Estructura de dades
Pràctica 2 - Estudi previ

Introducció

Abans de començar amb la implementació del projecte, cal considerar els objectius que ens hem plantejat. En aquest cas, se'ns proposa treballar amb diferents classes en un entorn de programació orientada a objectes (OOP) que simulen situacions reals a través de l'ús d'herències, interfícies i altres característiques. Entre altres exercicis que fiquen a prova el coneixement per tractar amb funcions de tipus genèric.

Descripció general

L'objectiu d'aquest projecte és modelar diferents estructures d'informació que gestionin persones, enginyers i departaments, contactes dins d'una agenda, i per últim una estructura que ens permeti emmagatzemar qualsevol tipus de dades.

Però l'objectiu principal és entendre al màxim els perques que van sorgint durant el projecte, i aconseguir l'optimització màxima fent ús de les eines que ens ofereix el llenguatge.

Classes implicades

El nostre projecte necessitarà diverses classes:

Person: Classe base que encapsula les propietats i comportaments bàsics d'una persona, com el nom i altres atributs generals.

Engineer: Subclasse de Person que representa un enginyer amb atributs i comportaments específics.

Department: Classe que modela un departament que pot contenir una llista d'enginyers, inclòs un cap de departament.

Storage: Classe genèrica que emmagatzema objectes de qualsevol tipus, proporcionant funcionalitats per afegir i copiar elements entre instàncies de Storage.

ContactAgenda i Contacte: Aquestes classes gestionaran una agenda de contactes, permetent afegir, buscar, eliminar i llistar els contactes amb l'ús d'iteradors.

Funcionalitats principals

1. Gestió de departaments i enginyers: El sistema serà capaç de gestionar diversos departaments, assignant un enginyer com a cap de cada departament i permetent afegir més enginyers a una llista. Aquesta funcionalitat també inclou la possibilitat de llistar els noms dels enginyers d'un departament.

2. Agenda de contactes: L'agenda ens permetrà gestionar els contactes amb les operacions bàsiques de qualsevol aplicació d'aquest tipus: afegir contactes (verificant si ja existeixen), eliminar-los i buscar-ne mitjançant el nom.

3. Emmagatzematge genèric: La classe Storage ens permetrà crear emmagatzematges de diferents tipus d'objectes (com nombres o persones), permetent operar amb ells de manera flexible.

Decisions de disseny

Herència i interfícies: Hem decidit utilitzar herència per establir les relacions entre classes com Person i Engineer, ja que aquests comparteixen característiques comunes però també presenten comportaments específics que justifiquen la seva especialització. Això facilita la reutilització de codi i assegura que tots els enginyers es tractin com a persones, però amb funcionalitats addicionals.

Genèrics: S'utilitzen genèrics en classes com Storage per oferir flexibilitat i assegurar la seguretat de tipus durant el procés d'emmagatzematge d'objectes.

Iteradors: En la implementació de l'agenda, farem servir ListIterator per proporcionar una manera eficient i flexible de recórrer i manipular els contactes.

Consideracions d'eficiència

Per garantir que les operacions siguin eficients, hem considerat l'ús de col·leccions de Java com ara LinkedList per a llistats dinàmics i ArrayList per a col·leccions on es requereixi un accés ràpid. A més, en operacions de cerca i modificació de dades dins de l'agenda o els departaments, s'ha de tenir en compte la complexitat temporal per evitar dissenys ineficients.

Conclusió

Aquest projecte ens permetrà practicar i consolidar conceptes fonamentals de la programació orientada a objectes, com l'herència, l'ús de col·leccions i la manipulació d'iteradors. Abans de començar amb la implementació, és essencial planificar bé la relació entre les classes i tenir clar quines operacions es realitzaran sobre les dades. Això ens permetrà escriure un codi més net, modular i mantenible.