Eikosograms and Their Software Implementation

By Glenn Lee

glee@math.uwaterloo.ca

Abstract

Eikosograms (see Cherry and Oldford, 2003) are diagrams that can be used to show the fundamental rules of probability such as Bayes' theorem. While Venn diagrams visually demonstrate abstract set operations and appear in almost all introductory probability text books, eikosograms are more useful in teaching probability theory and calculations, especially when two or three random variables are involved.

Software has been developed to use and construct eikosograms from data. After reviewing the basic concepts and usage of eikosograms, I will introduce the structure of the software and discuss the algorithms used.

1 Introduction

The eikosogram was first introduced by W.H.Cherry and R.W.Oldford in 2003 [1]. It is a unit square (area is 1) that 'visually grounds probability and naturally incorporates the rules of probability within its construction'[1]. It can visually identify independence relationships among two or more random variables and be used to derive Bayes' theorem.

The well-known Venn diagrams and Euler diagrams are useful in logic and set theory, and because of the close relationship between 'set' and 'event', Venn diagrams are used when we calculate the probability of an event. However, the diagram does not contain any quantitative information about probability, and may visually distort the statistical relation between events such as independence relations (see [1] 3.2). See [1] for more details about the history of ring diagrams including Venn and Euler, their relation to religious faith, and their pros and cons when used for teaching statistics. Two other diagrams, Outcome trees and Outcome diagrams, are also discussed in [1].

Eikosograms clearly demonstrate marginal, conditional and joint distributions of random variables as well as the relation between these. In addition, independence relations among variables can easily be seen. i.e. flatness usually means some kind of independence. It makes the dependence and independence relationships among random variables easy to understand, so statistical modeling and design are easy to study. See [2] for a complete discussion of the independence structure among three random variables demonstrated by eikosograms, and the graphical and log-linear models.

Eikosograms can also be used to resolve a number of probability problems and paradox. Oldford related [3] discusses examples including the gas station problem, cherry pie paradoxes, twice reversing Simpson's paradox, two-envelope paradox, the Monty Hall problem, the prisoner's dilemma and others.

The rest of this essay is organized as follows. Section 2 briefly reviews eikosograms, especially those features that are implemented in software.

Section 3 discusses the structure of the software using UML[1], the main algorithms and the data structures used to calculate values of the relevant quantities and store data.

The last section contains possible future work on the development of the software.

2 Eikosogram

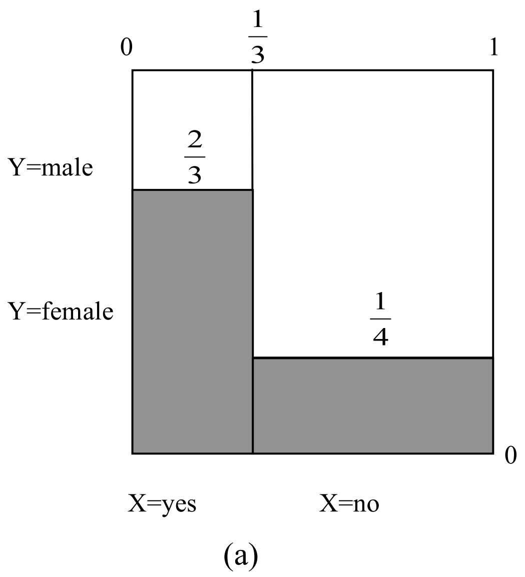
An eikosogram uses a unit square to display different kinds of probabilities of one or more random variables. The area of the unit square represents the probability 1. The unit square is divided

[1] UML is a standard for object-oriented modeling notations endorsed by the Object Management Group(OMG), and industrial consortium on object technologies.

into rectangles. The width of each rectangle represents the marginal probability of the corresponding random variable and the height is the conditional probability; thus the area of each rectangle matches the corresponding joint probability of all random variables involved by Bayes' theorem. The number of such rectangles is equal to the number of all possible combinations of different values of all random variables. The rectangles are aligned inside the unit square and divided into non-overlapping strips along the horizontal axis. Each strip is split into smaller rectangles based on the values of the 'vertical' random variable and distinguished by different colours or shadings.

2.1 Grounding probability

The following two examples of eikosograms will be used in section 2 and 3:



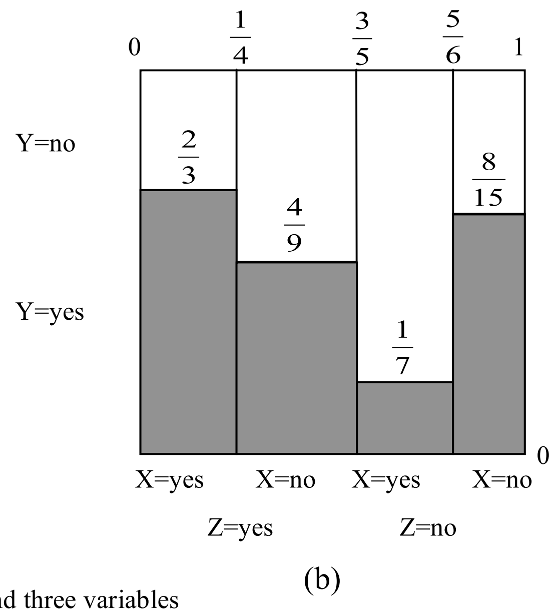


Figure 1: Eikosogram for two and three variables

Figure 1(a) involves two random variables X and Y; Y has two possible values 'male' and 'female' and X has two possible values 'yes' and 'no'. Since we have a unit square, the length of each side and the square area are all equal to 1, so they can be used to represent probabilities:

(i). marginal probability of X is the width/area of the vertical strip:

Pr(X=yes)=1/3 ; Pr(X=no)=1-1/3=2/3;

(ii).conditional probabilities of Y given X are the heights of the correspondent internal rectangles:

Pr(Y=female|X=yes)=2/3; Pr(Y=female|X=no)=1/4; Pr(Y=male|X=yes)=1-2/3=1/3 and Pr(Y=male|X=no)=1-1/4= 3/4 ;

(iii)joint probabilities of Y and X are the areas of the corresponding rectangles:

Pr(Y=female, X=yes)=Pr(Y=female|X=yes)*Pr(X=yes)=2/3*1/3=2/9;

Pr(Y=female, X=no)=Pr(Y=female|X=no)*Pr(X=no)=1/4*2/3=1/6;

Pr(Y=male, X=yes)=Pr(Y=male|X=yes)*Pr(X=yes)=1/3*1/3=1/9 and

Pr(Y=male, X=no)=Pr(Y=male|X=no)*Pr(X=no)=3/4*2/3=1/2;

(iv) marginal probability of Y are the sum of the corresponding rectangles' areas, which from (iii) are seen to be:

Pr(Y=female)=Pr(Y=female,X=yes)+Pr(Y=female,X=no)=2/9+1/6=7/18;

Pr(Y=male)= Pr(Y=male,X=yes)+Pr(Y=male,X=no)=1/9+1/2=11/18=1-7/18;

Figure 1(b) involves three random variables X,Y and Z, each with two possible values ‘yes’ and ‘no’. The width of 1st strip is Pr(X=yes, Z=yes)=1/4 and the height of the left shaded rectangle is Pr(Y=yes|X=yes, Z=yes)=2/3, therefore the area of that rectangle is the joint probability Pr(X=yes,Y=yes,Z=yes)=1/4*2/3=1/6. The marginal probability of Y=yes is now the sum of the areas of all the shaded rectangles and marginal probability of X is the sum of the width of the 1st and 3rd vertical strips. The marginal probability of Z can be read directly from the graph: Pr(Z=yes)=3/5.

In section 3, I will discuss how to calculate these probabilities and construct the eikosograms in Figure 1 from input data. See [1] section 4 for more details about the raindrop metaphor of eikosogram.

An eikosogram can apply to any number of random variables and each random variable can have any number of possible values. The following eikosogram illustrates the relationships among 4 random variables—W,X,and Z are binary while Y has three possible values.

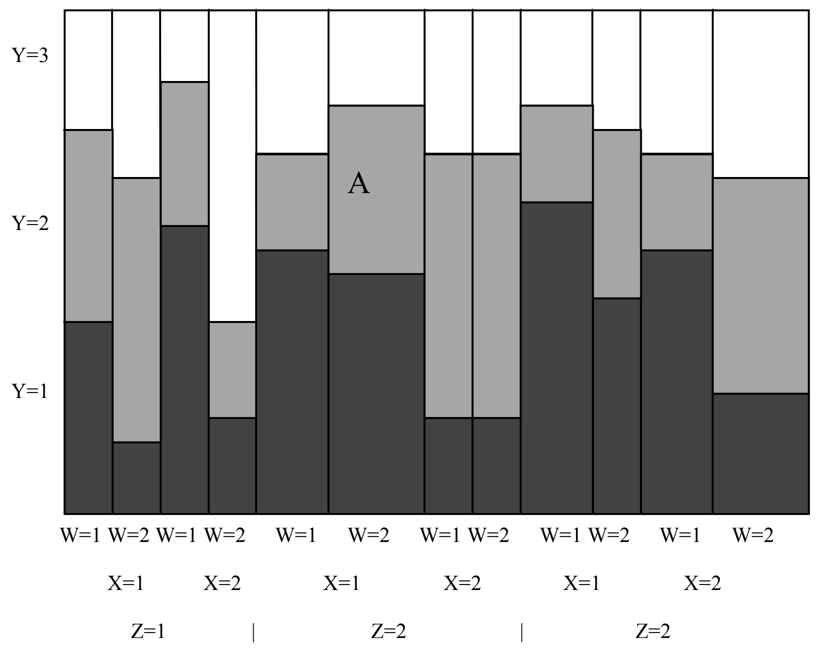


Figure 2: Eikosogram for multiple variables and values

To interpret this graph, for example, the area of rectangle ‘A’ is equal to the joint probability Pr(Y=2,W=2,X=1,Z=2).

2.2 Symmetry among random variables

We can assign any one random variable to the vertical axis and the rest to the horizontal axis. For example, for two variables there are two choices as shown in Figure 3 (a) and (b).

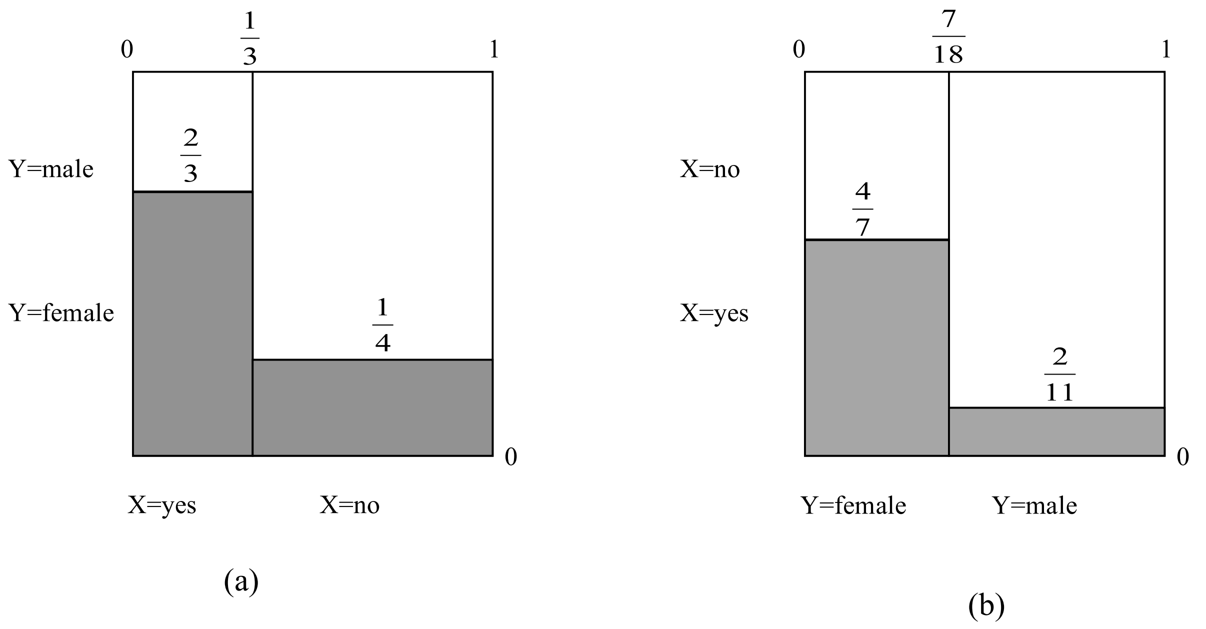


Figure 3: Different perspectives for two variables

Figure 3 (a) is the same as Figure 1 (a); if we assign the random variable X to the vertical axis, we get (b). Once (a) is given, (b) is fixed – all probabilities must be match. See [1] 4.1.3 for more details and the next section for the computation of the probabilities. Similarly for Figure 1(b), we can have the following three eikosograms:

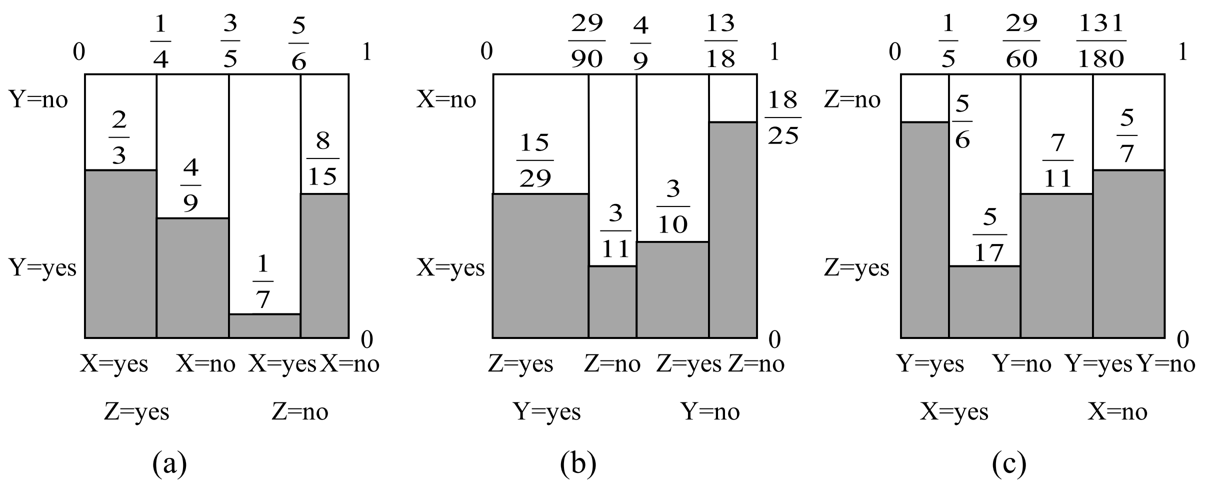


Figure 4: Different perspectives for three variables

Of course for eikosograms of three or more random variables, the order of variables assigned to the horizontal axis is arbitrary. In the software, the number of graphs displayed is equal to the number of random variables involved, with each variable being assigned to the vertical axis once on the order of the horizontal variables. See next section for details.

2.3 Water container metaphor and probabilistic independence

Dependence and independence relationships among random variables are obvious from eikosograms. Flatness across 2 or more rectangles means the equality of some conditional probabilities and hence some kind of conditional probabilistic (or unconditional if completely flat) independence. For example:

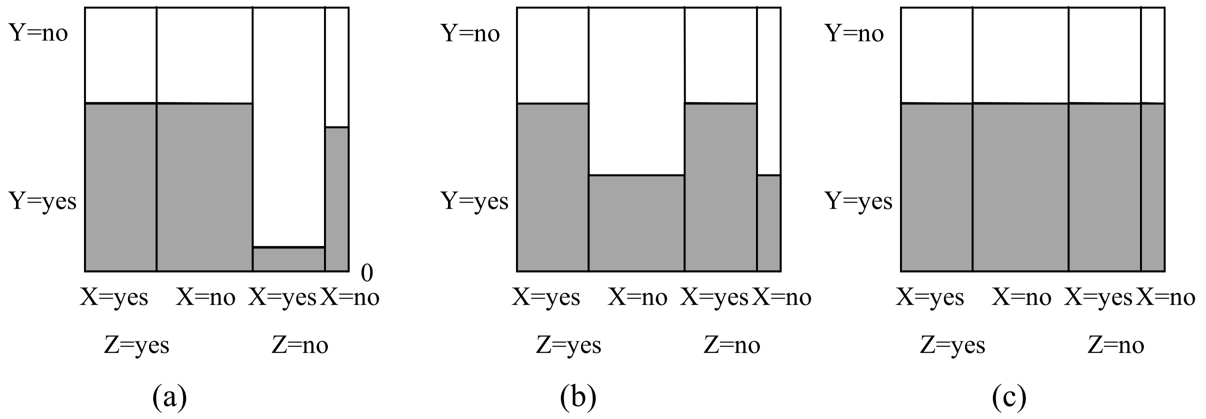


Figure 5: Independence in eikosogram

For Figure 5(a), Pr(Y=yes|X=yes,Z=yes) is equal to Pr(Y=yes|X=no,Z=yes), but Pr(Y=yes|X=yes,Z=no) is not equal to Pr(Y=yes|X=no,Z=no), similarly for Y=no, thus random variable Y is independent of X given Z=yes, but given Z=no, Y and X are dependent.
For Figure 5(b), Pr(Y=yes|X=yes,Z=yes) is equal to Pr(Y=yes|X=yes,Z=no), and Pr(Y=yes|X=no,Z=yes) is equal to Pr(Y=yes|X=no,Z=no), similarly for Y=no, thus random variable Y is independent of Z given X, but given Z, Y and X are dependent.

For Figure 5(c), all conditional probabilities of Y are equal, hence Y is independent of X and Y is independent of Z, but we don't know the relation between X and Z, we cannot say the three random variables are completely independent with each other. See [3] for more details.

We can imagine that an eikosogram is a water container where a barrier exists between any two adjacent vertical strips. When we remove the barrier, the level of water in both containers will become the same and represent some sort of conditional independence. This behavior has been simulated in the software and will be discussed in detail in the next section.

3. Software implementation

Eikosogram software is written in Java. The program reads in data from a disk file and generates graphs . I assume the random variables are categorical , the count is integer. The data input file has the following format:

variable_name1, value1,value2,...,valueN1
variable_name2, value1,value2,...,valueN2

```
…
variable_nameM, value1,value2,...,valueNm
count11,count12,...count1N (where count is integer and N=N2*...*Nm)
…
countk1,countk2,...countkN (where k=N1)
```

For example, the data input file for Figure 1(a) looks like this:

```
Y,female,male
X,yes,no
40,30
20,90
```

List 1.

This file may correspond to the following survey story:
The survey is about the attitude of students in a class to some school policy: the total number of students is 180, 110 male and 70 female; among the male, 20 students support the policy, 90 don't; while 40 female students support it and 30 don't. The following table shows the survey results:

Y \ X	yes	no	total
female	40	30	70
male	20	90	110
total	60	120	180

Table 1.

Data file for Figure 1(b) looks like this:

```
Y,yes,no
Z,yes,no
X,yes,no
210,196,42,112
105,245,252,98
```

List 2.
which corresponds to this table of results:

	Z=yes		Z=no	
	X=yes	X=no	X=yes	X=no
Y=yes	210	196	42	112
Y=no	105	245	252	98

Table 2.

3.1 Grounding probability and generating graphs

The probabilities are calculated by the ratios of the corresponding counts. The following graphs are the same as Figure 3 except the data values in Table 1 are shown in the corresponding rectangles:

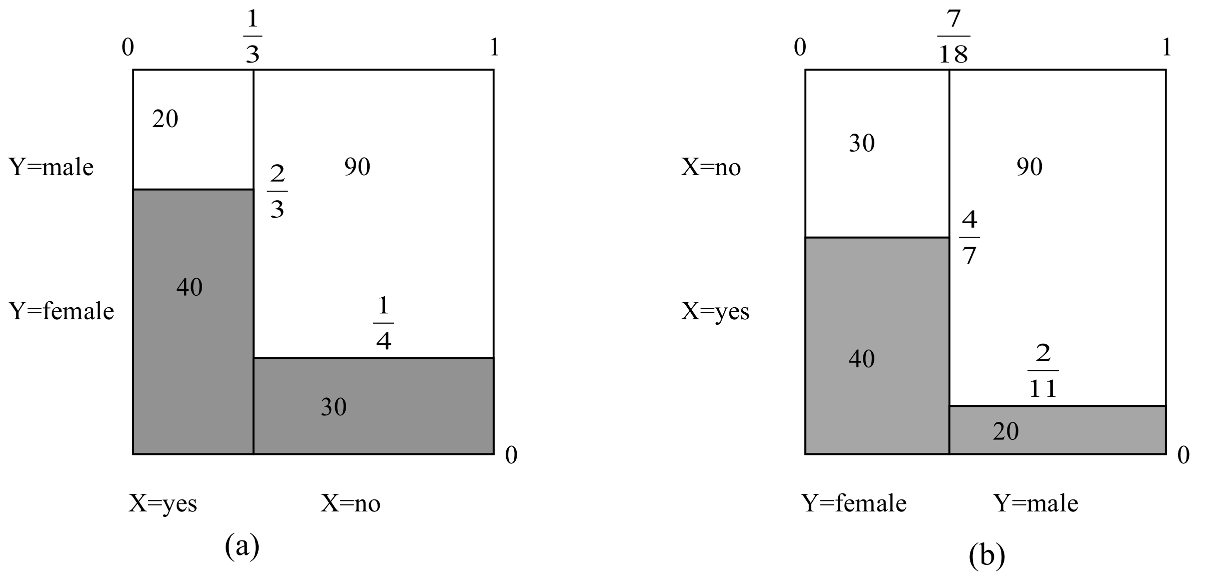


Figure 6: Grounding probabilities for two variables

The total sum is 180. For Figure 6 (a), the marginal probability of X is Pr(X=yes)=(20+40)/180=1/3; the conditional probability of Y given X is Pr(Y=female|X=yes)=40/(40+20)=2/3; similar computations apply for the rest of the probabilities.

The calculations for three or more variables are very similar to the above; the key here is to grab the data from the data input file and put it into the correct rectangle (which corresponds to the correct random variable values). The following graph is the same as Figure 4, but with the labels 1 to 8 in each rectangle. Rectangles that have the same labels correspond to the same probability count and should have the same area. In each graph, label 1 corresponds to the survey count 210, label 2: 196, label 3: 42, label 4: 112, label 5: 105, label 6: 245, label 7: 252, label 8: 98.

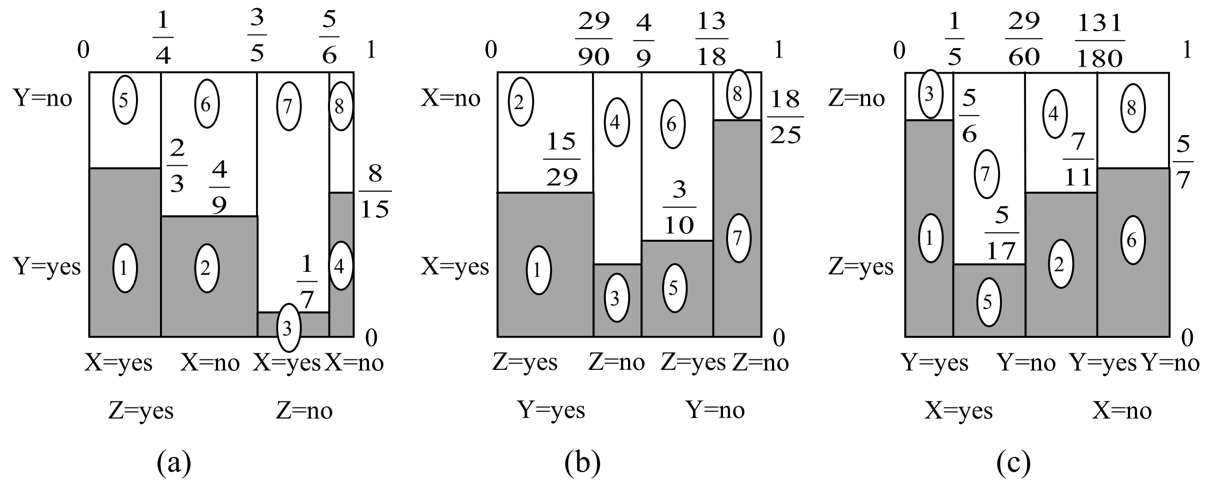


Figure 7: Different perspectives for three variables

It is a bit more involved if the number of variable is more than 3 and the number of values is more than 2, but the basic principles remain the same; to rearrange the rectangles in different perspectives, we identify them by the random variables and the values corresponding to them. The software can deal with any number of variables and any number of values for each. Strictly speaking, this is limited by the screen size and resolution, but very rarely we deal with more than 4 variables and more than three variable values when using eikosograms.

3.2 Symmetry among random variables

3.2.1 Data structure used for storing the raw data

The program uses ArrayList from Java collections framework (part of the java.util package) to store the raw data read from flat disk file. There are some advantages to using this:

(i). Memory efficient: only store the pointer (or reference) in the data structure;

(ii). Algorithm efficient and robust: the code of the collection framework has been extensively tested, and is very professional, thus efficient and succinct.

(iii). Easy and convenient to use: the add and remove item from list are very easy and there is no need to program the reference operations.

In the program, three ArrayLists are used:

1. alVarName: used to store variable names;
2. alVarValue: used to store variable values;
3. alNum: used to store the number of values for each variable;

For example: after reading the data file for figure 1(a), the content of the above ArrayLists are:

Data file(List 1):

```
Y,female,male
X,yes,no
40,30
20,90
```

ArrayList content :

```
alVarName:  Y,X
alVarValue: female,male, yes, no
alNum: 2,2
```

and for Figure 1(b):

Data file(List 2):

```
Y,yes,no
Z,yes,no
X,yes,no
210,196,42,112
105,245,252,98
```

```
alVarName: Y,Z,X
alVarValue: yes,no,yes,no,yes,no
alNum: 2,2,2
```

The alNum serves as a pointer when we grab values from alVarValue for variables in alVarName: for Figure 1(b), the first number '2' in alNum indicates the first 2 values in alVarValue which are 'yes' and 'no' are the values of the first random variable in alVarName which is 'Y', and so on.

The integer counts are stored in an array called 'probCount', the order of those number in the array corresponds to the contents of the above three ArrayLists. The computation and construction of an eikosogram are based on these data structures. E.g. for Figure 3, after generating (a), before generating (b), the content of the above data structures was changed as follow:

	Before generating (a)	After (a), before (b)
alVarName	Y,X	X,Y
alVarValue	female,male,yes,no	yes,no,female,male
alNum	2,2	2,2
probCount	40,30,20,90	40,20,30,90

Table 3.

Similarly for figure 4:

	Before (a)	After (a),before(b)	After(b), before(c)
alVarName	Y,Z,X	X,Z,Y	Z,Y,X
alVarValue	yes,no,yes,no,yes,no	yes,no,yes,no,yes,no	yes,no,yes,no,yes,no
alNum	2,2,2	2,2,2	2,2,2
probCount	210,196,42,112,105,245,252,98	210,42,105,252,196,112,245,98	210,105,196,245,42,252,112,98

Table 4.

The algorithm used is pretty straightforward; in the ArrayList, we circulate the contents, the number of steps of circulation being based on the content of ‘alNum’. The change of ‘probCount’ is a bit more complicated because the algorithm needs to deal with any number of variables each having any number of values. It becomes trivial after careful study of the situation. This algorithm is implemented in the method ‘reCalProbCount()’ in the class ‘Eikosogram’ of the source code file.

3.3 Water container metaphor and probabilistic independence

The simulation of the water container metaphor to form some sort of probabilistic independence is the main function of the program. The barriers are represented by small buttons on the top of the graph, like this:

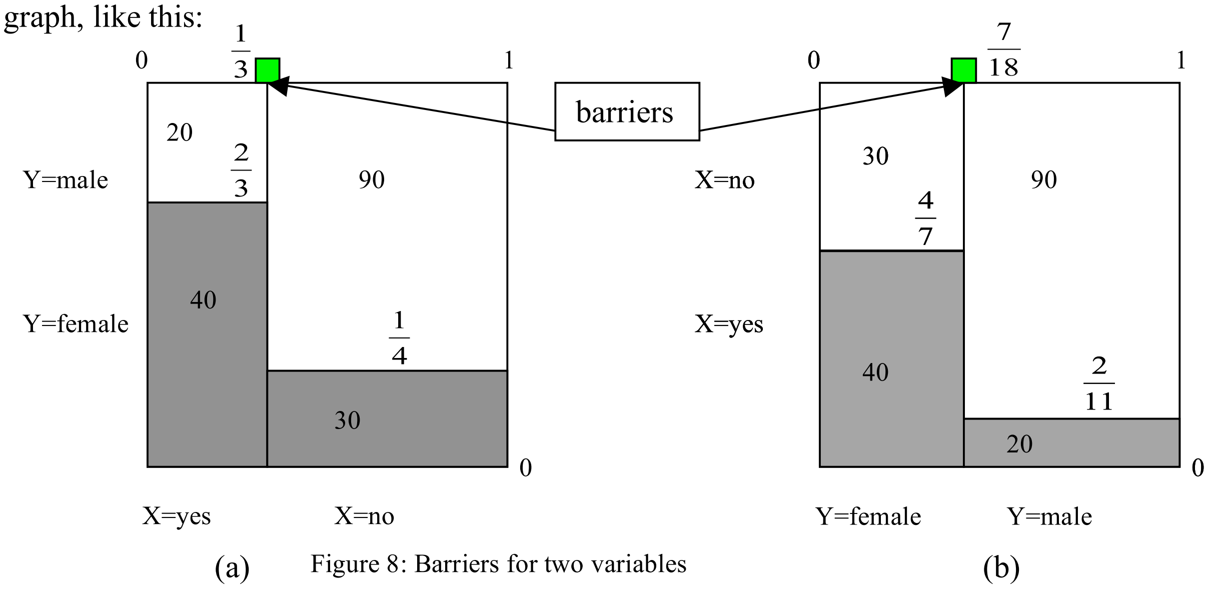


Figure 8: Barriers for two variables

When we click one of the buttons to remove the barrier, several situations need to be considered:

3.3.1 None of the other barriers ever clicked before

This is the easiest situation, i.e. the first time we click a barrier after the eikosogram is generated, we only need to consider the two vertical 'chambers' separated by the barrier clicked and, of course, all perspectives of the eikosogram need to be recalculated. The following are two examples:

Example 1. (See Figure 8) Click the barrier in Figure 8 (a): the rule here is to keep the marginal of X fixed, the sum of the corresponding data counts does not change. i.e. for Figure 9(a), the sum of the data count of the left strip is still 40+20=60, the sum of the lower two rectangles is still the 40+30=70, this simulates the behavior of removing the barrier from the water container, with the amount of water in the chambers separated by the barrier staying the same. To adjust the water level to be the same, the computation is:

Data count for Y=female, X=yes changed from 40 to (40+30)/3=70/3;
Data count for Y=female, X=no changed from 30 to (40+30)*2/3=140/3;
Data count for Y=male, X=yes changed from 20 to (20+90)/3=110/3;
Data count for Y=male, X=no changed from 90 to (20+90)*2/3=220/3;

This change must happen in the corresponding rectangles in (b) also. Recalculating the probabilities based on the changed data count, we get the following eikosograms after clicking the barrier in Figure 8(a) to arrive at the equal water levels as in Figure 9 (a):

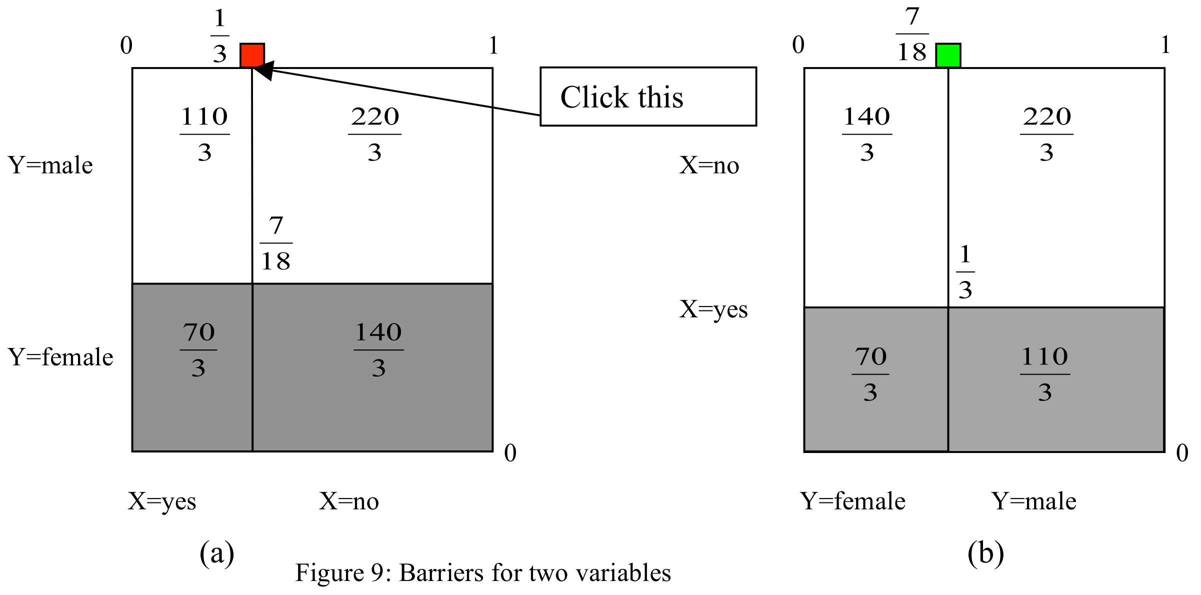


Figure 9: Barriers for two variables

The calculation of the probability is the same as before, e.g. Pr(Y=femal|X=yes)=(70/3)/(70/3+110/3)=7/18, etc. This is exactly what happened in the program. Clicking the barrier in (b) generates the same graph based on a similar analysis.

Example 2. Figure 10 is the same as figure 7 & 4, the data counts correspond to each rectangle are listed in Table 5.

	1	2	3	4	5	6	7	8
data count	210	196	42	112	105	245	252	98

Table 5.

The total is 1260.

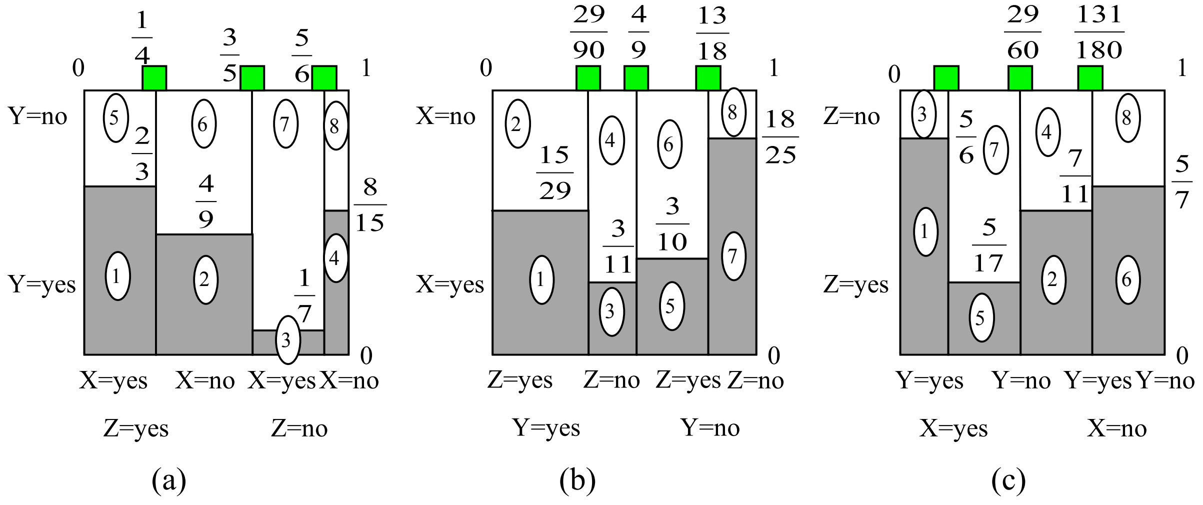


Figure 10: Eikosograms for three variables before removing barrier

click the left barrier in Figure 10 (a), the rectangles labeled 1,2,5,6 in (a)(b)(c) need to be recalculated. In (a). The recalculation is similar to example 1 but with an important little difference (pay attention to the '$\div \frac{3}{5}$' part):

Data count for Y=yes,X=yes,Z=yes (rectangle 1) changed from 210 to $(210+196)\times\frac{1}{4}\div\frac{3}{5}=\frac{1015}{6}$;

Data count for Y=yes,X=no,Z=yes (rectangle 2) changed from 196 to $(210+196)\times(\frac{3}{5}-\frac{1}{4})\div\frac{3}{5}=\frac{1421}{6}$;

Data count for Y=no,X=yes,Z=yes (rectangle 5) changed from 105 to $(105+245)\times\frac{1}{4}\div\frac{3}{5}=\frac{875}{6}$

Data count for Y=no,X=no,Z=yes (rectangle 6) changed from 245 to $(105+245)\times(\frac{3}{5}-\frac{1}{4})\div\frac{3}{5}=\frac{1225}{6}$

The calculation formula is readily derived from the rules discussed in Example 1, i.e. for graph (a) (on which the barrier was removed), keep the marginal of the joint distribution of X and Z fixed, and keep the sum of data account in corresponding rectangles fixed, which simulates the amount of water in tanks not changing when we remove barrier.

We summarize the new data distribution in the Table 6:

	1	2	3	4	5	6	7	8
data count	1015/6	1421/6	42	112	875/6	1225/6	252	98

Table 6.

Based on these data, recalculate all probabilities and regenerate all (a),(b),(c), we get

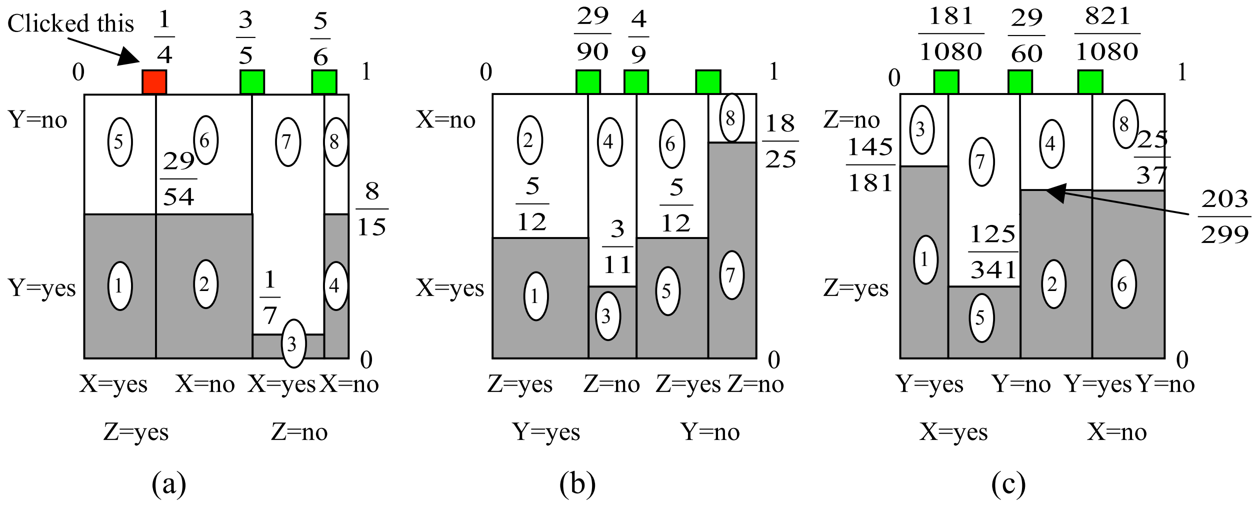


Figure 11: Eikosograms for three variables after removing the left barrier

Note that since the sum of the data count of rectangles 1 & 2, 5&6 , 1&5 and 2&6 don't change, the joint probabilities of X & Z and Z & Y (Figure 11 (a) and (b)) don't change, but the joint probability of Y and X does change since rectangle 1 changed and 3 didn't. E.g. Originally, $\Pr(Y = yes, X = yes) = \frac{210+42}{1260} = \frac{1}{5}$; after the click, rectangle 1 changed from 210 to 1015/6 and

$$\Pr(Y = yes, X = yes) = \frac{\frac{1015}{6}+42}{1260} = \frac{181}{1080};$$

Also note that in the last two strips of (c) , i.e. rectangle 2,4,6,8,

$$\Pr(Z = no \mid Y = yes, X = no) = \frac{\frac{1421}{6}}{\frac{1421}{6}+112} = \frac{203}{299} \approx .6789 \quad \text{and}$$

$$\Pr(Z = no \mid Y = no, X = no) = \frac{\frac{1225}{6}}{\frac{1225}{6} + 98} = \frac{25}{37} \approx .6757$$

The computer screen can not distinguish this difference, so it shows some 'flatness', but we cannot infer independence of Z and Y given X=no; this small 'drawback' is due to the equipment we used to display the graph, not to the eikosogram itself. The software implementation includes a button that can be used to switch between fraction and decimal display, because sometimes the fraction contains a huge numerator or denominator (see Examples in 3.3.3). Due to the above analysis, before making any conclusion of independence, we need to check the fractional display.

3.3.2 Some barriers have been removed already

Obviously the barriers have to be adjacent to the one we are about to click, otherwise we don't need to consider it since it does not affect what we are going to do. Let's continue the Example 2 in the last section. We now remove the middle barrier in (a). Since the left barrier has been removed already, the water tanks of 1,2,5 & 6 have to be considered as the same container. Again we don't change the joint distribution of X & Z in (a) and keep the sums of correspondent data counts the same as before. The rectangles 1,2,3,5,6,7 need to be recalculated (the data before this action is in Table 6 now):

Data count for Y=yes,X=yes,Z=yes (rectangle 1) changed from $\frac{1015}{6}$ to $(\frac{1015}{6} + \frac{1421}{6} + 42) \times \frac{1}{4} \div \frac{5}{6} = \frac{672}{5}$;

Data count for Y=yes,X=no,Z=yes (rectangle 2) changed from $\frac{1421}{6}$ to $(\frac{1015}{6} + \frac{1421}{6} + 42) \times (\frac{3}{5} - \frac{1}{4}) \div \frac{5}{6} = \frac{4704}{25}$;

Data count for Y=yes,X=yes,Z=no (rectangle 3) changed from 42 to $(\frac{1015}{6} + \frac{1421}{6} + 42) \times (\frac{5}{6} - \frac{3}{5}) \div \frac{5}{6} = \frac{3136}{25}$;

Similar calculations apply for rectangle 5, 6, 7. The new data counts are summarized in table 7:

	1	2	3	4	5	6	7	8
data count	672/5	4704/25	3136/25	112	903/5	6321/25	4214/25	98

Table 7.

Based on these data, recalculating all probabilities and regenerating all (a), (b), (c), we get

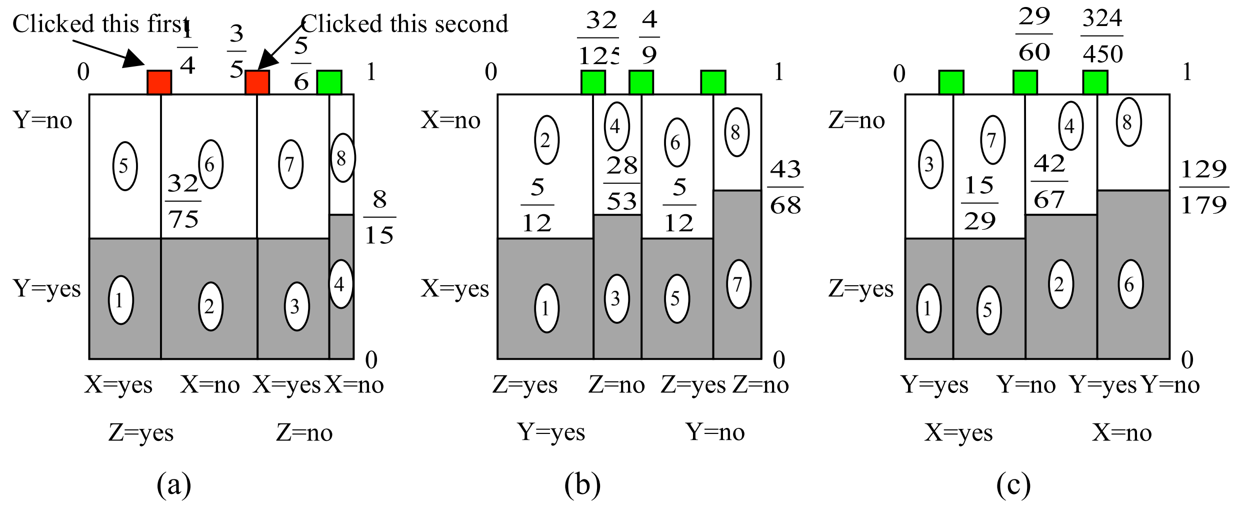


Figure 12: Eikosograms for three variables after removing the middle barrier

It's easy to verify the results using the same calculation as before applied to the data in Table 7.

If we remove the right barrier second, the calculation will be the same as 3.3.1 since the middle barrier still exists. If we remove the right barrier second and the middle one third, the calculation will be similar to the last example, only this time the rectangles 3,4,7,8 also need to be considered as the same tank.

Data count after removing left and right barriers in (a):

	1	2	3	4	5	6	7	8
data count	1015/6	1421/6	539/6	385/6	875/6	1225/6	1225/6	875/6

Table 8.

Data count after removing all barriers in (a):

	1	2	3	4	5	6	7	8
data count	140	196	392/3	280/3	175	245	490/3	350/3

Table 9.

It's easy to verify the probabilities in Figure 13 from the data in Table 8 and Figure 14 from Table 9.

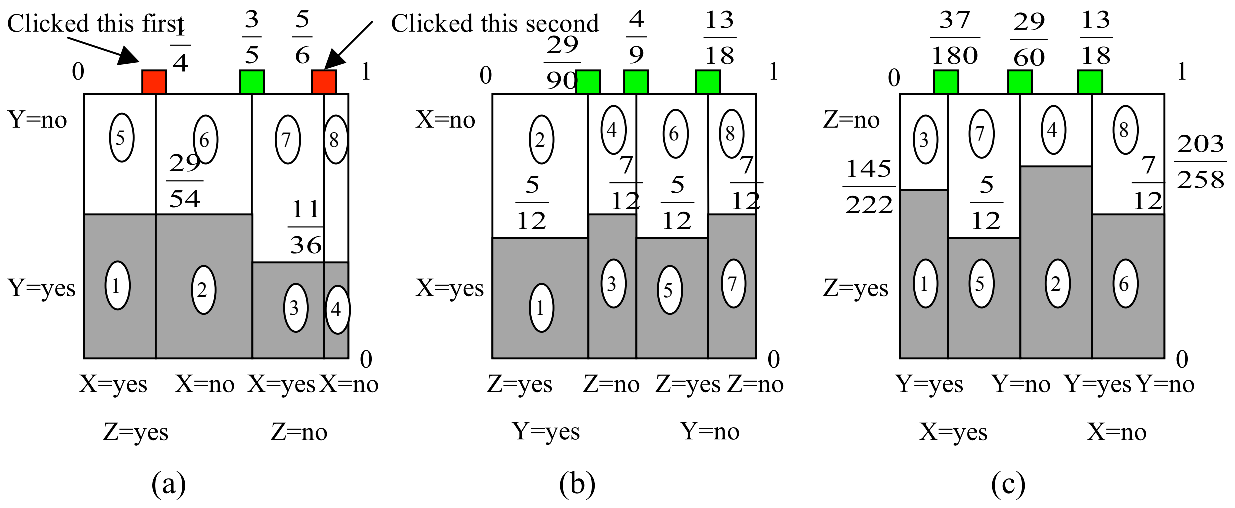


Figure 13: Eikosograms for three variables after removing the right barrier

We can easily see from Figure 13(a) that given Z, Y and X are independent; this relation is also shown in (b). No other independence relationships exist.

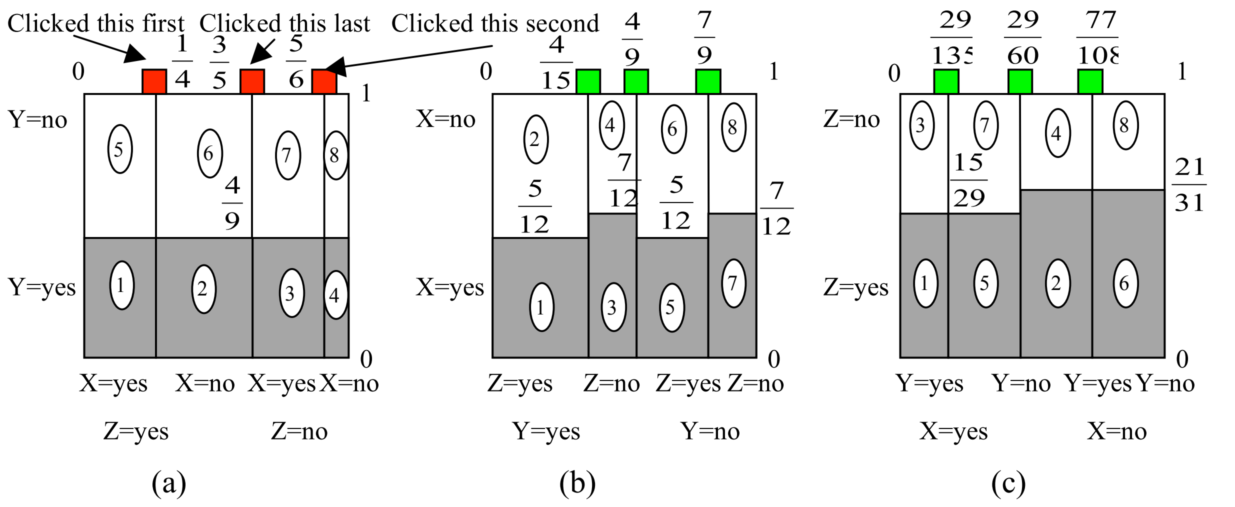


Figure 14: Eikosograms for three variables after removing all barriers in (a)

From Figure 14, we can see that, Y and X are independent given Z, which is also shown in (b); Y and Z are independent given X, shown in (c); No other independence relationships exist.

3.3.3 Remove barriers from a different container

In Figure 11, reproduced here as Figure 15(data as in Table 10), has the left barrier removed in (a). We now consider what happens when we follow this by removing the left barrier in (b). This removal affects the rectangles 1 & 2 but not 5 & 6.

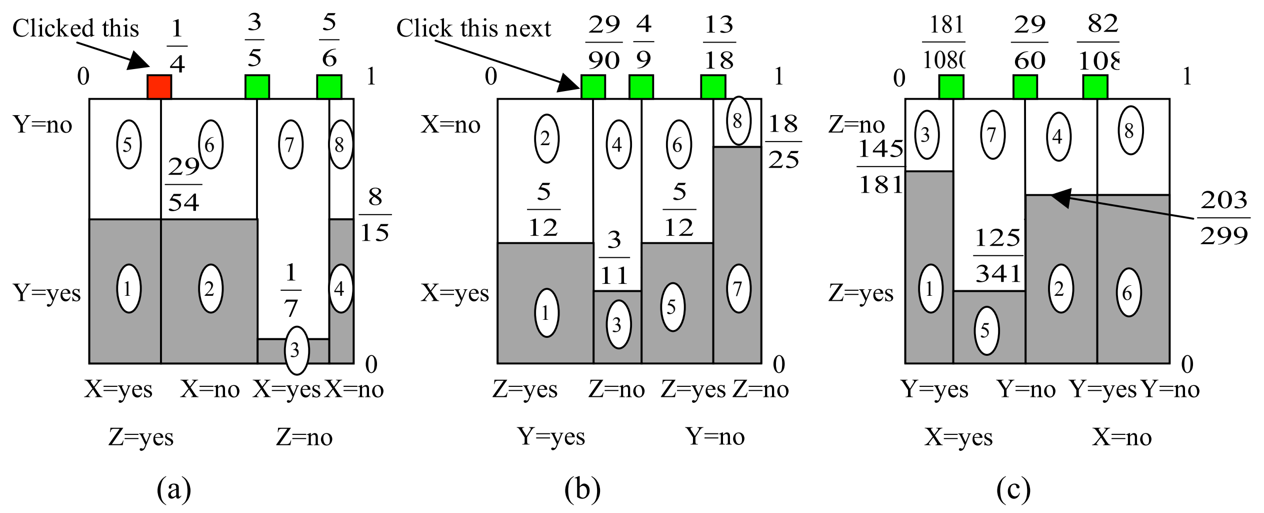


Figure 15: Eikosograms for three variables removing barriers from different graphs

	1	2	3	4	5	6	7	8
data count	1015/6	1421/6	42	112	875/6	1225/6	252	98

Table 10. Data after removing the left barrier in (a):

We apply the same method when we remove the barrier in (a). Then rectangles 1,2,3,4 will be changed in area but not 5,6, and so the flatness of rectangles 1,2 in (a) will be broken. Figure 16 shows the result:

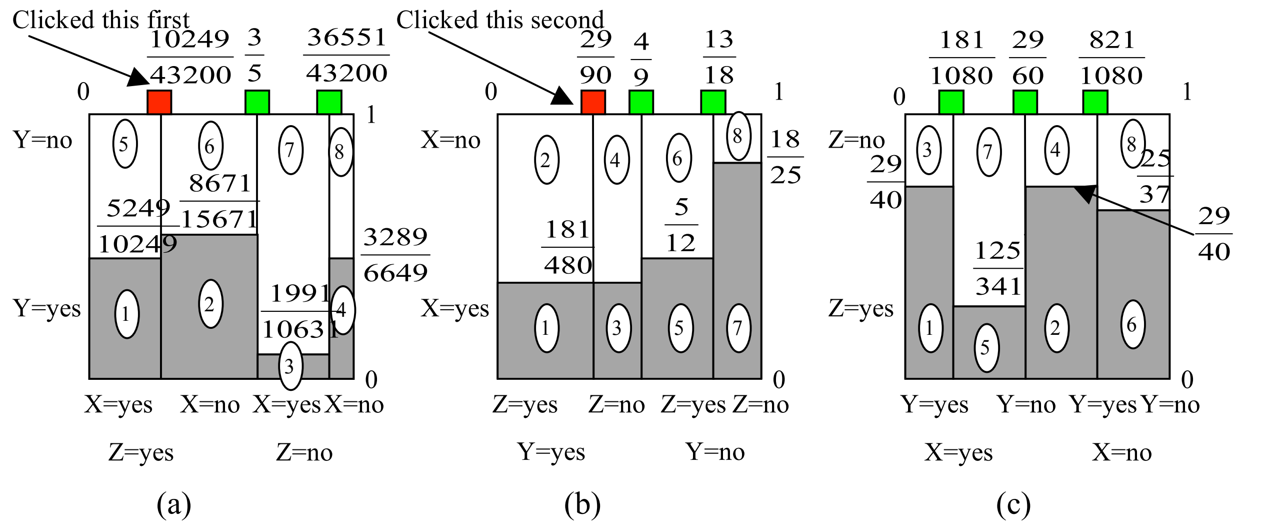


Figure 16: Eikosograms for three variables removing barriers from different graphs, as implemented

The data in each rectangle after removing the left barrier in (b) would then be as in Table 11:

	1	2	3	4	5	6	7	8
data count	36743/240	60697/240	13937/240	23023/240	875/6	1225/6	252	98

Table 11.

While simple, this approach has a significant drawback. Because the left barrier has already been removed, this was asserting a conditional independence and it would be best if this were preserved, that is the flatness should still be there after the action in (b). In other words, from a modeling perspective, it might be reasonable to keep the independence of Y and X given Z= yes when we further assert the independence of X and Z given Y=yes. Since the sum of rectangles 1&2 does not change after we remove the left barrier in (b), if we keep the flatness in (a), the position of the left barrier (which has been removed already) in (a) has to be changed, i.e. we have to adjust the joint distribution of X and Z for Z=yes. This will cause the change of rectangle 5&6 in all three eikosograms. For example, when we remove the left barrier in (b), not only the rectangle 1,2,3,4 will be adjusted, but also 5&6. This side effect is caused by our intention to keep the independence of Y and X given Z=yes. Here are some calculations and results:
We need to recalculate the joint probability of X=yes,Z=yes and data count of rectangle 5&6. (The sum of 1&2 is 406)

$\Pr(X = yes, Z = yes) = \frac{36743}{240} \times \frac{3}{5} \times \frac{1}{406} = \frac{181}{800}$;

data count in rectangle 5 is $350 \times \frac{181}{800} \times \frac{5}{3} = \frac{6335}{48}$;

data count in rectangle 6 is $350 \times (\frac{3}{5} - \frac{181}{800}) \times \frac{5}{3} = \frac{10465}{48}$

$\frac{36551}{43200}$ $\frac{29}{90}$ $\frac{4}{9}$ $\frac{13}{18}$ $\frac{18}{10}$ $\frac{4081}{8640}$ $\frac{64}{86}$

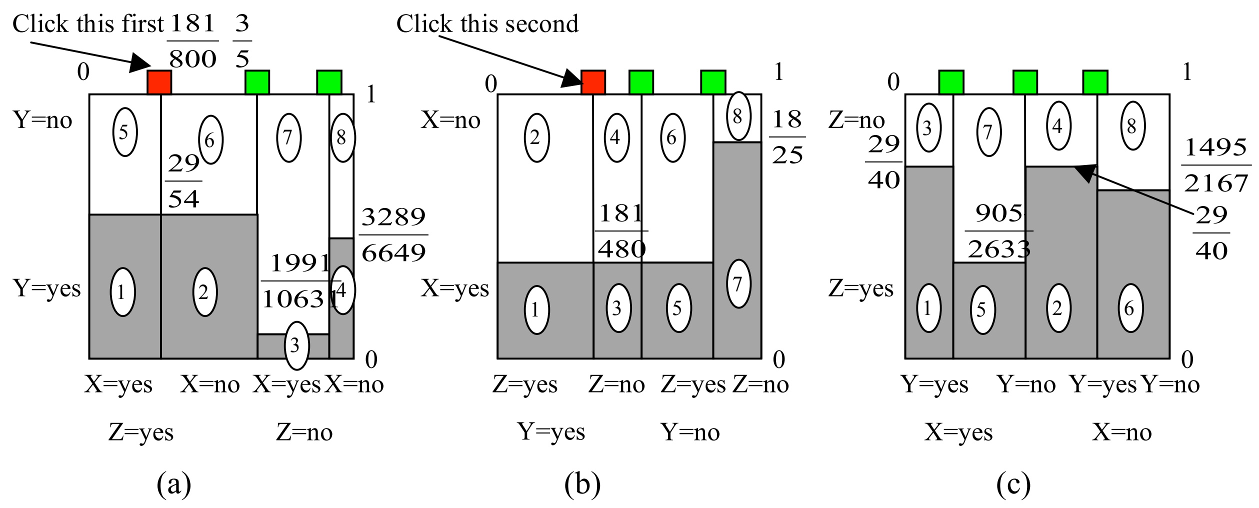


Figure 17: Eikosograms for three variables removing barriers from different graphs

The data count in each rectangle under the new independence restriction:

	1	2	3	4	5	6	7	8
data count	36743/240	60697/240	13937/240	23023/240	6335/48	10465/48	252	98

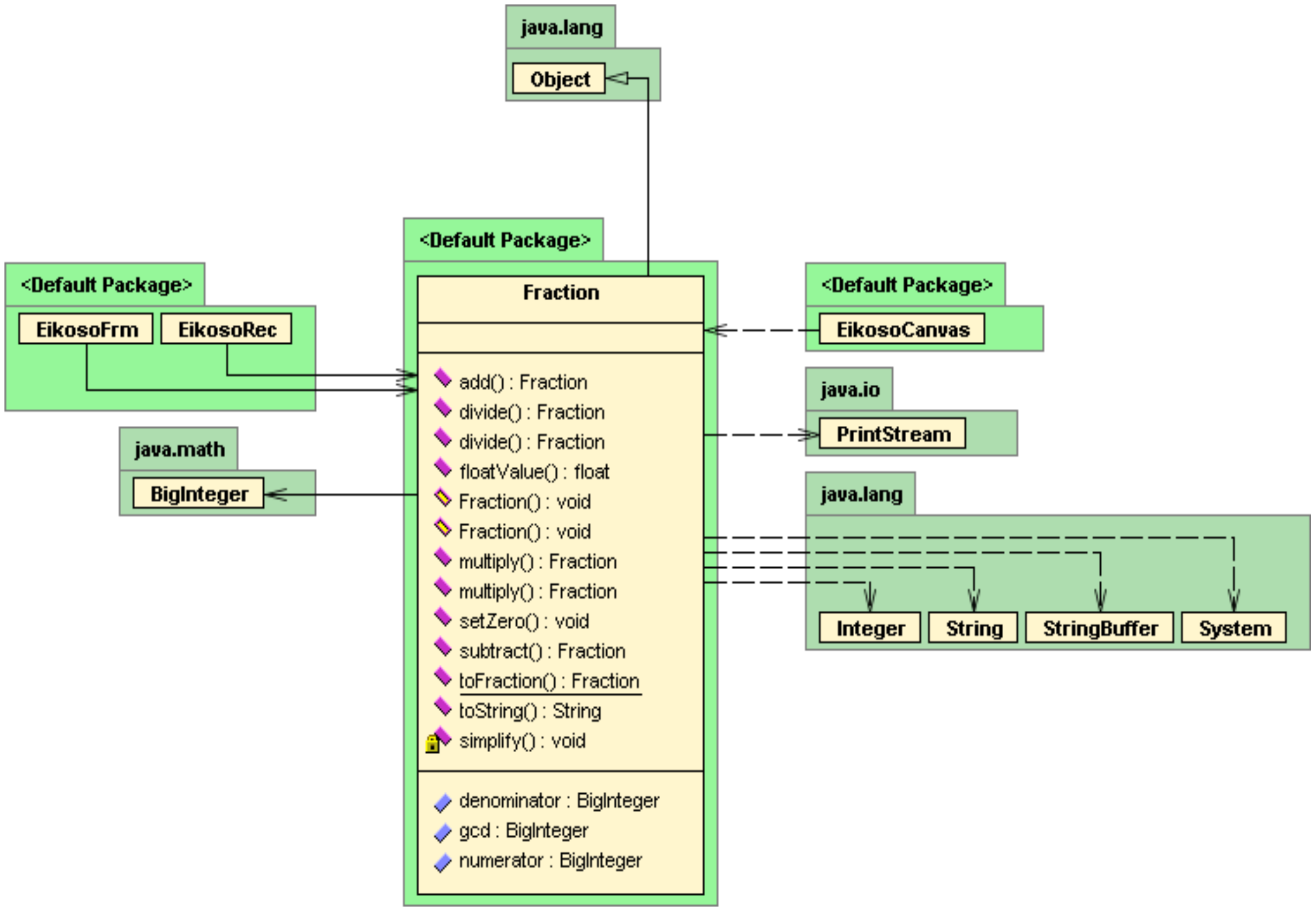
Table 12.

Now the problem is: Can we implement this type of behaviour in software assuming the user can remove (or reinstall) any barrier at any time? For example: continuing with Figure 17, what if the user removes the right barrier in (a), which will cause a change of 3,4,7,8 in (a), but in (b), since the left barrier has been removed already, the change in 3,4 will cause a change in 1,2 and that is related to 5,6 because of (a). How about we remove a barrier in (c) next? What about the eikosogram for four random variables and more than two values each (Figure 2)?

These problems remain open. The software implemented here always assumes that conditional independences are not necessarily preserved in one eikosogram when some are asserted (barriers removed) in another.

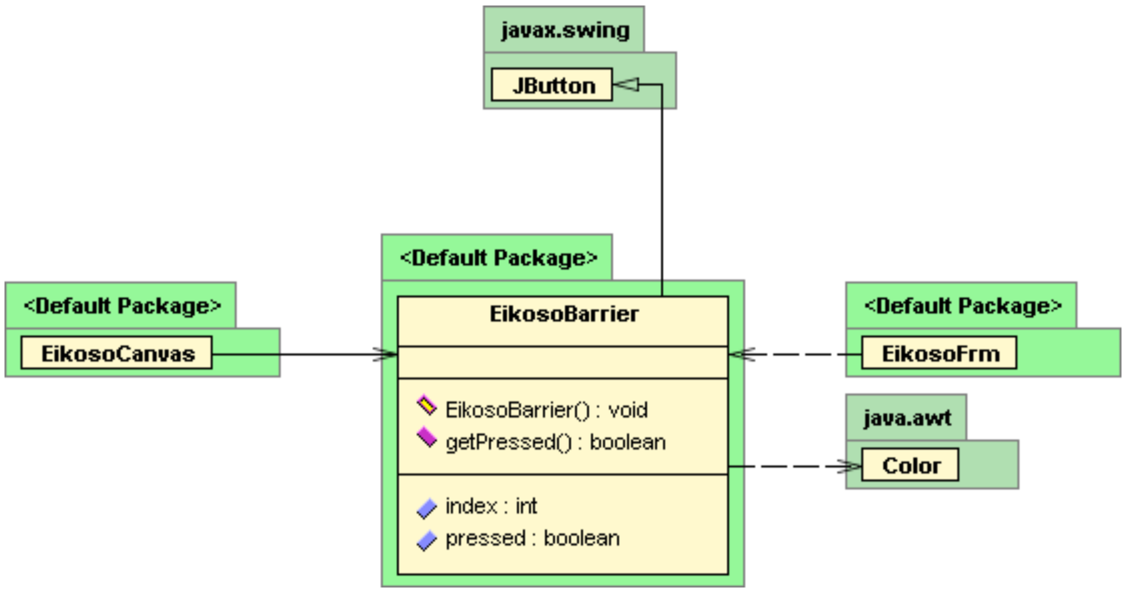
3.4 Software Structure

The software is written in Java and includes 6 classes: Eikosogram (main function), EikosoCanvas, EikosoRec, EikosoBarrier, EikosoInternalFrm and Fraction. All calculations in section 3.3 use rational arithmetic to avoid rounding errors. The **Fraction** class defines the rational computation and the method to display the fraction on the screen. Here is the class structure using UML:

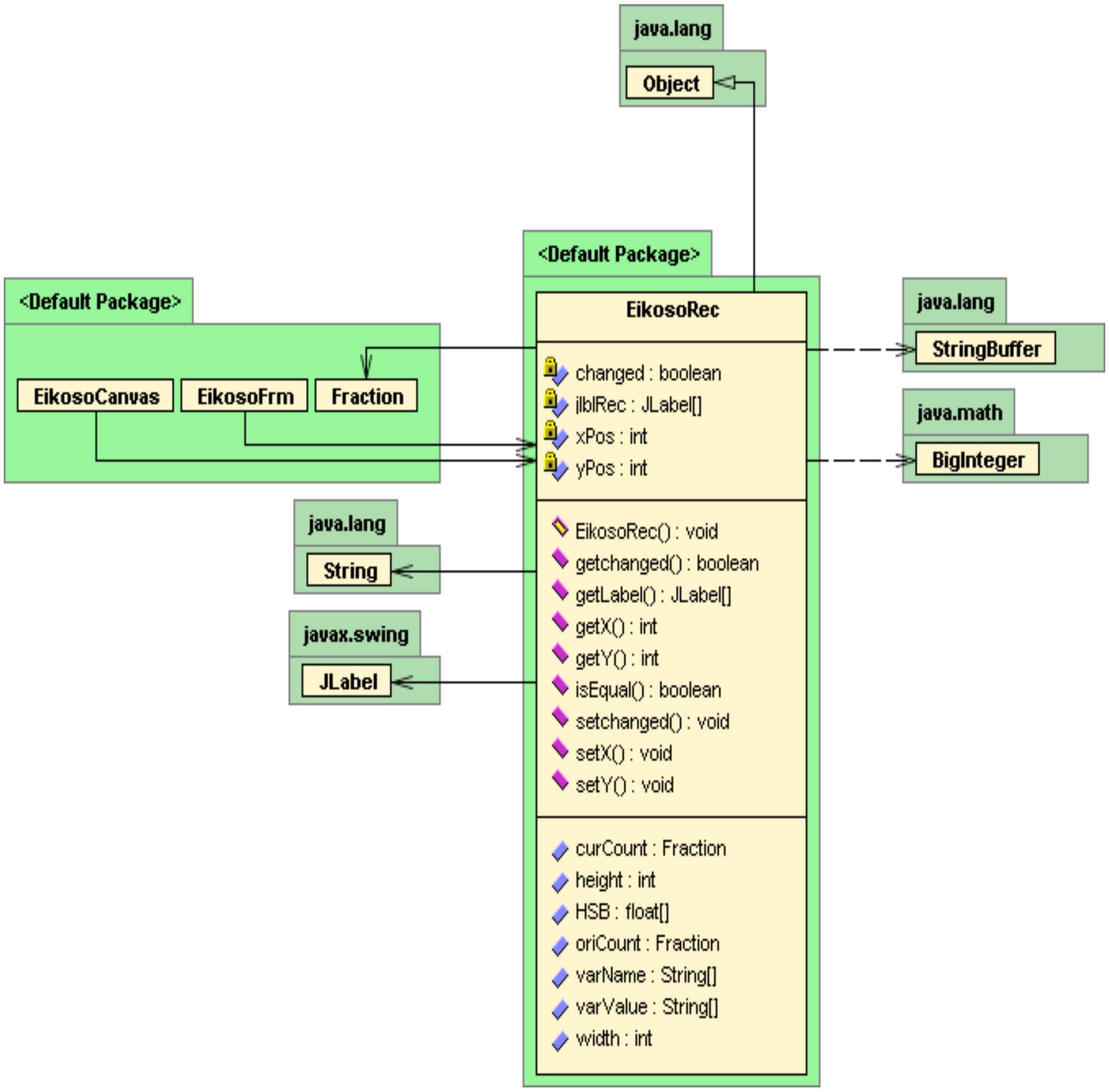


The key component of this class is the gcd (greatest common divisor); gcd is calculated using the gcd method of BigInteger class java.math.

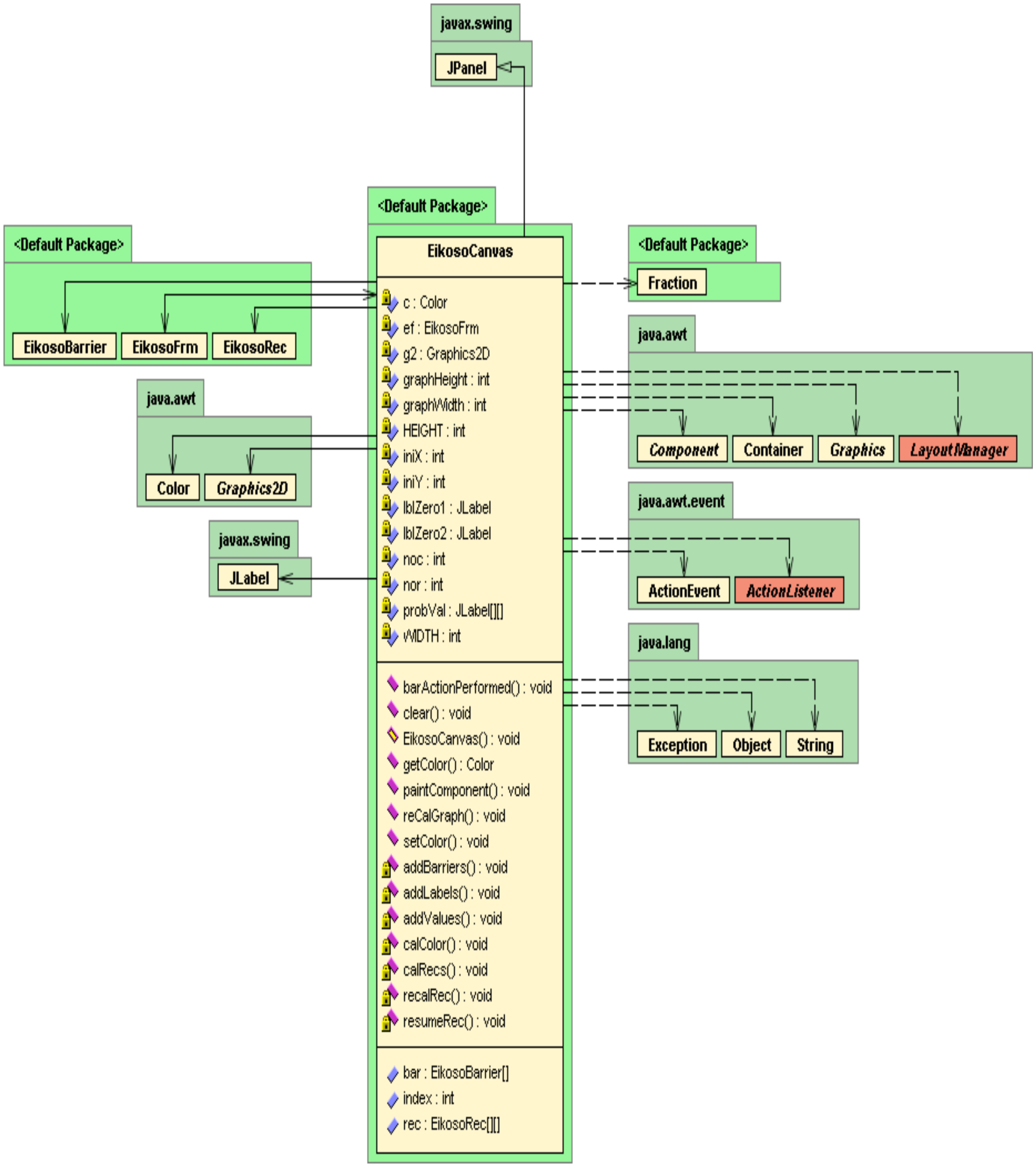
EikosoBarrier class defines characteristics and the basic behaviours of the barriers in each graph. Since it is clickable, this class is derived from the java swing JButton. The index field identifies the barrier on the graph and, if 'pressed', the colour of this barrier will change to red, otherwise the colour is green (may not true on mac or unix).



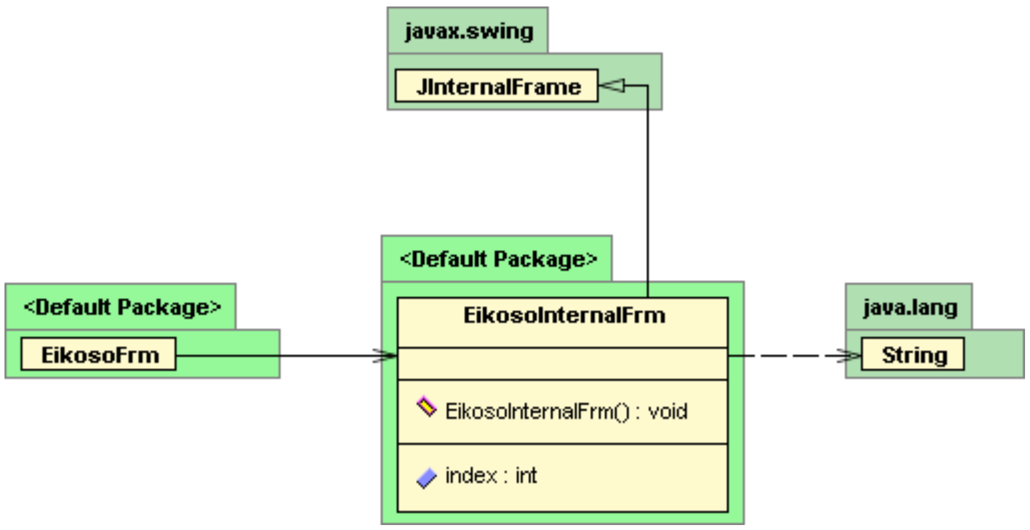
EikosoRec defines characteristic and behaviours of each small rectangle in the eikosogram. Each rectangle has a size given by its width and height and a position given by the left top coordinate: xPos and yPos. Also, the characteristic contains the colour or shade given by the HSB values and the data count corresponding to this rectangle. The data count includes an original one when the user first input the data so that the user can reinstall the barrier to resume the original graph, and a current data count calculated when the user removed a barrier. The rectangle is identified by the random variable value corresponding to it.



EikosoCanvas is the most important class of this application; all the calculations discussed in the last section of this paper are in this class. Each 'canvas' holds one graph which includes a two dimensional array of 'rectangles' such as Figure 17 (a). To display Figure 17, we need to generate three such canvas objects. Besides the rectangles, the 'canvas' also contains some barriers. The call back function 'barActionPerformed()' defines the actions after the user removes or reinstalls a barrier which in turn calls the 'reCalRec()' or 'resumeRec()' method to do the calculations mentioned before.



EikosoInternalFrm is a placeholder to hold ‘EikosoCanvas’, it only has one characteristic ‘index’ which identifies itself.



Eikosogram is the main function of this java application, it defines some basic functions such as changing colours and reading the data file to generate the original graph. The 'reCalProbCount()' method defines the calculations which generate different perspectives of an eikosogram. The 'reCalCanvas()' method is called by the EikosoCanvas method ' recalRec() ' to do the calculation on other perspectives of the graph when the user removes a barrier. The resize behaviour of the main frame is also defined here. (UML next page)

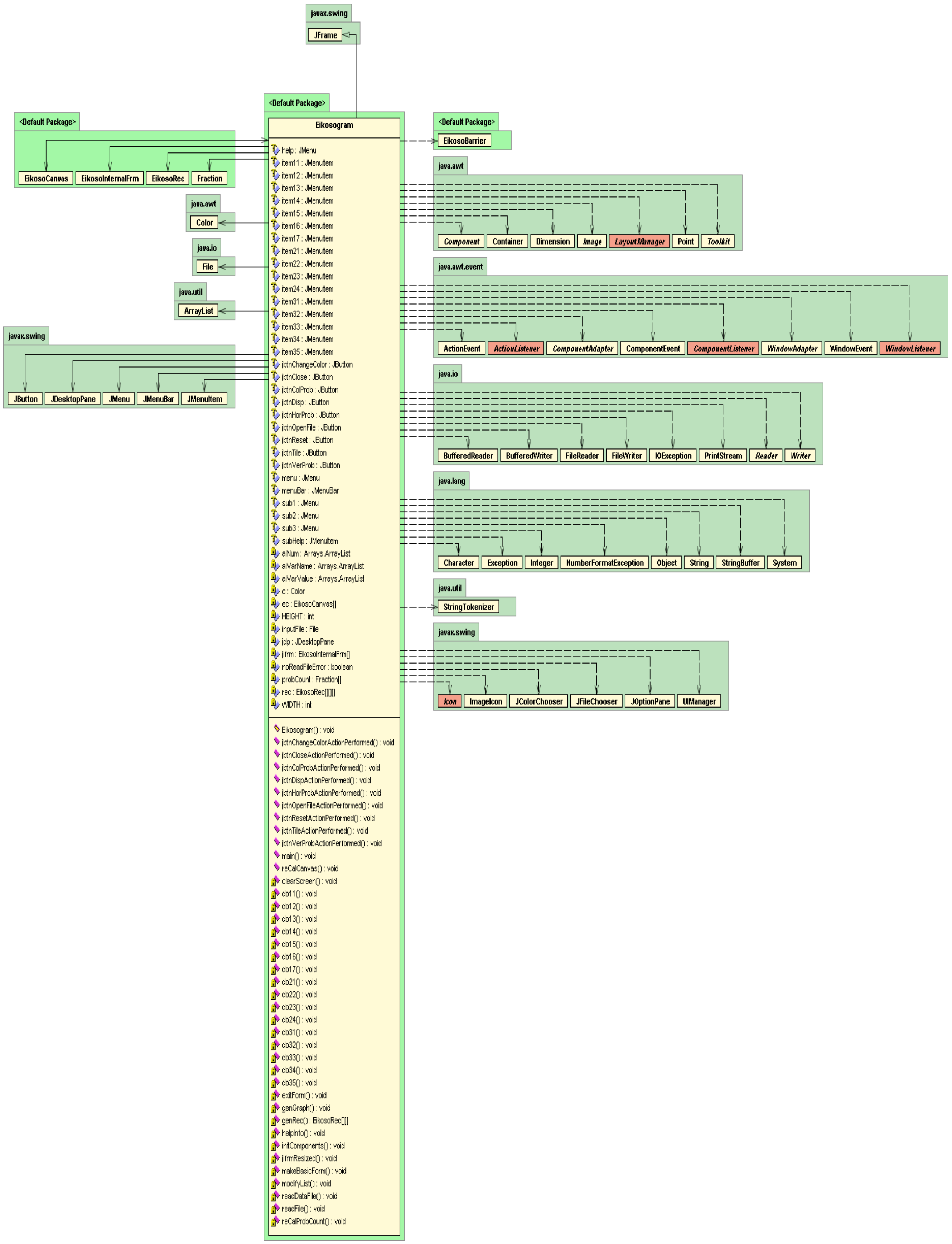
See appendix for the documents of these classes.

3.5 Examples in the software

The following examples are included in the software:
(i). Binary Association including: Perfect positive association; Part perfect positive association; Positive association; Independence; Negative association; Part perfect negative association; Perfect negative association;
(ii). Relating Events including: Coincident; Complementary; Mutually Exclusive; Independent;
(iii). Independence including: Y indep. X; Y indep. X given Z; Y indep. X given Z value; Not quite 3 way independence; 3 way independence;
For the details of these examples, see [1] section 4.3

4. Future work

The current version of the software has implemented everything until 3.3.3. It does not maintain user defined conditional independencies across graphs. That is when a barrier is removed from one eikosogram, the results do not respect barriers previously removed in other eikosograms. The future work of the software development therefore includes maintaining all the previous independence relation constraints when we remove a barrier. We may also study and compare the results of applying different probability models to analyze contingence table and the 'shape' of data visualized by eikosogram.


javax.swing
JFrame
<Default Package>
Eikosogram
help : JMenu
item11 : JMenuItem
item12 : JMenuItem
item13 : JMenuItem
item14 : JMenuItem
item15 : JMenuItem
item16 : JMenuItem
item17 : JMenuItem
item21 : JMenuItem
item22 : JMenuItem
item23 : JMenuItem
item24 : JMenuItem
item31 : JMenuItem
item32 : JMenuItem
item33 : JMenuItem
item34 : JMenuItem
item35 : JMenuItem
jbtnChangeColor : JButton
jbtnClose : JButton
jbtnColProb : JButton
jbtnDisp : JButton
jbtnHorProb : JButton
jbtnOpenFile : JButton
jbtnReset : JButton
jbtnTile : JButton
jbtnVerProb : JButton
menu : JMenu
menuBar : JMenuBar
sub1 : JMenu
sub2 : JMenu
sub3 : JMenu
subHelp : JMenuItem
alNum : Arrays.ArrayList
alVarName : Arrays.ArrayList
alVarValue : Arrays.ArrayList
c : Color
ec : EikosoCanvas[]
HEIGHT : int
inputFile : File
jdp : JDesktopPane
jifrm : EikosoInternalFrm[]
noReadFileError : boolean
probCount : Fraction[]
rec : EikosoRec[][][]
WIDTH : int
Eikosogram() : void
jbtnChangeColorActionPerformed() : void
jbtnCloseActionPerformed() : void
jbtnColProbActionPerformed() : void
jbtnDispActionPerformed() : void
jbtnHorProbActionPerformed() : void
jbtnOpenFileActionPerformed() : void
jbtnResetActionPerformed() : void
jbtnTileActionPerformed() : void
jbtnVerProbActionPerformed() : void
main() : void
reCalCanvas() : void
clearScreen() : void
do11() : void
do12() : void
do13() : void
do14() : void
do15() : void
do16() : void
do17() : void
do21() : void
do22() : void
do23() : void
do24() : void
do31() : void
do32() : void
do33() : void
do34() : void
do35() : void
exitForm() : void
genGraph() : void
genRec() : EikosoRec[][]
helpInfo() : void
initComponents() : void
jifrmResized() : void
makeBasicForm() : void
modifyList() : void
readDataFile() : void
readFile() : void
reCalProbCount() : void
<Default Package>
EikosoCanvas
EikosoInternalFrm
EikosoRec
Fraction
java.awt
Color
java.io
File
java.util
ArrayList
javax.swing
JButton
JDesktopPane
JMenu
JMenuBar
JMenuItem
<Default Package>
EikosoBarrier
java.awt
Component
Container
Dimension
Image
LayoutManager
Point
Toolkit
java.awt.event
ActionEvent
ActionListener
ComponentAdapter
ComponentEvent
ComponentListener
WindowAdapter
WindowEvent
WindowListener
java.io
BufferedReader
BufferedWriter
FileReader
FileWriter
IOException
PrintStream
Reader
Writer
java.lang
Character
Exception
Integer
NumberFormatException
Object
String
StringBuffer
System
java.util
StringTokenizer
javax.swing
Icon
ImageIcon
JColorChooser
JFileChooser
JOptionPane
UIManager

References:

[1]. Picturing Probability: the poverty of Venn diagrams, the richness of Eikosograms. W.H.Cherry and R.W. Oldford, University of Waterloo August 7,2003
[2]. Probability, problems, and paradoxes pictured by eikosograms. R.W.Oldford, University of Waterloo August 7, 2003
[3]. Understanding probabilistic independence and its modeling via Eikosograms and graphs. R.W.Oldford, University of Waterloo August 7, 2003

Appendix: Eikosogram software documents

Class EikosoCanvas

public class **EikosoCanvas**
extends javax.swing.JPanel

Title: EikosoCanvas.java

Description:

Copyright: Copyright (c) 2005

Company: university of waterloo

Version:
1.0
Author:
glenn lee

Constructor Detail

EikosoCanvas

```
public EikosoCanvas(EikosoRec[][] rec,
                    int index,
                    Eikosogram ef)
```

Create an EikosoCanvas object with rec.
Parameters:
`rec` - two dimensional EikosoRec on this canvas
`index` - index of this canvas
`ef` - the host of this canvas

Method Detail

getIndex

```
public int getIndex()
```

Returns:

Index of this canvas

---

addLabels

```
private void addLabels()
```

Add labels around the graph

---

addValues

```
private void addValues()
```

The probVal[][] is calculated in calRecs(), this method set location and add the probability value labels on the graph. Also need to add two extra "0" labels.

---

getBar

```
public EikosoBarrier[] getBar()
```

Returns:

Array of barriers of this canvas

---

barActionPerformed

```
public void barActionPerformed(java.awt.event.ActionEvent e)
```

This call back function reacts on the user's click one barrier.

Parameters:

`e` - the event that user click a barrier

---

resumeRec

```
private void resumeRec(EikosoBarrier curBar)
```

This method resumes the correspondent EikosoRec to its original status.

Parameters:

`curBar` - the barrier that user clicked

---

recalRec

`private void `**`recalRec`**`(EikosoBarrier curBar)`

Recalculate the correspondent EikosoRecs when users first click a barrier.

Parameters:

`curBar` - the barrier that user clicked

---

getRec

`public EikosoRec[][] `**`getRec`**`()`

Returns:

This canvas' eikosoRecs

---

setColor

`public void `**`setColor`**`(java.awt.Color c)`

Set the color of this canvas.

Parameters:

`c` - the color to be set for this canvas

---

getColor

`public java.awt.Color `**`getColor`**`()`

Returns:

the color of this canvas

---

calColor

`private void `**`calColor`**`()`

Assume instance variable 'Color c' has been set already, this method calculate colours for each of its EikosoRec

---

calRecs

```
private void calRecs()
```

Calculate recs' width, height, xPos, and yPos.

---

reCalGraph

```
public void reCalGraph(int width,
                       int height)
```

The host method to refresh a canvas

Parameters:

`width` - The width of this canvas

`height` - The height of this canvas

---

clear

```
public void clear()
```

Remove everything from this canvas

---

paintComponent

```
public void paintComponent(java.awt.Graphics g)
```

Draw this canvas

Parameters:

`g` - graphic instance used to draw.

Class Eikosogram

public class **Eikosogram**
extends javax.swing.JFrame

Title: Eikosogram.java

Description:

Copyright: Copyright (c) 2005

Company: university of waterloo

Version:
1.0
Author:
glenn lee

Constructor Detail

Eikosogram

`public Eikosogram()`
Create an empty frame.

Method Detail

makeBasicForm

`private void makeBasicForm()`
Generate the example menu and some general purpose buttons

---

jbtnChangeColorActionPerformed

`public void jbtnChangeColorActionPerformed(java.awt.event.ActionEvent e)`
Call back function for the change color button.
Parameters:
`e` - The event that user click the change color button

---

jbtnCloseActionPerformed

`public void jbtnCloseActionPerformed(java.awt.event.ActionEvent e)`
Call back function for the close button. used to exit the application
Parameters:
`e` - The event that user click the close button.

---

jbtnOpenFileActionPerformed

`public void` **`jbtnOpenFileActionPerformed`**`(java.awt.event.ActionEvent e)`

Call back function for the open file button. Display the file chooser, then read file.

Parameters:

`e` - The event that user click the open file button.

---

jbtnResetActionPerformed

`public void` **`jbtnResetActionPerformed`**`(java.awt.event.ActionEvent e)`

Call back function for the reset button. Resume the original eikosogram.

Parameters:

`e` - The event that user click reset button.

---

readDataFile

`private void` **`readDataFile`**`(java.lang.String str)`

Read disc file

Parameters:

`str` - File name

---

clearScreen

`private void` **`clearScreen`**`()`

Remove all internal frames from the main frame.

---

readFile

`private void` **`readFile`**`()`

Read data file operation

Throws:

`IOException` -

---

genGraph

`private void` **`genGraph`**`()`

Generate all eikosograms

---

reCalCanvas

```
public void reCalCanvas(int ind,
                        EikosoRec[][] changedRec)
```

This method is called by reCalRec() from EikosoCanvas, it recalculates all graphs except the 'ind' one using the new changed EikosoRec

Parameters:

`ind` - The index of canvas that has been changed already, other canvases need to be changed according to it.

`changedRec` - The two dimension array of the changed EikosoRec

---

exitForm

```
private void exitForm(java.awt.event.WindowEvent evt)
```

Exit the Application

Class EikosoRec

public class **EikosoRec**

Constructor Detail

EikosoRec

```
public EikosoRec(Fraction count,
                 java.lang.String[] varName,
                 java.lang.String[] varValue)
```

Create a new EikosoRec object

Parameters:

`count` - Data count corresponds to this rectangle

`varName` - Random variable names

`varValue` - Random variable values correspond to 'varName'

Method Detail

getVarName

```
public java.lang.String[] getVarName()
```

Returns:

The variable names of this rectangle

---

getVarValue

`public java.lang.String[] getVarValue()`

Returns:

The variable values of this rectangle

---

setchanged

`public void setchanged(boolean b)`

Set the flag if this rectangle changed or not

Parameters:

`b` - True if changed, false if not

---

getchanged

`public boolean getchanged()`

Returns:

If this rectangle changed

---

getLabel

`public javax.swing.JLabel[] getLabel()`

Returns:

The label corresponds to this rectangle

---

getX

`public int getX()`

Returns:

The X coordinates of the left top corner of this rectangle

---

getY

`public int getY()`

Returns:

The Y coordinates of the left top corner of this rectangle

---

setX

`public void setX(int x)`

Set the X coordinate of the left top corner of this rectangle

Parameters:

`x` - The X coordinates

---

setY

`public void setY(int y)`

Set the Y coordinate of the left top corner of this rectangle

Parameters:

`y` - The Y coordinates

---

getWidth

`public int getWidth()`

Returns:

The width of this rectangle

---

getHeight

`public int getHeight()`

Returns:

The height of this rectangle

---

getHSB

`public float[] getHSB()`

Returns:

The hsb color of this rectangle

---

setHeight

```
public void setHeight(int height)
```

Set the height of this rectangle

Parameters:

`height` - The height about to be set

---

setWidth

```
public void setWidth(int width)
```

Set the width of this rectangle

Parameters:

`width` - The width about to be set

---

setHSB

```
public void setHSB(float h,
                   float s,
                   float b)
```

Set the hsb color of this rectangle

Parameters:

`h` - h component of color
`s` - s component of color
`b` - b component of color

---

getOriCount

```
public Fraction getOriCount()
```

Returns:

The original data count corresponds to this rectangle

---

getCurCount

```
public Fraction getCurCount()
```

Returns:

The current data count corresponds to this rectangle

---

setCurCount

```
public void setCurCount(Fraction count)
```

Set the current data count of this rectangle after the rectangle changed

Parameters:

`count` - The data count about to be set

---

setOriCount

```
public void setOriCount(Fraction count)
```

Set the original data count of this rectangle

Parameters:

`count` - The data count about to be set

---

isEqual

```
public boolean isEqual(java.lang.String[] names,
                       java.lang.String[] values)
```

Test if two rectangles correspond to the same data count

Parameters:

`names` - The variable name of the rectangle this compared

`values` - The correspondent variable value

Returns:

True if 'this' corresponds to the same data count with the rectangle compared

---

Class EikosoBarrier

public class **EikosoBarrier**
extends javax.swing.JButton

Title: EikosoBarrier

Description:

Copyright: Copyright (c) 2005

Company: university of waterloo

Version:

1.0

Author:

glenn lee

Constructor Detail

EikosoBarrier

`public EikosoBarrier(int ind)`

Create a barrier object

Parameters:

`ind` - The index of this barrier in one perspective of eikosogram

---

Method Detail

getIndex

`public int getIndex()`

Returns:

The index of this barrier

---

getPressed

`public boolean getPressed()`

Returns:

True if this barrier has been removed, false otherwise

---

setPressed

`public void setPressed(boolean b)`

Set the status of this barrier: removed or not

Parameters:

`b` - True if this barrier has been removed, false if not

Class EikosoInternalFrm

public class **EikosoInternalFrm**
extends javax.swing.JInternalFrame

Title: EikosoInternalFrm.java

Description:

Copyright: Copyright (c) 2005

Company: university of waterloo

Version:
1.0
Author:
glenn lee

Constructor Detail

EikosoInternalFrm

```
public EikosoInternalFrm(java.lang.String title,
                         boolean b1,
                         boolean b2,
                         boolean b3,
                         boolean b4,
                         int ind)
```

Create an internal frame to hold one perspective of eikosogram

Parameters:
`title` - The title of this frame
`b1` - Resizable
`b2` - Closable
`b3` - maximizable
`b4` - Iconifiable
`ind` - The index of this frame

Method Detail

getIndex

```
public int getIndex()
```

Returns:
The index of this frame

Class Fraction

public class **Fraction**

Title: Fraction

Description:

Copyright: Copyright (c) 2005

Company: university of waterloo

Version:
1.0
Author:
glenn lee

Constructor Detail

Fraction

```
public Fraction(java.lang.String numerator,
                java.lang.String denominator)
```

Create one fraction

Parameters:
`numerator` - The numerator of this fraction
`denominator` - The denominator of this fraction

Fraction

```
public Fraction()
```

Create the fraction '0'

Method Detail

floatValue

```
public float floatValue()
```

The float value of this fraction

Returns:

The float value

---

setZero

`public void `**`setZero`**`()`

Set this fraction to '0'

---

toString

`public java.lang.String `**`toString`**`()`

The string display of this fraction

Returns:

The string

---

simplify

`private void `**`simplify`**`()`

Divide both numerator and denominator by their gcd

---

toFraction

`public static Fraction `**`toFraction`**`(java.lang.String s)`

Convert a string to a fraction

Parameters:

`s` - The string about to be converted

Returns:

A fraction object of the string

---

getNumerator

`public java.math.BigInteger `**`getNumerator`**`()`

Returns:

The numerator of this fraction

---

getDenominator

```
public java.math.BigInteger getDenominator()
```
Returns:
The denominator of this fraction

---

setNumerator

```
public void setNumerator(java.math.BigInteger b)
```
Set the numerator of this fraction

Parameters:
`b` - The numerator about to be set

---

setDenominator

```
public void setDenominator(java.math.BigInteger b)
```
Set the denominator of this fraction

Parameters:
`b` - The denominator about to be set

---

getGcd

```
public java.math.BigInteger getGcd()
```
Returns:
The gcd of numerator and denominator of this fraction

---

setGcd

```
public void setGcd(java.math.BigInteger num,
                   java.math.BigInteger den)
```
Set the gcd of this fraction

Parameters:
`num` - The numerator of a fraction to be set
`den` - The denominator of a fraction to be set

---

add

```
public Fraction add(Fraction f)
```
Add 'this' to another fraction f

Parameters:

`f` - A fraction to be added

Returns:

The sum of two fractions

---

subtract

```
public Fraction subtract(Fraction f)
```

Subtract f from 'this'

Parameters:

`f` - The fraction to be subtracted

Returns:

The result of the subtraction

---

multiply

```
public Fraction multiply(Fraction f)
```

Multiply 'this' by a fraction f

Parameters:

`f` - The fraction to be multiplied

Returns:

The result of the multiplication

---

multiply

```
public Fraction multiply(int i)
```

Multiply 'this' by a integer i

Parameters:

`i` - The integer to be multiplied

Returns:

The result of the multiplication

---

divide

```
public Fraction divide(Fraction f)
```

Divide 'this' by a fraction f

Parameters:

`f` - The fraction to be divided

Returns:

The result of the division; null if f is zero

---

divide

`public Fraction divide(int i)`

Divided 'this' by an integer i

Parameters:

`i` - The integer to be divided

Returns:

The result of the division; null if i is zero
