Univerzitet u Kragujevcu

Fakultet inženjerskih nauka



**Softverski inženjering**

Projektni zadatak:

Crtač grafika

Student: Predmetni nastavnik:

Mladen Radojević 630/2017 Nenad Filipović

Tijana Šušteršič

**Sadržaj**

|  |  |
| --- | --- |
| **1 Uvod** ........................................................................................................................  **2 Opis sistema** ..............................................................................................................  **3 Osnovni koncepti** …………………………………………………………………………..  3.1 Mapiranje koordinatnog sistema  3.2 Skaliranje osa  3.3 Diskretizacija funkcije i crtanje  **4 Realizacija sistema** ...................................................................................................  4.1 Klasa Point ............................................................................................................  4.2 Klasa MainCanvas ……………………………………………………………………...  4.3 Klasa RenderEngine …………………………………………………………………….  4.4 Klasa CanvasInput ………………………………………………………………………  4.5 Klasa UserInterface ……..……………………………………………………………...  4.6 Klasa Function ………………………………………….……………………………….  4.7 Klasa DrawingOptionsWindow…………………………………………………………  4.8 Klasa Communicator ……………………………………………………………………  4.9 Klasa MainWindow ………………………………………………………………………  4.10 Klasa App ……………………………………………………………………………….  **5 UML dijagrami** ............................................................................................................  5.1 Dijagram slučajeva korišćenja ................................................................................  5.2 Dijagram sekvenci ..................................................................................................  5.3 Dijagram aktivnosti .................................................................................................  5.4 Dijagram stanja ......................................................................................................  5.5 Dijagram klasa .......................................................................................................  5.6 Dijagram objekta ....................................................................................................  **6 Korišćenje sistema** ...................................................................................................  6.1 Korisnički interfejs ………………………………………………………………………  6.2 Format unosa funkcija ………………………………………………………………….  **7 Zaključak** ...................................................................................................................  **8 Literatura** …………………………………………………………………………………… | ***3***  ***3***  ***3***  *3*  *4*  *5*  ***6***  *6*  *6*  *8*  *8*  *11*  *17*  19  23  26  *27*  ***28***  *28*  *29*  *31*  *32*  *32*  *33*  ***34***  *34*  *39*  *41*  *42* |

***1. Uvod***

Računare danas koristimo više nego ikad. Razne vrste softvera su postala naša svakodnevnica, oni nam plaćaju račune, omogućavaju nam da kupujemo sedeći na udobnoj fotelji, naučnicima olakšava proračune i mnogo drugo. Softversko inženjerstvo je relativno nova disciplina, računari su postojali i pre nje, međutim, softver koji se tada pisao nije bio preterano obiman, ali kasnije kad su se pravili specijalizovani softveri sa razna preduzeća uključujuci bankarstvo kao i avio, naftnu i druge kompanije,tada je već bilo jasno, potreban je sistematizovan pristup kreiranja softvera. U ovom izlaganju bice objašnjen pristup kreiranja softvera zaduženog za crtanje grafika funkcije.

***2. Opis sistema***

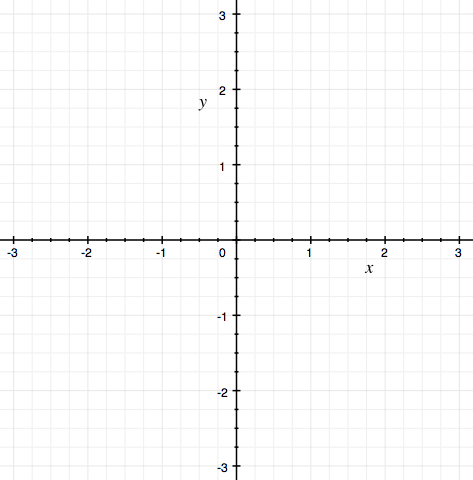
Matematičke funkcije srećemo gotovo u svim poljima nauke kao i naravno u čistoj teorijskoj matematici, bilo da se radi o studentima, profesorima pa čak i istrazivačima u nekoj kompaniji ili organizaciji, biti u mogućnosti vizualizovati različite proračune i modele je važan faktor u stvaranju intuicije o nekoj pojavi pa samim tim i naše razumevanje o istoj. Naš sistem treba da omogući korisniku da iscrta standardne matematičke funkcije kao i njihove kompozicije proizvoljne dubine. Skaliranje apcise i ordinate je takođe proizvoljno i korisniku treba da pruži mogućnost da priblizi(zumira) ili udalji određeni deo grafika koji ga interesuje, kao i pomeranje u bilo kom pravcu koordinatnog sistema. U daljem izlaganju bice objašnjeni osnovni problemi sa kojima se susrećemo, kao i sama realizacija u programskom jeziku *Javi*.

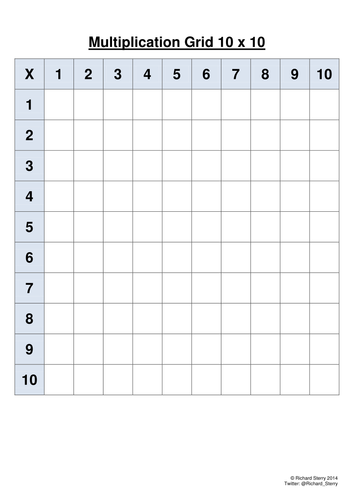
***3. Osnovni koncepti***

Crtanje grafike na računaru moze biti jako zahtevno, susrećemo se sa mnoštvom problema. Ovo poglavlje je namenjeno da objasni neke od tih problema kao i naravno da predloži rešenje za iste.

***3.1 Mapiranje koordinatnog sistema***

Mi ćemo za crtanje koristiti platno (instancu klase *Canvas* koja će biti kasnije objašnjena), međutim, osnovni problem kod toga jeste da to platno možemo posmatrati kao tabelu piksela. Pogledajmo najpre slike 3.1.1 i 3.1.2.





Slika 3.1.1: Koordinatni sistem Slika 3.1.2: Tabela

Dakle, kod koordinatnog sistema x-osa raste sa leva na desno i to nam ne pravi problem jer je to slučaj i sa tabelom, međutim, y-osa kod koordinatnog sistema raste odozdo-nagore što je suprotno od tabele. Takođe sa predstavljanje i negativnih delova koordinatnog sistema kod tabele neophodno je translirati gornji-levi ugao tako da on dođe u centru.

Označimo širinu i dužinu tabele kao ***width*** i ***height***respektivno. Neka su ***x*** i ***y*** koordinate koje odgovaraju koordinatnom sistemu a **u** i **v** koordinate koje odgovaraju tabeli. Tada definisemo preslikavanje kao:

***(u, v) = (x + width/2, -y + height/2)*** (1)

Jednačina (1) nam omogućava da izračunavanja tačaka funkcije čuvamo u standardnom formatu kao tačke koordinatnog sistema, a kada želimo da ih iscrtamo možemo koristiti ovu formulu kako bi to zaista i prikazali na platnu.

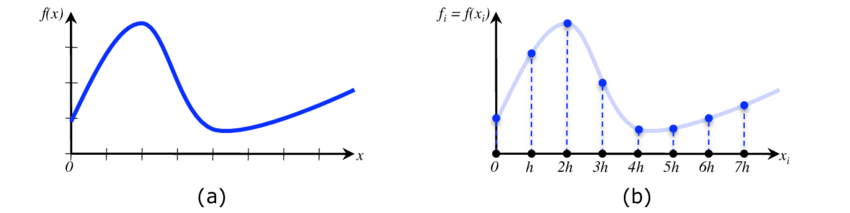
***3.2 Skaliranje osa***

Odnosi se na to koliko piksela izdvajamo da bi smo prikazali jednu jedinicu. Ako želimo da prikažemo funkciju na intervalu x ꞓ (-5, 5) sa širinom *width,* to žnači da će svaka jedinica imati po *width/10* piksela. To je u redu ako je taj interval tog reda, međutim sta ako želimo da prikažemo funkciju na nekom većem intervalu, recimo x ꞓ (-5000, 5000), to znači da je širina jedne jedinice sada *width/10000.* Ako uzmemo u obzir činjenicu da će naše platno uglavnom biti manje veličine recimo oko 500px(za širinu), dolazimo do problema. Naime, razlomak 500/10000 ce biti manji od 1, pošto broj piksela koji nije ceo broj nema smisla program ce u najboljem slučaju dati netačne informacije(mada postoji šansa i da ce se srušiti). Rešenje za ovaj problem jeste da se skaliranje ne odnosi na pojedinačne jedinice, nego radije na skup jedinica. Sada kada imamo interval od ukupno 1000 jedinica iscrtaćemo samo 10 podeoka, od kojih ce svaki da sadrži u sebi 100 jedinca. Opravdanje za to je činjenica da kada korisnik vidi funkciju na nekom većem intervalu njemu onda nije najbitnija preciznost za jednu jedinicu. U slučaju da jeste, korisnik bi trebalo da približi samo deo koji ga zanima, jer ce tada preciznost biti najveća. Dakle, udaljavanjem grafa dobijamo veći pregled funkcije, ali je i preciznost manja.

Ovo je implementirano putem posebnih promenjivih ***scalex*** i ***scaley***. Naime, ako imamo width = 500, rangeX = 10 (abs(maxX - minX)) i scalex = 2, tada tada cemo imati 500/(10/2) piksela po podeoku, tj. ukupno 5 podeoka sa od po 100px. Analogno je za y-osu.

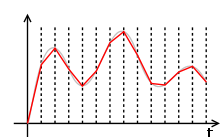
***3.2 Diskretizacija funkcije i crtanje***

Iako bi bilo super da ispitamo vrednost funkcije u svakoj tački, to nažalost, nije moguće. Umesto toga uzećemo samo neke tačke i vrednosti između njih ćemo aproksimirati. Dakle diskretizovaćemo funkciju sa korakom **h**. Postupak je prikazan na slici 3.2.1.



Slika 3.2.1: Diskretizacija funkcije

Međutim kako sada iscrtati ovu funkciju na platnu kada znamo samo vrednosti u nekim tačkama? Ono što ćemo mi uraditi jeste da između svaka dva susedna odabiraka nacrtamo pravu liniju. Pogledajmo na primeru na slici 3.2.2 (funkcija ne odgovara prethodnom primeru, uzeta je nova).



Slika 3.2.2: Aproksimacija funkcije

Poprilično blizu originalnoj funkciji zar ne? Kada je ***h*** tj. u našoj realizaciji pod nazivom ***step*** jako malo, onda je i ova aproksimacija bolja. Takođe impelemntacija ove metode je poprilično jednstavna, sve sto treba da uradimo je da iskoristimo jednačinu prave kroz dve tačke.

(2)

Međutim, mi ćemo u konkretnoj realizaciji da koristimo gotovu funkciju *drawLine(),* međutim, smatramo da je važno imati ideju kako je moguće impelentirati je, pa je iz tog razloga jednačina (2) data.

***4. Realizacija sistema***

Sistem ćemo realizovati u objektno orijentisanom programskom jeziku *Javi.* Program je prirodno GUI tipa, prema tome, odlučili smo se za korišcenje klasa iz paketa *swing.*

***4.1 Klasa Point***

Sasvim jednostavna klasa koju cemo koristiti za skladištenje pojedinacnih tačaka funkcije, dakle, mi zapravo čuvamo konačan niz tačaka u nadi da ćemo time aproksimirati funkciju. Treba napomenuti i to da vrednosti koje ovde čuvamo nisu mapirane za reprezentaciju u *Canvas*-u, prema tome, potrebno je ekplicitno mapirati ove vrednosti kada pokušavamo da ih zaista i iscrtamo.

1. **public** **class** Point {
3. **public** **double** x;
4. **public** **double** y;
6. **public** Point(**double** x, **double** y){
7. **this**.x = x;
8. **this**.y = y;
9. }
10. }

***4.2 Klasa MainCanvas***

Ova klasa predstavlja nase platno na kome crtamo koordinatne ose i funkciju. Klasa nasleđuje *Canvas,* i razlog zašto pravimo ovu klasu jeste da bi osim mogućnosti koje nam pruža Canvas mogli i da vodimo računa o tome kojim bojama se crtaju određeni delovi grafika, korisnik ih može menjati u bilo kom trenutku.

1. **import** java.awt.\*;
2. **import** java.awt.image.BufferStrategy;
4. **public** **class** MainCanvas **extends** Canvas {

7. // Default colors
8. **private** Color backgroundColor = **new** Color(45,56,60);
9. **private** Color drawColor = Color.YELLOW;
10. **private** Color axisColor = Color.WHITE;

13. **public** BufferStrategy bs;
14. **public** Graphics g;

17. **public** MainCanvas(**int** WindowdWidth, **int** WindowHeight){
18. **super**();
19. Dimension dim = **new** Dimension(WindowdWidth\*2/3, WindowHeight);
20. **this**.setPreferredSize(dim);
22. //default background
23. **this**.setBackground(backgroundColor);
25. }

28. // Setters for colors
29. **public** **void** setAxisColor(Color axisColor) {
30. **this**.axisColor = axisColor;
31. }
33. **public** **void** setBackgroundColor(Color backgroundColor) {
34. **this**.backgroundColor = backgroundColor;
35. **this**.setBackground(**this**.backgroundColor);
36. }
38. **public** **void** setDrawColor(Color drawColor) {
39. **this**.drawColor = drawColor;
40. }

43. **public** Color getBackgroundColor() {
44. **return** backgroundColor;
45. }
47. **public** Color getDrawColor() {
48. **return** drawColor;
49. }
51. **public** Color getAxisColor() {
52. **return** axisColor;
53. }
55. }

***4.3 Klasa RenderEngine***

Ova klasa nam omogućava da osvežavamo bilo koji objekat čija klasa impelentira *Updatable* interfejs. U našem slučaju to ce biti klasa *Communicator.* Dakle funkcija *update* ce biti pozvana na svakih *delay* milisekundi. Razlog za postojanje ove klase je činjenica da prikaz iscrtane funkcije nije statičan, kada korisnik želi da priblizi(udalji) neki deo ili prevuče mišem u želji da pomeri grafik, crtež bi istog trenutka trebao biti izmenjen sa uključenim novim parametrima. Iako možda postoji deo vremena u kome korisnik ne želi da menja prikaz, ovim ipak razdvajamo te dve funkcionalnosti, pa su i održavanje i eventualne izmene znatno olakšane.

1. **import** javax.swing.Timer;
2. **import** java.awt.event.ActionEvent;
3. **import** java.awt.event.ActionListener;

6. **public** **class** RenderEngine **implements** ActionListener {
8. **private** **int** delay = 10;
9. **private** Timer timer;
11. **private** Updatable target;
13. **public** RenderEngine(Updatable target){
14. // Add reference to canvas, direct communication
15. **this**.target = target;
17. timer = **new** Timer(delay, **this**);
18. timer.start();
19. }
21. @Override
22. **public** **void** actionPerformed(ActionEvent e) {
23. target.update();
24. }
25. }

***4.4 Klasa CanvasInput***

Iako se razni ulazi od strane miša i tastature provlače u više delova programa, ova klasa ima namenu samo da ostvari funkcionalnosti za dve, približavanje(udaljavanje) i translaciju grafa. Translaciju radimo tako što pamtimo prošle parametre grafika (opsege x i y-ose) i u zavisnosti gde se kursor miša nalazi procenjujemo u kom pravcu je povučen, pa dodajemo određen broj na trenutne vrednosti *maxX*, *minX*, *maxY* i *minY*. Udaljavanje i približavanje radimo tako što u isto vreme smanjimo ili povećamo vrednost trenutnih parametra u zavisnosti u kom pravcu je točkic miša okrenut. Treba napomenuti da vrednost koju dodajemo na trenutne parametre zavisi od njih samih, jer recimo ako je trentuni opseg za x osu koja je prikazana (-10000, 10000) nema smisla smanjiti opseg (recimo ako je približavanje u pitanju) na (- 9999, 9999) zato jer je taj rezultat jedva vidljiv. Isto vazi i za trasnlaciju.

1. **import** java.awt.event.\*;
3. **public** **class** CanvasInput {
5. **private** MainCanvas canvas;
6. **private** Communicator communicator;
8. // Needed for dragging
9. **private** **double** minXprev;
10. **private** **double** maxXprev;
12. **private** **double** minYprev;
13. **private** **double** maxYprev;
15. **private** **int** mousePreviosX;
16. **private** **int** mousePreviosY;
18. **private** **boolean** dragged = **false**;
20. **public** CanvasInput(MainCanvas canvas, Communicator communicator){
21. **this**.canvas = canvas;
22. **this**.communicator = communicator;
24. // Initialize
25. minXprev = communicator.minX;
26. maxXprev = communicator.maxX;
28. minYprev = communicator.minY;
29. maxYprev = communicator.maxY;
31. // Create event listeners!
32. canvas.addMouseListener(**new** MouseListener() {
33. @Override
34. **public** **void** mouseClicked(MouseEvent e) {
35. minXprev = communicator.minX;
36. maxXprev = communicator.maxX;
38. minYprev = communicator.minY;
39. maxYprev = communicator.maxY;
40. }
42. @Override
43. **public** **void** mousePressed(MouseEvent e) {}
45. @Override
46. **public** **void** mouseReleased(MouseEvent e) {
47. minXprev = communicator.minX;
48. maxXprev = communicator.maxX;
50. minYprev = communicator.minY;
51. maxYprev = communicator.maxY;
53. **if**(dragged) {
54. communicator.fillBuffer();
55. dragged = **false**;
56. }
57. }
59. @Override
60. **public** **void** mouseEntered(MouseEvent e) {}
62. @Override
63. **public** **void** mouseExited(MouseEvent e) {}
64. });
66. // Logic for dragging. Done
67. canvas.addMouseMotionListener(**new** MouseMotionListener() {
68. @Override
69. **public** **void** mouseDragged(MouseEvent e) {
71. **double** offsetX = e.getX() - mousePreviosX;
72. communicator.minX = minXprev - communicator.scalex\*(offsetX / communicator.numOfPixelsX);
73. communicator.maxX = maxXprev - communicator.scalex\*(offsetX / communicator.numOfPixelsX);
75. **double** offsetY = e.getY() - mousePreviosY;
76. communicator.minY = minYprev + communicator.scaley\*(offsetY / communicator.numOfPixelsY);
77. communicator.maxY = maxYprev + communicator.scaley\*(offsetY / communicator.numOfPixelsY);
79. dragged = **true**;
81. }
83. @Override
84. **public** **void** mouseMoved(MouseEvent e) {
86. mousePreviosX = e.getX();
87. mousePreviosY = e.getY();
88. maxXprev = communicator.maxX;
89. minXprev = communicator.minX;
90. maxYprev = communicator.maxY;
91. minYprev = communicator.minY;
93. }
94. });
96. // scrool options!
97. canvas.addMouseWheelListener(**new** MouseWheelListener() {
98. @Override
99. **public** **void** mouseWheelMoved(MouseWheelEvent e) {
100. **double** scalex = Math.abs(communicator.maxX - communicator.minX);
101. **double** scaley = Math.abs(communicator.maxY - communicator.minY);
103. minXprev =  communicator.minX;
104. maxXprev =  communicator.maxX;
106. minYprev =  communicator.minY;
107. maxYprev =  communicator.maxY;
109. **if**(e.getWheelRotation() == 1){
110. communicator.minX -= scalex\*0.02;
111. communicator.maxX += scalex\*0.02;
112. communicator.minY -= scaley\*0.02;
113. communicator.maxY += scaley\*0.02;
115. }
116. **else** {
117. communicator.minX += scalex \* 0.02;
118. communicator.maxX -= scalex \* 0.02;
119. communicator.minY += scaley \* 0.02;
120. communicator.maxY -= scaley \* 0.02;
122. }
124. communicator.fillBuffer();
126. }
128. });
130. canvas.addComponentListener(**new** ComponentListener() {
131. @Override
132. **public** **void** componentResized(ComponentEvent e) {
133. communicator.canvasWidth = e.getComponent().getWidth();
134. communicator.canvasHeight = e.getComponent().getHeight();
135. }
137. @Override
138. **public** **void** componentMoved(ComponentEvent e) {}
140. @Override
141. **public** **void** componentShown(ComponentEvent e) {}
143. @Override
144. **public** **void** componentHidden(ComponentEvent e) {}
145. });
147. }
149. }

***4.5 Klasa UserInterface***

Sve komponente sa kojima korisnik može da ima interakciju su uključeni u ovoj klasi. Svaka komponenta je ručno pozicionirana bez upotrebe *window buildera.* Iz ove klase korisnik može da unosi funkciju, bira opsege x i y-ose koje ce biti prikazane kao i da pristupi prozoru u kome može da menja parametre iscrtavanja grafa.

1. **import** javax.swing.\*;
2. **import** javax.swing.event.ChangeEvent;
3. **import** javax.swing.event.ChangeListener;
4. **import** java.awt.\*;
5. **import** java.awt.event.\*;
7. **public** **class** UserInterface **extends** JPanel{
9. **public** JTextField functionInput;
11. **private** **int** uiWidth;
12. **private** **int** uiHeight;

15. **private** Communicator communicator;
17. **private** DrawingOptionsWindow optionWindow;
19. **public** **void** setCommunicator(Communicator comm){
20. **this**.communicator = comm;
21. **this**.optionWindow.comm = comm;
22. }
24. **public** UserInterface(**int** windowWidth, **int** windowHeight) {
25. **super**();
26. GridBagConstraints gc = **new** GridBagConstraints();
28. **this**.uiWidth = (**int**) (windowWidth / 3.5);
29. **this**.uiHeight = windowHeight;
30. **this**.setPreferredSize(**new** Dimension(**this**.uiWidth, **this**.uiHeight));
31. **this**.setLayout(**new** GridBagLayout());
32. **this**.setBorder(BorderFactory.createTitledBorder("Options"));
33. **this**.setBackground(Color.lightGray);
34. **this**.optionWindow = **new** DrawingOptionsWindow();
36. gc.gridx = 0;
37. gc.gridy = 0;
38. gc.weightx = 0.9;
39. gc.weighty = 0.9;
40. gc.insets.top = 15;
41. JButton optionsBtn = **new** JButton("Drawing Options");
42. optionsBtn.setPreferredSize(**new** Dimension(130, 30));
43. **this**.add(optionsBtn, gc);

46. JPanel inputFunctionPanel = **new** JPanel();
47. inputFunctionPanel.setPreferredSize(**new** Dimension(200, 100));
48. inputFunctionPanel.setLayout(**new** GridBagLayout());
49. inputFunctionPanel.setBorder(BorderFactory.createTitledBorder("Input function "));
50. GridBagConstraints gcf = **new** GridBagConstraints();
52. gcf.gridx = 0;
53. gcf.gridy = 0;
54. gcf.weightx = 0.8;
55. gcf.weighty = 0.8;
56. gcf.insets.top = 10;
57. gcf.anchor = GridBagConstraints.FIRST\_LINE\_START;
58. JLabel labelForFunction = **new** JLabel("Function:  ");
59. inputFunctionPanel.add(labelForFunction, gcf);
61. gcf.insets.top = 0;
62. gcf.gridx = 1;
63. gcf.gridy = 0;
64. gcf.anchor = GridBagConstraints.LINE\_END;
65. functionInput = **new** JTextField(10);
66. functionInput.setPreferredSize(**new** Dimension(100, 25));
67. inputFunctionPanel.add(functionInput, gcf);
69. gcf.gridx = 0;
70. gcf.gridy = 1;
71. gcf.gridwidth = 2;
72. gcf.anchor = GridBagConstraints.LAST\_LINE\_END;
73. JButton drawBtn = **new** JButton("Draw");
74. drawBtn.setBackground(Color.green);
75. inputFunctionPanel.add(drawBtn, gcf);
77. gc.gridx = 0;
78. gc.gridy = 1;
80. **this**.add(inputFunctionPanel, gc);
82. // set boundaries logic
83. JPanel boundariesPanel = **new** JPanel();
84. boundariesPanel.setPreferredSize(**new** Dimension(200, 100));
85. GridBagConstraints gcb = **new** GridBagConstraints();
86. boundariesPanel.setLayout(**new** GridBagLayout());
87. boundariesPanel.setBorder(BorderFactory.createTitledBorder("Set boundaries for X "));
89. gcb.gridx = 0;
90. gcb.gridy = 0;
91. gcb.weightx = 0.9;
92. gcb.weighty = 0.9;
93. gcb.insets.top = 10;
94. gcb.anchor = GridBagConstraints.FIRST\_LINE\_START;
95. JLabel labelForMin = **new** JLabel("X Min: ");
96. boundariesPanel.add(labelForMin, gcb);
98. gcb.gridx = 1;
99. gcb.gridy = 0;
100. gcb.anchor = GridBagConstraints.FIRST\_LINE\_END;
101. JSpinner minSpinnerX = **new** JSpinner(**new** SpinnerNumberModel(-5, Integer.MIN\_VALUE, Integer.MAX\_VALUE, 0.5 ));
102. minSpinnerX.setPreferredSize(**new** Dimension(80, 25));
103. boundariesPanel.add(minSpinnerX, gcb);

106. gcb.gridx = 0;
107. gcb.gridy = 1;
108. gcb.anchor = GridBagConstraints.FIRST\_LINE\_START;
109. JLabel labelForMax = **new** JLabel("X Max: ");
110. boundariesPanel.add(labelForMax, gcb);
112. gcb.gridx = 1;
113. gcb.gridy = 1;
114. gcb.anchor = GridBagConstraints.LINE\_END;
115. JSpinner maxSpinnerX = **new** JSpinner(**new** SpinnerNumberModel(5, Integer.MIN\_VALUE, Integer.MAX\_VALUE, 0.5 ));
116. maxSpinnerX.setPreferredSize(**new** Dimension(80, 25));
117. boundariesPanel.add(maxSpinnerX, gcb);
119. gc.gridx = 0;
120. gc.gridy = 2;
122. **this**.add(boundariesPanel, gc);
124. // set boundaries for Y logic here
125. JPanel boundariesPanel2 = **new** JPanel();
126. boundariesPanel2.setPreferredSize(**new** Dimension(200, 100));
127. GridBagConstraints gcb2 = **new** GridBagConstraints();
128. boundariesPanel2.setLayout(**new** GridBagLayout());
129. boundariesPanel2.setBorder(BorderFactory.createTitledBorder("Set boundaries for Y "));
131. gcb2.gridx = 0;
132. gcb2.gridy = 0;
133. gcb2.weightx = 0.9;
134. gcb2.weighty = 0.9;
135. gcb2.insets.top = 10;
136. gcb2.anchor = GridBagConstraints.FIRST\_LINE\_START;
137. JLabel labelForMin2 = **new** JLabel("Y Min: ");
138. boundariesPanel2.add(labelForMin2, gcb2);
140. gcb2.gridx = 1;
141. gcb2.gridy = 0;
142. gcb2.anchor = GridBagConstraints.FIRST\_LINE\_END;
143. JSpinner minSpinnerY = **new** JSpinner(**new** SpinnerNumberModel(-5, Integer.MIN\_VALUE, Integer.MAX\_VALUE, 0.5 ));
144. minSpinnerY.setPreferredSize(**new** Dimension(80, 25));
145. boundariesPanel2.add(minSpinnerY, gcb2);
147. gcb2.gridx = 0;
148. gcb2.gridy = 1;
149. gcb2.anchor = GridBagConstraints.FIRST\_LINE\_START;
150. JLabel labelForMax2 = **new** JLabel("Y Max: ");
151. boundariesPanel2.add(labelForMax2, gcb2);
153. gcb2.gridx = 1;
154. gcb2.gridy = 1;
155. gcb2.anchor = GridBagConstraints.LINE\_END;
156. JSpinner maxSpinnerY = **new** JSpinner(**new** SpinnerNumberModel(5, Integer.MIN\_VALUE, Integer.MAX\_VALUE, 0.5 ));
157. maxSpinnerY.setPreferredSize(**new** Dimension(80, 25));
158. boundariesPanel2.add(maxSpinnerY, gcb2);
160. gc.gridx = 0;
161. gc.gridy = 3;
163. **this**.add(boundariesPanel2, gc);





170. drawBtn.addActionListener(**new** ActionListener() {
171. @Override
172. **public** **void** actionPerformed(ActionEvent e) {
173. communicator.funkcija = **new** Function(functionInput.getText());
174. communicator.fillBuffer();
175. }
176. });


180. minSpinnerX.addChangeListener(**new** ChangeListener() {
181. @Override
182. **public** **void** stateChanged(ChangeEvent e) {
183. **double** value = (**double**)minSpinnerX.getValue();
184. **if**(communicator.maxX > value) {
185. communicator.minX = value;
186. } **else** {
187. JOptionPane.showMessageDialog(**null**, "Value must be less then maximum X");
188. communicator.minX = communicator.maxX - 1;
189. minSpinnerX.setValue(communicator.minX);
190. }
191. communicator.maxX = (**double**)maxSpinnerX.getValue();
192. communicator.minY = (**double**)minSpinnerY.getValue();
193. communicator.maxY = (**double**)maxSpinnerY.getValue();
194. }
195. });



200. maxSpinnerX.addChangeListener(**new** ChangeListener() {
201. @Override
202. **public** **void** stateChanged(ChangeEvent e) {
203. **double** value = (**double**)maxSpinnerX.getValue();
204. **if**(communicator.minX < value){
205. communicator.maxX = value;
206. } **else** {
207. JOptionPane.showMessageDialog(**null**, "Value must be greater then minimal X");
208. communicator.maxX = communicator.minX + 1;
209. maxSpinnerX.setValue(communicator.maxX);
210. }
211. communicator.minX = (**double**)minSpinnerX.getValue();
212. communicator.minY = (**double**)minSpinnerY.getValue();
213. communicator.maxY = (**double**)maxSpinnerY.getValue();
215. }
216. });


220. minSpinnerY.addChangeListener(**new** ChangeListener() {
221. @Override
222. **public** **void** stateChanged(ChangeEvent e) {
224. **double** value = (**double**)minSpinnerY.getValue();
225. **if**(communicator.maxY > value) {
226. communicator.minY = value;
227. } **else** {
228. JOptionPane.showMessageDialog(**null**, "Value must be less then maximum Y");
229. communicator.minY = communicator.maxY - 1;
230. minSpinnerY.setValue(communicator.minY);
231. }
232. communicator.minX = (**double**)minSpinnerX.getValue();
233. communicator.maxX = (**double**)maxSpinnerX.getValue();
234. communicator.maxY = (**double**)maxSpinnerY.getValue();
235. }
236. });
238. maxSpinnerY.addChangeListener(**new** ChangeListener() {
239. @Override
240. **public** **void** stateChanged(ChangeEvent e) {
242. **double** value = (**double**)maxSpinnerY.getValue();
243. **if**(communicator.minY < value){
244. communicator.maxY = value;
245. } **else** {
246. JOptionPane.showMessageDialog(**null**, "Value must be greater then minimal Y");
247. communicator.maxY = communicator.minY + 1;
248. maxSpinnerY.setValue(communicator.maxY);
249. }
250. communicator.minX = (**double**)minSpinnerX.getValue();
251. communicator.maxX = (**double**)maxSpinnerX.getValue();
252. communicator.minY = (**double**)minSpinnerY.getValue();
253. }
254. });
256. optionsBtn.addActionListener(**new** ActionListener() {
257. @Override
258. **public** **void** actionPerformed(ActionEvent e) {
259. optionWindow.setVisible(**true**);
260. }
261. });
263. functionInput.addActionListener(**new** ActionListener() {
264. @Override
265. **public** **void** actionPerformed(ActionEvent e) {
266. functionInput.setBackground(Color.WHITE);
267. }
268. });
270. functionInput.addMouseListener(**new** MouseListener() {
271. @Override
272. **public** **void** mouseClicked(MouseEvent e) {
273. functionInput.setBackground(Color.WHITE);
274. }
276. @Override
277. **public** **void** mousePressed(MouseEvent e) {
279. }
281. @Override
282. **public** **void** mouseReleased(MouseEvent e) {
284. }
286. @Override
287. **public** **void** mouseEntered(MouseEvent e) {
289. }
291. @Override
292. **public** **void** mouseExited(MouseEvent e) {
294. }
295. });
297. functionInput.addKeyListener(**new** KeyListener() {
298. @Override
299. **public** **void** keyTyped(KeyEvent e) {
301. }
303. @Override
304. **public** **void** keyPressed(KeyEvent e) {
305. **if**(e.getKeyCode() == KeyEvent.VK\_ENTER)
306. {
307. communicator.funkcija = **new** Function(functionInput.getText());
308. communicator.fillBuffer();
309. }
310. }
312. @Override
313. **public** **void** keyReleased(KeyEvent e) {
315. }
316. });
318. }
320. }

***4.6 Klasa Function***

Moglo bi se reći da je ova klasa srž programa. Funkciju čuvamo u tekstualnom formatu gde je „x“ promenjiva. Kada želimo da izračunamo vrednost funkciju u nekoj tački, tada za svako pojavljivanje „x“ u tekstu menjamo sa konkretnim brojem i koristimo parser izraza kako bi dobili vrednost u toj tački. Ovo nam omogućava da evaluiramo ogroman broj izraza sačinjenih od osnovnih funkcija i njihovih kompozicija. Treba skrenuti pažnju na to da parser koji koristimo radi samo sa formatom gde je decimalni separator tačka. Različiti sistemi mogu koristiti drugačiji format, zato ekplicitno navodimo koji format želimo da ne bi došlo do problema. Ova klasa implementira *Computable* interfejs.

1. **import** java.text.DecimalFormat;
2. **import** java.text.NumberFormat;
3. **import** java.util.Locale;
5. **public** **class** Function **implements** Computable {

8. **private** String functionString;
10. **private** DecimalFormat df;

13. @Override
14. **public** **double** compute(**double** x) **throws** RuntimeException {
16. String exp = functionString.toLowerCase().replaceAll("x", "(" + df.format(x) + ")");
17. **return** eval(exp);
19. }
21. **public** Function(String funcString){
22. NumberFormat nf = NumberFormat.getNumberInstance(Locale.US);
23. **this**.df = (DecimalFormat)nf;
24. **this**.df.setMaximumFractionDigits(15);
25. **this**.df.applyPattern("###.##############");
26. **this**.functionString = funcString;
27. }
29. **public** **void** setFunction(String funcString){
30. **this**.functionString = funcString;
31. }
33. //fixed!
34. **public** **static** **double** eval(**final** String str) {
35. **return** **new** Object() {
36. **int** pos = -1, ch;
38. **void** nextChar() {
39. ch = (++pos < str.length()) ? str.charAt(pos) : -1;
40. }
42. **boolean** eat(**int** charToEat) {
43. **while** (ch == ' ') nextChar();
44. **if** (ch == charToEat) {
45. nextChar();
46. **return** **true**;
47. }
48. **return** **false**;
49. }
51. **double** parse() {
52. nextChar();
53. **double** x = parseExpression();
54. **if** (pos < str.length()) **throw** **new** RuntimeException("Unexpected: " + (**char**)ch);
55. **return** x;
56. }
58. // Grammar:
59. // expression = term | expression `+` term | expression `-` term
60. // term = factor | term `\*` factor | term `/` factor
61. // factor = `+` factor | `-` factor | `(` expression `)`
62. //        | number | functionName factor | factor `^` factor
64. **double** parseExpression() {
65. **double** x = parseTerm();
66. **for** (;;) {
67. **if**      (eat('+')) x += parseTerm(); // addition
68. **else** **if** (eat('-')) x -= parseTerm(); // subtraction
69. **else** **return** x;
70. }
71. }
73. **double** parseTerm() {
74. **double** x = parseFactor();
75. **for** (;;) {
76. **if**      (eat('\*')) x \*= parseFactor(); // multiplication
77. **else** **if** (eat('/')) x /= parseFactor(); // division
78. **else** **return** x;
79. }
80. }
82. **double** parseFactor() {
83. **if** (eat('+')) **return** parseFactor(); // unary plus
84. **if** (eat('-')) **return** -parseFactor(); // unary minus
86. **double** x;
87. **int** startPos = **this**.pos;
88. **if** (eat('(')) { // parentheses
89. x = parseExpression();
90. eat(')');
91. } **else** **if** ((ch >= '0' && ch <= '9') || ch == '.') { // numbers
92. **while** ((ch >= '0' && ch <= '9') || ch == '.') nextChar();
93. x = Double.parseDouble(str.substring(startPos, **this**.pos));
94. } **else** **if** (ch >= 'a' && ch <= 'z') { // functions
95. **while** (ch >= 'a' && ch <= 'z') nextChar();
96. String func = str.substring(startPos, **this**.pos);
97. x = parseFactor();
98. **if** (func.equals("sqrt")) x = Math.sqrt(x);
99. **else** **if** (func.equals("sin")) x = Math.sin(x);
100. **else** **if** (func.equals("cos")) x = Math.cos(x);
101. **else** **if** (func.equals("tan")) x = Math.tan(x);
102. **else** **throw** **new** RuntimeException("Unknown function: " + func);
103. } **else** {
104. **throw** **new** RuntimeException("Unexpected: " + (**char**)ch);
105. }
107. **if** (eat('^')) x = Math.pow(x, parseFactor()); // exponentiation
109. **return** x;
110. }
111. }.parse();
112. }
113. }

***4.7 Klasa DrawingOptionsWindow***

Instancu ove klase je moguce napraviti iz istance klase *UserInterface*. U njoj se nalaze dodatne opcije za podešavanje parametra, u ovom slučaju se odnosi na vizualni prikaz grafa. Dakle, opcije koje su na raspolaganju korisnicima su promene boje delova platna, prikazivanje i sakrivanje numeracija osa kao i izbor samog kvaliteta grafa. Ovaj objekat se u toku životnog ciklusa programa kreira samo jedanput i ne uništava se do samog kraja programa. Kada se klikne na dugme *close* objekat je zapravo samo sakriven, ponovno otkrivanje je moguce iz instance klase *UserInterface.*

1. **import** javax.swing.\*;
2. **import** javax.swing.event.ChangeEvent;
3. **import** javax.swing.event.ChangeListener;
4. **import** java.awt.\*;
5. **import** java.awt.event.\*;

8. **public** **class** DrawingOptionsWindow **extends** JFrame {
10. **public** Communicator comm;

13. **private** **void** hideWindow(){
14. **this**.setVisible(**false**);
15. }
17. **public** DrawingOptionsWindow()
18. {
19. **super**("Drawing Options");
20. GridBagConstraints gc = **new** GridBagConstraints();
21. **this**.setSize(**new** Dimension(400, 280));
22. **this**.setResizable(**false**);
23. **this**.setLayout(**new** GridBagLayout());
24. JPanel chooseColorPanel = **new** JPanel();
25. chooseColorPanel.setPreferredSize(**new** Dimension(150, 150));
26. chooseColorPanel.setBorder(BorderFactory.createTitledBorder("Choose colors "));
28. GridBagConstraints gcp = **new** GridBagConstraints();
29. chooseColorPanel.setLayout(**new** GridBagLayout());
30. JButton canvasBackgroundBtn = **new** JButton("Background");
31. JButton canvasAxisBtn = **new** JButton("Axis");
32. JButton canvasFunctionBtn = **new** JButton("Function");
33. gcp.gridx = 0;
34. gcp.gridy = 0;
35. canvasBackgroundBtn.setPreferredSize(**new** Dimension(120, 25));
36. chooseColorPanel.add(canvasBackgroundBtn, gcp);
38. gcp.gridx = 0;
39. gcp.gridy = 1;
40. canvasAxisBtn.setPreferredSize(**new** Dimension(120, 25));
41. chooseColorPanel.add(canvasAxisBtn, gcp);
43. gcp.gridx = 0;
44. gcp.gridy = 2;
45. canvasFunctionBtn.setPreferredSize(**new** Dimension(120, 25));
46. chooseColorPanel.add(canvasFunctionBtn, gcp);
48. gc.gridx = 0;
49. gc.gridy = 0;
50. gc.insets.right = 30;
51. // gc.weighty = 0.9;
52. // gc.weightx = 0.9;
53. **this**.add(chooseColorPanel, gc);
54. gc.insets.right = 0;
56. // Creating panel for step drawing!
57. JPanel stepChooser = **new** JPanel();
58. stepChooser.setBorder(BorderFactory.createTitledBorder("Choose step "));
59. stepChooser.setLayout(**new** GridBagLayout());
60. stepChooser.setPreferredSize(**new** Dimension(130, 80));
61. GridBagConstraints gcp2 = **new** GridBagConstraints();
62. JLabel stepLabel = **new** JLabel("Step:  ");
63. JSpinner stepSpinner = **new** JSpinner(**new** SpinnerNumberModel(0.05, 0.01, 1, 0.01));
64. stepSpinner.setPreferredSize(**new** Dimension(60, 25));
65. JButton spinerBtn = **new** JButton("Ok");
66. spinerBtn.setPreferredSize(**new** Dimension(60, 20));
67. gcp2.gridx = 0;
68. gcp2.gridy = 0;
69. stepChooser.add(stepLabel, gcp2);
70. gcp2.gridx = 1;
71. gcp2.gridy = 0;
72. stepChooser.add(stepSpinner, gcp2);
73. gcp2.gridx = 1;
74. gcp2.gridy = 1;
75. gcp2.insets.top = 10;
76. stepChooser.add(spinerBtn, gcp2);
78. gc.gridx = 1;
79. gc.gridy = 0;
80. gc.anchor = GridBagConstraints.FIRST\_LINE\_START;
81. **this**.add(stepChooser, gc);
83. //toggle axis numbers
84. JCheckBox axisNumberX = **new** JCheckBox("Show x-axis");
85. axisNumberX.setSelected(**true**);
86. gc.gridx = 1;
87. gc.gridy = 0;
89. gc.insets.top = 50;
90. gc.anchor = GridBagConstraints.LINE\_START;
91. **this**.add(axisNumberX, gc);
93. JCheckBox axisNumberY = **new** JCheckBox("Show y-axis");
94. axisNumberY.setSelected(**true**);
95. gc.gridx = 1;
96. gc.gridy = 0;
98. gc.insets.top = 100;
99. gc.anchor = GridBagConstraints.LINE\_START;
100. **this**.add(axisNumberY, gc);

103. JButton closeBtn = **new** JButton("Close");
104. gc.gridx = 1;
105. gc.gridy = 2;
106. gc.anchor = GridBagConstraints.SOUTHEAST;
107. gc.insets.top = 20;
108. **this**.add(closeBtn, gc);
110. **this**.comm = comm;

113. canvasBackgroundBtn.addActionListener(**new** ActionListener() {
114. @Override
115. **public** **void** actionPerformed(ActionEvent e) {
116. Color color = JColorChooser.showDialog((Component)e.getSource(), "Background color", comm.canvas.getBackgroundColor());
117. **if**(color != **null**)
118. comm.canvas.setBackgroundColor(color);
119. }
120. });
122. canvasAxisBtn.addActionListener(**new** ActionListener() {
123. @Override
124. **public** **void** actionPerformed(ActionEvent e) {
125. Color color = JColorChooser.showDialog((Component)e.getSource(), "Axis color", comm.canvas.getAxisColor());
126. **if**(color != **null**)
127. comm.canvas.setAxisColor(color);
128. }
129. });
131. canvasFunctionBtn.addActionListener(**new** ActionListener() {
132. @Override
133. **public** **void** actionPerformed(ActionEvent e) {
134. Color color = JColorChooser.showDialog((Component)e.getSource(), "Function color", comm.canvas.getDrawColor());
135. **if**(color != **null**)
136. comm.canvas.setDrawColor(color);
137. }
138. });

141. closeBtn.addActionListener(**new** ActionListener() {
142. @Override
143. **public** **void** actionPerformed(ActionEvent e) {
144. hideWindow();
145. }
146. });
148. axisNumberX.addChangeListener(**new** ChangeListener() {
149. @Override
150. **public** **void** stateChanged(ChangeEvent e) {
151. comm.showXaxis = axisNumberX.isSelected();
152. }
153. });
155. axisNumberY.addChangeListener(**new** ChangeListener() {
156. @Override
157. **public** **void** stateChanged(ChangeEvent e) {
158. comm.showYaxis = axisNumberY.isSelected();
159. }
160. });
162. spinerBtn.addActionListener(**new** ActionListener() {
163. @Override
164. **public** **void** actionPerformed(ActionEvent e) {
165. comm.step = (**double**)stepSpinner.getValue();
166. comm.fillBuffer();
167. }
168. });


172. }
173. }

***4.8 Klasa Communicator***

Ovo je verovatno najkompleksnija klasa u celom projektu, predstavlja okosnocu izmedju drugih klasa a ujedno i sadrži funkciju koja iscrtava grafik. Glavne funkcije jesu *fillBuffer* i *update. FillBuffer* funkcija se poziva kada je potrebno rekalkulisati tačke iznova zbog promene parametra(recimo promenio se maxX). Medjutim u toj funkciji se tačke smeštaju u standardnom koordinatnom sistemu. Za prikaz na platnu eksplicitno je u funkciji *update* odrađeno mapiranje i skaliranje objašnjeno u prethodnom poglavlju. Takodje implementiran je i mehanizam numeracija osa. Dakle, usled različitog skaliranja i numeracija ose ce se razlikovati. Treba uočiti da ova klasa impelementira *Updatable* interfejs, instanca ove klase zapravo predstavlja *target* u klasi *RenderEngine.*

1. **import** javax.swing.\*;
2. **import** java.awt.\*;
3. **import** java.util.ArrayList;
5. **public** **class** Communicator **implements** Updatable {
7. **public** MainCanvas canvas;

10. // Wait for passing!
11. **public** Computable funkcija;
13. **public** **double** step = 0.05;
15. // For faster entry
16. **public** **int** canvasWidth;
17. **public** **int** canvasHeight;
19. // Public for direct communication
20. // Default is 10 in every direction
21. **public** **double** minX = -5;
22. **public** **double** maxX = 5;
24. **public** **double** minY = -5;
25. **public** **double** maxY = 5;
27. **public** **boolean** showXaxis = **true**;
28. **public** **boolean** showYaxis = **true**;
30. **public** **int** numOfPixelsX;
31. **public** **int** numOfPixelsY;
33. // just test for now!
34. **public** **int** scalex = 1;
35. **public** **int** scaley = 1;

38. **private** UserInterface userInterface;
39. **private**  ArrayList<Point> dataBuffer;
41. // Need to be public for other classes do chose when to update this
42. **public** **void** fillBuffer() {
44. **if**(dataBuffer == **null**) dataBuffer = **new** ArrayList<>();
45. **if**(funkcija == **null**) **return**;
46. // buffer size has points of range size for both ends
47. dataBuffer.clear();
48. **double** x = (**double**) minX - Math.abs(minX - maxX);
50. **try** {
51. **while** (x <= (**double**) maxX + Math.abs(minX - maxX)) {
52. dataBuffer.add(**new** Point(x, funkcija.compute(x)));
53. x += scalex\*step;
54. }
55. }
56. **catch**(Exception ex){
57. userInterface.functionInput.setBackground(Color.ORANGE);
58. JOptionPane.showMessageDialog(**null**, "Invalid input for a function.");
59. dataBuffer = **null**;
60. funkcija = **null**;
61. }
62. }
64. **public** Communicator(MainCanvas canvas, UserInterface userInterface)
65. {
66. **this**.canvas = canvas;
67. **this**.userInterface = userInterface;
69. **this**.canvasWidth = canvas.getWidth();
70. **this**.canvasHeight = canvas.getHeight();
72. numOfPixelsX = (**int**)(canvas.getWidth()/Math.abs(maxX - minX));
73. numOfPixelsY = (**int**)(canvas.getHeight()/Math.abs(maxY - minY));
75. **this**.dataBuffer = **new** ArrayList<>();
76. }

79. @Override
80. **public** **void** update() {
81. canvas.bs = canvas.getBufferStrategy();
82. **if**(canvas.bs == **null**){
83. canvas.createBufferStrategy(3);
84. **return**;
85. }
86. canvas.g = canvas.bs.getDrawGraphics();
87. //canvas.g.setColor(canvas.getBackgroundColor());
88. canvas.g.clearRect(0, 0, canvasWidth, canvasHeight);
89. // Draw here!!!
91. //////////////////////////////////////////////////////////////////////////////////////////////////
92. //draw axis! Support scaling
94. **double** rangeX = Math.abs(maxX - minX);
95. **double** rangeY = Math.abs(maxY - minY);
97. // Calculate scale for X
98. scalex = 1;
99. **while**(rangeX / scalex > 10){
100. scalex++;
101. }
103. // Calculate scale for Y
104. scaley = 1;
105. **while**(rangeY / scaley > 10){
106. scaley++;
107. }
109. **double** scaledRangeX = rangeX/scalex;
110. **double** scaledRangeY = rangeY/scaley;
111. numOfPixelsX = (**int**)(canvas.getWidth()/scaledRangeX);
112. numOfPixelsY = (**int**)(canvas.getHeight()/scaledRangeY);
114. canvas.g.setColor(canvas.getAxisColor());

117. // Draw X axis
118. canvas.g.drawLine(0, (**int**)(maxY/scaley\*numOfPixelsY), canvasWidth, (**int**)(maxY/scaley\*numOfPixelsY));
119. // Draw Y axis
120. canvas.g.drawLine(-(**int**)(minX/scalex\*numOfPixelsX) , 0, -(**int**)(minX/scalex\*numOfPixelsX), canvasHeight);

123. // Draw numerals for X-axis
124. **int** startNumberX = (**int**)(minX);
125. **while**(startNumberX % scalex != 0){startNumberX++;}
127. **if**(minX > 0){
128. **if**((**int**)(minX) < startNumberX) {startNumberX -= scalex;}
129. }

132. **for**(**int** i = 0; i <= (**int**)scaledRangeX; i++) {
133. **if** (i != (**int**) (Math.abs(minX) / scalex) || minX > 0)
134. canvas.g.fillRect(i \* numOfPixelsX - (**int**) ((minX / scalex - (**int**) (minX / scalex)) \* numOfPixelsX) - 1, (**int**) (maxY / scaley \* numOfPixelsY) - 3, 3, 6);
136. **if** (showXaxis) {
137. canvas.g.drawString(String.valueOf(startNumberX), i \* numOfPixelsX - (**int**) ((minX / scalex - (**int**) (minX / scalex)) \* numOfPixelsX), (**int**) (maxY / scaley \* numOfPixelsY) + 15);
138. startNumberX += scalex;
139. } **else** startNumberX += scalex;
140. }
141. // Numerals for X-axis END

144. // Draw numerals for Y-axis
145. **int** startNumberY = (**int**)(maxY);
146. **while**(startNumberY % scaley != 0){startNumberY--;}
148. **if**(maxY < 0){
149. **if**((**int**)(maxY) > startNumberY) {startNumberY += scaley;}
150. }
152. **for**(**int** i = 0; i <= (**int**)scaledRangeY; i++){
153. **if**(i!=(**int**)(Math.abs(maxY/scaley)) || maxY < 0)
154. canvas.g.fillRect(-(**int**)(minX/scalex\*numOfPixelsX) - 3, i\*numOfPixelsY + (**int**)((maxY/scaley - (**int**)(maxY/scaley))\*numOfPixelsY) - 1, 6, 3);
156. **if**(startNumberY == 0){ startNumberY-=scaley; **continue**;}
157. **if**(showYaxis) {
158. canvas.g.drawString(String.valueOf(startNumberY), -(**int**) (minX/scalex \* numOfPixelsX) - 30 , i \* numOfPixelsY + (**int**) ((maxY/scaley - (**int**)(maxY/scaley)) \* numOfPixelsY) + 5);
159. startNumberY -= scaley;}
160. **else** startNumberY-= scaley;
161. }
163. // Numerals for Y-axis END
165. // Draw points of a function BEGIN
167. canvas.g.setColor(canvas.getDrawColor());
168. **if**(dataBuffer != **null** && dataBuffer.size() > 1) {
169. Point first = dataBuffer.get(0);
170. Point second = dataBuffer.get(1);
171. **for** (**int** i = 2; i < dataBuffer.size(); i++) {
173. canvas.g.drawLine((**int**) (first.x \* numOfPixelsX/scalex - minX/scalex \* numOfPixelsX), -(**int**) (first.y \* numOfPixelsY/scaley - maxY/scaley \* numOfPixelsY), (**int**) (second.x \* numOfPixelsX/scalex - minX/scalex \* numOfPixelsX), -(**int**) (second.y \* numOfPixelsY/scaley - maxY/scaley \* numOfPixelsY));
174. Point temp = dataBuffer.get(i);
175. first = second;
176. second = temp;
177. }
178. }
180. // Draw points END
182. // End drawing
183. canvas.bs.show();
184. canvas.g.dispose();
185. }
186. }

***4.9 Klasa MainWindow***

Ova klasa nam služi da bi integrisali sve prethodne komponente u jedan sistem. Dakle kreirane su instance i ujedno razmenili neophodne reference između njih.

1. **import** javax.swing.\*;
2. **import** java.awt.\*;
4. **public** **class** MainWindow **extends** JFrame {
6. **private** UserInterface userInterface;
7. // public for communication with drawingOptions
8. **public** MainCanvas canvas;
10. **private** Communicator communicator;
12. **private** CanvasInput canvasInput;
14. **private** RenderEngine render;

17. **public** MainWindow(**int** windowWidth, **int** windowHeight, String title)
18. {
19. **super**(title);
20. **this**.setSize(**new** Dimension(windowWidth, windowHeight));
22. // Layout settings
23. **this**.setLayout(**new** BorderLayout());
25. // Adding UserInterface
27. **this**.userInterface = **new** UserInterface(windowWidth, windowHeight);
28. **this**.add(userInterface, BorderLayout.WEST);
30. // Adding Canvas, rest of space
31. **this**.canvas = **new** MainCanvas(windowWidth, windowHeight);
32. **this**.add(canvas);
33. **this**.pack();
35. //set up Communicator
36. **this**.communicator = **new** Communicator(canvas, userInterface);
38. // Set up Render
39. **this**.render = **new** RenderEngine(communicator);
41. // Set up Event listeners
42. **this**.canvasInput = **new** CanvasInput(**this**.canvas, **this**.communicator);
44. **this**.userInterface.setCommunicator(communicator);
46. }
48. }

***4.10 Klasa App***

Ovo je klasa koju biramo kao target pri kompajliranju. Naime, ona kreira novi MainWindow koji je u ovom slučaju 800x500 (u pikselima). Ovde treba napomenuti da je ova veličina data iz razloga jer će osigurati da se *scale factor* izmedju x i y ose biti isti, međutim moguće je proizvoljno promeniti veličinu prozora(i u runtime-u), program ce raditi bez problema, međutim treba uzeti u obzir da ce se *scale factor* između osa verovatno razlikovati. Drugo što treba uočiti jeste da se radi bolje odazivnosti kreiranje *MainWindow*-a odvija u zasebnoj niti.

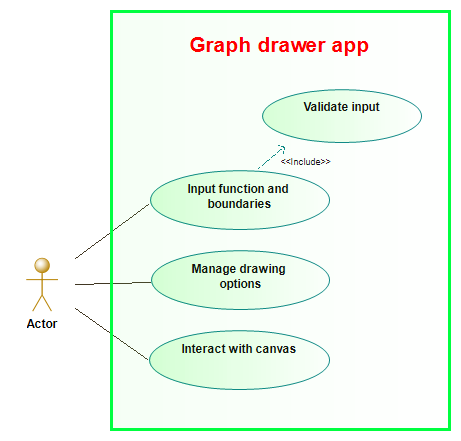
1. **import** javax.swing.\*;
2. **import** java.awt.\*;
4. **public** **class** App {
6. **public** **static** **void** main(String[] args) {
7. SwingUtilities.invokeLater(**new** Runnable() {
8. @Override
9. **public** **void** run() {
10. MainWindow window = **new** MainWindow(800, 500, "GraphDrawer by Mladen");
11. window.setDefaultCloseOperation(WindowConstants.EXIT\_ON\_CLOSE);
12. window.setMinimumSize(**new** Dimension(100, 500));
13. window.setVisible(**true**);
15. }
16. });
18. }
19. }

***5. UML dijagrami***

Objedinjeni jezik za modelovanje ili UML (eng. Unified Modeling Lanuage) je u polju softverskog inženjerstva standardni jezik za vizuelno prikazivanje objektnog modela. UML je opšti jezik za modelovanje pomoću kog se preko grafičkih simbola pravi apstraktni model sistema poznat kao UML jezik. U daljem izlaganju cemo prikazati neke od ovih dijagrama za nas konkretni sistem.

***5.1 Dijagram slučajeva korišćenja***

Dijagram slučajeva korisćenja prikazuje skup slučajeva korišćenja i aktera. Koristi da specificira neku funkcionalnost i ponašanje nekog subjekta. Slučaj korisćenja predstavlja opis skupa sekvenci akcija, uključujuci varijante, koje sistem obavlja da bi proizveo vidljiv rezultat od vrednosti za pojedinog aktera. Ovaj dijagram za naš konkretni sistem je dat na slici 5.1.1

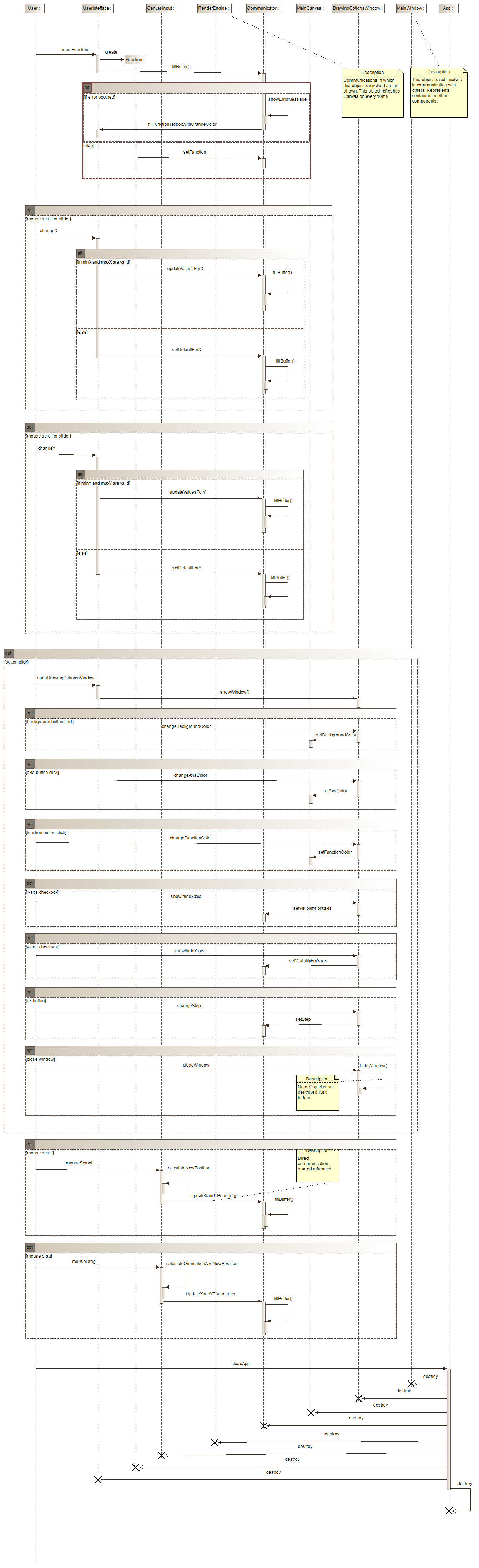


Slika 5.1.1: Dijagram slučajeva korišćenja

Kao što vidimo na dijagramu, postoje tri glavna slučajeva korišćenja. ***Input function and boundaries*** se odnosi na unos funkcije i granica za ose. ***Manage drawing options*** se odnosi na dodatni prozor u kome je moguće menjati boje delova grafika, prikazati ili sakriti numeraciju osa ili pak promeniti kvalitet isrtavanja grafika. ***Interact with canvas*** se odnosi na približavanju i udaljavanju grafika kao i translaciju, dakle ovo se odnosi na komunikaciju korisnika i platna putem miša.

***5.2 Dijagram sekvenci***

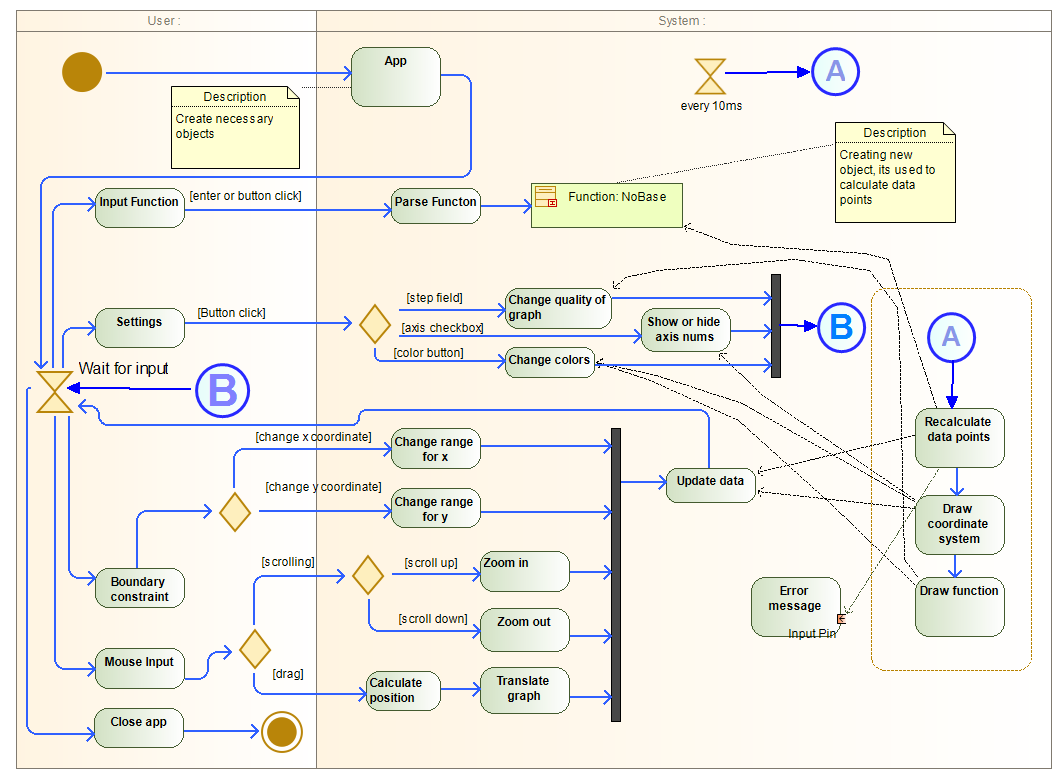
Ovaj dijagram prikazuje komunikaciju između skupa objekta, koja se ostvaruje porukama koje objekti međusobno razmenjuju u cilju ostvarivanja očekivanog ponašanja. Takođe, detaljno opisuje kako se operacije izvode tj. koje poruke se šalju i kada. Dijagram sekvenci koji odgovara našem sistemu dat je na slici 5.2.1.



Slika 5.2.1: Dijagram sekvenci

***5.3 Dijagram aktivnosti***

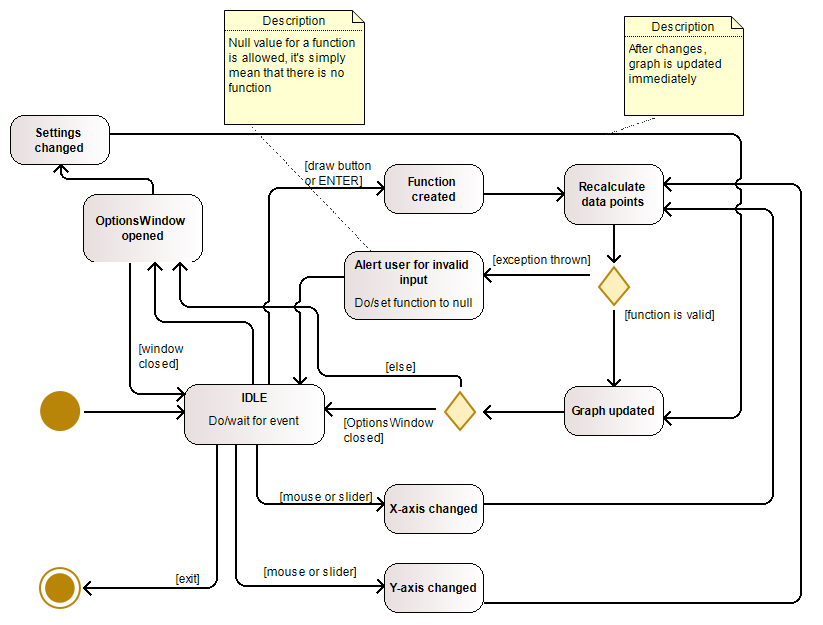
Dijagrami aktivnosti su namenjeni modeliranju dinamičkih aspekata(ponašanja) sistema. Prikazuju tok aktivnosti koju izvršavaju objekti kao i tok objekata između aktivnosti. Dakle, uočićemo aktivnosti i probati da modeliramo tok između njih. Dijagram koji odgovara našem sistemu je dat na slici 5.3.1.



Slika 5.3.1: Dijagram aktivnosti

***5.4 Dijagram Stanja***

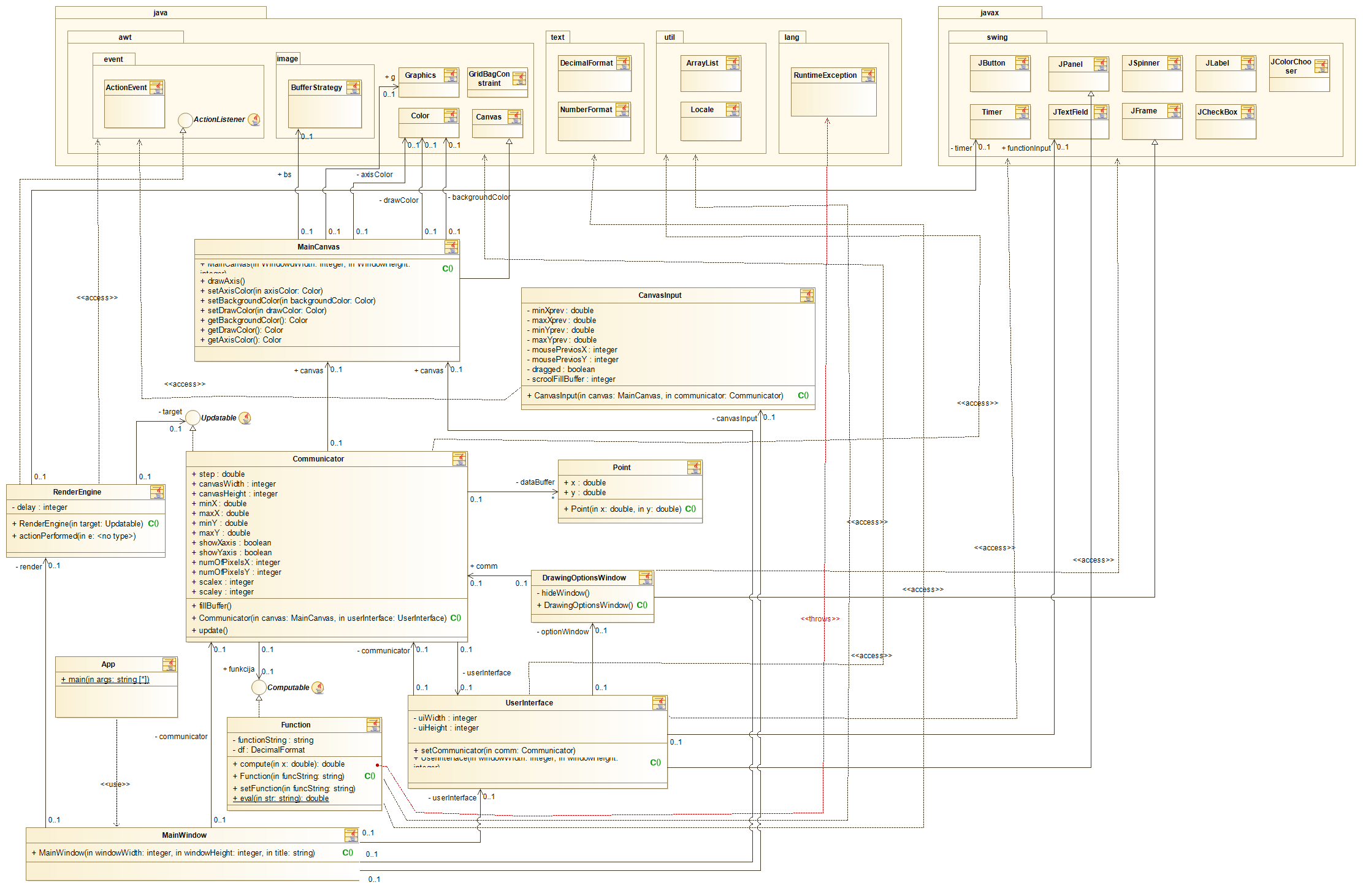
Ovaj dijagram predstavlja ponašanje sistema koji je sastavljen od konačnog broja stanja. Ponašanje koje specifira sekvence stanja kroz koja prolazi. Dijagram stanja modelira ponašanje nekog entiteta ili protokol interakcije. Za početak je potrebno da prepoznamo stanja kroz koje naš sistem prolazi. Iako u programu postoji deo koji se odnosi na učestalo osvežavanje platna, to nećemo prikazati na ovom dijagramu. Umesto toga uvodimo stanje *‘Graph updated’*, to stanje nam ožnačava da je došlo do promene vizualnog prikaza canvas-a koje korisnik vidi. Ovaj dijagram je dat na slici 5.4.1.



Slika 5.4.1: Dijagram stanja

***5.5 Dijagram Klasa***

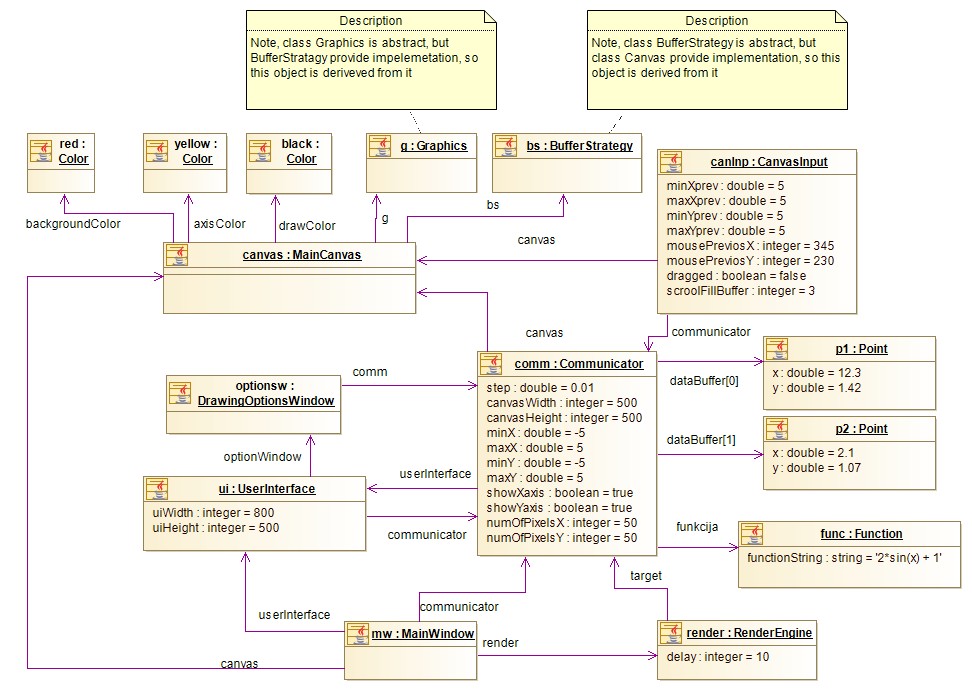
Dijagram klasa prikazuje skup klasa, interfejsa, saradnji, I strugih stvari strukture, povezanim relacijama. Predstavlja graf obrazovan od temena(stvari) povezanih granama(relacijama). Klase koje smo kreirali za naš sistem su objašnjene u ranijem izlaganju, ovde ćemo ih prikazati grafički. Dijagram koji odgovara našem sistemu je prikazan na slici 5.5.1.



Slika 5.5.1: Dijagram klasa

***5.6 Dijagram Objekta***

Dijagrami objekta prikazuju primerke(objekte) apstrakcija(klasa) I njihove veze preko kojih objekti mogu da komuniciraju. Ono što je važno reći o njima jeste da predstavljaju pogled na sistem u jednom trenutku. Pomaže nam u razumevanju modela, ima dokumentacionu svrhu. Dijagram objekta koji odgovara našem sistemu dat je na slici 5.6.1.



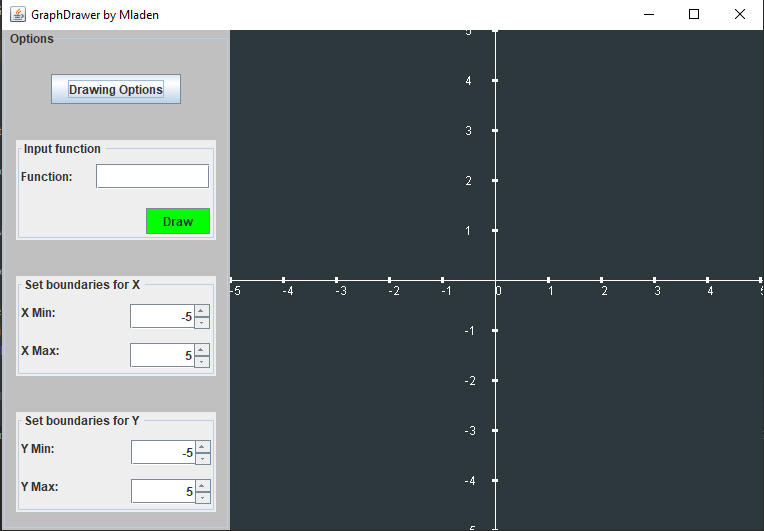
Slika 5.6.1: Dijagram objekta

***6. Korišćenje sistema***

U ovom poglavlju je dato kratko upustvo za korišćenje našeg sistema, biće posebno izdvojen korisnički interfejs, gde ćemo objasniti koje su sve opcije korisniku na raspolaganju, kao i sam format unosa funkcije.

***6.1 Korisnički interfejs***

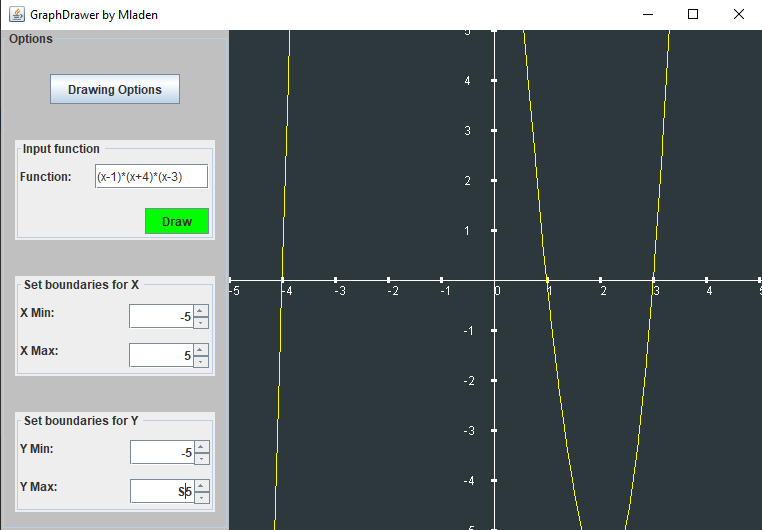
Za početak, krenućemo od početnog prozora. Pogledajmo sliku 6.1.1.



Slika 6.1.1: Korisnički interfejs glavnog prozora

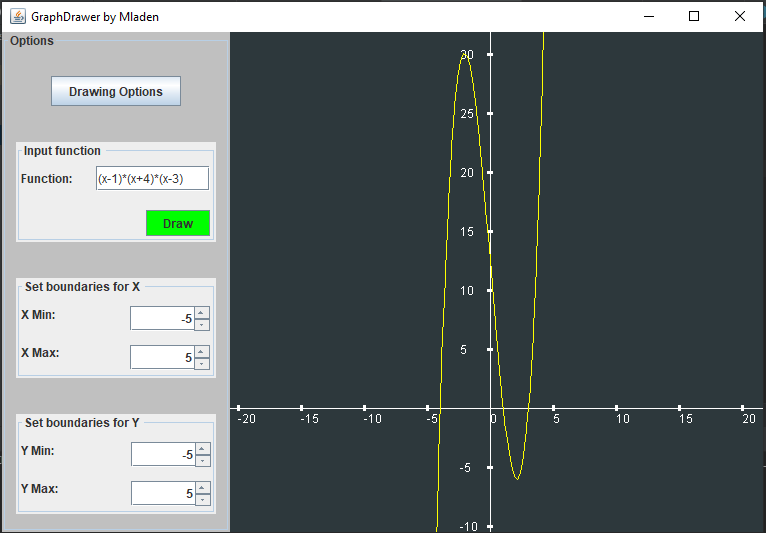
Kao što vidimo, korisnički interfejs početnog prozora je jednostavan. Imamo polje u kome treba da unesemo funkciju, pritiskom na taster ***ENTER*** ili klikom na dugme ***Draw*** iscrtaćemo unetu funkciju. Osim toga, ispod se nalaze polja gde možemo da odaberemo minimalne i maksimalne vrednosti za obe ose, u slučaju da opseg vrednosti (max - min) za ose nije isti, skaliranje se vrši automatski, pa korisnik mora imati u vidu da ose nisu identično skalirane, jedan podeok na x-osi može da predstavlja 10 jedinica dok kod y-ose moze biti 20. U slučaju pogrešno unetih vrednosti za funkciju ili vrednosti granica osa nisu validne, program će nas obavestiti propratnom porukom na ekranu. Ako je greška u izboru vrednosti za ose, onda će se postaviti podrazumevana vrednost, dakle tako da razlika između min i max bude tačno 1. Pogledajmo sada primer iscrtane funkcije na slici 6.1.2, za ovaj primer iscrtaćemo funkciju:

**(x-1)\*(x+4)\*(x-3)**



Slika 6.1.2: Isrctana funkcija (x-1)\*(x+4)\*(x-3)

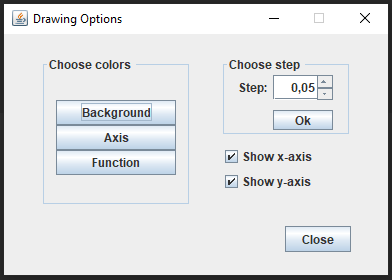
Međutim, imamo mali problem. Ne vidimo ekstremume ove funkcije. Možemo to da rešimo na dva načina. Prvi je vec objašnjen, u pitanju je promena vrednosti za min i max osu. Ali umesto toga, možemo koristiti i skrol miša kako bi udaljili grafik. Treba obratiti pažnju da se skrolovanjem miša ne menjaju vrednosti polja za unos min i max vrednosti. To je urađeno iz razloga ako kasnije budemo želeli da se vratimo na te vrednosti. Dakle, pogledajmo sada sliku 6.1.3 gde smo udaljili grafik da bi videli ekstremume funkcije, i obratimo pažnju na to da se vrednosti polja za unos min i max osa nisu promenila.



Slika 6.1.3: Udaljena funkcija sa slike 6.1.2

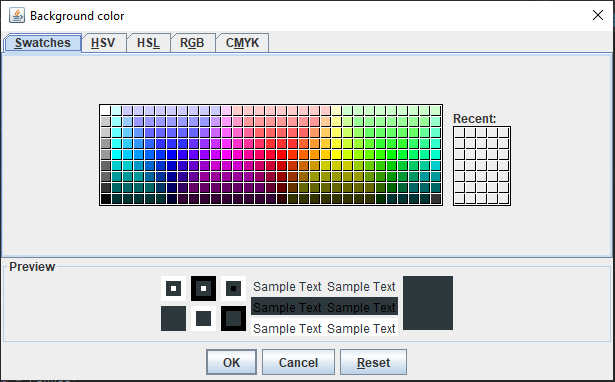
Sada je bolje, obratimo pažnju na to i da je slika osim sto je udaljena ujedno i pomerena malo prema dole, da bi oba ekstremuma mogli da vidimo jasno. Dakle, osim unosa funkcije i biranje veličine osa, omogućena nam je i direktan interakcija sa grafikom, skrol miša za približavanje i udaljavanje (u zavisnosti na koju stranu skrolujemo) i kilikni-zadrži princip za translaciju grafa.

Pogledajmo sada dugme **’Drawing Opstions**’, klikom na ovo dugme otvara nam se prozor kao na slici 6.1.4.



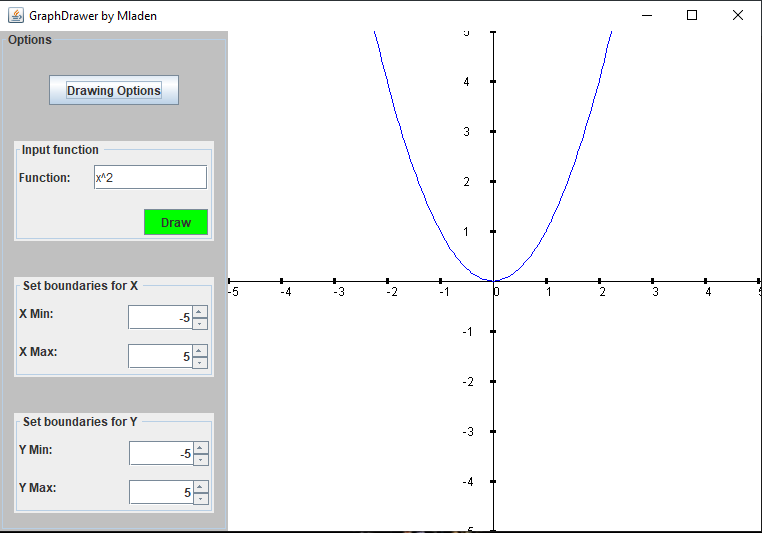
Slika 6.1.4: Izgled prozora *Drawing Options*

Ovde imamo par opcija koje odnose na boju, biranje step-a funkcije (to nam zapravo predstavlja kvalitet, što manji broj to bolje), i opcije za biranje da li numerisati ose ili ne, to su polja za štikliranje. Klikom na neko dugme koje odnosi na promenu boja (recimo da smo kliknuli na dugme *Background*), dobijamo prozor kao na slici 6.1.5.



Slika 6.1.5: Izgled prozora za menjanje boja

Dakle, imamo jako širok spektar boja za izbor, kada izaberemo neku boju klikom na dugme **OK** potvrđujemo promenu i boja pozadine je istog trenutka promenuta. Isti je protokol i za menjanje boja osa, kao i boja iscrtane funkcije. Pogledajmo na slici 6.1.6 kako ce izgledati funkcija **x^2** ako promenimo boju pozadine u belu boju, ose u crnu, a funkciju u pravu boju.



Slika 6.1.6: Funkcija x^2 sa izmenjenim podrazumevanim bojama

Treba još napomenuti da program (bar za sada) ne razrešava domene funkcija. Iz tog razloga program ne može da garantuje tačne vrednosti u tačkama u kojima funkcija nije definisana.Ako je to slučaj , program će najverovatnije porasti prema beskonačnosti u tim tačkama. Tako da ako je ipak potrebno skicirati neku funkciju recimo 1/x, onda je potrebno posmatrati samo vrednosti u kojima je funkcija definisana, a u nuli, i blizu nule ignorisati.

***6.2 Format unosa funkcija***

Pre smo iscrtali neke funkcije a da nismo objasnili u stvari koji format funkcija program prihvata. To ćemo pokušati da uradimo ovde.

Promenjiva je uvek ‘**x**’, program implicitno sve pretvara u mala slova pa nije bitno da li koristimo velika ili mala slova. Osnovne aritmetičke operacije: sabiranje, oduzimanje, množenje i deljenje se označavaju simbolima: **+**, **-**, **\*** , **/** respektivno. Omoguceno je i korišnjenje zagrada **(**,**)** proizvoljne dubine. Dakle sledece funkcije su validne:

* **X\*23**
* **23\*12/3** (izraz ne mora da ima x u sebi)
* **X\*x\*12** (obratiti paznju da je prvo x *uppercase*)
* **(x\*2)\*(12 - 1) + 1**
* **((((x+1))+1))** (višak zagrada ne predstavlja problem)
* **((x+1)\*2)/(x+2)** (uvek se preporučuje korišćenje zagrada da osiguramo redosled operacija)

Međutim, sledeći izrazi nisu validni:

* **Xx** (iako smo možda mislili da ih pomnožimo, za program je ovo greska!)
* **(x+2)(x-1)** (slično kao malopre, operacija između dve zagrade mora da se navede)
* **((x+1) + 1))** (zatvorili smo jednu zagradu vise!)

Osim ugnježdenih izraza koje mozemo praviti prateći pravila, imamo i u opticaju korišćenje nekih od standardnih funkcija: kvadratni koren, sinus, kosinus, tanges, i stepen. U našem programu su ove funkcije date kao: **sqrt**, **sin**, **cos**, **tan** i **^** respektivno.

Neki od validnih izraza koje sada uključuju i funkcije su:

* **sin(x)**
* **2^(x+1)**
* **sin(3.14)** (kao argument funkcije ne mora biti promenjiva!)
* **sin((23\*x) + 1)** (kao argument funkcije može biti bilo koji izraz)
* **cos(tan(2\*x))** (kompozicija funkcija je dozvoljena do proizvoljne dubine!)
* **tan((sin(x))^2 + cos(sin(2\*x)))**

Iz funkcija koje imamo je moguće izvesti i za apsolutnu vrednost, dakle:

|x| -> **sqrt(x^2)**

**7. Zaključak**

Podsetimo se prvo šta smo do sada uradili. Na početku smo definisali osnovnu namenu sistema, zatim razvili osnovne koncepte za realizaciju, napravili objektni model sistema i dokumentovali pomoću UML dijagrama, i na kraju realizovali u programskom jeziku *Javi*. Razmotrimo ovaj scenario, recimo da sada želimo da uključimo još par ljudi u projekat, kako im najbrže objasniti kako program funkcioniše? Odgovor je: UML dijagrami. Ali zašto ne samo kod? Razlog tome jeste da bi na taj način izgubili dragoceno vreme, sigurno će imati puno pitanja a možda bi i prolazili kroz delove koji uopšte nisu relavantni za deo na kojem će oni raditi. Dakle, prilikom projektovanja softvera zaključujemo da je dokumentacija veoma važan aspekt razvoja, naročito zbog drugih ljudi koji tek treba da se uključe u projekat. Druga stvar koju smo mozda i spomenuli ranije, potrebno je što vise razdvojiti implementacije komponenti sistema. Razlozi za to su brojni, neki među važnijim jeste lakše održavanje, omogućavanje testiranja jedinica, s čime se može lako utvrditi koji deo je ‘zakazao’, druga stvar jeste povećanje paralelizma, kada su komponente razdvojene onda više timova može paralelno raditi na delovima sistema i na kraju integrisati u jednu celinu.

# *8. Literatura*

[1] “Portal za elektronsko učenje“: Moodle, Jul 2020, <http://moodle.fink.rs/>

[2] Greg Hewgill, “How to evaluete a math expression given in string form?”, jul 2020, <https://stackoverflow.com/questions/3422673/how-to-evaluate-a-math-expression-given-in-string-form>

[3] “Unified Modeling Language”: *Wikipedia: The Free Encyclopedia*. Wikimedia Foundation, Aug. 2020, https://en.wikipedia.org/wiki/Unified\_Modeling\_Language