アプリケーション名と概要

アプリケーション名：AlMetry（アルメトリー）

このアプリケーションは、数式を入力することでその数式に対応するグラフを描画し、そのグラフを png 形式で保存することができる。数式を媒介変数表示で$x(t)=\{t\text{の式}\}, y(t)=\{t\text{の式}\}$のようにテキストフィールドに入力し、「描画」ボタンを押下することですることで、その数式の文字列から数学的な意味での関数を生成し、グラフを描画する。描画の際にグラフの軸のスケールを変更することもできる。また、「保存」ボタンを押下することで描画したグラフとそのグラフの媒介変数表示の式を png 形式の画像ファイルとして出力でき、そのファイル名は「filename」のテキストフィールドで指定できる。このアプリケーションで使用できる定数、演算子、関数を表 1 に示す。なお、テキストフィールドに入力された半角スペース、全角スペースは無視される。

表 1 ：使用できる定数、演算子、関数

テキストフィールド上での表記	一般的な表記、意味	使用例
+	+、足し算を表す演算子	t+1 t+sin(t)
*	×、掛け算を表す演算子 ※ 2*t などを 2t と省略して書くことが可能	3*t 2*log(t)
^	累乗を表す演算子	t^3+t^2 sin(t)^2
-	-、引き算を表す演算子 ※ -1*t を省略して-t と書くことが可能	(t-1)^2 1-cos(t) -e^t
sin()	sin()、正弦関数 ※()は必須	sin(t+2) sin(2*pi*t)
cos()	cos()、余弦関数 ※()は必須	cos(t^2) 3cos(2pi*t)
tan()	tan()、正接関数	tan(t)^(-1)
log()	log()、自然対数関数	log(t) -log(2t)
pi	π、円周率を表す定数	5sin(2*pi*t) 4cos(2pi*t)
e	e、ネイピア数（自然対数の底）を表す定数	e^t log(2e)
()	括弧、合成関数などを作成するときに使用	sin(t*(t^2+1)) (t^2+1)^(-1)

グラフの軸のスケールを変更するには、X scale（x 軸のスケール）、Y scale（y 軸のスケール）をコンボボックスで指定する。スケールとは、グラフのグリッド線 1 目盛り分の数値であり 1、5、10、50、100、1000 から選ぶことができる。図 1 、2 、3 にこのアプリケーションを用いて描画したグラフの例を示す。図 1 は x=3cos(t)、y=3sin(2t)のグラフである。このように、軸は緑色、グラフは水色で描画され、目盛りには数値がふられる。また、右下にはそのグラフを表す式が描画される。図 2 のように x＝ t とすれば、y を x の関数のように扱うこと

ができる。また、図３のようにカージオイドなどの有名な曲線を描くことができる。

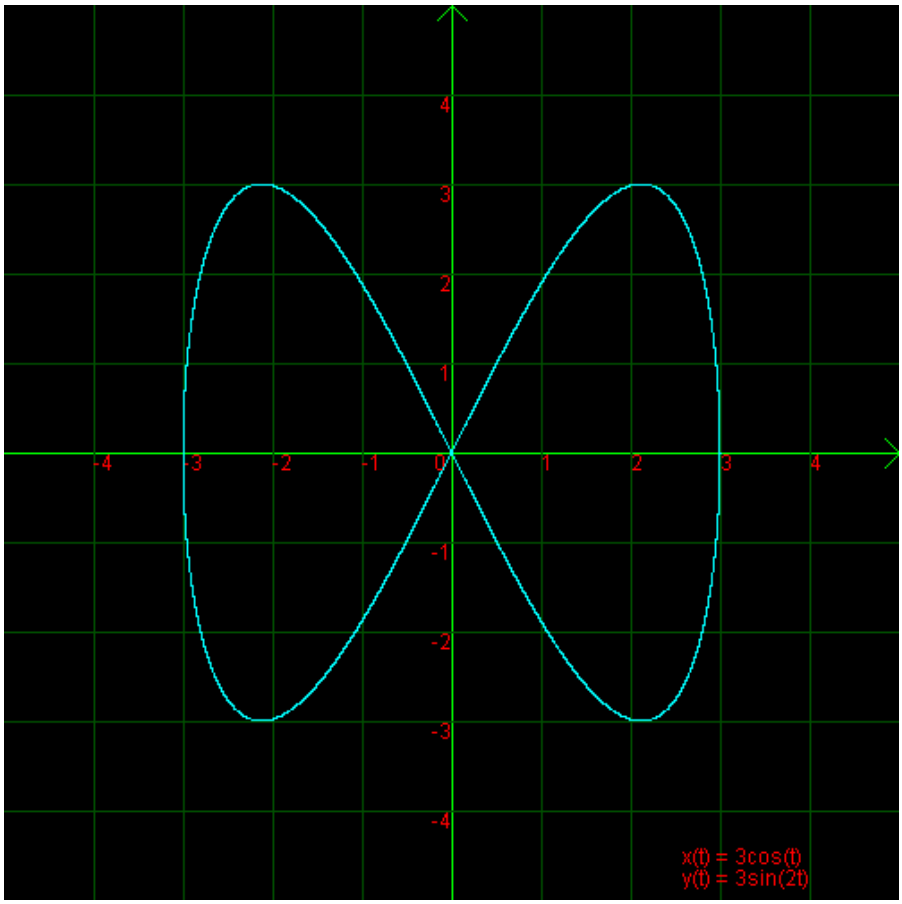


図１：x=3cos(t)、y=3sin(2t)のグラフ

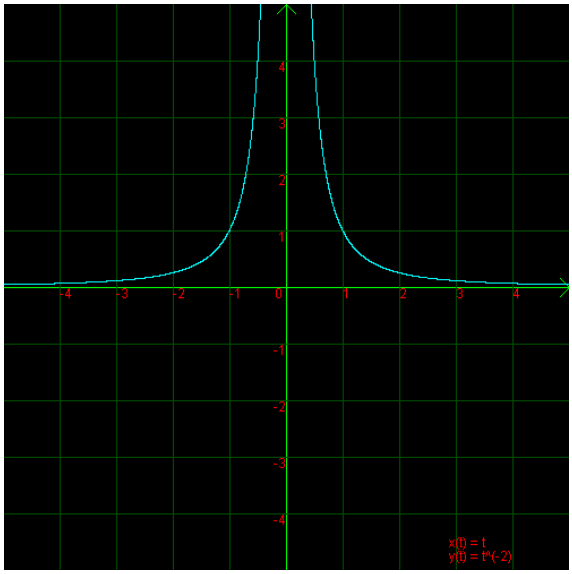


図２：x=t、y=t^(-2)のグラフ

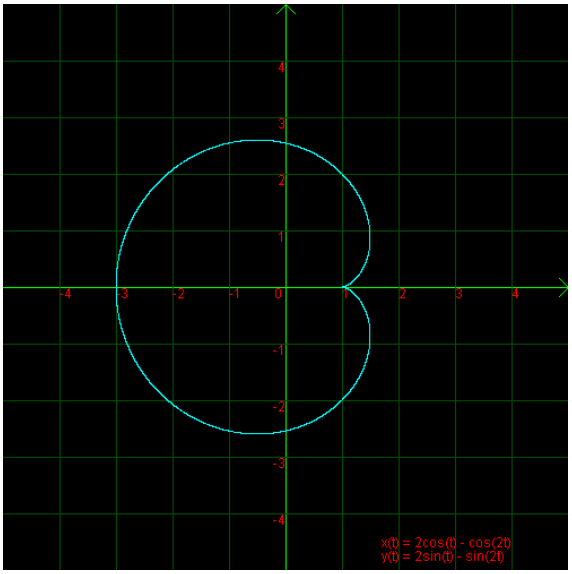
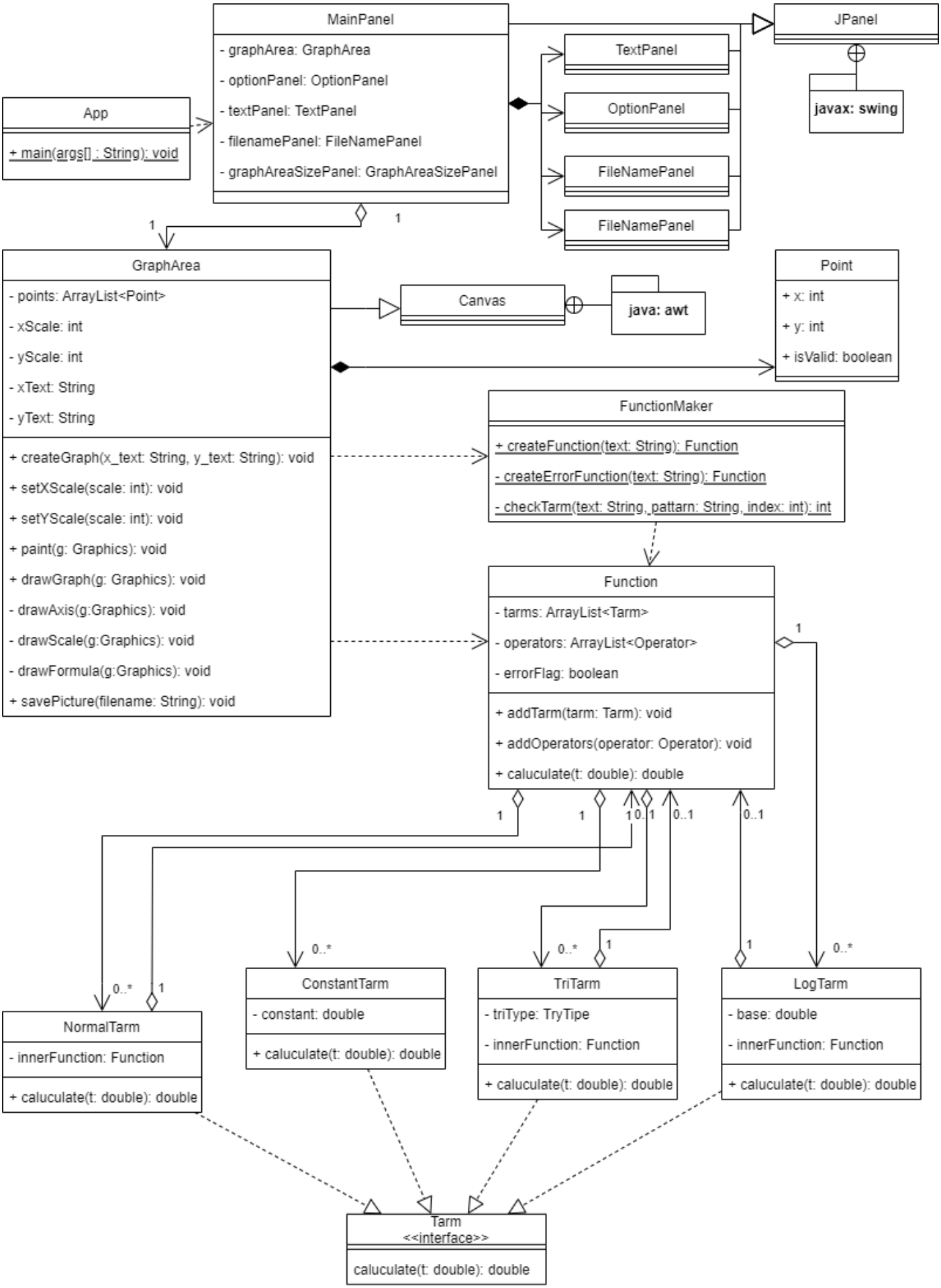
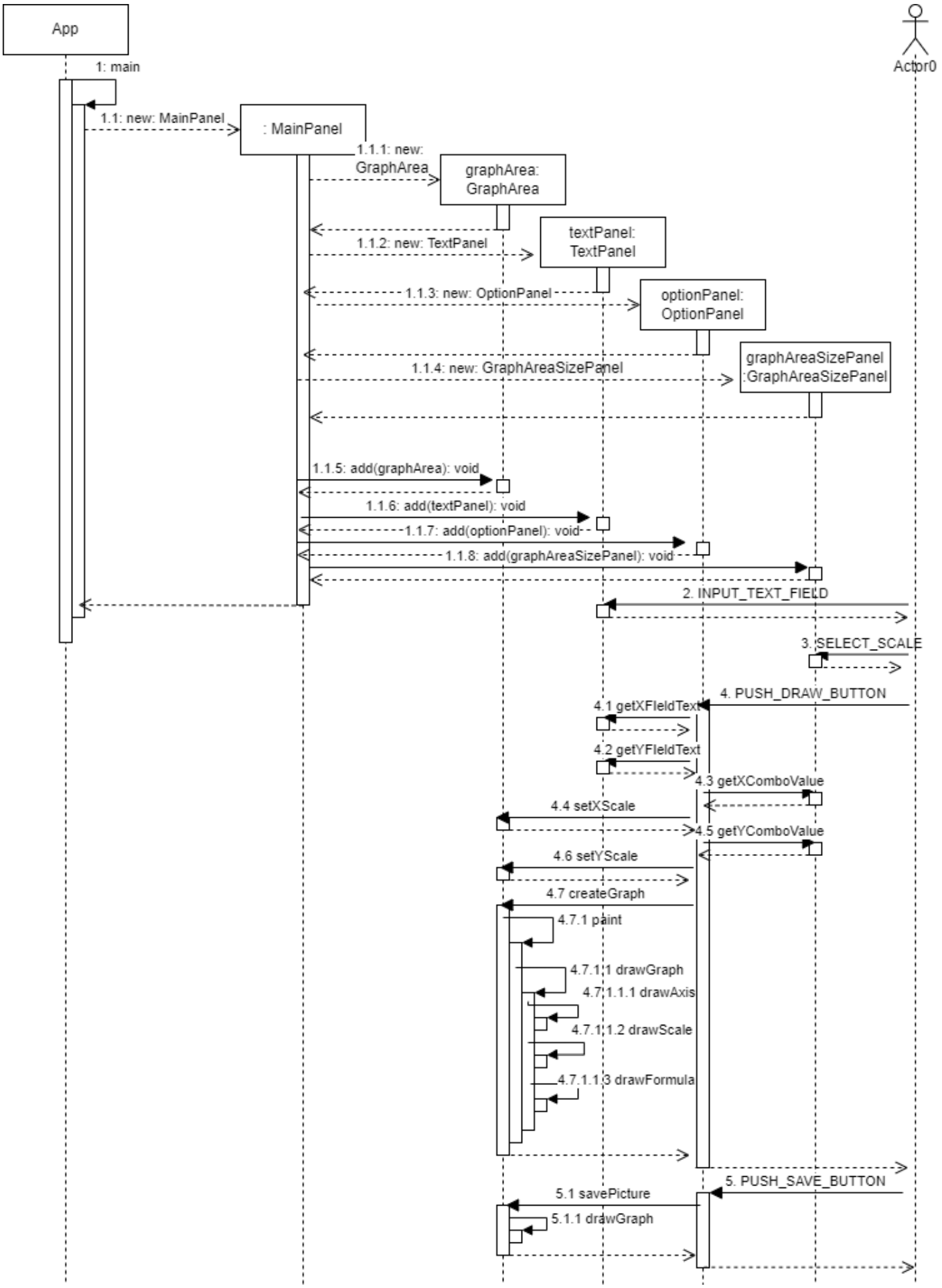


図３：x= 2cos(t) - cos(2t)、y= 2sin(t) - sin(2t)のグラフ

アプリケーションのクラス図



アプリケーションのシーケンス図



プログラムの要件をどのように満たしたか

1．マウスでボタンをクリックしたり、テキストフィールドにキーボードで文字を入力したりする。
2．paint メソッドをオーバーライドし、線や文字を描画することでグラフを描画する。
3．グラフの軸の目盛りに値が表示される。グラフに式が表示される。
4．描画ボタンと保存ボタンを使用する。

1～4 以外の機能

・コンボボックスで軸のスケールを変更できる
・描画したグラフを画像ファイルとして保存できる

ソースコードの解説と工夫した点

このアプリケーションは4つの java ファイル“App.java”、“GraphArea.java”、“FunctionMaker.java”、“Terms.java”で構成される。App.java は main メソッドとパネルやボタンなどのコンポーネントの配置などの処理がまとめられている。GraphArea.java はグラフの描画に関する処理がまとめられている。FunctionMaker.java は関数の生成に関する処理がまとめられている。Terms.java は単項式のクラスがまとめられている。

このアプリケーションを作るにあたって最も難しかったのは、テキストフィールドに入力された文字列をどのようにして数学的な関数として解釈させるかという点である。この解釈する過程の実装は大きく分けて 2 つの要素で構成される。

一つ目の要素は数学的な関数のふるまいを記述することである。関数は入力 t の値に応じて値を返すものである。数学的な意味での関数を表す Function クラスと、そのクラス内に t を代入したときの式の値を計算し出力する calculate メソッドを作成した。グラフを描画する点を求める際、for 文で t の値を変化させ Function クラスの calculate メソッドで式の値を計算することで、数学的な関数のふるまいを実現した。しかし、関数には様々な種類が考えられるため、Function クラス単体では、実現できる関数の種類に限界があると考えた。例えば、$x(t) = t^2 + 2t + 3$のような 2 次関数をつくるとなると、t^2の係数の情報、tの係数の情報、定数項の情報を Function クラスのフィールドとして持っておく必要があるが、それだけでは 3 次関数以上の関数を表すことができない。配列の n 番目に n 次の項の係数の情報を記録する方法も考えられるが、それだと$t^{0.5}$のような非整数を指数部に持つ関数を表現できない。また、$x(t) = (t + 4)^2 + 1$のような合成関数を表現することが難しい。また、$x(t) = t * \sin(t)$のような 2 種類の関数の積で表される形式の関数を表現することが難しい。

この問題を解決するのが Term インターフェースである。Term インターフェースは単項式を表すインターフェースで、Function クラスと同様に calculate メソッドを持つ。Term インターフェースを実装した TriTerm クラスや、LogTerm クラスによって、三角関数や対数関数を表すことができる。TriTerm クラスがもつ情報は三角関数の種類と内部の関数である。これにより、TriTerm クラスは$\sin(t + 1)$のような合成関数の形式を扱うことができる。$\sin(t + 1)$をこのクラスで扱う場合、三角関数の種類はsin、内部の関数は$t + 1$となる。Function クラスのオブジェクトをフィールドに持つことで内部の関数の情報を持つことができる。TriTerm クラスの calculate メソッドは、Function クラスのオブジェクトの calculate メソッドで得た値をuとすると、$\sin(u)$を計算してその値を返す。このように、Term インターフェースを実装したクラスは、Function クラスのオブジェクトをフィールドに持つことで合成関数を表現することができる。

Function クラスは Term インターフェースのオブジェクトの List と、それらの単項式の間の演算子の情報を表す Operators のリストを持つ。たとえば、$t^2 + 2t + 3$を Function クラスで表すとき、Term インターフェースのオブジェクトの List は、{t, 2, 2, t, 3}となり、Operators のリストは{^, +, *, +}となる。Function クラスの calculate メソッドでは、Operators のリストの情報を頼りに、累乗->掛け算->足し算の順で計算を行う。

その計算アルゴリズムについて説明する。まず、Term インターフェースのオブジェクトの List の {t, 2, 2, t,

3}でそれぞれのオブジェクトの calculate メソッドを実行し、項ごとの値を計算する。Operators のリストから「^」の要素を左から順に探索し、見つかったところのインデックスを n とすると n 番目と n+1 番目項の値について累乗の計算を行う。最後まで探索すると、次に「*」の演算子について、最後に「+」の演算子について計算する。すると、最終的に要素が 1 つの項の値の List が得られ、その残った要素が求める関数の値である。この処理の過程の例を表 2 に示す。

表 2 ：Function クラスの calculate メソッドのアルゴリズムの計算例

	項の値のリスト(t=1 のとき)	Operators のリスト
0 回目	{1, 2, 2, 1, 3}	{^, +, *, +}
1 回目	{1^2=1, 2, 1, 3} = {1, 2, 1, 3}	{+, *, +}
2 回目	{1, 2*1=2, 3} = {1, 2, 3}	{+, +}
3 回目	{1+2=3, 3} = {3, 3}	{+}
4 回目	{3+3=6} = {6}	{}

まず、t=1 の場合は、{t, 2, 2, t, 3} の t に 1 が代入されるので、{1, 2, 2, 1, 3} というリストが得られる。1 回目の計算では、Operators のリストの 0 番目に「^」を発見し、項の値のリストの 0 番目と 1 番目の値を累乗計算する (1^2=1)。こうして、項の値のリストの 0 番目が新たに「1」となり、計算が終わった「^」は Operators のリストから消去される。続いて、「*」の演算子を優先するので、Operators のリストの 1 番目に「*」を発見し、項の値のリストの 1 番目と 2 番目の値を掛け算する (2*1=2)。こうして、項の値のリストの 1 番目が新たに「2」となり、計算が終わった「*」は Operators のリストから消去される。以下、同様に Operators のリストが空になるまで計算を続ける。こうして要素が「6」のみのリスト得られ、6 が求める関数の値である。

二つ目の要素は、文字列を数学的な関数として変換することである。ユーザからは 2sin(t+1)のような文字列が与えられるが、当然文字列のままでは計算を行うことができない。そこで、FunctionMaker クラスで文字列から Function クラスのオブジェクトを作成する処理を行う。FunctionMaker クラスの createFunction メソッドは、引数に文字列をとり、戻り値に Function クラスのオブジェクトを返す。createFunction メソッドでは、まず半角、全角スペースをすべて消去する。次に、for 文で文字列の先頭の文字から順に探索をはじめ、その文字の種類によって処理を変える。まず数値かそれ以外で処理を分岐する。数値が来たら、数値以外の文字がくるまで次の文字を調べ、数値が書かれている部分文字列を取得する。そしてその部分文字列を Double クラスの parseDouble メソッドで double 型にする。数値以外が来たら、その文字によって処理を分岐させる。例えば、「+」などの演算子が来たら、Function クラスの addOperators メソッドで Operators リストに「+」を加える。また、「s」などの文字列が来たら、checkTerm メソッドで、現在のインデックスから始まる文字列が「sin」かどうか調べ、sin() の中の文字列を元に sin の項の内部の関数を生成する。そして、sin の項を Function クラスの addTarm メソッドで追加する。また、createFunction メソッドでは途中で想定外の入力が現れたとき、処理を中断しエラーフラグを立てた関数を生成し、return する。エラーフラグが立っている関数は、描画処理を行わないようにプログラムしてある。

以上の 2 つの要素によって、かなり高い自由度で関数を生成することができる。この点がアプリケーションの要点であり、工夫した点である。