0.5 pts.** de la nota obtenida en este ejercicio en función de la calidad del código proporcionado.

***(2.5 pts.: 0.25 pts. GameMapExceptionTest, 0.25 pts. PositionExceptionTest, 0.33 pts. GameMapTypeTest, 0.33 pts. GameMapTest 0.34 pts. PositionTest, 0.33 pts. GameMapTypeToStringTest, 0.33 pts. GameMapEqualsToStringTest i 0.34 pts. PositionEqualsToStringTest)***

**Exercici 2 (5 punts)**

Abans de començar has de:

Mirar el vídeo anomenat “Herència” que trobaràs al *site* UOCoders.

**Obre el projecte PAC4Ex2 d'IntelliJ**. Al *package* edu.uoc.pac4 del directori /src/test/java està l'arxiu amb els tests unitaris que et proporcionem. Copia el paquet edu.uoc.pac4 del directori /src/main/java de l'Exercici 1 d'aquesta PAC4 una vegada l'hagis completat.

En aquest exercici ampliem el programa anterior afegint noves classes, interfícies i mètodes. El nou diagrama de classes UML el trobaràs en el zip de l'enunciat amb el nom PAC4Ex2.png. És important que el consultis sovint ja que hi ha informació, com els modificadors d'accés dels elements, que és possible que no es comentin en aquest enunciat.

A continuació anem a comentar cadascun dels elements que apareixen en el diagrama de classes anterior:

**Interface Movable**

Aquesta interfície declara la signatura dels mètodes que ha de tenir qualsevol classe els

objectes de la qual siguin capaços de moure's.

**Class EntityException**

Aquesta classe ha de definir tots els atributs i constructors que apareixen en el diagrama UML i, a més, heretar de la classe Exception igual que en l'exercici anterior. Ha d'estar dins del package edu.uoc.pac4.exception.

**Class Entity**

**Aquesta classe és abstracta**, és a dir, **no es pot instanciar** i hauran de ser les seves classes filles les que s'instanciïn. Per això, cal declarar-la com abstract. A més, representa el concepte d'entitat que pot haver en un mapa, ja sigui un personatge jugable o un enemic.

Per a la seva implementació, cal seguir la definició que s'ha proporcionat en el diagrama UML. A més, cal tenir en compte les següents consideracions:

●  La classe té un atribut anomenat vid, que significa *virtual id* en anglès. Aquest ens permet identificar de manera única a cada entitat del mapa i s'ha d'assignar automàticament en el constructor de la classe. A més, no pot haver-hi salts entre els vid assignats. Per tant, en cas que durant l'execució del constructor es produeixi alguna excepció, no s'haurà d'assignar ni incrementar el vid.

●  La classe Entity ha d'implementar la interfície Movable, però no haurà d'implementar el mètode move. Hauran de ser les classes filles les que s'encarreguin d'això.

●  **setName**: Si el nom rebut per paràmetre (name) és null o no conté cap caràcter (String buida), el mètode ha de llançar una excepció de tipus EntityException amb el missatge "Name cannot be null or empty". En cas contrari, ha d'assignar el nom de l'entitat.

●  **setLevel**: Si el nivell rebut per paràmetre (level) és inferior a 1 o superior al màxim permès (MAX\_LEVEL), el mètode ha de llançar una excepció de tipus EntityException amb el missatge "Level must be between 1 and 99". En cas contrari, ha d'assignar el nivell de l'entitat.

●  **setPosition**: Si la posició rebuda per paràmetre (position) és null, el mètode ha de llançar una excepció de tipus EntityException amb el missatge "Position cannot be null". En cas contrari, ha d'assignar la posició de l'entitat.

●  **setMaxHP**: Si el valor màxim de punts de vida rebut per paràmetre (maxHP) és inferior a 1, s'assignarà un 1 com a valor. En cas contrari, s'assignarà el valor rebut per paràmetre. És a dir, el valor mínim que pot adoptar aquest paràmetre és 1.

●  **setCurrentHP**: Si el valor actual de punts de vida rebut per paràmetre (currentHP) és inferior a 0, s'assignarà 0 com a valor i, en cas que el valor rebut sigui superior al màxim de punts de vida del personatge (maxHP), s'assignarà aquest últim com a valor actual de punts de vida. En qualsevol altre cas, s'assignarà el valor rebut per paràmetre. És a dir, el valor d'aquest atribut ha d'estar dins del rang [0, maxHP].

**Class GameMap**

Aquesta classe ha sofert una lleugera modificació respecte a l'exercici anterior, ja que s'han afegit 3 mètodes nous i un nou atribut. Fins ara, sempre s'ha utilitzat un array de Java per emmagatzemar múltiples elements en un únic atribut, però en aquest cas, s'utilitzarà una altra estructura una mica més complexa: **una taula de dispersió (Hashtable)**.

Aquesta estructura pertany a la interfície Map de Java i **permet emmagatzemar elements a partir d'una estructura coneguda com a clau-valor (Key-Value)**. La clau és la manera única de poder identificar a l'element emmagatzemat (i sol ser algun atribut de la classe que sigui únic per a cada objecte d'aquesta classe) i, com és d'esperar, el valor és l'element emmagatzemat. D'aquesta manera i gràcies a l'ús d'una funció anomenada hash, es pot aconseguir una complexitat constant tant en la lectura com en l'escriptura de dades.

En aquest exercici només aprendrem a usar aquesta estructura, però podeu investigar més sobre ella en la documentació oficial de Java: https://docs.oracle.com/javase/8/docs/api/java/util/Hashtable.html.

Per això, hauràs de declarar l'atribut entities com un Map<Integer, Entity> (on Integer és el tipus de les claus i Entity és el tipus dels valors) i, en la seva assignació, crear una instància de tipus Hashtable. A més, fixa't que en aquesta classe s'ha definit el vid com un Integer en comptes de com un int. Aquest tipus d'implementació és degut a que la clau del Hashtable ha de ser una classe i no un tipus de dada primitiu de Java, com ho és int. Per aquesta raó, s'utilitzarà Integer, que representa un valor enter igual que int, però actuant com un objecte d'una classe.

Finalment, per a la implementació dels tres nous mètodes, **hauràs d'utilitzar mètodes de la classe Hashtable per afegir, eliminar i consultar elements.** Per això, et recomanem visitar l'enllaç anterior de la documentació oficial sobre la classe Hashtable. A més, hauràs de seguir en tot moment la definició d'aquests mètodes en el UML.

**Interface Speakable**Aquesta interfície declara la signatura dels mètodes que ha de tenir qualsevol classe els

objectes de la qual siguin capaços de parlar.

**Enum Empire**

● Els valors literals de l'enumeració són: SHINSOO, CHUNJO i JINNO. Cadascun

d'aquests literals representa un regne del continent.

**Class PlayerException**

Aquesta classe ha de definir tots els atributs i constructors que apareixen en el diagrama UML i, a més, heretar de la classe Exception igual que en l'exercici anterior. Ha d'estar dins del *package* edu.uoc.pac4.exception.

**Class Player**

Aquesta classe implementa el comportament d'un personatge jugable del joc. Al igual que passava amb la classe Entity, aquesta classe torna a ser abstracta. A més, hauràs de tenir en compte les següents consideracions durant la seva implementació:

●  Aquesta classe ha d'heretar la classe Entity i implementar la interfície Speakable. En aquest cas, només haurà d'implementar la funcionalitat del mètode sayName, delegant en les classes filles la implementació dels altres mètodes de la interfície Speakable.

●  **setCurrentGold** y **setCurrentExperience**: Ambdós mètodes han d'assignar els corresponents atributs, però, en cas que el paràmetre rebut (currentGold o currentExperience, respectivament) sigui inferior a 0, el valor assignat serà un 0. És a dir, els atributs als quals es fa referència mai podran tenir un valor negatiu.

●  **setVitality**, **setIntelligence**, **setStrength** y **setAgility**: Tots aquests mètodes permeten assignar els valors dels atributs d'un personatge jugable. En cas que el paràmetre rebut en qualsevol dels mètodes (vitality, intelligence, strength o agility, respectivament) sigui inferior a 0 o superior al valor màxim que poden tenir els atributs (MAX\_STAT), es deurà acotar el valor a aquest rang. És a dir, aquests atributs sempre hauran de tenir un valor contingut dins del rang [0, MAX\_STAT].

●  **setEmpire**: En cas que el regne rebut per paràmetre (empire) sigui null, el mètode ha de llançar una excepció de tipus PlayerException amb el missatge "Empire cannot be null". En cas contrari, ha d'assignar el regne rebut.

●  **sayName**: Aquest mètode de la interfície Speakable ha de retornar el nom de l'entitat a la qual fa referència seguit de dos punts i un espai. És a dir, ha de seguir el següent format:

"name: "  
On name és elnomdel’entitat(i.e.valordel’atributnamedelaclasseEntity).

**Class Warrior, Ninja, DarkMage i Shaman**

Aquestes classes són les classes instanciables de tots els personatges jugables del joc. És a dir, totes han d'heretar de la classe Player i, pel fet de ser instanciables, ja no han de ser abstractes.

Cadascuna d'elles conté els valors base de cada classe, a més, de les implementacions dels mètodes que hereta. Concretament, ha d'implementar el mètode move de la interfície Movable i els dos mètodes restants de la interfície Speakable.

Per tant, aquestes classes permeten generar especialitzacions de personatges dins del joc, amb els seus atributs de raça i les seves pròpies frases abans d'entrar en batalla o just abans de morir.

Per a la implementació de totes elles has de seguir les definicions de l'UML i tenir en compte les següents consideracions:

● **move**: Un personatge no pot moure's, en cada moviment que faci, més enllà de la distància màxima (MAX\_STEP). És a dir, només podrà moure's si, des de la seva posició actual fins a la posició a la qual vol desplaçar-se (la que es rep per paràmetre) la distància entre elles és inferior a la distància màxima (MAX\_STEP). En cas que pugui moure's, el mètode ha d'actualitzar la posició del personatge amb la posició rebuda per paràmetre (position) i retornar true, en cas contrari no ha d'actualitzar la posició i ha de retornar false.

● **battleCry**: Aquest mètode ha de retornar una String amb el nom del personatge seguit de dos punts i un espai i una frase segons la raça del personatge que correspon a un crit de guerra abans de la batalla. Seguidament, es detallen les frases segons la raça:

○ **Warrior**:"Tovictoryordeath!"  
○ **Ninja**:"Strikeswift,strikesilent!" ○ **DarkMage**:"Letthedarknessreign!"  
○ **Shaman**:"Spiritsguideustovictory!"

Així, un exemple d’aquesta crida d’aquest mètode per part d’un guerrer amb el nom Invincible, seria:

Invincible: To victory or death!  
● **deathCry**: Aquest mètode ha de retornar una String amb el nom del personatge

seguit de dos punts i un espai i una frase segons la raça del personatge que correspon al crit just abans de morir en batalla. Seguidament, es detallen les frases segons la raça:

○ **Warrior**:"Ishallreturn,strongerthanever..." ○ **Ninja**:"Silence,atlast..."  
○ **DarkMage**:"Thedarkartswillliveon..."  
○ **Shaman**:"Mysouljoinstheancestors..."

Així, un exemple d’aquesta crida d’aquest mètode per part d’un ninja amb el nom Ambush, seria:

Ambush: Silence, at least...

**Interface TransCloneable**Aquesta interfície declara la signatura dels mètodes que ha de tenir qualsevol classe els

objectes dels quals siguin capaços de transformar-se en un altre enemic.

**Class Enemy**

Aquesta classe modela el comportament bàsic dels enemics del joc. Cada tipus d'enemic s'identifica mitjançant un identificador id comú a tots els enemics del mateix tipus. Per exemple, tots els orcs tenen el mateix id. Encara que existeixen subclasses que s'especialitzen en tipus específics d'enemics, l'atribut id resulta útil en situacions on el sistema necessita realitzar cerques en la memòria, particularment quan només es disposa de l'identificador i no del tipus de classe directament. Això facilita la identificació i manipulació de les dades dels enemics durant l'execució del joc, especialment quan les consultes a la memòria es basen en identificadors provinents de la base de dades.

A més, la classe conté els atributs necessaris per poder determinar la recompensa que s'obté en acabar amb ell (minGold, maxGold i experience), el dany que infringeix (minDamage i maxDamage) i el líder del seu grup (groupLeader). Aquest últim atribut és especialment interessant ja que tots els enemics del joc apareixen a través d'un grup. És a dir, quan apareixen en el mapa, ho fan en grup. Dins d'aquest grup, un d'ells és el líder i, perquè la resta d'enemics no se separen del líder, tenen una referència al líder del grup per poder calcular la distància que tenen a ell i acostar-se en cas que fos necessari.

Per tant, les consideracions que s'han de tenir en compte a l'hora d'implementar aquesta classe són:

● Aquesta classe no és instanciable i, a més, hereta la classe Entity.

● **setId**: Aquest mètode assigna l'identificador rebut per paràmetre (id) sense cap tipus de validació, ja que és un valor que resideix en la base de dades i ja ha estat validat prèviament.

●  **setGold**: Aquest mètode assigna els valors als atributs minGold i maxGold alhora. D'una banda, si el valor d'or mínim rebut per paràmetre (minGold) és inferior a 0, el mètode ha d'assignar 0 a l'atribut minGold. En cas contrari, assignarà el valor rebut per paràmetre. És a dir, l'atribut minGold mai podrà tenir un valor negatiu. D'altra banda, si el valor d'or màxim rebut per paràmetre (maxGold) és inferior al valor d'or mínim (minGold), el mètode ha d'assignar com a valor d'or màxim el valor que tingui el paràmetre d'or mínim (minGold). És a dir, maxGold mai podrà tenir un valor inferior al que té minGold. Formalment:

0 ≤ minGold ≤ maxGold

●  **setExperience**: Si el valor rebut per paràmetre (experience) és inferior a 0, el mètode ha d'assignar 0 a l'atribut experience. En cas contrari, ha d'assignar el valor rebut per paràmetre. És a dir, experience mai podrà tenir un valor inferior a 0.

●  **setDamage**: Aquest mètode assigna els valors als atributs minDamage i maxDamage alhora seguint la mateixa estratègia que el mètode setGold. D'una banda, si el valor de dany mínim rebut per paràmetre (minDamage) és inferior a 0, el mètode ha d'assignar 0 a l'atribut minDamage. En cas contrari, assignarà el valor rebut per paràmetre. És a dir, l'atribut minDamage mai podrà tenir un valor negatiu. D'altra banda, si el valor de dany màxim rebut per paràmetre (maxDamage) és inferior al valor de dany mínim (minDamage), el mètode ha d'assignar com a valor de dany màxim el valor que tingui el paràmetre de dany mínim (minDamage). És a dir, maxDamage mai podrà tenir un valor inferior al que té minDamage. Formalment:

0 ≤ minDamage ≤ maxDamage

●  **setGroupLeader**:Had'assignarlareferènciaallíderdelgrup.Encasquel'enemic que s'estigui creant sigui el líder del grup, el valor de l'atribut groupLeader ha de ser null. És a dir, no s'ha d'aplicar cap tipus de validació per a aquest atribut, ja que pot adoptar un valor null o una referència a un altre enemic prèviament ja creat.

●  **isFarFromGroupLeader**: Ha de retornar true si l'enemic es troba lluny del líder del seu grup. En cas contrari, false. Si l'enemic que invoca aquest mètode és un líder d'un grup, el mètode ha de retornar false. En cas contrari, s'haurà de calcular la distància d'aquest enemic al líder del grup mitjançant les seves posicions. Es considera que un enemic està lluny del líder del grup si la seva distància amb ell és superior a 50.0 (MAX\_GROUP\_LEADER\_DISTANCE).

**Class OrcReborn**

Aquesta classe implementa el comportament d'un OrcReborn, un tipus d'enemic la classe del qual sí que és instanciable i hereta d'Enemy. Per tant, no ha de ser abstracta i, a més, haurà d'implementar aquells mètodes pendents de les interfícies implementades per les seves classes pare.

Com tota classe que hereta d'una altra, necessita tenir una crida al constructor de la classe pare dins del seu propi constructor.

Per implementar la classe heu de seguir les següents especificacions.

* ●  La classe ha d'incloure tots els atributs que es defineixen en el diagrama UML proporcionat.
* ●  **move**: Un OrcReborn no pot moure's més que la distància màxima (MAX\_STEP). És a dir, només podrà moure's si, des de la seva posició actual fins a la posició a la qual vol desplaçar-se (la que es rep per paràmetre) la distància entre elles és inferior a la distància màxima (MAX\_STEP). En cas que pugui moure's, el mètode ha d'actualitzar la posició de l'enemic per la posició rebuda per paràmetre (position) i retornar true, en cas contrari no ha d'actualitzar la seva posició i ha de retornar false.

**Class Orc**

Aquesta classe implementa el comportament d'un Orc, un tipus d'enemic que té la mateixa classe pare que el tipus OrcReborn. A més, segueix les mateixes regles especificades a la classe anterior.

Per acabar d'implementar la classe heu de seguir les següents especificacions:

* ●  La classe ha d'implementar les interfícies TransCloneable i Cloneable.
* ●  La classe ha d'incloure tots els atributs que es defineixen en el diagrama UML

proporcionat.

* ●  **move**: Un Orc no pot moure's més que la distància màxima (MAX\_STEP). És a dir, només podrà moure's si, des de la seva posició actual fins a la posició a la qual vol desplaçar-se (la que es rep per paràmetre) la distància entre elles és inferior a la distància màxima (MAX\_STEP). En cas que pugui moure's, el mètode ha d'actualitzar la posició de l'enemic per la posició rebuda per paràmetre (position) i retornar true, en cas contrari no ha d'actualitzar la seva posició i ha de retornar false.
* ● **clone**: Aquest mètode ha de clonar la instància que l'invoqui utilitzant el mètode clone de la classe Object. Ara bé, tingueu en compte que aquest mètode fa una còpia de tots els atributs que té la classe. És a dir, per atributs com la posició de l'entitat (position) el mètode faria una còpia de la referència a aquesta posició, per la qual cosa l'enemic clonat compartiria una mateixa instància de posició amb l'enemic que executa aquest mètode.
* Per poder evitar això, és important que es cree una nova posició a partir de les dades que té aquesta posició. D'aquesta manera, ens assegurem de que la posició de l'enemic clonat ja no apuntarà a la mateixa que la de l'altre enemic i, en cas de modificar una, no s'altera l'altra.
* Fixeu-vos que el mateix ocorreria amb la referència al mapa al qual pertany aquesta posició. És a dir, ocorre amb tots els atributs que té incloent aquells que són d'altres classes. En aquest cas en concret, ens interessa mantenir aquesta referència sense duplicar-la, ja que el mapa continuarà sent el mateix per a tots els enemics per molt que aquests es clonin.
* Per tant, en aquest mètode, a més de clonar la instància que el crida, només hauràs de crear una nova instància per a la posició per evitar els problemes esmentats anteriorment.
* ● **transClone**: Una instància d'Orc té la capacitat d'invocar enemics al seu voltant a través d'una habilitat. Concretament, invoca a un OrcReborn a la mateixa posició on es troba. Per això, aquest mètode ha de clonar l'enemic que crida a aquest mètode (seguint les especificacions que s'han descrit al mètode clone) per poder crear un nou Orc. Un cop s'hagi clonat, ja es podrà crear la nova instància de la classe OrcReborn que s'ha de retornar. Fixeu-vos que, en aquest cas, la posició tindrà els mateixos valors, però no tindrà la mateixa referència.
* Finalment, ha d'actualitzar la referència al líder del grup. Si bé és cert que amb el mètode clone s'hauria d'haver clonat la referència al líder, existeix un cas on pot crear conflicte: quan es clona un líder de grup. Al clonar un líder de grup, aquesta referència apuntaria a null, provocant que aquest enemic clonat fos també un líder de grup. Per tant, en cas que la referència al líder del grup sigui null, s'ha d'actualitzar a la referència de la instància que invoca aquest mètode (this).

–