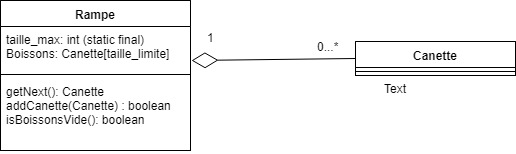
Exo 1:

1. 
2. J’utilise ArrayList pour stocker plusieurs Canettes dans une Rampe .

A close up of a map

Description generated with high confidence

1. On donne une nouveau attribut dans la classe Canette : add :boolean = false (en défault). C’est-à-dire si une Canette est contenue dans une Rampe, on l’indique qu’il est stocké ( on met la value de « add » qui est « true »).

A screenshot of a cell phone

Description generated with very high confidence

La méthode getAdd() est de vérifier une Cannette qui est stocké dans une Rampe ou non.

public boolean addCanette(Canette boisson){

if(!boisson.getAdd()){

Boissons.add(boisson);

boisson.setAdd(true); // set add = true

return true;

}

return false;

}

Je pense que la Rampe travail selon la mécanisme LIFO (comme une pile). Pour récupérer la premièr Canette, on récupère le dernier élément de Rampes et en même temps on le surrime dans le ArrayList<>.

public Canette getNext(){

try{

Canette Canette\_Recuper = new Canette();

Canette\_Recuper = Boissons.get(Boissons.size());

Rampes.remove(Boissons.size());

return Canette\_Recuper;

} catch(Exception e){

System.out.println(" La Rampe est vide");

}

}

Exo 2 :

1. A close up of text on a white background

   Description generated with very high confidence
2. Je pense que le Hashmap<K, V> est la structure de données qui est la mieux apdaté parce qu’un distributeur peut stocker plusieurs Rampes et chaque Rampe correspond à un nombre unique ( afin de saisir un numero qui permet d’obtenir la boisson). On stock donc emsemble 2 informations : 1) le numéro et 2) un Rampe , donc je choisis le HashMap<K,V>
3. public Canette servir(int numero){

Canette boisson\_servir = new Canette();

if(Rampes.get(numero).isDesBoissonsVide())

return boisson\_servir = Rampes.get(numero).getNext();

return boisson\_servir = null;

}

Exo 3 :

1. A screenshot of a cell phone

   Description generated with high confidence

La méthode credit(int argent) , il est du même méthode setArgent(int argent) ( insérer de l’argent ⬄ mettre la valeur « argent » ).

Méthode Similiaire debiter() , on rend tous des argents ⬄ renvoie la valeur « argent » et maintenant argent = 0.

Saisir(int numero) : affiche le prix de produit qui correspond son numéro.

Servir(int numero) : il est une méthode override de la classe Distributeur.

1. Son code maison, je le comprends pas
2. Je pense qu’on vérifie d’abord la valeur d’argent dans le Distributeur. Si il reste l’argent dans la machine, on pourra rendre, sinon c’est pas possible

###Code

public int debiter(){

if(this.argent > 0){

return this.argent && this.argent = 0;

}

return 0;

}

d)

\* La méthode servir() effectue 2 étapes :

1) il vérifie que l’argent dans le distributeur est supérieur à la prix de produit

2) il vérifie que dans la Rampe qui correspond avec son numero, la boisson est vide ou non .

#Override

public Canette servir(int numero){

Canette boisson\_servir = new Canette();

if(this.argent >= saisir(numero)){

if(!Rampes.get(numero).isDesBoissonsVide()){

this.argent = this.argent - saisir(numero);

return boisson\_servir = Rampes.get(numero).getNext();

}

}

return boisson\_servir = null;

}