9.11.2015

Nils Gawlik
(Other group members: Jannes Brunner, Oliver Christ)

Lab Report for the 3rd November 2015

Pre-Lab Solutions:

P1:

I googled "java data types" and found https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html.

Java's data types are: byte, short, int, long, float, double, boolean, char, String

P2:

- byte/short/int/long
- String
- byte/short/int/long
- int
- byte/short/int/long
- String
- String
- char
- float/double

P3:

You could argue for both sides. You could say that the *book* is a class from which you can create an *understanding of the narrative* in the readers head; in that case the slightly different interpretations of the narrative of different readers would be the objects.

It seems to make more sense though to say that the class is the *abstract idea of an book*. My book then is an object of the class Book, *different books* are different objects of the same class, they share attributes like page count, content, size, name, etc..

P4:

The class is Book. It has two formal parameters. Their types are String and double.

Lab work:

Getting BlueJ started (I already installed it in the pre lab) was as simple as executing BlueJ.exe, opening the project was as easy as opening the package.bluej file with BlueJ.

9.11.2015

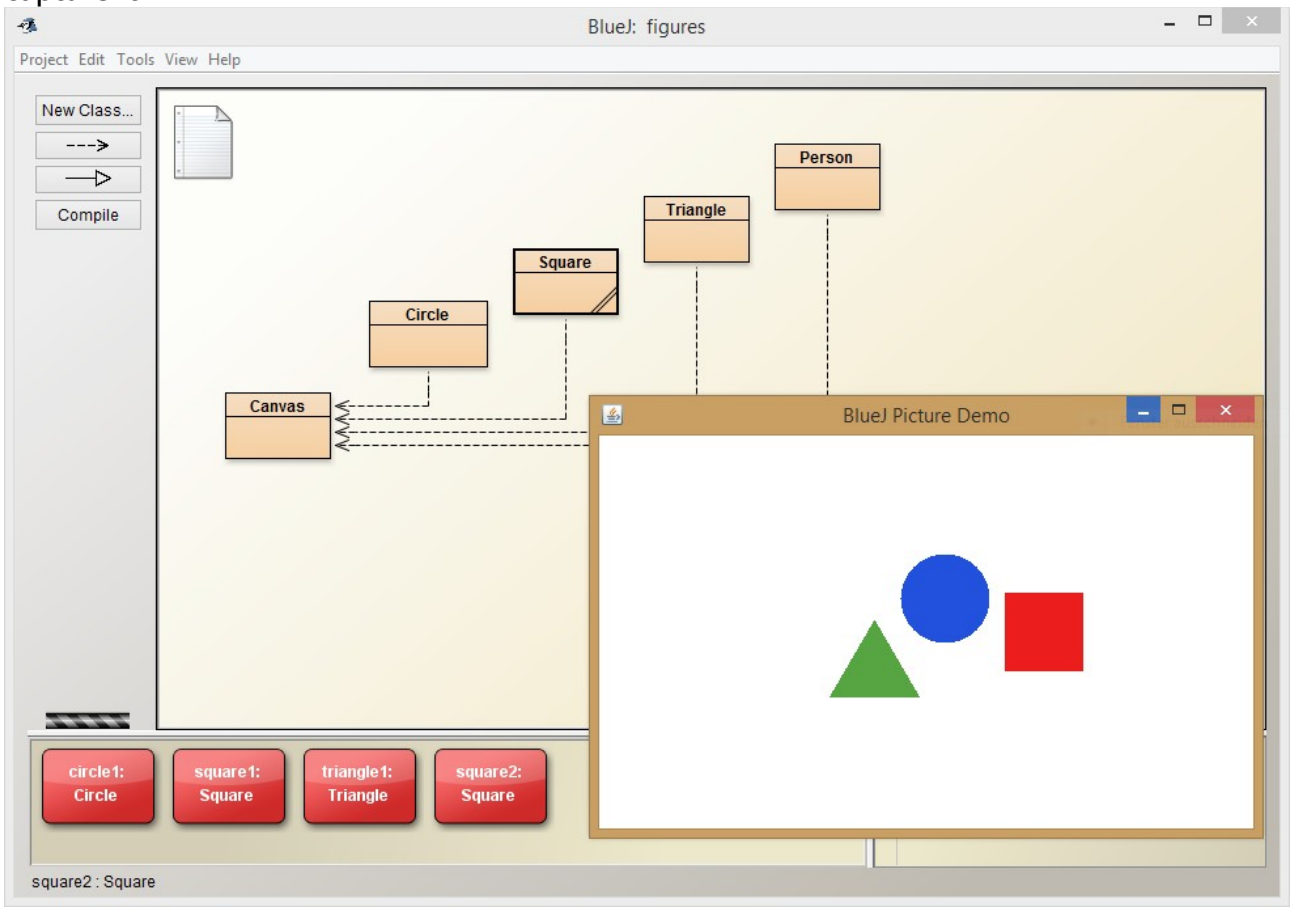
Nils Gawlik
(Other group members: Jannes Brunner, Oliver Christ)

The program asked for the path to the Java JDK and I selected the path to the JDK which was shipped with BlueJ (/BlueJ/jdk), which worked.

1.:

I opened the Figures project, I created all the shapes by creating new objects of the classes (right click on the class card → new ...). I also called the getCanvas() method on Canvas. I then also had to call the makeVisible() method on all objects (right click on object card → makeVisible()) or else they would not get displayed.

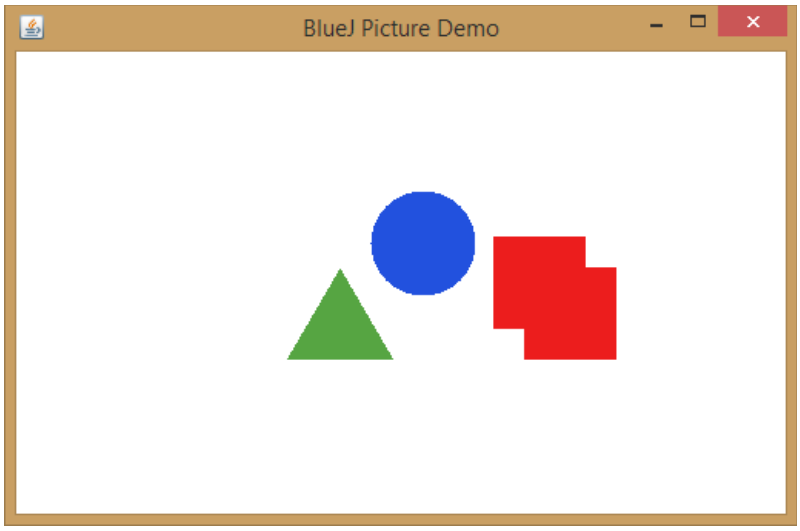
Here you see a screenshot of the scene afterwards, I used the Microsoft Snipping Tool to capture it:



Apparently the second square is perfectly overlapping the first. To solve this I called moveRight() and moveDown() on the square by right-clicking it's object card and selecting the method.

9.11.2015

Nils Gawlik
(Other group members: Jannes Brunner, Oliver Christ)



2.:

I now want to change the circle and the triangle both to red. To do this I call the method changeColor("red") on them by right-clicking them and writing "red" in the attribute field.

3.:

When you inspect the objects you see that they all have the attributes xPosition, yPosition, color and isVisible. On top of that the circle has a diameter attribute, the squares have a size attribute and the triangle has a height and width attribute.

They all have have the same value for the color attribute (from the previous task), the two Squares have different xPosition and yPosition attributes (from the first task). All other shapes actually have different positions because the standard positions defined in the classes are different for each shape.

4.:

While the constructor is syntactically correct, semantically it does not make sense. The local variable price is not used for anything and gets deleted after the constructor is "done".

In the context of the ticket machine project it makes sense to remove the int before price in the constructor; this way no new variable is declared. Price is actually already an attribute of the TicketMachine class. The change will make it so that the attribute price is set to the given value.

5.:

After replacing the constructor, there are no errors. But after some testing you can see that the ticket prize is always zero, no matter what value you give the constructor. This confirms my point from task 4; the price always defaults to zero – the value that Java gives all integers at declaration.

9.11.2015

Nils Gawlik
(Other group members: Jannes Brunner, Oliver Christ)

6.:

If you look at the source code of Kara01, you can see that the class defines no constructor. This means, that it inherits it's constructor from it's parent class, which is Kara. This is indicated by the "extends Kara" part in the declaration of the class.

KaraWorld01 has it's own constructor. The first thing the constructor does is calling the parent's class constructor using super(…). The parent constructor expects three arguments, while KaraWorld01's expects none. The arguments passed to super() are taken from constants declared in KaraWorld01.

The constructor then sets the paint order of the different classes using the inherited method setPaintOrder(...). This function takes classes as arguments; to pass the classes the notation Class.class (e.g. Kara.class) is used. The order in which the classes are passed as arguments determines the paint order. All this information can be found in the description of the method.

Then Greenfoots method setSpeed(int speed) is called, this sets the execution speed of the whole Kara simulation (in this case to 20).

Finally the prepare() function is called. This function seems to further structure the world creation process (and is necessary to support the "save the world" functionality). This method creates Kara, adds her to the world and turns her into the right direction.

7.:

For this exercise I first of all downloaded the "better-ticket-machine". I wanted to rewrite refundBalance method to give back coins. The first hurdle was the question how those coins should be returned. I decided to use an array, which stores the different amounts of coins (position 0: 2€ coins, position 1: 1€ coins, …). I also created a static array to hold the values of the different coins in cents: {200, 100, 50, 20, 10, 5, 2, 1}.

The variable "total" holds, for further explanation, the amount of money to be returned. The basic idea of the algorithm was to fit as many 2€ coins as possible into total and substract those coins from total. Then proceed with the 1€ coins. , 50 cent coins, ..., and so on all the way to 1 cent.

This is achieved by using a for loop to go through the array holding the coin values. At the beginning of each loop I divide the total through the coin value. Since they're both integers this will give me the number of times the coin fits into the total. I write the number into my array and subtract the number of times multiplied with the coin value from the total. The loop then does the same with the next smaller coin value and so on. When the for loop ends, the total should be zero and the array is filled with the numbers of coins.

I did some mistakes while implementing the algorithm, for example having an unnecessary extra for loop. But they are hard to explain and didn't give me any insights, so I am not going into details right here.