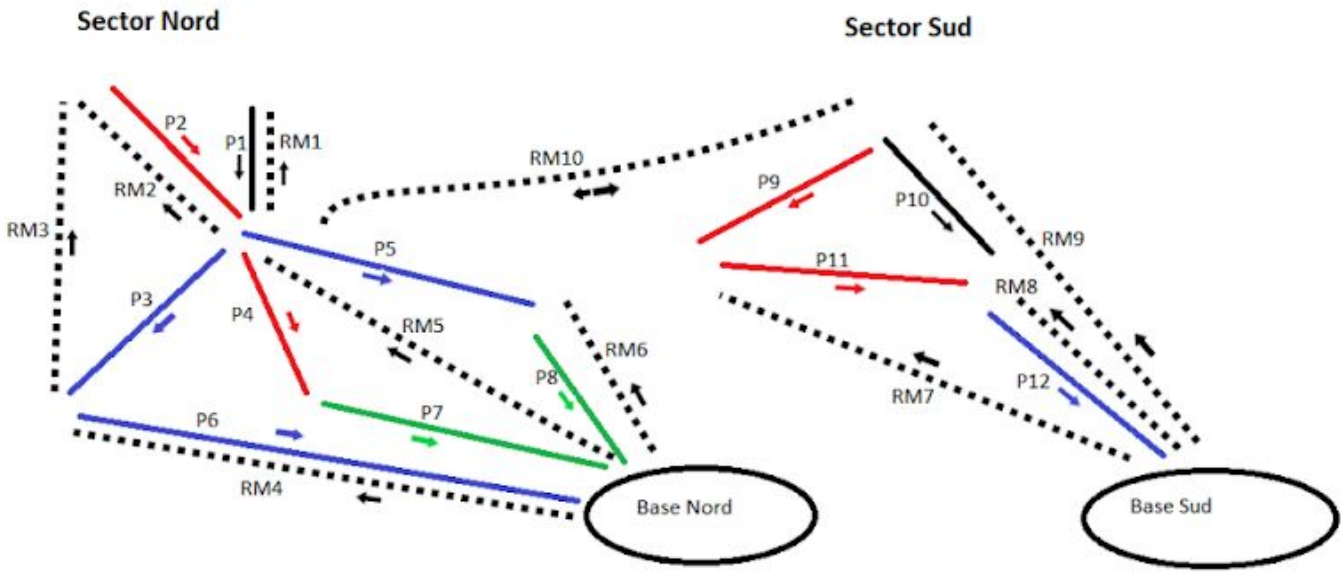
Pràctica 1: L'Estació d'Esquí Vall2000



Oscar de Caralt Roy MC

Pau Segura Baños MC

ENGINYERIA INFORMÀTICA UB

Primer de carrera: Programació 2

15/03/2021

ÍNDEX:

Introducció:

La pràctica que hem fet durant aquestes tres setmanes consisteix a construir un programa que modeli la informació corresponent a l'Estació d'Esquí Vall2000 segons les condicions meteorològiques que l'usuari introdueixi per teclat. I gestionar, a partir d'aquestes, si es pot accedir o no a determinades pistes.

Per a realitzar això es requereixen uns coneixements bàsics de POO (programació orientada a objectes), herència de classes, de polimorfisme, de creació de classes amb els seus respectius atributs i d'encapsulament.

A l'apartat d'implementació de classes hi ha una explicació de la classe, però l'explicació de cada mètode està inclosa als respectius javadocs.

1. Implementació de classes

Classe pista: classe que conté un constructor per construir l'objecte pista. Conté els atributs: nom, sector, color, longitud, estatDeLaNeu, estat i dependències.
Nom: serveix per identificar cada pista.
Sector: on està col·locada.
Color: nivell de dificultat de cada pista: Verda, Blava, Vermella o Negra.
Longitud: longitud de cada pista.
estatDeLaNeu: estat de la neu que pot variar entre Pols, Dura o Primavera.
estat: atribut que dirà si la pista està oberta o tancada. És un atribut molt important.
Dependencies: aquí es dirà de quin remuntador depèn cada pista. Si un remuntador està fora de servei, la pista del qual depèn també ho estarà.

També conté els mètodes actualitzaEstat, setters i getters de cada atribut, a afegirDependencies i toString.

Classe LlistaPistes: classe que serveix per anar afegint una a una les pistes creades a la classe pista, a un ArrayList. Tenim els mètodes actualitzaEstat, afegirPista, getPista, llistarPistes i CalculaKmsPista. Conté objectes de classe Pista.

Classe Remuntador: també conté un constructor per construir l'objecte remuntador. Conté els atributs: nom, sector, estat, velocitatdelVentMax i visibilitat.
Nom: serveix per identificar cada remuntador.
Sector: a quin sector de l'estació està col·locat.
Estat: si està en funcionament. Depèn de la velocitat del vent i la visibilitat.
velocitatdelVentMax: si la velocitat del vent introduïda per teclat supera aquest límit l'estat passarà a Fora de Servei.
Visibilitat: boolean que determina si el remuntador es veuria afectat per un canvi de visibilitat.
També conté els mètodes tipus, diferents setters i getters per cada atribut, actualitzaEstat i toString.

Classe LlistaRemuntadors: classe que serveix per anar afegint un a un els remuntadors creats a la classe Remuntador, a un ArrayList. Hi ha els mètodes actualitzaEstat, afegirRemuntador, totsForadeServei, llistarRemuntadors i getNoms. Conté objectes de classe Remuntadors.

Classe Telefèric: classe filla de Remuntador, a partir de la paraula reservada *extends*. Podem definir el tipus d'aquests remuntadors. Utilitza un constructor amb els mateixos atributs que a la classe mare. Això ho fa amb la paraula reservada *super*.

Classe Telecadira: classe filla de Remuntador, a partir de la paraula reservada *extends*. Podem definir el tipus d'aquests remuntadors. Utilitza un constructor amb els mateixos atributs que a la classe mare. Això ho fa amb la paraula reservada *super*.

Classe Telecabina: classe filla de Remuntador, a partir de la paraula reservada *extends*. Podem definir el tipus d'aquests remuntadors. Utilitza un constructor amb els mateixos atributs que a la classe mare. Això ho fa amb la paraula reservada *super*.

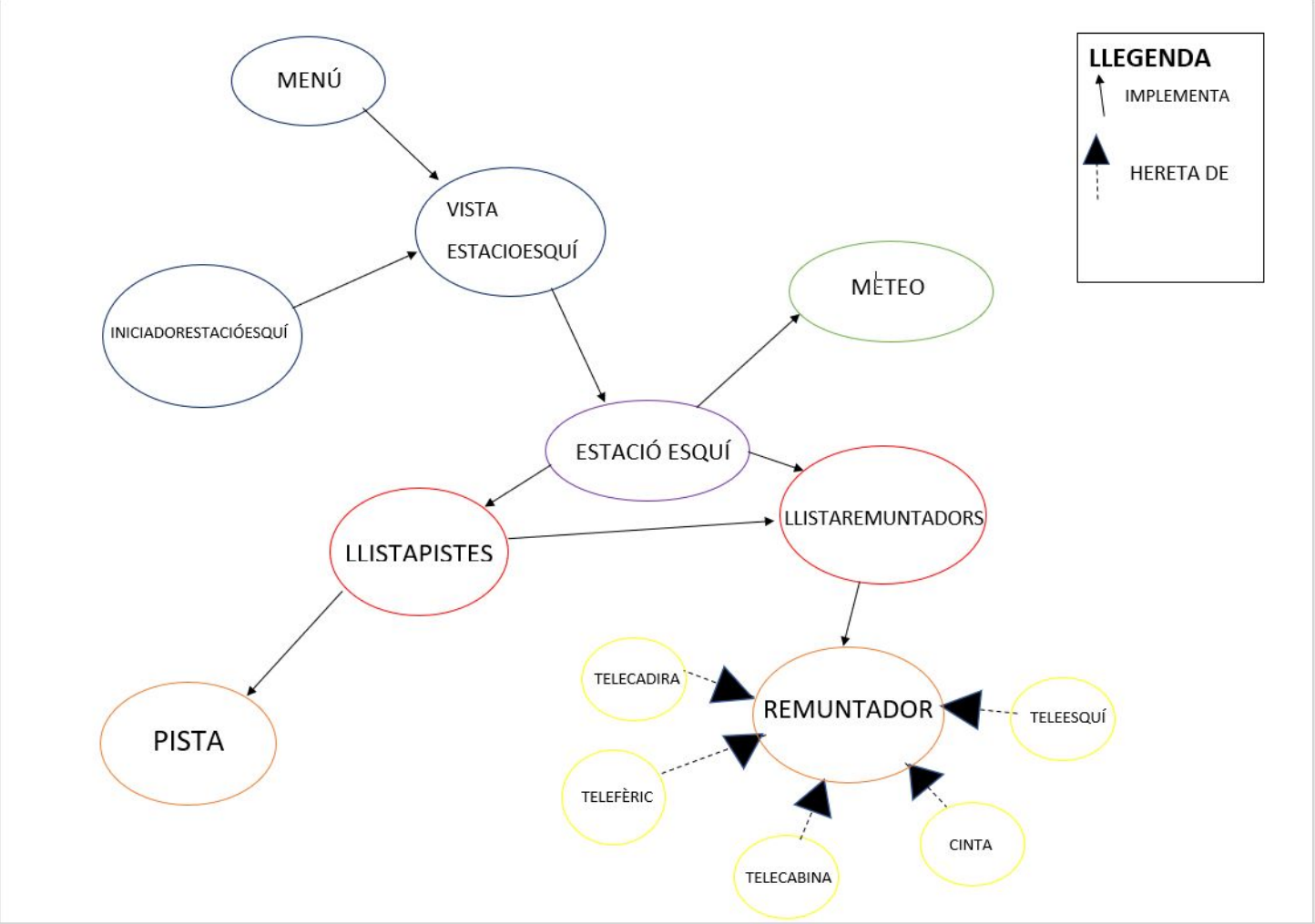
Classe Teleesqui: classe filla de Remuntador, a partir de la paraula reservada *extends*. Podem definir el tipus d'aquests remuntadors. Utilitza un constructor amb els mateixos atributs que a la classe mare. Això ho fa amb la paraula reservada *super*.

Classe cintaTransportadora: classe filla de Remuntador, a partir de la paraula reservada *extends*. Podem definir el tipus d'aquests remuntadors. Utilitza un constructor amb els mateixos atributs que a la classe mare. Això ho fa amb la paraula reservada *super*.

Classe Meteo: classe on es defineixen les condicions meteorològiques de l'estació a partir d'un constructor amb els atributs velocitatdelVent i visibilitat i els seus respectius setters i getters. A més hi ha un mètode toString.

Classe EstacióEsquí: per començar, en aquesta classe declarem objectes de classe LlistaRemuntadors, LlistaPistes, Meteo, Pista, Remuntador. Strings visibilitat i estació i un int VelocitatdelVent. En aquesta classe s'inicialitzen totes les dades de les pistes i remuntadors i s'afegeixen les respectives dependències. També hi ha un constructor per l'estació d'esquí amb els atributs String estació, velocitatdelVent i visibilitat amb els seus getters i setters.
També inclou els mètodes, actualitzaEstatNeu, calculaTotalKm, modificaVelocitatdelVent, modificaVisibilitat, reportMeteo i actualitzaEstat.

2. Dibuixar el diagrama de relacions entre les classes que has utilitzat a la teva pràctica. No cal incloure la llista d'atributs i mètodes



(diagrama de relacions entre classes a través del word)

3. Identificació i explicació dels atributs de la classe EstacioEsqui.

Meteo meteo: per poder modificar la velocitat del vent o la visibilitat. Aquests canvis podrien provocar que alguns remuntadors variessin el seu estat i per tant deixessin d'estar operatius i en conseqüència algunes pistes també.

Pista pista: objecte pista per poder construir les pistes amb els seus respectius atributs.

Remuntador remuntador: objecte remuntador per poder construir els remuntadors amb els seus respectius atributs. També per poder establir les dependències entre els remuntadors i les pistes.

LlistaPistes llistaPistes: per poder afegir les pistes creades prèviament a l'ArrayList, amb el mètode afegirPista, actualitzar l'estat de la neu i per mostrar els km totals.
També per poder accedir a les funcions de la classe llistaPistes.

LlistaRemuntadors LlistaRemuntadors: per poder afegir els remuntadors creats prèviament, a ArrayList, amb el mètode afegirRemuntador. També per poder accedir a les funcions de la classe llistaPistes.

String Estacio: per inicialitzar l'estació amb el seu constructor. Més tard s'inicialitza a la classe IniciadorEstacióEsquí amb els atributs **String** estació, **String** visibilitat (indica la visibilitat de l'estació) i **int** velocitatdelVent(indica la velocitat del vent).

.

4. Com es podria haver desenvolupat el projecte sense les classes LlistaPistes i LlistaRemuntadors? Quin seria l'impacte en l'estructura del vostre codi?

Com la interpretació d'aquesta pregunta és bastant ambigua, assumirem que la pregunta va en la següent direcció: Com podríeu desenvolupar el projecte sense utilitzar les classes LlistaPistes i LlistaRemuntadors però podent utilitzar els atributs d'aquestes.

Llavors, el que podríem haver fet seria implementar les ArrayLists de LlistaPistes i LlistaRemuntadors a les classes Pista i Remuntador respectivament, i haver-les omplert. Un cop haguéssim tingut les ArrayLists hauríem d'haver variat la majoria dels mètodes de les classes Pista i Remuntadors i en comptes de fer petits mètodes que servissin per a una pista en concret, hauríem d'haver recorregut l'Array en tots i fer els canvis recorrent l'Array en cada cas. I, tot i ser una mica més difícil d'implementar, hauríem pogut arribar al mateix punt que tal com hem realitzat la pràctica.

L'estructura del nostre codi, hauria variat, ja que no disposaríem de les classes LlistaPistes i LlistaRemuntadors i per tant hauríem d'haver fet que la classe Pista i la Classe Remuntador implementessin de la classe EstacioEsqui. A més a més, la facilitat per a la lectura de codi també s'hagués vist afectada de manera negativa a l'haver-hi menys implementacions.

5. Que passaria si cada objecte de tipus Pista creés els seus propis remolcadors (dels quals depèn) i els guardéssiu en la seva llista de remolcadors?

El que passaria si cada objecte de tipus Pista creés els seus propis remolcadors (dels quals depèn) i els guardéssim en la seva llista de remolcadors seria que alguns mètodes resultarien prescindibles i encara que la implementació d'aquests objectes fos una mica més complicada, podríem arribar a estalviar bastant codi.

6. Detallar les proves realitzades per comprovar el correcte funcionament de la pràctica, resultats obtinguts i accions derivades.

Per tal de confirmar que el programa funcionés correctament buscàvem els punts crítics de canvi d'estat en els remuntadors i les pistes. Per exemple, augmentant la velocitat del vent a 100 km/h i comprovant que l'estat de tots els remuntadors fos "Fora de Servei" i automàticament les pistes canviessin a "Tancada". Més tard posàvem la velocitat per sobre del límit d'alguns remuntadors però per sota d'altres, com ara a uns 40 km/h. El funcionament era correcte ja que els RM1,RM2,RM3 quedaven fora de servei ja que la seva velocitat límit és de 35 km/h. Més tard modifiquem la visibilitat i la posem en bona per veure si el RM9 canvia d'estat, i afirmativament l'estat varia a "En Servei". Totes les pistes també varien correctament en consonància amb els remuntadors dels quals depenen.
També provem a canviar l'estat de la neu d'alguna pista, canviant l'estat a un dels tres permesos: Dura, Pols i Primavera, i confirmem que funciona correctament.

Els resultats han resultat bastant satisfactoris, malgrat que hi ha un petit error al mètode on modificàvem la visibilitat. Ens imprimeix dues vegades la frase: *"Quina visibilitat hi ha? Bona o Dolenta*" només quan hem provat una altra opció del menú abans que aquesta, però si la cridem al començament no s'imprimeix doble. Tot i això, la resta del programa funciona perfectament.

Bàsicament, el que fèiem era: en cas que sorgís algun error durant les comprovacions, anàvem al mètode en qüestió i l'intentàvem arreglar amb el *debugger.* Al principi als mètodes modificaVisibilitat i modificaVelocitatdelVent els havíem fet amb un *do while*, però s'imprimia dues vegades el system.out.print que teníem posat. Vam canviar el *do while* per un simple *while* i l'error a modificaVelocitatdelVent es va solucionar.

7. Observacions generals

En començar el treball no sabíem gaire bé ni com ni per on començar. Quan vam aprendre que era una classe ja vam començar amb les classes Pista i Remuntador.
Vam anar a poc a poc, però sense parar. Realitzant i completant totes aquelles funcions que podíem a causa del cos de coneixement que teníem en aquell moment. I a mesura que anàvem aprenent cada cop més sobre POO a les classes (tant presencials com virtuals) vam poder avançar fins a arribar al punt actual. On tenim tot el programa fet i ens sentim realitzats i satisfets de com ha quedat el treball gràcies al nostre esforç i al nostre temps i el del professor de les classes teorico-pràctiques, que ens va ajudar a resoldre bastants dubtes i va tenir molta paciència amb nosaltres.

D'aquest treball, el que més podríem destacar, són els mètodes actualitzaEstat que són el que més ens va costar (no els vam saber fer fins a l'última classe teorico-pràctica), però al final de la pràctica hem pogut entendre tots els conceptes que ens demanaven.
Gràcies a aquesta pràctica hem pogut aprendre a com crear les classes amb els seus constructors i getters i setters. També hem pogut reforçar els coneixements sobre l'herència de classes i els mètodes que implementen d'una classe a una altra.