PRÀCTICA 3
Programant orientat a objectes

Disseny i programació orientada a objectes

Semestre 20192

Estudis d'Informàtica, Multimèdia i Telecomunicació

Presentació

Aquesta Pràctica aprofundeix en els conceptes de classe, objecte i relacions entre objectes i classes (herència i implementació) del paradigma de programació orientada a objectes (POO). A més posa l'accent en el disseny de software mitjançant diagrames de classes UML.

Objectius

Els objectius que es desitgen aconseguir amb aquesta Pràctica són:

- Practicar els conceptes estudiats a l'assignatura.
- Entendre les especificacions d'un problema donat i descriure una solució mitjançant un diagrama de classes UML.
- Entendre i posar en pràctica la relació de generalització/especialització (o herència) entre classes.
- Comprendre i utilitzar interfícies.
- Usar el programari Dia per dibuixar diagrames de classes.

Enunciat

Aquesta Pràctica està formada per 3 exercicis.

Abans de començar, és recomanable llegir el mòdul <<*Herència (relació entre objectes)*>> dels apunts.

Per fer l'últim apartat de l'Exercici 3 hauràs d'usar Dia, un programari que permet crear diagrames de classes UML, entre altres tipus de diagrames. Dia és un programari de suport i, òbviament, té limitacions en crear els diagrames de classes. Per això, cal saber el que s'està fent a cada moment. Dia és aliè a Eclipse. Per tant, primer has d'instal·lar-ho i configurar-ho en el teu ordinador. Per a això, vés-hi a la següent pàgina web segons sigui el teu sistema operatiu:

- Windows: http://dia-installer.de/download/index.html.
- Linux: http://dia-installer.de/download/linux.html.
- MacOS: http://dia-installer.de/download/macosx.html.

Exercici 1 – Ixcepciones personalitzades (1 punt)

En les classes `Item` i `Tank` hem utilitzat la classe pròpia de l'API de Java anomenada `Exception` per llançar excepcions quan es produïa algun cas anòmal. Això és correcte, però quan en un programa es donen excepcions que són particulars del problema/context que tractem, és adequat crear les nostres pròpies excepcions o excepcions personalitzades (en anglès, *custom exceptions* o *user-defined exceptions*) que siguin adequades per a les necessitats del nostre programa.

No farem les nostres pròpies excepcions si les que porta Java per defecte ja ens van bé i són bastant significatives.

En el nostre cas farem excepcions per controlar les anomalies que es puguin donar amb els ítems. Evidentment, com et pots imaginar, Java no porta excepcions que representin anomalies que es donen amb els ítems d'un aquari. Les excepcions de Java són més generalistes.

Arribats a aquest punt, 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 d'herència. Doncs així (de senzill) és com crearem les nostres pròpies excepcions, creant una classe que hereti de la classe `Exception`. Per tant, en aquest exercici codificaràs la classe `ItemException` que heretarà d'`Exception`. Se't demana seguir els següents passos per codificar la classe `ItemException`:

1. Amb Eclipse, obre el projecte `PRAC3_ex1` que et donem.
2. Una vegada obert el projecte, crea una classe nova anomenada `ItemException` (fitxer `ItemException.java`) que hereti de la classe `Exception`.
3. Dins de la classe `ItemException` codifica el seu constructor per defecte. En aquest cas estàs heretant d'`Exception` i, per tant, has de cridar al constructor per defecte de la classe pare, és a dir, al constructor per defecte de la superclasse `Exception`.
4. Trasllada els atributs de tipus de `String` que contenen els missatges d'excepció del fitxer `Item.java` al fitxer `ItemException.java`, ja que a partir d'ara pertanyeran a aquesta classe (`ItemException`).

> **Nota**: Recorda, dels apunts del mòdul <<Herència (relació entre classes)>> que els constructors de les classes filles (o també anomenades *derivades* o *subclasses*) han de cridar d'una manera especial al constructor de la classe pare (també anomenada *superclase* o *base*).

(0.25 punts)

5. Ara implementa un constructor públic amb un argument anomenat `msg` de tipus `String`. Aquest constructor ha de cridar al constructor amb un argument de tipus `String` que té la classe pare `Exception`.

(0.25 punts)

6. Ara modifica la classe `Item` que se't dóna dins del projecte `PRAC3_ex1` perquè en comptes d'usar la classe `Exception` de Java per llançar les excepcions utilitzi la nova classe `ItemException`. Dit d'una altra manera, on en `Item.java` posa `Exception`, ara has de posar `ItemException`. La classe `Item` que et proporcionem és fruit d'haver fet la Pràctica anterior.

(0.25 punts)

7. Has de fer exactament el mateix que has fet amb `ItemException` per crear una altra excepció anomenada `TankException`. Els missatges d'error que apareixen en `Tank` ara han d'estar centralitzats en `TankException`.

(0.25 punts)

Si has realitzat el pas 6, ara el `catch` que hi ha en el mètode `main` del fitxer `Check.java` hauria de capturar excepcions `ItemException`, en comptes de tipus `Exception`. Si posessis `Exception` (com abans del pas 6), en lloc d'`ItemException`, també et funcionaria, doncs `ItemException` hereta d'`Exception` i `Exception` captura totes les excepcions que heretin d'ella. Per això és important que si posem diversos `catch`, l'últim capturi `Exception`, doncs en cas contrari una excepció pròpia mai seria capturada pel seu `catch`, en ser l'excepció pròpia que hem creat filla d'`Exception`.

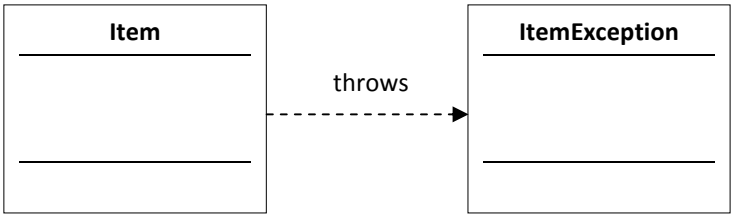
Per saber si has fet bé aquest exercici, has de veure que quan es llança una excepció, la instrucció `e.printStackTrace()` indiqui per pantalla (i.e. Console) que el tipus d'excepció és `ItemException` en

comptes de `java.lang.Exception`. Crea situacions que llanci excepcions, per exemple, posar l'altura d'un ítem igual a `0`.

Si vols saber més sobre les excepcions, llegeix els següents recursos:

- Apartat “Excepcions” de la guia preliminar de Java.
- http://www.javatpoint.com/exception-propagation
- https://examples.javacodegeeks.com/java-basics/exceptions/java-custom-exception-example/
- https://docs.oracle.com/javase/8/docs/api/java/lang/exception.html

Potser t'estàs preguntant com es representa la relació entre la classe `Item` i `ItemException` en un diagrama de classes UML. Doncs bé, per representar que una classe llança una excepció, habitualment, s'utilitza un tipus de relació que no hem vist en els apunts anomenada *dependència*. La *dependència* es dibuixa de la següent manera:



Es tracta d'una línia discontínua que va de la classe `Item` cap a la classe `ItemException`. Per emfatitzar 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. `Item`) utilitza el servei d'una altra classe B (p.ex. `ItemException`) per poder funcionar íntegrament. És la classe A (p.ex. `Item`) la que coneix l'existència de la classe B (p.ex. `ItemException`). S'assembla bastant a una associació binària unidireccional, però a nivell de codificació, com 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 instancia la classe B però no guarda la instància en un atribut de la classe. Això és precisament el que succeeix amb les excepcions, creem la instància de tipus `Exception` (o derivada d'`Exception`, com `ItemException`) 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 classe que la instància.

Et comentem l'existència d'aquest tipus d'associació anomenada *dependència* perquè t'adonis 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.

Exercici 2 – Especialitzant els ítems (3 punts)

Nota: Per fer aquest exercici és necessari haver completat l'Exercici 1. Una vegada completat, has de crear el projecte `PRAC3_ex2` i copiar tots els fitxers de l'Exercici 1, excepte `Check.java` (no és necessari).

Com ja saps, en el nostre tanc existeixen ítems, però el concepte d'ítem és massa *abstracte*. Per tant, anirem matisant els diferents tipus d'ítems que ens trobem en el nostre tanc. Aquests ítems poden ser animals (`Animal`), menjar (`Food`), algues (`Kelp`) i, fins i tot, alguna joguina aquàtica (`SubmarineToy`). Així doncs, estem davant un clar cas d'herència o generalització/especialització. La classe pare (superclasse) en el nostre projecte és l'`Item`. A continuació anirem explicant cadascuna de les noves classes que han sorgit i com es relacionen entre si. També hi haurà classes que ja hem anat utilitzant en les pràctiques anteriors que patiran lleugers canvis.

Item

Tots els ítems que es troben en un aquari tenen els atributs `xCoord`, `yCoord`, `length`, `height`, `spriteImage` i un identificador (`id`). A partir d'ara els ítems no tindran un atribut `status` de tipus `ItemStatus` perquè existeixen classes filles en les quals no té sentit aquest status. Per exemple, una joguina aquàtica no pot estar malalta o morta. No obstant això, sorgiran altres tipus enumerats més específics que es relacionaran amb les classes que corresponguin.

Per la mateixa raó que `status`, els ítems, en general, tampoc tindran l'atribut `energy`, encara que sí ho tindran alguns ítems que concretarem mitjançant subclasses.

Els mètodes que comparteixen tots els ítems són els setters/getters dels atributs que es mantenen de les pràctiques anteriors. El mètode `toString` ha patit una variació donat que els atributs corresponents a l'`status` i l'energia (`energy`) ja no pertanyen a la classe `Item` en aquesta nova versió. Així doncs, en el mètode `toString` es mostrarà la longitud (`length`) i amplària (`height`), en lloc de l'`energy` i l'`status`, respectivament.

Finalment, dir que els ítems continuen guardant una referència a l'aquari al que pertanyen (`tank`). La nova signatura del constructor de la classe `Item` ha de ser:

```
protected Item(double xCoord, double yCoord, String spriteImage, double length, double height, Tank tank) throws ItemException
```

Tingues en compte que:

a) Cada atribut té els seus *getter* i *setter* públics, excepte `setId` que és `privat` i el constructor que ara és `protected`.

b) Modifica el mètode `toString()` de manera que retorni un `String` amb la informació de `length` i `height`, en comptes d'`energy` i `status`, respectivament.

c) A partir d'ara no instanciarem cap objecte de la classe `Item`.

Animal

Els animals són una especialització d'`Item` en la qual cadascun tindrà els següents atributs:

- Gènere (`gender`) el valor de la qual només podrà ser FEMALE o MALE.
- Edat (`age`), que serà de tipus enter.
- Un booleà anomenat `facingRight` que indica si l'animal està orientat cap a la dreta o cap a l'esquerra.
- La velocitat (`speed`) de tipus `double` a la qual es desplaça.
- La quantitat de menjar que requereix cada vegada que menja (`requiredFoodQuantity`). Serà de tipus `double`.
- Un valor anomenat `thresholdReverse` que determinarà la probabilitat que l'animal canviï d'orientació mentre es desplaça. Aquest valor serà un `double`.
- La energia (`energy`) que l'animal té al moment present. Serà un enter amb valor inicial igual a `100` (perquè funcioni correctament la lògica del mètode `setEnergy`).
- Finalment, els animals tindran l'estat de l'animal (`status`). Els seus valors seran els mateixos que els de l'antic `ItemStatus`, ja que definirà l'estat de salut de l'animal. Ara l'`enum` en comptes d'anomenar-se `ItemStatus` es dirà `AnimalStatus`.
- S'han de construir tots els *setters* i *getters* per als nous atributs d'aquesta classe, que seran públics. En el cas dels *setters* i

getters dels atributs `energy` i `status`, aquests es comporten com havíem definit fins ara a la classe `Item`. És a dir, has de traslladar el codi de la classe `Item` cap a la classe `Animal`. De la mateixa manera, els missatges relatius a les excepcions provocades per aquests mètodes has de situar-los a la classe `AnimalException`.

Respecte als nous *setters* deus tenir en compte les següents característiques:

- L'edat d'un animal no pot ser negativa, en aquest cas s'haurà de disparar una excepció de tipus `AnimalException` (aquesta classe l'expliquem a continuació en aquest enunciat) amb el següent missatge: `"[ERROR] Animals' age cannot be a negative value!!"`. Aquest missatge haurà d'estar a la classe `AnimalException` en un atribut anomenat `MSG_ERR_AGE_VALUE`.
- El llindar per girar-se ha d'estar comprès entre els valors `0` i `1` (tots dos inclosos). En cas d'assignar un valor invàlid, s'ha de disparar una excepció de tipus `ItemException` amb el següent missatge: `"[ERROR] Threshold reverse cannot be negative either greater than 1!!"`. Aquest missatge ha d'estar a la classe `ItemException` en un atribut anomenat `MSG_ERR_THRESHOLD_VALUE`.
- La velocitat ha de ser major que `0`, en cas contrari es dispararà una excepción de tipus `ItemException` amb el següent missatge: `"[ERROR] Speed cannot be 0 or negative!!"`. Aquest missatge ha d'estar a la classe `ItemException` en un atribut anomenat `MSG_ERR_SPEED_VALUE`.
- El mètode *getter* de `facingRight` en retornar un valor booleà, es dirà `isFacingRight`.

D'altra banda, existeix un mètode anomenat `reverse` que únicament fa que l'atribut `facingRight` prengui el valor contrari al que té al moment de ser cridat.

Per la seva banda, els mètodes `getStatus`, `setStatus`, `getEnergy`, `setEnergy`, `incEnergy`, `decEnergy` i `isDead` han de situar-se ara

a la classe filla (`Animal`), respectant els modificadors d'accés i la seva lògica (només cal adaptar `ItemException` per `AnimaException`).

Finalment, aquesta classe tindrà un sol constructor i serà amb paràmetres. L'ordre dels paràmetres ha de ser: `xCoord`, `yCoord`, `spriteImage`, `length`, `height`, `gender`, `age`, `speed`, `requiredFoodQuantity`, `thresholdReverse`, `status`, `energy` i `tank`. Quan un animal es crea, aquest sempre està orientat cap a la dreta.

AnimalException

L'excepció `AnimalExcepción` no és una especialització d'`Exception` sinó que és una especialització d'`ItemException`. Per tant haurà d'heretar d'`ItemException`. Has de tenir en compte els missatges d'error concrets relatius a un `Animal`. Per tant, tots aquells missatges d'error concrets de la classe `Animal` hauran de ser traslladats a la classe `AnimalExcepcion` i eliminats de la classe `ItemException`. En els missatges, lògicament, on posa “item”, ha de posar “animal”.

Food

La classe `Food` (menjar) és una especialització d'`Item` que té dos atributs propis:

- La velocitat per la qual es mou en l'aquari (`speed`), que serà de tipus `double`. Aquest atribut tindrà els seus respectius *getter* i *setter* que es comportaran de la mateixa manera que l'especificat a la classe `Animal` (llegeix, de nou, les seves descripcions) A més, quan es creï qualsevol objecte d'aquesta classe, el seu atribut `speed` s'inicialitzarà amb el valor `1`.
- Un `boolean` que indiqui si ha estat menjat (`eaten`) per algú. Per a aquest atribut existeixen els següents mètodes: (1) `isEaten` que ens permet consultar el valor de l'atribut; (2) un mètode `eaten` que assigna el valor `true` a l'atribut. No existeix un *setter* a l'ús.
- L'energia (`energy`) que proporciona a l'animal que la hi menja. El valor d'aquest atribut serà un `int` entre `1` i `10`, tots dos inclosos. Aquest atribut tindrà els seus mètodes *getter* i *setter* corresponents. El *setter* serà privat. En cas de llançar-se una excepció, aquesta serà de tipus `Exception` i el seu missatge

serà: `"[ERROR] Food cannot be less than 1 either greater than 10!!"`.

Finalment, dir que aquesta classe només té un constructor i serà amb paràmetres. L'ordre d'aquests serà: `xCoord`, `yCoord`, `length`, `height`, `energy` i `tank`. El valor de `spriteImage` serà sempre el mateix: `"./images/food/seed.png"` (ara com ara no pensis perquè és aquest el valor, ho entendràs en la propera pràctica).

SubmarineToy

La classe `SubmarineToy` ha de tenir tres atributs propis:

- La velocitat per la qual es mou en l'aquari (`speed`) que serà del tipus `double`. El valor per defecte d'aquest atribut és `1`.
- Un `boolean` anomenat `facingRight` el valor del qual per defecte serà `true`.
- Un `double` entre `0` i `1` (tots dos inclosos) anomenat `thresholdReverse`. El valor per defecte és `0.0003`.

Els tres atributs anteriors tindran els seus respectius *getter* i *setter* que es comportaran d'igual manera que a la classe `Animal` (llegeix les seves descripcions).

A més, igual que ocorre a la classe `Animal`, la classe `SubmarineToy` té un mètode anomenat `reverse` que assigna a l'atribut `facingRight` el valor contrari que té en aquest moment.

Finalment comentar que aquesta classe tindrà un sol constructor i serà amb paràmetres, l'ordre dels quals serà: `xCoord`, `yCoord`, `length`, `height` i `tank`. El valor de `spriteImage` serà sempre el mateix: `"./images/submarine/submarine.png"` (ara com ara no pensis perquè és aquest el valor, ho entendràs en la propera pràctica).

Kelp

La classe `Kelp` ha de tenir un atribut privat denominat `growStep` que s'inicialitzarà amb el valor `50`. Aquesta classe, ara com ara, no tindrà cap *getter* ni *setter*. Aquesta classe només tindrà un constructor i aquest serà amb paràmetres. L'ordre dels paràmetres serà: `xCoord`, `yCoord`, `length`, `height` i `tank`. El valor de `spriteImage` serà sempre el mateix: `"./images/kelp/kelp_default.png"` (ara com

ara no pensis perquè és aquest el valor, ho entendràs en la propera pràctica).

TankException

La classe `TankException` és la mateixa que has creat en l'exercici anterior. Així doncs, només has de copiar-la.

Tank

Copia la classe `Tank` que has creat després de completar l'Exercici 1. Veuràs que apareix un error en el mètode `removeDeadItems`. Això es deu al fet que ara la classe `Item` no conté el mètode `getStatus`. Així doncs, modifica el mètode `removeDeadItems` perquè elimini tots aquells ítems que puguin estar morts.

A partir de l'explicació anterior, se't demana:

a) Codifica la classe `Animal` perquè es comporti com indica l'enunciat. Fes el mateix amb `AnimalStatus` i `AnimalException`.
(1 punt: 0.8 punts `Animal`; 0.2 punts `AnimalException`)

b) Codifica la classe `SubmarineToy`.
(0.50 punts)

c) Codifica la classe `Food`.
(0.25 punts)

d) Codifica la classe `Kelp`.
(0.25 punts)

e) Codifica els canvis necessaris a la classe `Item`, `ItemException` i `Tank`.
(1 punt: 0.5 punts `Item`; 0.5 punts `Tank`)

Nota: Si per a un atribut, mètode o argument no s'indica el seu tipus i/o la seva visibilitat (i.e. nivell d'accés), hauràs de ser tu qui decideixi què és el que millor s'ajusta segons les característiques del problema/programa que estem resolent. El mateix si hi ha algun detall en la implementació d'un constructor i/o mètode.

Exercici 3 – Ampliant el nostre aquari (herencia, interfícies i enum) (6 punts)

Nota: Per fer aquest exercici és necessari haver completat l'Exercici 2. Una vegada completat, has de crear el projecte `PRAC3_ex3` i copiar tots els fitxers.

En aquest exercici anem a ampliar el nostre disseny i a continuar estenent el nostre aquari. Cal tenir en compte que quan en la propera pràctica fem la part gràfica, aquesta serà 2D.

Fins ara, tots els animals que es troben en un tanc són objectes de la classe `Animal`. Això és massa genèric. Por això, anem a diferenciar entre peixos (`Fish`) i cargols (`Snail`).

A més, t'hauràs adonat que alguns mètodes relacionats amb el desplaçament es repeteixen tal qual en diferents classes. Això ens fa adonar-nos que realment el que necessitem és crear una interfície denominada `Movable` que implementaran totes les classes que hagin de desplaçar-se per l'aquari.

Movable

Tots els objectes que es desplacen per l'aquari han d'implementar una interfície denominada `Movable`. Aquesta interfície defineix el següent conjunt de mètodes i constants que han de satisfer les classes que la implementin:

- Tindrà dues constants anomenades `TANK_PANE_WIDTH` i `TANK_PANE_HEIGHT` que determinen l'ample i alt dels nostres tancs a nivell visual (quan en la propera pràctica les mostrem gràficament per pantalla). Els valors d'aquestes dues constants són `404` i `346`, respectivament.
- El mètode `moveLeft` que serveix per moure un element a l'esquerra. No retorna res.
- El mètode `moveRight` que serveix per moure un element a la dreta. No retorna res.
- El mètode `moveUp` que serveix per moure un element cap amunt. No retorna res.
- El mètode `moveDown` que serveix per moure un element cap avall. No retorna res.

- Un mètode anomenat `collideWithTank` que permet saber si l'element ha col·lidit amb el tanc. Aquest mètode retorna un valor de tipus `Collision`.
- Un mètode anomenat `update` que serveix para determinar el moviment de cada objecte pel tanc. Dit d'una altra manera, és el cicle d'actualització de l'objecte. Així doncs, cada cert temps, el nostre programa (el de la propera pràctica) haurà de cridar a aquest mètode per actualitzar l'estat de l'objecte en funció de diferents aspectes. Aquest mètode no retorna res.
- Mètodes `getSpeed`, `setSpeed`, `getThresholdReverse` *i* `setThresholdReverse`. Respecte a les excepcions, és important destacar que els mètodes `setSpeed` i `setThresholdReverse` disparen una excepció del tipus `MovableException` (l'expliquem a continuació).
- El mètode `isFacingRight` que permet saber si s'està desplaçant cap a la dreta o no. La lògica és que retorni `true` si està mirant cap a la dreta i `false` en cas contrari.
- El mètode `reverse` que permet girar d'orientació/sentit. No retorna res.

MovableException

Aquesta classe heretarà d'`Exception` i serà l'encarregada de modelar les excepcions relatives als moviments. En concret s'han de gestionar aquests dos tipus d'errors:

- **MSG_ERR_SPEED_VALUE:** `"[ERROR] Speed cannot be 0 or negative !!"`.
- **MSG_ERR_THRESHOLD_VALUE:** `"[ERROR] Threshold reverse cannot be negative either greater than 1!!"`.

En l'Exercici 2 aquests dos missatges estan a la classe `ItemException` i ara han d'estar només a la classe `MovableException`.

Collision

És possible que els elements que es mouen pateixin col·lisions amb altres elements de l'aquari o amb el propi aquari. Per tant existirà un tipus `Collision` els valors de la qual ens indicaran per on està col·lidint un element: `LEFT`, `RIGHT`, `TOP`, `BOTTOM` i `NO_COLLISION`.

Tank

La classe `Tank` ara ha de tenir un nou mètode anomenat `getFood` que no rep cap argument. Aquest mètode retorna una `List<Item>` que inclourà només aquells ítems que hi ha en el tanc que siguin de la classe `Food`.

Animal

La classe `Animal` ara ha d'implementar la interfície `Movable`.

Els moviments cap a l'esquerra, dreta, a dalt i a baix, han d'actualitzar la localització dels animals en el tanc utilitzant les coordenades en les quals es troba i la velocitat a la qual es desplaça. Tingues en compte que l'eix d'ordenades `Y` és invers a com estàs acostumat/a en matemàtiques. És a dir, la coordenada `(0,0)` és la cantonada superior esquerra de "la teva pantalla" i els valors positius d'`Y` van cap avall i els negatius cap amunt. L'eix d'abscisses `X` es comporta com al món matemàtic (valors positius a la dreta i negatius a l'esquerra).

L'algorisme del mètode `collideWithTank` que determina la col·lisió entre l'animal i el tanc és el següent:

- Si l'animal està dirigint-se cap a l'esquerra i està a menys de 3 unitats de xocar amb el tanc, llavors el mètode ha de retornar una col·lisió del tipus `LEFT`.
- La col·lisió a la dreta (`RIGHT`) es dóna quan l'animal està orientat a la dreta i a una distància menor a `60` unitats.
- Les col·lisions superiors (`TOP`) i inferiors (`BOTTOM`) són similars a les anteriors, però els marges seran `5` unitats per al marge superior i `45` per al marge inferior.
- En cas contrari, no hi haurà col·lisió.

El mètode `update` no ha d'implementar-se a la classe `Animal`. És més, no té ni que aparèixer en el codi de la classe `Animal`.

La classe `Animal` inclourà 2 nous mètodes públics que no tindran cos (i.e. no escriurem claus `{}` per a aquests mètodes): `getOxygenConsumption` (no rep es i retorna un `int`) i `breathe` (no rep res i retorna `void`).

Així mateix aquesta classe inclourà un nou mètode públic anomenat `eat` que no rep res i retorna `void`. L'algorisme de menjar (`eat`) consisteix en el següent:

- Si l'animal està mort, no menja (no fa res).
- Si està viu, mira si per a qualsevol objecte de la classe `Food` que estigui en el mateix tanc que ell, aquest està a una distància menor de 5 i 8 posicions en els eixos `X` i `Y`, respectivament. Si és així, l'animal es menjarà aquest objecte `Food`, actualitzant l'estat de l'aliment a “menjat”. En menjar-se aquest menjar, l'animal veurà incrementat la seva energia la quantitat d'energia que tingui el menjar que s'acaba de menjar.

Per donar informació rellevant en fer un `print` d'un objecte `Animal`, anem a sobreescriure el mètode `toString` de manera que retorni el següent `String`: `"(X, I) F G"`, on `X` i `Y` són les coordenades `x` i `Y`, respectivament. `F` és el la lletra `"R"` o `"L"` segons si l'animal està mirant/orientat cap a la dreta (Right) o cap a l'esquerra (Left); i `G` és el valor del gènere.

Finalment, veuràs que el constructor d'`Animal` llança (`throws`) una excepció de tipus `Exception`, la qual cosa no ens permet diferenciar entre els tipus d'excepcions que llança. Així doncs, hem d'especificar què excepcions pot llançar el constructor d'`Animal` (escriu-los en ordre jeràrquic, i.e. de la classe més concreta a la més general). Això permet, d'una banda, escriure un codi que explica exactament el que fa i, per tant, més fàcil de mantenir. D'altra banda, indica a l'usuari de la classe quines excepcions concretes pot esperar (`Exception` és massa genèric).

SubmarineToy

La classe relacionada amb els submarins de joguina també implementa la interfície `Movable` i, per tant, ha de codificar tots els mètodes declarats en `Movable`.

- Els mètodes dels moviments (`moveUp`, `moveLeft`, `moveRight` i `moveDown`) així com el mètode `collideWithTank` es comporten exactament igual que a la classe `Animal`.
- Ara als submarins de joguina els hi declararem un atribut nou denominat `balanceMove` (balanç del moviment) de tipus `int` el valor del qual per defecte és `0`. Aquest atribut no tindrà ni *getter* i ni *setter*.
- També afegirem una constant privada de tipus `int` denominada `TURN_BALANÇ_MOVE` amb el valor `100`. No té ni *getter* ni *setter*.
- Cada objecte `SubmarineToy` realitza una actualització del seu estat usant el mètode `update`. Aquest mètode `update` cridarà a un mètode privat anomenat `move` que no retorna res. L'algorisme d'aquest mètode privat consisteix en el següent:
  - La joguina submarina pot col·lidir amb el tanc tant per l'esquerra (`LEFT`), dreta (`RIGHT`), part superior (`TOP`) o el fons (`BOTTOM`). També és possible que la joguina no col·lideixi amb el tanc.
  - Si el balanç del moviment (`balanceMove`) és major que `0`, llavors es mourà cap avall, i li restarem `1` unitat a l'atribut `balanceMove`.
  - Si el balanç del moviment és menor que `0`, llavors es mourà cap amunt i el balanç s'augmentarà en `1` unitat.
  - En cas que el balanç sigui `0`, l'algorisme es complica lleugerament donat que si la joguina col·lideix amb el tanc tant a l'esquerra com a la dreta, el que ha de fer és girar-se cap al costat contrari. A més si col·lideix amb la part superior o inferior del tanc, el balanç de moviment s'assignarà a la constant `TURN_BALANÇ_MOVE` positivament o al valor de la constant negativament, respectivament. D'altra banda, si no hi ha cap col·lisió, llavors es calcularà un valor de decisió aleatori entre `0` i `1` (valor de tipus `float`; has de buscar com crear un valor aleatori en Java) que:
    - en cas que sigui menor que el llindar de gir (`thresholdReverse`) farà que la joguina es giri (i.e. canviï d'orientació).

- En cas que sigui major, llavors la joguina es desplaçarà en l'orientació que es trobi i es tornarà a calcular un nou valor aleatori de manera que si aquest valor és menor a `0.48`, llavors es desplaçarà, a més, cap amunt i, en cas contrari, es desplaçarà cap avall.

Food

El menjar també implementa la interfície `Movable`. No obstant això, només haurem de codificar:

- `moveDown`: ja que el menjar és llançat des de dalt del tanc i baixa fins al fons.
- `collideWithTank`: en aquesta ocasió la col·lisió amb el tanc únicament té la possibilitat de col·lidir amb el fons (això si succeeix si es troba a menys de `20` unitats del fons del tanc). Així doncs, el menjar o col·lideix per sota o no col·lideix.
- `update`: flama a un mètode privat anomenat `sink`. El mètode `sink` crida al mètode `moveDown` si no està col·lidint per sota.
- `setSpeed` i `getSpeed`: aquests dos mètodes mantenen el codi de l'Exercici 2.
- La resta dels mètodes de la interfície `Movable` no tenen cap sentit a la classe `Food`. Així doncs, haurem d'escriure el codi mínim perquè la implementació de la interfície `Movable` no doni errors. En el cas de `isFacingRight`, aquest retornarà `false` i `getThresholdReverse` retornarà `0`.

El constructor de `Food` indica que només llança (`throws`) la superclasse `Exception` quan en veritat llança aquesta i subtipus d'aquesta. Posa el llistat d'excepcions, de més concreta a més general.

Fish

Els peixos són un tipus d'`Animal`. Aquesta classe serà abstracta. Els peixos poden tenir un `color` el valor del qual indicarà el color predominant del peix. Els valors de l'atribut `color` vénen determinats per l'`enum Color`: `YELLOW`, `RED`, `GREEN`, `BLUE`, `GRAY`, `WHITE`, `BLACK`, `ORANGE`, `BRONZE` i `NOT_DEFINED`. Aquest `enum` tindrà un únic constructor que tindrà un paràmetre. Aquest paràmetre permet indicar el codi hexadecimal associat al color (és un `String`): `YELLOW (#FFFF00)`, `RED (#FF0000)`, `GREEN (#00FF00)`, `BLUE`

(#0000FF), GRAY (#888888), WHITE (#FFFFFF), BLACK (#000000), ORANGE (#FF8300), BRONZE (#CD7F32) i NOT_DEFINED (#). Finalment l'enum Color tindrà dos mètodes: (1) getHexCode que retornarà el codi hexadecimal del valor, i (2) getColor que donat un codi hexadecimal retornarà el valor corresponent de l'enum o, en cas de no existir, retornarà el valor NOT_DEFINED.

Tornant a la classe Fish, dir que l'atribut color tindrà els seus corresponents mètodes *getter* i *setter* públics.

El constructor de la classe Fish serà protected i tindrà la següent signatura:

```
protected Fish(double xCoord, double yCoord, String spriteImage, double length, double height, Gender gender, int age, double speed, double requiredFoodQuantity, float thresholdReverse, AnimalStatus status, int energy, Color color, Tank tank)
```

És a dir, els paràmetres en el mateix ordre que el constructor d'Animal però amb l'atribut color entre els paràmetres energy i tank.

Cada Fish realitza una actualització del seu estat quan des d'un programa principal (que ja farem) cridi al seu mètode update. El mètode update de Fish, de moment, executa dues accions privades en el següent ordre: nedar (swim) i menjar (eat; mètode heretat de la superclasse Animal).

L'algorisme de nedar (swim) consisteix en el següent:

- El peix pot col·lidir amb el tanc tant per l'esquerra (LEFT), dreta (RIGHT), part superior (TOP) o el fons (BOTTOM). També és possible que el peix no col·lideixi amb el tanc.
- Si el peix està mort (AnimalStatus.Dead) i no ha col·lidit encara amb el fons del tanc, llavors es mourà cap avall (anirà en direcció al fons del tanc). En cas contrari, no farà res.
- En cas que el peix no estigui mort, si està col·lidint amb el tanc per l'esquerra o per la dreta, el que ha de fer és girar-se cap al costat contrari.
- En cas que el peix no estigui mort, si està col·lidint amb la part superior del tanc, el seu moviment serà cap avall.
- En cas que el peix no estigui mort, si està col·lidint amb la part inferior del tanc, el seu moviment serà cap amunt.

- Finalment, si el peix està viu i no es produeix col·lisió alguna, el que ha de succeir és que es calculi un valor aleatori del tipus `float` entre `0` i `1` (tots dos inclosos) que determini la decisió del moviment del peix.
  - En cas que aquest valor sigui inferior al valor de llindar per girar-se (`thresholdReverse`), el peix es girarà.
  - En cas que la decisió sigui igual o superior al llindar de gir (`thresholdReverse`), es tornarà a calcular un nou valor de decisió. Amb aquest nou valor poden succeir diverses circumstàncies:
    - Si el valor és igual o superior a `0.9`, el peix es mourà cap avall.
    - Si el valor està entre `0.8` (inclòs) i `0.9` (no inclòs), el peix es mourà cap amunt.
    - En cas contrari, el moviment del peix dependrà de si el peix està mirant/orientat a la dreta o a l'esquerra. En cas de trobar-se mirant cap a la dreta, el peix es mourà cap a la dreta. En caso contrari, el peix es mourà cap a l'esquerra.

Malgrat que `Fish` herata d'`Animal`, no sobreescriurem els dos mètodes abstractes de la superclasse `Animal` (i.e. `getOxygenConsumption` i `breathe`), doncs deixarem la seva implementació per a les subclasses de `Fish`.

Finalment comentar que la classe `Fish` sobreescriu el mètode `toString` de la classe `Object` de manera que imprimeix exactament el mateix que la superclasse `Animal` però amb el següent text al final: `" : C"`, on `C` és el valor de l'atribut `color`. És a dir:

`"(X, Y) F G : C"`.

Snail

Els cargols són un subtipus d'`Animal`. Només té un constructor i és amb paràmetres, l'ordre dels quals ha de ser: `xCoord`, `length`, `height`, `gender`, `age`, `energy` i `tank`. El valor d'`yCoord` serà sempre el valor de `TANK_PANE_HEIGHT - 30`. Així mateix el valor de `spriteImage` serà per a tots els objectes: `"./images/snail/snail.png"`.

És important destacar que els caragols només s'arrosseguen pel fons de l'aquari (`crawl`) de dreta a esquerra i viceversa, mai cap amunt o cap avall. Per tant, cal evitar que donat un objecte de tipus `Snail` aquest pugui pujar o baixar.

D'altra banda, en el mètode `update` que permet actualitzar l'estat d'un cargol, només s'executarà el mètode `crawl`. L'algorisme d'arrossegar-se és el següent:

- En primer lloc el mètode ha de comprovar si el cargol col·lideix per l'esquerra o a la dreta. Si és així, canviarà d'orientació (i.e. es girarà).
- A continuació calcularà un valor aleatori entre `0` i `1` (tots dos inclosos). Si aquest valor és inferior a `0.00000003`, el cargol canviarà d'orientació. En cas contrari, seguirà el seu rumb.
- Finalment, l'última acció que farà l'algorisme és comprovar com està orientat el cargol. Si està mirant a la dreta, ho mourà a la dreta. En cas contrari, ho mourà cap a l'esquerra.

La classe `Snail` ha de sobreescriure els mètodes `getOxygenConsumption` i `breathe`. Per al primer simplement retornarà el valor `1`. En el cas de `breathe`, simplement obrirem claus i escriurem en el cos del mètode el comentari `//TODO`.

A partir d'aquí, se't demana:

a) Codifica `Movable`, `MovableException` i `Collision`.

(0.4 punts `Movable`)

b) Codifica el mètode `getFood` a la classe `Tank`.

(0.4 punts)

c) Actualitza la classe `Animal`.

(1 punt: 0.4 pts. `collideWithTank`; 0.5 pts. `eat`; 0.1 pts. `toString`)

d) Actualitza la classe `SubmarineToy`.

(0.4 punts mètode `move`)

e) Actualitza la classe `Food`.

(0.4 punts per la classe en general)

f) Codifica `Fish` i `Color`.

(0.9 punts: 0.4 punts mètode `swim`; 0.15 punts la resta de `Fish`; 0.25 punts `Color`)

g) Codifica `Snail`.

(0.5 punts: 0.3 punts mètode `crawl` i 0.2 punts la resta)

h) Utilitzant el programari Dia, dibuixa el diagrama de classes UML complet del software de gestió de l'aquari implementat fins a aquest exercici inclòs. Has de generar un fitxer `.dia` i una imatge del diagrama en format `.png`. Reflexiona primer abans de codificar el problema.

(2 punts)

Nota per a l'apartat (h): no has d'incloure informació referent a `packages` ni tampoc has d'incloure classes pròpies de Java, p.ex. `Exception`, `ArrayList`, etc. D'igual manera, no has d'incloure les classes que fan referència a excepcions personalitzades (p.ex. `AnimalException`). Tot això no se sol posar perquè, en cas contrari, el diagrama seria immens.

Per declarar un atribut que sigui una col·lecció d'objectes, utilitza els claudàtors (nomenclatura dels *arrays*), p.ex. `people:Person[]`. És en l'etapa d'implementació on decideixes si aquesta col·lecció és una `ArrayList`, un `HashMap`, un *array*, etc. Recorda que els diagrames de classes UML han de ser el més independents del llenguatge com sigui possible.

Recursos

Aquesta Pràctica se centra especialment en els conceptes estudiats en el mòdul *<<Herència (relació entre classes)>>*, però inevitablement treballa també els conceptes vistos en els mòduls anteriors. També disposes a l'aula de dues guies de Java (una en estat preliminar).

És molt normal que sorgeixin dubtes quan es programa, encara que portem molts anys programant amb un llenguatge i un paradigma concrets. És per això que cal tenir molt assimilada la competència de buscar informació.

Avui dia, el millor lloc on trobar informació i de manera ràpida és Internet. En el cas de Java, tens tota la documentació de la seva API (i.e. llibreries amb classes ja fetes) en la següent web: https://docs.oracle.com/en/java/javase/13/docs/api/index.html. També pots utilitzar fòrums com *stackoverflow*, tant la versió anglesa com l'espanyola:

- Anglès: http://stackoverflow.com/questions/tagged/java
- Espanyol: http://es.stackoverflow.com/questions/tagged/java

O utilitzar directament el cercador Google: www.google.com

Si vols ampliar coneixements, sempre pots comprar llibres sobre POO i Java en alguna llibreria o demanar-los prestats a qualsevol biblioteca (incloent la de la UOC, http://biblioteca.uoc.edu).

Si malgrat buscar informació, no soluciones els teus problemes, pots fer preguntes teòriques, demanar aclariments dels enunciats o preguntar sobre aspectes tècnics (p.ex. Java) en el fòrum de l'aula. **No pots utilitzar els fòrums ni cap altre mitjà per demanar a un altre company la solució del que es demana als enunciats. Tant la persona que demana com qui proporciona la solució, seran penalitzades seguint la normativa acadèmica de la UOC.**

Criteris de valoració

El repartiment dels punts d'aquesta Pràctica és el següent:

- Exercici 1: 1 punt.
- Exercici 2: 3 punts.
- Exercici 3: 6 punts.

El detall del repartiment de punts en cada exercici està a l'enunciat.

Format i data de lliurament

S'ha de lliurar un fitxer *.zip, el nom del qual ha de seguir aquest patró: `loginUOC_PRAC3.zip`. Per exemple: *dgarciaso_PRAC3.zip*. Aquest fitxer comprimit ha d'incloure els següents elements:

- El projecte `PRAC3_ex1` completat seguint les peticions i especificacions de l'Exercici 1.
- El projecte `PRAC3_ex2` completat seguint les peticions i especificacions de l'Exercici 2.
- El projecte `PRAC3_ex3` completat seguint les peticions i especificacions de l'Exercici 3.
- Per a l'apartat (h) de l'Exercici 3, inclou on directori/carpeta anomenat `UML` amb el diagrama de classes de l'Exercici 3 complet realitzat amb Dia (fitxer `*.dia`) i la imatge en format PNG del diagrama realitzat.

L'últim dia per lliurar aquesta Pràctica és l'**17 de maig de 2020 a les 23:59**. Qualsevol Pràctica lliurada més tard serà considerada com no presentada.