EPHEC
Louvain-la-Neuve

Nom : .....

Prénom : .....

Classe : 2TL1 - 2TL2

Interrogation de janvier 2015

Langage avancé de programmation

Professeur : V. Van den Schrieck

Date : 12/1/2015 **Matériel autorisé** : -

Nombre de pages : 27 **Durée** : 120 minutes

Répondre aux questions directement sur ce document.

Acquis d'apprentissage évalués

- Utiliser le vocabulaire adéquat pour expliquer et discuter les concepts orientés-objets
- Analyser une situation en l'exprimant dans un formalisme orienté-objet (en l'occurrence, UML)
- Développer une solution logicielle dans le langage Java sur base de spécifications détaillées (textuelles ou UML)
- Utiliser une méthodologie, des outils et des techniques de collaboration afin de mener à bien un projet de développement java en équipe

Pondération

	Points pour la partie	Points de l'étudiant
Architecture	8	
POO	12	
Bonnes pratiques	6	
Programmation réseau	5	
Programmation concurrente	3	
Total	**34**	
Total	**20**	

Contexte

Le programme sur lequel l'examen se base est un logiciel permettant la gestion d'un magasin de cookies.
Le magasin possède un compte banquaire contenant un montant de départ (investissement), un stock initialement vide, et des recettes par défaut. A chaque commande d'un type de cookies, le logiciel doit vérifier s'il y a les ingrédients suffisants pour le type de cookie demandé. Si ce n'est pas le cas, mais que le solde du compte le permet, des ingrédients sont automatiquement commandés et rajoutés au stock, et la commande est honorée. Si le solde est insuffisant, les cookies du type demandé ne sont pas produits.
Le programme possède deux interfaces : Une interface graphique, et une interface réseau. Les deux fonctionnent en parallèle, ce qui fait que des commandes peuvent être effectuées via la fenêtre GUI ou via un client réseau.
Vous trouverez en fin du feuillet les classes qui composent ce programme. Commencez par les lire attentivement, avant de répondre aux questions. Les classes relatives aux exceptions ne sont pas fournies, mais vous pouvez considérer qu'elles sont disponibles. Certaines implémentation de méthodes et certaines spécifications triviales ne sont pas montrées par souci d'alléger la lecture. Certaines classes ne sont pas absolument utiles pour répondre aux questions, mais ont néanmoins été incluses au fascicule pour vous permettre d'avoir une vue d'ensemble de l'application.

Architecture (/5)

Question 1 [/4] : UML

Dessinez le diagramme UML du package cookies.magasin. Dans ce diagramme doivent figurer aussi bien les classes du package que les classes de l'API Java qui y sont explicitement mentionnées. Attention à bien respecter les conventions pour la représentation des relations. N'hésitez pas à ajouter des précisions à votre diagramme en commentaires.

Question 2 [/2] : MVC
Le logiciel est structuré selon le modèle MVC. Faites un schéma (non UML) représentant les relations entre les différents packages, dans le cadre de ce modèle. Expliquez également ce qu'apporte chacune des relations à l'architecture MVC.

Question 3 [/2] : Design Patterns
Des design patterns ont été utilisés pour construire ce code. Donnez-en deux, et pour chacun, détaillez-le brièvement et expliquez comment et pourquoi il est utilisé dans le cadre du logiciel.

Programmation orientée-objet (/13)

Question 4 [/2] : Utilisation d'objets

Ecrivez la méthode ajouterRecettesParDefaut() de la classe MagasinCookies, afin que, par défaut, le magasin propose dans sa carte des cookies au cacao. Pour un cookie au cacao, il faut :

- Une dose de farine (disponible par doses de 10 grammes, à 2 euros le kilo)
- Une dose de beurre (disponible par doses de 10 grammes, à 3 euros le kilo)
- Une dose de sucre (disponible par doses de 10 grammes, à 1 euros le kilo)
- Deux doses d'oeufs (disponible par doses de 10 grammes, à 5 euros le kilo)
- Deux doses de cacao (disponible par doses de 2 grammes, à 15 euros le kilo)

```
public static void ajouterRecettesParDefaut{




}
```

Question 5 [/4] : Héritage
Le propriétaire du magasin de Cookies a développé un nouveau concept : Celui d'un cookie avec un message personnalisé imprimé dessus. Ce cookie sera facturé 20% plus cher qu'un cookie normal de la même recette.
Vous devez implémenter cette modification pour permettre la commande de cookies avec message personnalisé, en apportant un minimum de changements aux classes existantes. Sans rentrer dans les détails d'implémentation, il vous est demandé de :

1. expliquer le principe orienté-objet utilisé pour résoudre le problème
2. lister les modifications apportées à l'interface des classes (signature et spécifications des méthodes publiques)
3. lister les modifications apportées à chacune des classes

Question 6 [/2] : Polymorphisme
Dans la classe ListeIngredients, la méthode ajouter() utilise la méthode containsKey() de la classe HashMap pour détecter si un ingrédient est déjà présent dans la liste. Imaginons que nous ayons déjà 10 doses d'un ingrédient appelé *sucre* dans la liste, et que nous voulions rajouter trois doses d'un ingrédient également appelé *sucre*.

1. Expliquez en détails comment la méthode containsKey() exploite le *polymorphisme* pour traiter ce cas de figure.

2. Quels sont les critères utilisés pour détecter que l'ingrédient que l'on tente d'ajouter est déjà dans la liste?

Si vous devez mentionner des extraits de code, indiquez le nom de la classe et la méthode correspondante.

```
/**
 * Modifie la liste d'ingredients en ajoutant la quantite indiquee a l'ingredient i.
 * Si i n'etait pas encore dans la liste, il y est ajoute
 * @param i Un objet Ingredient non nul
 * @param quantite le nombre de doses de l'ingredient a ajouter (> 0)
 */
public void ajouter(Ingredient i, int quantite){
    if(liste.containsKey(i)){
        liste.put(i, liste.get(i)+quantite);
    }
    else{
        liste.put(i, quantite);
    }
}
```

Question 7 [/2] : Structures de données
Listez toutes les structures de données utilisées par le programme, et pour chacune, expliquez pourquoi elle a été choisie en fonction de l'usage qu'il en est fait, et si ce choix est pertinent.

Question 8 [/2] : Exceptions
Imaginons que, depuis le client réseau, une commande soit passée pour des cookies à la noix de cajou, alors que le magasin n'en propose pas à la carte. Expliquez en détails ce qui va se passer au niveau de la cascade d'appels de méthodes : Qui va détecter le problème? Sous quelle forme sera-t'il relayé, étape par étape, jusqu'au client?

Bonnes pratiques de programmation (/6)

Question 9 [/2] : Documentation
Documentez la méthode retirer(Ingredient i, int quantite) de la classe ListeIngredients.

```java
/**
  *
  *
  *
  *
  *
  *
  *
  *
  *
  *
  *
  */
  public void retirer(Ingredient i, int quantite) throws StockInsuffisantException{
    if(!liste.containsKey(i) || liste.get(i) < quantite){
        throw new StockInsuffisantException();
    }
    liste.put(i, liste.get(i)-quantite);
  }
```

Question 10 [/2] : Tests
Expliquez en quoi consistent les tests Black-Box, et quelle est la différence avec les tests White-Box. Proposez ensuite un ensemble de tests Black-Box pour la méthode retirer(Ingredient i, int quantite) de la classe ListeIngredients, sous forme d'un **tableau valeurs d'entrée (paramètres ou état de l'objet/du programme) - résultat attendu** (pas de code java). Basez-vous sur votre réponse à la question précédente.

Question 11 [/2] : Qualité du code

La méthode valeurIngredients a été implémentée en dépit du bon sens. Heureusement, les spécifications ont été écrites par une autre personne, à l'esprit moins tortueux. Proposez une nouvelle implémentation de cette méthode qui respecte les critères de qualité d'un bon code source. Listez les principes de bonne programmation que vous avez appliqués.

```
/**
 * Calcule la valeur des ingredients contenus dans la liste.
 * @return La valeur des ingredients en euros
 */
public double valeurIngredients(){
    double c = 0;
for(Map.Entry<Ingredient, Integer> e : liste.entrySet()){
    Ingredient MaVariable = e.getKey();
    double o;int X = e.getValue();  //X = quantite
    o = MaVariable.getPrixUnitaire(); //o = prix
    if(X > 0){ c = c+ (X*o);
            }}return c;
}
```

Programmation réseau (/5)

Question 12 [/2] : Principes

Le logiciel de gestion de commandes de cookies est une application distribuée utilisant des sockets.
Pourquoi a-t-on choisi cette solution plutôt que des DatagramSocket? Expliquez la différence entre les deux, et expliquez pour quoi le premier choix est préférable dans cette situation.

Question 13 [/3] : Fonctionnement

Expliquez le fonctionnement du client réseau du magasin de cookies, en complétant les 5 points de commentaires.

```
public static String acheteCookies(String nomCookie, int numCookies) throws IOException{
    //1.
    //
    Socket socket = new Socket(serverIP, port);
    //2.
    //
    BufferedReader in = new BufferedReader(
                        new InputStreamReader(
                                socket.getInputStream()));
    PrintWriter out = new PrintWriter(
                      new BufferedWriter(
                              new OutputStreamWriter(
                                      socket.getOutputStream())),true);
    //3.
    //
    out.println(nomCookie);
    out.println(numCookies);
    //4.
    //
    String result = in.readLine();
    //5.
    //
    in.close();
    out.close();
    socket.close();
    return result;
}
```

Programmation concurrente (/3)

Question 14 [/3] : Parallélisation du serveur

Le magasin doit pouvoir recevoir des commandes depuis l'interface GUI et depuis le réseau. Or, par défaut, il n'y a qu'un seul *fil d'exécution* en java.

1. Quel mécanisme a été utilisé pour permettre au serveur et à la GUI de fonctionner en parallèle? Expliquez.

2. La programmation concurrente peut poser certains problèmes d'intégrité des données si elle est appliquée sans précaution. Expliquez ici quelles données pourraient potentiellement poser problème. Est-ce que le programme a été concu pour éviter ça? Si oui, qu'est ce qui a été fait, et pourquoi. Si non, que faire pour arranger ça? Indiquez précisément la modification à appliquer, et à quel(s) endroit(s) du programme.

Classes

Classe MagasinCookies

```
package cookies.magasin;

import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import java.util.Observable;
import java.util.Map.Entry;

public class MagasinCookies extends Observable{
    private String nom;
    private Map<String, RecetteCookie> recettesCookies;
    private Compte compte;
    private ListeIngredients stock;
    private ArrayList<String> ventes;
    public MagasinCookies(){

    }
    public MagasinCookies(String nom, int investissement){
        this.nom = nom;
        this.stock = new ListeIngredients();
        this.compte = Compte.getInstance(investissement);
        this.ventes = new ArrayList<String>();
        recettesCookies = new HashMap<String, RecetteCookie>();
        ajouterRecettesParDefaut();
    }
    public MagasinCookies(String nom,int investissement,
                          Map<String, RecetteCookie> recettes ){
        this.nom = nom;
        this.stock = new ListeIngredients();
        this.compte = Compte.getInstance(investissement);
        this.ventes = new ArrayList<String>();
        this.recettesCookies = recettes;
    }
    /**
     * Ajoute une recette de cookie au magasin
     * @param r : Une recette correctement initialis e avec
     * les ingr dients n cessaires.
     */
    public void ajouterRecette(RecetteCookie r){
        recettesCookies.put(r.getDescriptif(), r);
        setChanged();
        notifyObservers();
    }


    /**
     * Fabrique numCookies selon la recette donn  s'il y a
     * assez d'ingr dients en stock ou s'il est possible de compl ter le stock.
     * Le stock est modifi  en fonction des ingr dients pr lev s.
     * Le solde du compte est modifi  en fonction des ingr dients
     * qu'il a fallu achet  et du paiement des cookies fabriqu s.
     * @param nomRecette : le nom de la recette du cookie demand
     * @param numCookies : Le nombre de cookies demand  (>0)
     * @return Les cookies demand s, sous forme de tableau d'objets Cookie.
     * @throws SoldeInsuffisantException
     * @throws StockInsuffisantException
     * @throws RecetteInconnueException
     */
    public Cookie[] fabriquerCookies(String nomRecette, int numCookies)
                                                throws SoldeInsuffisantException,
                                                       StockInsuffisantException,
                                                       RecetteInconnueException {
        Cookie [] newCookies = new Cookie[numCookies];
        //1. R cup rer la recette
        RecetteCookie recette = recettesCookies.get(nomRecette);
        if(recette==null)
```

```
            throw new RecetteInconnueException();
        //2. Prendre les ingr dients en effectuant les achats n cessaires
        pr leverDansStock(recette, numCookies);
        //3. Fabriquer et vendre les cookies
        for(int i = 0; i<numCookies; i++){
            newCookies[i] = new Cookie(recette);
            vendre(newCookies[i]);
        }
        setChanged();
        notifyObservers();
        return newCookies;
    }
    public double getSolde(){...}
    public Map<String, RecetteCookie> getRecettes() {...}
    public ListeIngredients getStock(){...}
    public ArrayList<String> getVentes(){...}
    public String getNom() {...}

    /**
     * Calcule la valeur du stock courant
     * @return la somme du cout des ingr dients en stock
     */
    public double valeurStock(){
        return stock.valeurIngredients();
    }

    /**
     * Calcule le bilan financier du magasin,
     *    savoir le solde du compte et la valeur du stock
     * @return le bilan calcul
     */
    public double getBilan(){
        return getSolde() + valeurStock();
    }
    /**
     * Augmente le solde du compte du magasin
     * @param montant
     */
    public void ajouterInvestissement(int montant){
        compte.depot(montant);
        setChanged();
        notifyObservers();
    }
    /**
     * Le prix du cookie est encaiss  sur le solde du compte
     * @param c
     */
    private void vendre(Cookie c){
        compte.depot(c.prix());
        //R cup rer la date de la vente :
        Date now = Calendar.getInstance().getTime();
        SimpleDateFormat format = new SimpleDateFormat("d/M/yy hh:mm");
        ventes.add(format.format(now) + " " + c );

    }
    /**
     * @param recette : La recette sp cifiant les ingr dients    pr lever
     *          dans le stock pour un cookie
     * @param numCookies : Le nombre de cookies    produire
     * @result S'il y a suffisamment de fonds pour compl ter le stock au besoin, le stock
     * est diminu  de la quantit  d'ingr dients indiqu s dans la recette
     * @throws SoldeInsuffisantException :  S'il n'y a pas suffisamment
     * de fonds pour compl ter le stock, le stock est inchang  et l'exception est lanc e.
     * @throsw StockInsuffisantException : Cette exception n'est normalement jamais
     * lanc e puisque l' tat  du stock est v rifi  avant le pr l vement     *
     */
    private void pr leverDansStock(RecetteCookie recette, int numCookies)
                                                throws  SoldeInsuffisantException {
        //Si le stock est incomplet, on ach te les ingr dients manquants
        completerStock(recette, numCookies);
        //Le stock est suffisant, donc on pr l ve ce dont on a besoin
        for(Entry<Ingredient, Integer> entry : recette.getIngredients()){
            try{
                stock.retirer(entry.getKey(), entry.getValue()*numCookies);
```

```
            }
            catch (StockInsuffisantException e){
                System.err.println("Erreur : Stock insuffisant \
                                    malgr  v rification pr alable");
            }
        }
    }

    /**
     * Renvoie les achats    effectuer pour pouvoir effectuer la recette indiqu e
     * @param recette est non null
     * @param numCookies >0
     * @return la liste des ingr dients en rupture de stock, avec le nombre de
     *  kilos    acheter pour chacun.
     */
    private ListeIngredients ingredientsAAcheter(RecetteCookie recette, int numCookies){
        ListeIngredients ingredientsManquants = new ListeIngredients();
        for(Map.Entry<Ingredient, Integer> entry : recette.getIngredients()){
            Ingredient ingredient = entry.getKey();
            int unitesNecessaires = entry.getValue() * numCookies;
            int unitesManquantes = unitesNecessaires - stock.quantite(ingredient);

            if(unitesManquantes > 0){
                //Calculer la quantite par kilo a acheter
                int aAcheter = (int) (ingredient.getNbUniteParKilo() \
                        * Math.ceil(unitesManquantes / ingredient.getNbUniteParKilo()));
                ingredientsManquants.ajouter(ingredient, aAcheter);
            }
        }
        return ingredientsManquants;
    }
    /**
     * Ach te la quantit  d'ingr dients indiqu e dans la liste d'ingr dients manquants
     * si le solde le permet.
     * @param ingredientsManquants : La liste des ingr dients    compl ter et
     *          la quantit  demand e pour chacun
     * @post Si le solde est suffisant pour permettre de renflouer les stocks etant donn
     *  que les achats se font au kilo, apr s ex cution de la m thode, le stock contient
     *  au minimum la quantit  demand e pour chaque ingr dient
     * @throws SoldeInsuffisantException si le solde est insuffisant pour effectuer les achats.
     * Le stock est alors inchang .
     */
    private void completerStock(RecetteCookie recette, int numCookies)
                                                    throws SoldeInsuffisantException{
        ListeIngredients ingredientsAAcheter = ingredientsAAcheter(recette, numCookies);
        if(ingredientsAAcheter.valeurIngredients()>compte.solde())
            throw new SoldeInsuffisantException();

        for(Entry<Ingredient, Integer> entry : ingredientsAAcheter){
            acheterIngredientAuKilo(entry.getKey(), entry.getValue());
        }
    }

    /**
     *
     * @param i
     * @param numKilos
     * @throws SoldeInsuffisantException
     */
    private void acheterIngredientAuKilo(Ingredient i, int quantite)
                                            throws SoldeInsuffisantException{
        compte.retrait(i.getPrixUnitaire()*quantite);
        stock.ajouter(i, quantite);
    }
    private void ajouterRecettesParDefaut(){
        //A compl ter
    }


}
```

Classe ListeIngredients

```
package cookies.magasin;

import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Map.Entry;

public class ListeIngredients implements Iterable<Entry<Ingredient, Integer>> {
    Map <Ingredient, Integer> liste;

    public ListeIngredients(Map<Ingredient, Integer> liste) {
        super();
        this.liste = liste;
    }
    public ListeIngredients(){
        liste = new HashMap<Ingredient, Integer>();
    }
    /**
     * Modifie la liste d'ingr dients en ajoutant la quantit  indiqu e    l'ingr dient i.
     * Si i n' tait  pas encore dans la liste, il y est ajout
     * @param i Un objet Ingr dient non nul
     * @param quantite le nombre de doses de l'ingr dient    ajouter (> 0)
     */
    public void ajouter(Ingredient i, int quantite){
        if(liste.containsKey(i)){
            liste.put(i, liste.get(i)+quantite);
        }
        else{
            liste.put(i, quantite);
        }
    }
    /**
     *
     * A Compl ter
     */
    public void retirer(Ingredient i, int quantite) throws StockInsuffisantException{
        if(!contient(i,quantite)){
            throw new StockInsuffisantException();
        }
        liste.put(i, liste.get(i)-quantite);
    }


    /**
     * Cette m thode v rifie si le stock contient au minimum le nombre de doses
     * indiqu e en param tre pour l'ingr dient i.
     * @param i : un ingr dient
     * @param quantite : le nombre de doses d'ingr dient demand .
     * @return vrai si l'ingr dient est pr sent en quantit  suffisante, faux sinon.
     */
    public boolean contient(Ingredient i, int quantite){
        if(!liste.containsKey(i) || liste.get(i) < quantite){
            return false;
        }
        return true;
    }
    /**
     * Calcule la valeur des ingr dients contenus dans la liste.
     * @return La valeur des ingr dients en euros
     */
    public double valeurIngredients(){
        double c = 0;
for(Map.Entry<Ingredient, Integer> e : liste.entrySet()){
        Ingredient MaVariable = e.getKey();
        double o;int X = e.getValue();  //X = quantite
        o = MaVariable.getPrixUnitaire(); //o = prix
        if(X > 0){ c = c+ (X*o);
                }}return c;
    }
```

```
    /**
     * Renvoie un it rateur permettant de parcourir la liste d'ingr dients
     */
    public Iterator<Entry<Ingredient, Integer>> iterator(){
        return liste.entrySet().iterator();
    }
    /**
     * Renvoie le nombre de doses en stock pour l'ingr dient donn
     * @param ingredient
     * @return le nombre de doses en stock de ingredient
     */
    public int quantite(Ingredient ingredient) {
        if (!liste.containsKey(ingredient))
            return 0;
        return liste.get(ingredient);
    }
    public String toString(){
        String result = "";
        for(Entry<Ingredient, Integer> entry : liste.entrySet()){
            result += (entry.getKey() + " - " +  entry.getValue() + " unit s\n");
        }
        return result;
    }
}
```

Classe Ingredient

```
package cookies.magasin;

public class Ingredient {
    String name;
    double qteBaseGr;
    double prixKilo;

    public Ingredient(String name, double qteBaseGr, double prixKilo) {
        super();
        this.name = name;
        this.qteBaseGr = qteBaseGr;
        this.prixKilo = prixKilo;
    }

    public String getName() {
        return name;
    }

    public double getPrixUnitaire() {
        return (prixKilo/1000.0)*qteBaseGr;
    }
    public String toString(){
        return name;
    }
    public double getNbUniteParKilo(){
        return 1000.0/qteBaseGr;
    }

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + ((name == null) ? 0 : name.hashCode());
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Ingredient other = (Ingredient) obj;
        if (name == null) {
            if (other.name != null)
                return false;
        } else if (!name.equals(other.name))
            return false;
        return true;
    }


}
```

Classe RecetteCookie

```
package cookies.magasin;

public class RecetteCookie {
    private String descriptif;
    private ListeIngredients ingredients;

    public RecetteCookie(String descriptif) {
        super();
        this.descriptif = descriptif;
        ingredients = new ListeIngredients();
    }

    public ListeIngredients getIngredients() {
        return ingredients;
    }
    public void addIngredient(Ingredient i, int quantite){
        ingredients.ajouter(i, quantite);
    }
    public double cout(){
        return ingredients.valeurIngredients();
    }

    @Override
    public String toString() {
        return "Cookie [descriptif=" + descriptif + ", prix()=" + cout() + "]";
    }


    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result
                + ((descriptif == null) ? 0 : descriptif.hashCode());
        return result;
    }


    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        RecetteCookie other = (RecetteCookie) obj;
        if (descriptif == null) {
            if (other.descriptif != null)
                return false;
        } else if (!descriptif.equals(other.descriptif))
            return false;
        return true;
    }


    public String getDescriptif() {
        return descriptif;
    }

}
```

Classe Cookie

```
package cookies.magasin;

public class Cookie {
    public final double BENEFICE = 0.25;
    RecetteCookie recette;
    public Cookie(RecetteCookie recette) {
        this.recette = recette;
    }

    public double prix(){
        return recette.cout() * (1+BENEFICE);
    }
    public String toString(){
        return this.recette.getDescriptif() + " - " + prix();
    }
}
```

Classe Compte

```
package cookies.magasin;

public final class Compte {
    private double solde;
    private static Compte uniqueInstance  = null;

    private Compte() {
    }
    public final static Compte getInstance(double soldeInitial) {
        if (uniqueInstance == null) {
            uniqueInstance = new Compte(); }
            uniqueInstance.solde = soldeInitial;
            return uniqueInstance;
    }

    public void retrait(double montant) throws SoldeInsuffisantException{
        if(solde >= montant){
            solde = solde - montant;
        }
        else {
            throw new SoldeInsuffisantException();
        }

    }

    public void depot(double montant){
        solde = solde + montant;
    }

    public double solde() {
        return solde;
    }

    public String toString(){
        return "Solde du compte : " + solde;
    }
}
```

Classe MagasinCookiesMain

```
package cookies.main;

import cookies.controleur.CookieGUIControleur;
import cookies.gui.MagasinFrame;
import cookies.magasin.MagasinCookies;
import cookies.netServer.CookieServer;

public class MagasinCookiesMain {
    public static void main(String [] args){

        MagasinCookies monMagasin = new MagasinCookies("CookieFactory", 1000);
        CookieGUIControleur ctrl = new CookieGUIControleur(monMagasin);
        MagasinFrame gui =  new MagasinFrame(monMagasin, ctrl);
        ctrl.addGUI(gui);
        CookieServer server = new CookieServer(monMagasin);
        Thread t = new Thread(server);
        t.start();
    }
}
```

Classe CookieGUIControleur

```
package cookies.controleur;

import java.util.Map;
import javax.swing.JTextField;
import cookies.gui.MagasinFrame;
import cookies.magasin.Cookie;
import cookies.magasin.MagasinCookies;
import cookies.magasin.RecetteInconnueException;
import cookies.magasin.SoldeInsuffisantException;
import cookies.magasin.StockInsuffisantException;


public class CookieGUIControleur {
    MagasinCookies magasin;
    MagasinFrame gui;
    public CookieGUIControleur(MagasinCookies magasin) {
        this.magasin = magasin;
    }
    public void commande(Map<String, JTextField> commande){
        double aPayer = 0;
        for(Map.Entry<String, JTextField> entry : commande.entrySet()){
            String typeCookie = entry.getKey();
            int numCookies = Integer.parseInt(entry.getValue().getText());
            try {
                if(numCookies>0){
                    Cookie[] cookies = magasin.fabriquerCookies(typeCookie, numCookies);
                    for(Cookie c : cookies){
                        aPayer += c.prix();
                    }
                }
            } catch (SoldeInsuffisantException e) {
                gui.getPanelVente().
                    afficheResultat("Erreur : Stock insuffisant pour "+typeCookie);
            } catch (StockInsuffisantException e) {
                gui.getPanelVente().afficheResultat("Erreur : Stock insuffisant");
            } catch (RecetteInconnueException e) {
                gui.getPanelVente().afficheResultat("Erreur : Recette inconnue");
            }
            entry.getValue().setText("0");
        }
        gui.getPanelVente().afficheResultat("Montant de la transaction : "+aPayer + " euros");

    }
    public void addGUI(MagasinFrame gui) {
        this.gui = gui;
    }
}
```

Classe MagasinFrame

```
package cookies.gui;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTabbedPane;

import cookies.controleur.CookieGUIControleur;
import cookies.magasin.MagasinCookies;

public class MagasinFrame extends JFrame  {
    PanelVente panelVente;
    PanelStock panelStock;
    PanelComptes panelCompte;
    public MagasinFrame(MagasinCookies m, CookieGUIControleur ctrl ) {
        super(m.getNom());

        // Parametrisation de la fenetre
        setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        setSize (500, 600);
        panelVente = new PanelVente(m, ctrl);
        panelStock = new PanelStock(m, ctrl);
        panelCompte = new PanelComptes(m, ctrl);

        JTabbedPane tabbedPane = new JTabbedPane();
        tabbedPane.addTab("Vente", panelVente);
        tabbedPane.addTab("Stock", panelStock);
        tabbedPane.addTab("Comptes", panelCompte);

        // On place la boite principale sur la fenetre
        getContentPane().add(tabbedPane);
        setVisible(true);
    }

    public PanelVente getPanelVente() {
        return panelVente;
    }

}
```

Classe PanelComptes

```
package cookies.gui;

import java.awt.Color;
import java.awt.Dimension;
import java.util.ArrayList;
import java.util.Observable;
import java.util.Observer;

import javax.swing.Box;
import javax.swing.JLabel;
import javax.swing.JPanel;

import cookies.controleur.CookieGUIControleur;
import cookies.magasin.MagasinCookies;

public class PanelComptes extends JPanel implements Observer{
    MagasinCookies magasin;
    CookieGUIControleur controleur;
    Box verticalBox;
    public PanelComptes(MagasinCookies magasin, CookieGUIControleur controleur) {
        this.magasin = magasin;
        this.controleur = controleur;
        magasin.addObserver(this);
        draw();
    }

    private void draw(){
        this.removeAll();
        verticalBox = Box.createVerticalBox();
        JLabel entete = new JLabel("Bilan financier");
        entete.setAlignmentX(CENTER_ALIGNMENT);

        verticalBox.add(entete);
        verticalBox.add(Box.createRigidArea(new Dimension(20, 20)));

        Box ligne = Box.createHorizontalBox();
        ligne.add(new JLabel("Valeur du stock : "));
        ligne.add(new JLabel(magasin.valeurStock() + " euros"));
        verticalBox.add(ligne);

        ligne = Box.createHorizontalBox();
        ligne.add(new JLabel("Solde du compte : "));
        ligne.add(new JLabel(magasin.getSolde() + " euros"));
        verticalBox.add(ligne);

        ligne = Box.createHorizontalBox();
        ligne.add(new JLabel("Bilan : "));
        ligne.add(new JLabel(magasin.getBilan() + " euros"));
        verticalBox.add(ligne);

        verticalBox.add(getVentes());

        this.add(verticalBox);
        this.setBackground(Color.white);
    }
    private Box getVentes(){
        ArrayList<String> ventes = magasin.getVentes();
        Box tabVentes = Box.createVerticalBox();
        for(String vente : ventes){
            tabVentes.add(new JLabel(vente));
        }
        return tabVentes;
    }

    @Override
    public void update(Observable o, Object arg) {
        draw();
    }
}
```

Classe PanelVente

```
package cookies.gui;

[import caches pour alleger l affichage]

public class PanelVente extends JPanel implements ActionListener{
    MagasinCookies magasin;
    Map<String, JTextField> choixCookie;
    JLabel resultat;
    JButton okButton;
    JButton cancelButton;
    CookieGUIControleur controleur;
    public PanelVente(MagasinCookies magasin, CookieGUIControleur controleur) {

        this.magasin = magasin;
        this.controleur = controleur;

        choixCookie = new HashMap<String, JTextField>();
        Box verticalBox = Box.createVerticalBox();
        JLabel entete = new JLabel("Choisissez vos cookies :");
        entete.setAlignmentX(CENTER_ALIGNMENT);
        verticalBox.add(entete);
        verticalBox.add(Box.createRigidArea(new Dimension(20, 20)));

        ajouterTableauChoix(verticalBox);

        verticalBox.add(Box.createRigidArea(new Dimension(20, 20)));
        resultat = new JLabel();
        verticalBox.add(resultat);
        verticalBox.add(Box.createRigidArea(new Dimension(20, 20)));
        okButton = new JButton("Commander");
        okButton.addActionListener(this);
        cancelButton = new JButton("Reset");
        cancelButton.addActionListener(this);
        Box ligneBoutons = Box.createHorizontalBox();
        ligneBoutons.add(okButton);
        ligneBoutons.add(cancelButton);
        verticalBox.add(ligneBoutons);
        this.add(verticalBox);
        this.setBackground(Color.white);
    }
    public void ajouterTableauChoix(Container c){
        for(Map.Entry<String, RecetteCookie> entry : magasin.getRecettes().entrySet()){
            String nomRecette = entry.getKey();
            Box ligne = Box.createHorizontalBox();
            JTextField field = new JTextField("0",3);
            choixCookie.put(nomRecette, field);
            field.setMaximumSize(new Dimension(30, 30));
            field.setAlignmentX(RIGHT_ALIGNMENT);
            ligne.add(new JLabel(nomRecette + " : "));
            ligne.add(Box.createHorizontalGlue());
            ligne.add(field);
            ligne.setBorder(BorderFactory.createLineBorder(Color.GRAY));
            c.add(ligne);
        }
    }
    public void actionPerformed (ActionEvent event)
    {
        if(event.getSource()==okButton){
            controleur.commande(choixCookie);
        }
        if(event.getSource()==cancelButton){
            for(Map.Entry<String, JTextField> entry : choixCookie.entrySet()){
                entry.getValue().setText("0");
            }
        }
    }
    public void afficheResultat(String string) {
        resultat.setText(string);
    }
}
```

Classe PanelStock

```
package cookies.gui;

import java.awt.Color;
import java.awt.Container;
import java.awt.Dimension;
import java.util.Map;
import java.util.Observable;
import java.util.Observer;

import javax.swing.BorderFactory;
import javax.swing.Box;
import javax.swing.JLabel;
import javax.swing.JPanel;

import cookies.controleur.CookieGUIControleur;
import cookies.magasin.Ingredient;
import cookies.magasin.MagasinCookies;

public class PanelStock extends JPanel implements Observer{
    MagasinCookies magasin;
    CookieGUIControleur controleur;
    Box verticalBox;
    public PanelStock(MagasinCookies magasin, CookieGUIControleur controleur) {
        this.magasin = magasin;
        this.controleur = controleur;
        magasin.addObserver(this);
        draw();
    }
    public void draw(){
        this.removeAll();
        Box verticalBox = Box.createVerticalBox();
        JLabel entete = new JLabel("Etat des stocks");

        entete.setAlignmentX(CENTER_ALIGNMENT);

        verticalBox.add(entete);
        verticalBox.add(Box.createRigidArea(new Dimension(20, 20)));

        ajouterTableauStock(verticalBox);

        verticalBox.add(Box.createRigidArea(new Dimension(20, 20)));
        JLabel valeur = new JLabel();
        valeur.setText(magasin.valeurStock() + " euros");
        verticalBox.add(valeur);
        verticalBox.add(Box.createRigidArea(new Dimension(20, 20)));
        this.add(verticalBox);
        this.setBackground(Color.white);
    }

    private void ajouterTableauStock(Container c){
        for(Map.Entry<Ingredient, Integer> entry : magasin.getStock()){
            Ingredient ing = entry.getKey();
            int nbUnites = entry.getValue();
            Box ligne = Box.createHorizontalBox();
            ligne.add(new JLabel(ing.getName()+ " - "));
            ligne.add(new JLabel(" "+ing.getPrixUnitaire()+ " euros/dose - "));
            ligne.add(new JLabel(""+nbUnites + " doses en stock"));

            c.add(ligne);
        }
    }

    @Override
    public void update(Observable o, Object arg) {
        draw();

    }


}
```

Classe CookieServer

```
package cookies.netServer;

[Import caches pour alleger l affichage]

public class CookieServer implements Runnable{
    boolean stop = false;
    MagasinCookies mag;

    public CookieServer(MagasinCookies mag){
        this.mag = mag;
    }
    public void stop(){
        this.stop=true;
    }
    public void run(){
        try{
            Socket soc;
            ServerSocket s = new ServerSocket(12345);

            //MagasinCookies mag = new MagasinCookies("Cookie's World", 1000);
            while(!stop){
                //Arrivee d'une requete : Ouverture de la connexion et des flux de communication
                soc = s.accept();
                BufferedReader in = new BufferedReader(new InputStreamReader(
                                                        soc.getInputStream()));
                PrintWriter out = new PrintWriter(
                        new BufferedWriter(
                                new OutputStreamWriter(soc.getOutputStream())), true);
                //Traitement de la requete
                String nomRecette = in.readLine();
                if(nomRecette == "QUIT"){
                    break;
                }
                int numCookies = Integer.parseInt(in.readLine());
                Cookie[] cookies;
                String result = "";
                try {
                    cookies = mag.fabriquerCookies(nomRecette, numCookies);
                    double prix = 0;
                    for(Cookie c : cookies){
                        prix += c.prix();
                        result += (c+" - ");
                    }
                    out.print("Voici les cookies demand s , la facture est de "+prix
                                + " euros. Bon app tit! (" + result+ ")");

                } catch (SoldeInsuffisantException e) {
                    out.println("Erreur : stock insuffisant pour la transaction.");
                } catch (RecetteInconnueException e){
                    out.println("Erreur : La recette n'est pas    la carte du magasin");
                } catch (StockInsuffisantException e) {
                    e.printStackTrace();
                }
            }
            s.close();
        }
        catch(IOException e)
        {
            System.out.println("Erreur IO : Le serveur a crash ");
        }
    }
}
```

Classe CookieClient

```java
package cookies.netClient;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.Socket;

public class cookieClient {
    static String serverIP;
    static int port;
    /**
     * Cette methode effectue une commande de cookies au magasin de cookie
     * @param nomCookie : Le nom du cookie command
     * @param numCookies : Le nombre de cookies de type nomCookie command s
     * @return le resultat de la commande
     * @throws IOException en cas d'erreur lors de la communication
     */
    public static String acheteCookies(String nomCookie, int numCookies) throws IOException{
        //1.
        //
        Socket socket = new Socket(serverIP, port);
        //2.
        //
        BufferedReader in = new BufferedReader(
                                new InputStreamReader(
                                    socket.getInputStream()));
        PrintWriter out = new PrintWriter(
                              new BufferedWriter(
                                      new OutputStreamWriter(
                                              socket.getOutputStream())),true);
        //3.
        //
        out.println(nomCookie);
        out.println(numCookies);
        //4.
        //
        String result = in.readLine();
        //5.
        //
        in.close();
        out.close();
        socket.close();
        return result;
    }
    public static void main(String[] args) {
        serverIP="127.0.0.1";
        port = 12345;
        try {
            System.out.println(acheteCookies("Cookie au cacao", 2));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```