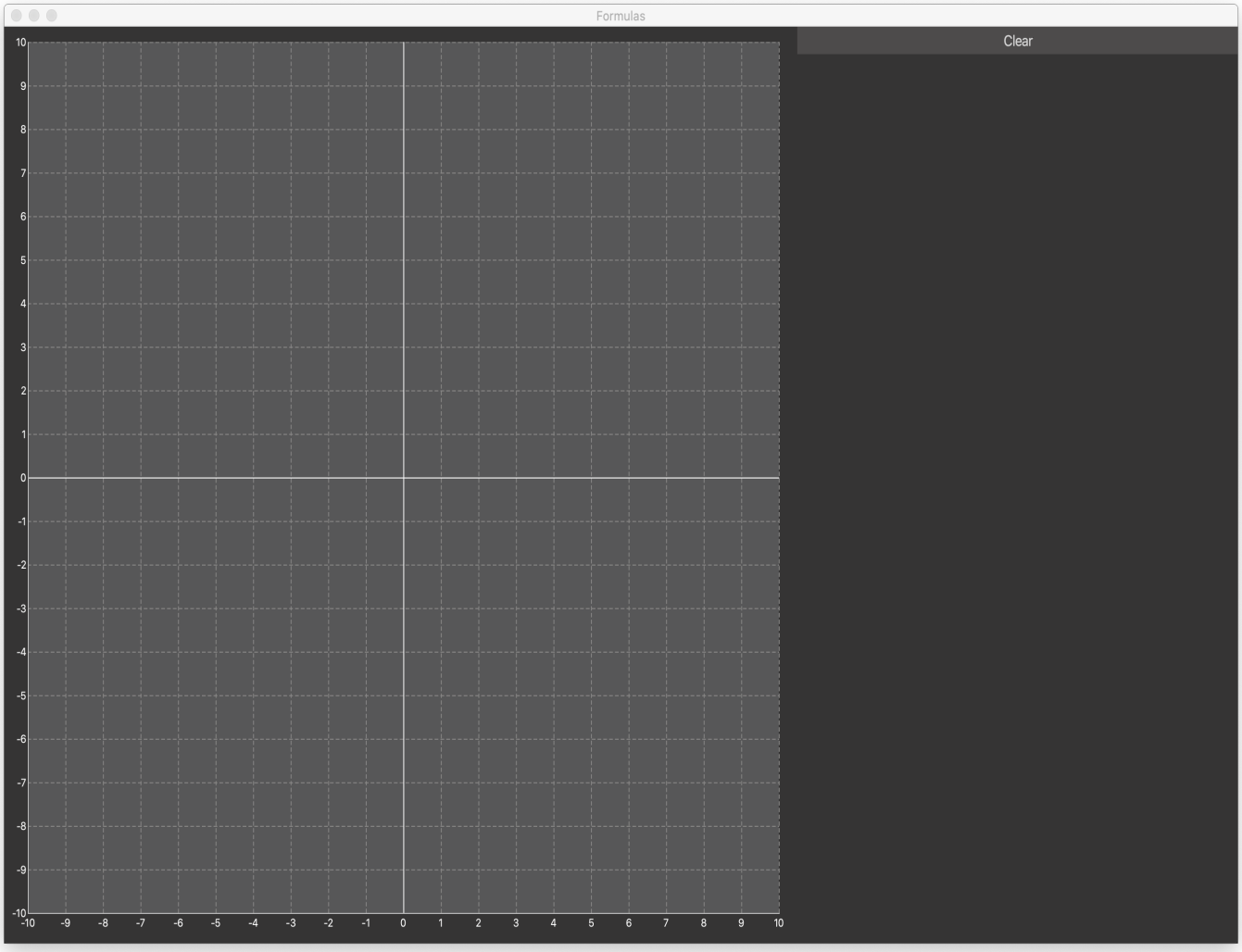
1 Formules

1.1 Consignes pour démarrer le TP

Comme pour les TP précédents, on va utiliser git pour la gestion de versions. Il vous faut donc vous reporter aux consignes du premier TP.

Une fois le dépôt téléchargé, vous pouvez compiler et exécuter le programme en cliquant deux fois sur `formula-***` -> `application` -> `run` (en n'oubliant pas d'ajouter votre projet en tant que projet *Gradle*). Vous devriez obtenir l'affichage suivant.



Lien vers la classroom du TP : lien.

1.2 Objectif

Dans cet exercice, vous allez implémenter des classes pour générer des formules mathématiques. Chaque classe correspondra à un type de formule (constantes, variable, addition, multiplication, ...).

Chaque classe devra implémenter l'interface Formula suivante :

```
public interface Formula {

  /**
   * Compute the value of the formula
   *
   * @param xValue the value of the variable x
   * @return the value of the function when the variable x has value {@code xValue}
   */
  double eval(double xValue);

  /**
   * Compute a {@code String} representation of the formula.
   * @return the formula as a {@code String}
   */
  String toString();

  /**
   * Compute the derivative of the formula.
   * @return the derivative of the formula
   */
  Formula derivative();
}
```

Une classe implémentant `Formula` devra donc avoir trois fonctionnalités :

— le calcul de sa valeur étant donnée une valeur pour la variable x : méthode `eval`,
— la représentation en chaîne de caractères de la formule : méthode `toString`,
— le calcul de sa dérivée sous la forme d'une autre formule : méthode `derivative`.

1.3 Constante

1.3.1 Le contrat

Vous allez commencer par définir une classe `Constant` représentant une constante. Cette classe permettra de construire une formule correspondant à une constante. Cette classe implémentera l'interface `Formula` et devra contenir :

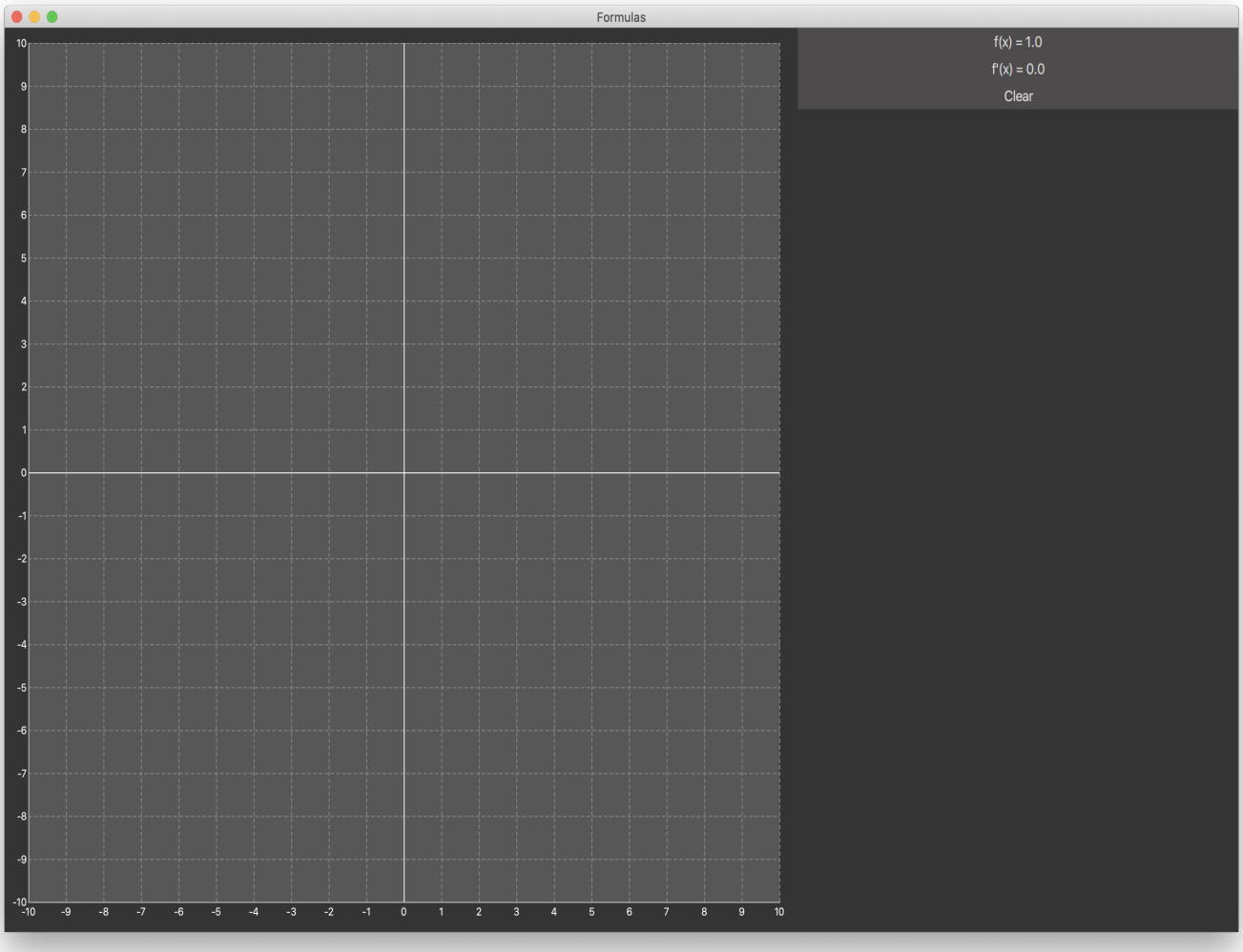
— un constructeur `public Constant(double value)` permettant de créer une constante avec une certaine valeur,
— une méthode `public double eval(double xValue)` qui devra toujours retourner la valeur de la constante,
— une méthode `public String toString()` qui devra retourner la chaîne de caractères correspondant à la constante,
— une méthode `public Formula derivative()` qui devra retourner une formule qui est la dérivée de la constante (indice, pour n'importe quelle fonction $f(x) = c$ avec c constante, la dérivée de la fonction f est $f'(x) = 0$).

1.3.2 Le test

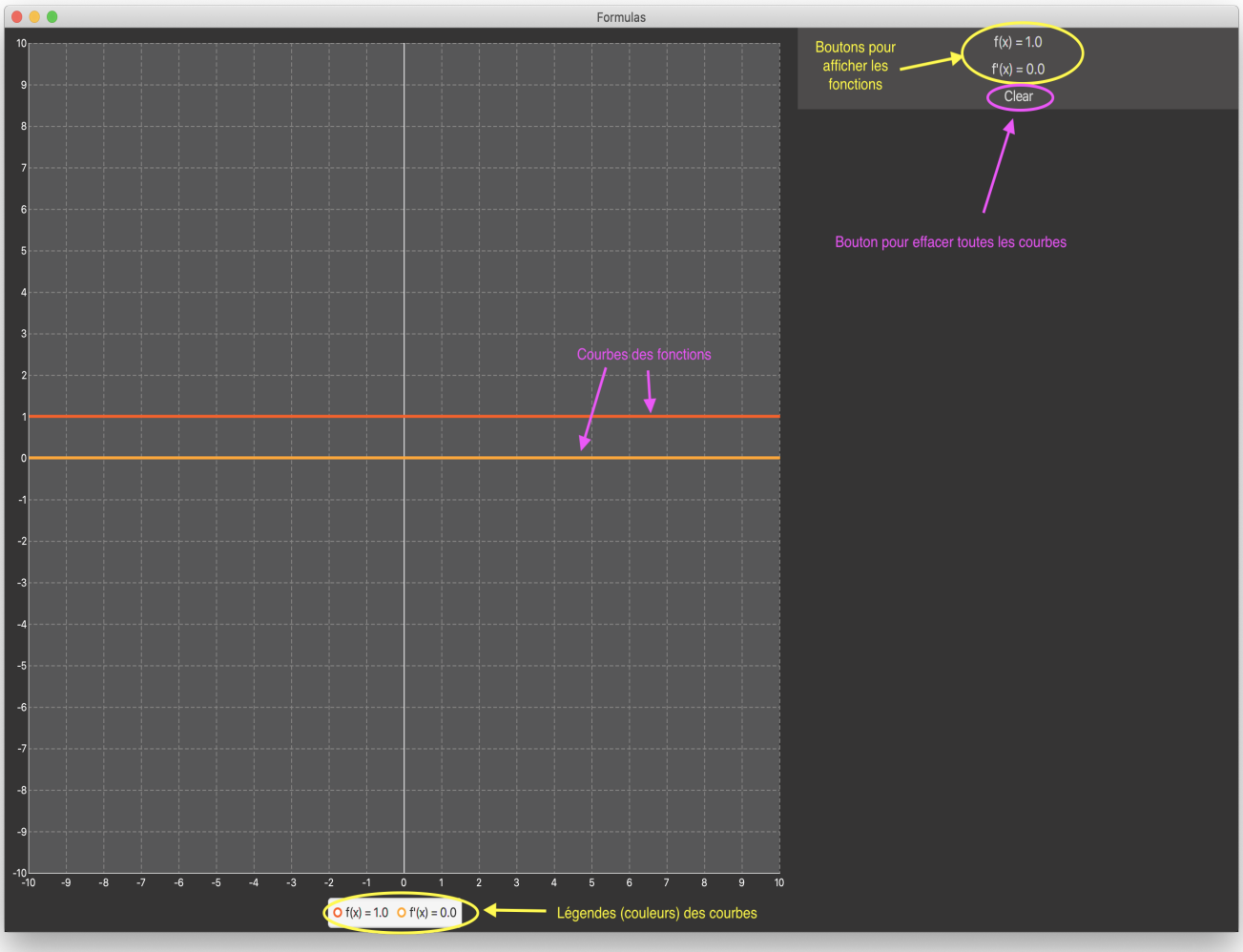
Pour tester votre classe constante, vous aller créer une fonction constante dans le logiciel de dessin de fonctions. Pour cela, il vous faut modifier le constructeur de la classe `viewer.FunctionList` pour rajouter les deux lignes suivantes à la place du `//TODO` :

```
PlottableFunction function = new PlottableFunction(new Constant(1), "f");
addFunctionAndItsDerivative(function);
```

Ces deux lignes permettent respectivement :
— de créer une fonction f dont la formule est une constante égale à 1 et ayant comme nom f,
— puis de rajouter la fonction f et sa dérivée f'.
Vous devriez obtenir l'affichage suivant.



Pour afficher les fonctions, il suffit de cliquer sur les boutons correspondant. Vous devriez obtenir l'affichage suivant :



1.4 Variable x

1.4.1 Le contrat

Vous allez maintenant définir une classe `VariableX` représentant une variable x. Cette classe permettra de construire une formule correspondant à la variable x.

Cette classe implémentera l'interface `Formula` et devra contenir :

— un constructeur `public VariableX()` permettant de créer une variable,

— une méthode `public double eval(double xValue)` qui devra toujours retourner la valeur `xValue`,

— une méthode `public String toString()` qui devra retourner la chaîne de caractères correspondant à la variable x et donc la chaîne de caractère `"x"`,

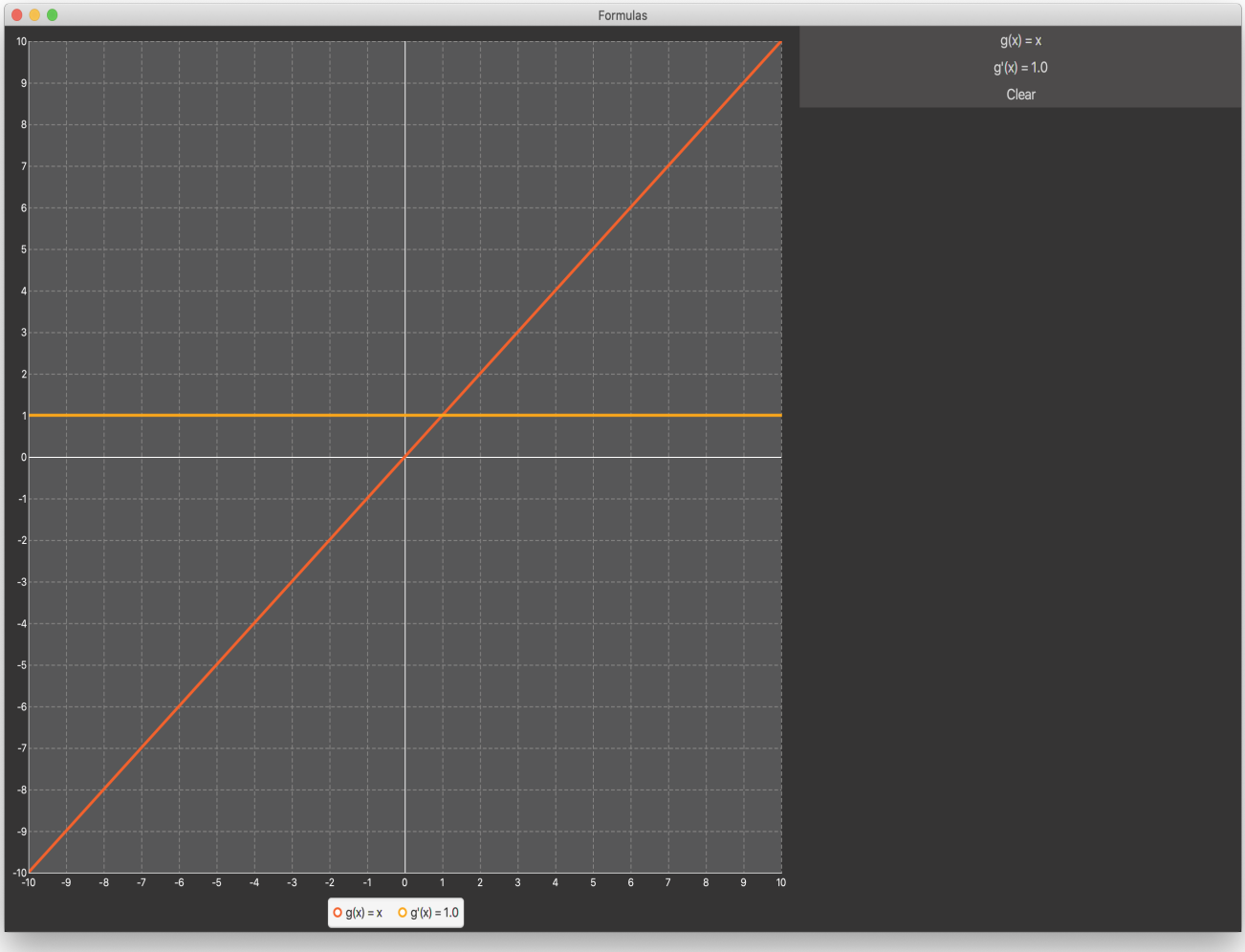
— une méthode `public Formula derivative()` qui devra retourner une formule qui est la dérivée en x de la variable x (indice, pour une fonction $g(x) = x$, la dérivée de la fonction g est $g'(x) = 1$).

1.4.2 Le test

Pour tester votre classe variable, vous aller créer une fonction égale à x dans le logiciel de dessin de fonctions. Pour cela, il vous faut modifier le constructeur de la classe `viewer.FunctionList` pour rajouter les deux lignes suivantes à la place du `//TODO` :

```
PlottableFunction function = new PlottableFunction(new VariableX(), "g");
addFunctionAndItsDerivative(function);
```

Ces deux lignes permettent respectivement :
— de créer une fonction dont la formule est égale à x et ayant comme nom g,
— puis de rajouter la fonction g et sa dérivée g'.
Vous devriez obtenir l'affichage suivant.



1.5 Addition

1.5.1 Le contrat

Vous allez maintenant définir une classe `Addition` représentant une addition de deux formules. Cette classe permettra de construire une formule correspondant à la somme de 2 formules.

Cette classe implémentera l'interface `Formula` et devra contenir :
— un constructeur `public Addition(Formula leftMember, Formula rightMember)` permettant de créer une addition des deux formules données en arguments correspondants aux deux membres sommés,
— une méthode `public double eval(double xValue)` qui devra toujours retourner la sommes des valeurs des deux membres de l'addition,

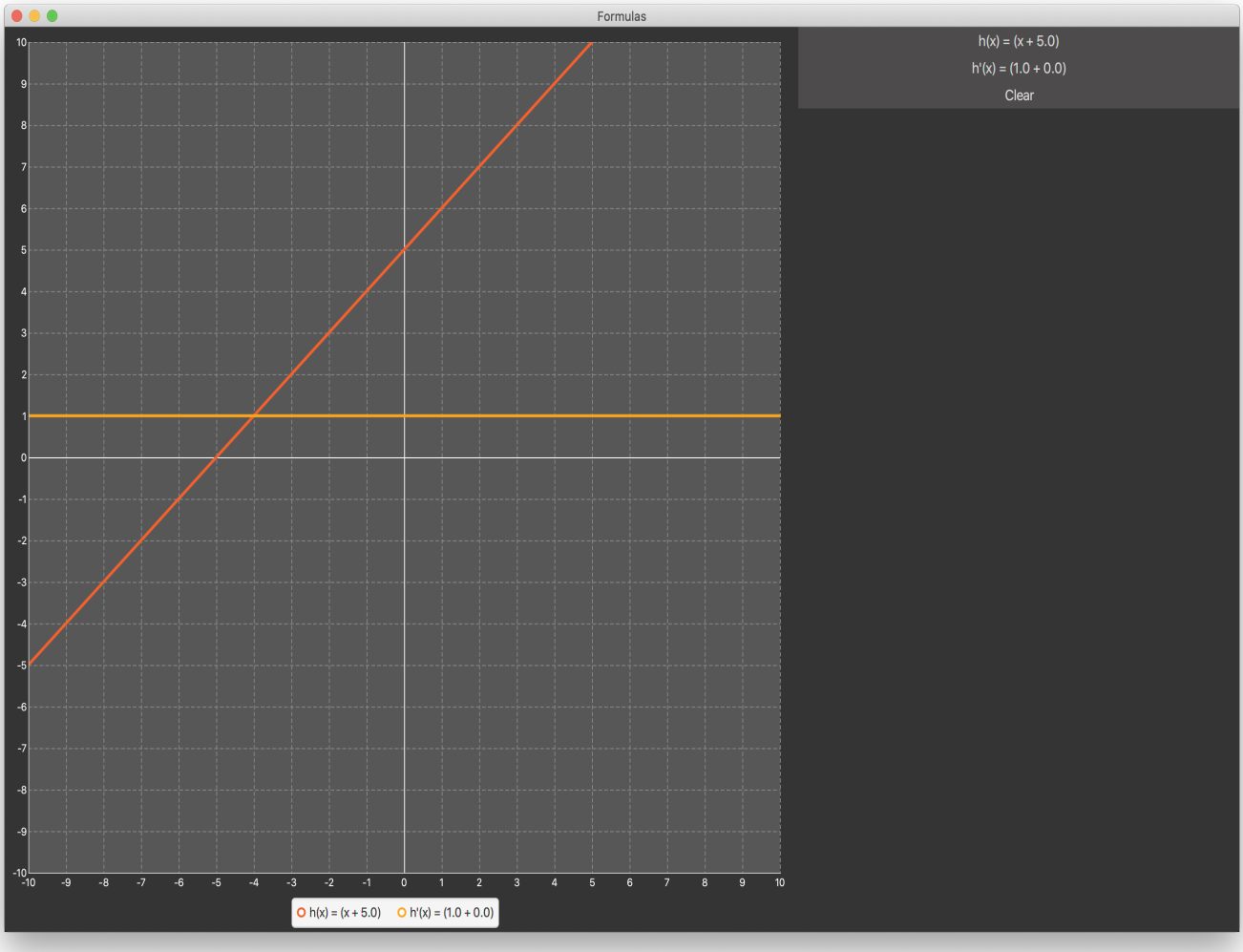
— une méthode `public String toString()` qui devra retourner la chaîne de caractères correspondant à l'addition, soit la chaîne de caractères correspondant au membre de gauche, concaténée avec le symbole `+` puis concaténé au membre de droite,
— une méthode `public Formula derivative()` qui devra retourner une formule qui est la dérivée en x de la variable x (indice, pour une fonction $h(x) = f(x) + g(x)$, la dérivée de la fonction h est $h'(x) = f'(x) + g'(x)$).

1.5.2 Le test

Pour tester votre classe addition, vous aller créer une fonction égale à $x + 5$ dans le logiciel de dessin de fonctions. Pour cela, il vous faut modifier le constructeur de la classe `viewer.FunctionList` pour rajouter les deux lignes suivantes à la place du `//TODO` :

```
PlottableFunction function = new PlottableFunction(new Addition(new VariableX(), new Constant(5)), "
addFunctionAndItsDerivative(function);
```

Ces deux lignes permettent respectivement :
— de créer une fonction dont la formule est égale à $x +5 $ et ayant comme nom h,
— puis de rajouter la fonction h et sa dérivée h'.
Vous devriez obtenir l'affichage suivant.



1.6 Autres opérations

Écrivez les classes suivantes qui implémente l'interface `Formule` (n'oubliez pas de les tester) :
— `Multiplication` (multiplication de deux formules),
— `Division` (division d'une formule par une autre),
— `Subtraction` (soustraction d'une formule par une autre),
— `Opposite` (opposé d'une formule),
— `Cosine` (cosinus d'une formule),
— `Sine` (sinus d'une formule),
— `Exponential` (fonction exponentielle d'une formule, c'est-à-dire, e puissance la formule) et
— `Logarithm` (fonction logarithme naturel d'une formule).

Attention !

Certaines classes que vous avez écrites ont beaucoup en commun. Comme nous l'avons vu dans le cours, la répétition est quelque chose qu'un bon programmeur essaye d'éviter. Dès que vous repérez une répétition, utilisez les techniques vues en cours (délégation, utilisation d'interfaces, classes abstraites et extension) pour l'éviter au maximum.

1.7 Tâches optionnelles

— Rajouter dans l'interface `Formula` une méthode `boolean isConstant()` qui retourne vrai si la formule correspond à une constante (Un instance de `Constant` est constante et tout opérations utilisant que des constantes est aussi constante).
— Rajouter dans l'interface `Formula` une méthode `Formula simplifiedFormula()` qui retourne une formule simplifiée qui correspond à la formule de départ. L'idée est de rendre la formule plus simple en remplaçant entre autre une somme de deux constantes par la constante égal à la somme. Par exemple, la formule simplifiée de `new Addition(new Addition(new Constant(2), new Addition(new Constant(2)), new VariableX())` sera `new Addition(new Constant(4), new VariableX())`. Vous pouvez aussi imaginer d'autres simplifications en prenant en compte les éléments neutres et absorbant des opérations, les règles de distributivité et d'associativité,...