Antonio Rosado

Prof. Chien

IOOP Section #4

Lab #2

**By submitting this work, I certify that this assignment is my own work, and that I have not copied in part or whole from online sources or plagiarized the work of other students.**

**2.6**

public class Student

{

}

Public class LabClass

{

}

**2.7**

Yes, it matters because the entire code will not operate properly as it won’t recognize the class as public. When trying to compile the program, errors will appear. An explanation for one says that <identifier> was expected.

**2.8**

Yes, it is possible, everything works the same.

**2.9**

It is not possible to leave out “class” because none of the methods and objects can operate without a class. You cannot compile without a class being present.

**2.10**

Fields:

* balance, price, total

Constructor:

* TicketMachine

Methods:

* getBalance();
* getPrice();
* insertMoney(int amount);
* printTicket()

**2.11**

The two features of the constructor that make it look significantly different from the methods of the class are that a constructor has no return type, and the constructor matches the class name.

**2.12**

private int count; integer

private Student representative; string

private Server host; string

**2.13**

private Boolean alive; alive

private Person tutor; tutor

private Game game; game

**2.14**

None of them would be class names because classes need to be declared as public, so they cannot be private.

**2.15**

The order is vital because a whole bunch of errors appear with different orderings of the field declaration. The only one that works properly is the one shown.

**2.16**

It is always necessary to have a semicolon at the end of a field declaration. If there is not one present, errors would appear, and the code cannot execute or compile properly.

**2.17**

private int status;

**2.18**

The constructor belongs to the class Student.

**2.19**

The constructor has two parameters, those being a string and a double.

**2.20**

It seems from the constructor’s parameters that the Book class serves to describe an object book based on a title that is a string, and a price that is a double.

**2.21**

name = petsName;

**2.22**

public Date(String month, int day, int year)

**2.23**

There is only one difference between the getBalance method and the getPrice method and that is that although they are both accessor methods, they return different information. getBalance returns the balance, and getPrice returns the price.

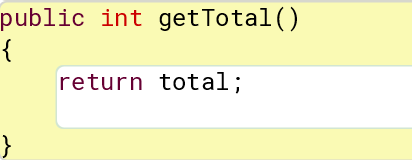
**2.24**

I would characterize the call as “How much was put into the ticket machine?”

**2.25**

If getBalance is changed to getAmount, the code is able to compile. This shows me that although it is not required for the name of the accessor method to be the same as the field associated with it, it would still be a good idea to make them associated for less confusion and sloppiness.

**2.26**

****

**2.27**

The error message explains that a return statement was expected, and this is because the method does not contain “void” and when it doesn’t, a return statement is expected and required.

**2.28**

getPrice is an accessor method meaning it returns something, whereas printTicket has “void” in its header meaning it returns nothing, and is also a mutator method because it changes the value of “total” as well.

**2.29**

Both methods do not have return statements as a result of their void types, which essentially tells the code that no values are returned.

**2.31**

It’s easy to tell that setPrice is a method and not a constructor based off the fact that it seeks to change the specified value, whereas a constructor creates one.

**2.32**

price = cost;

**2.33**

public void increase(int points)

{

score = score + points;

}

**2.34**

Yes, it is a mutator method because it does not return any values, and it changes the value of a previous variable.

**2.35**

public void discount(int amount)

{

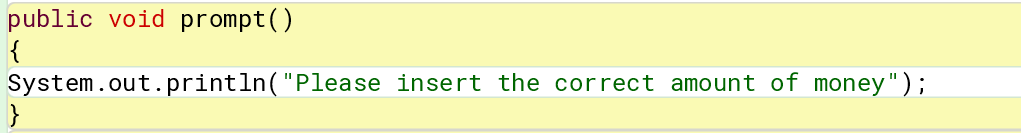
amount = amount - price;

}

**2.36**

My cat has green eyes.

**2.37**

****

**2.38**

# price cents.

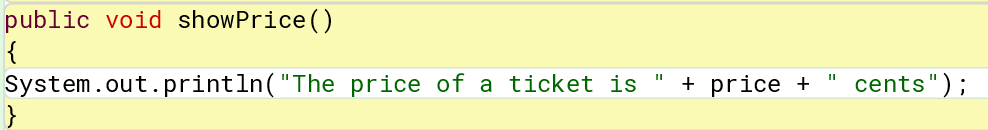
**2.39**

# price cents.

**2.40**

Neither of the two above variations could be used to show the price of the tickets because the parenthesis around price make it a string instead of printing the value that the variable price holds.

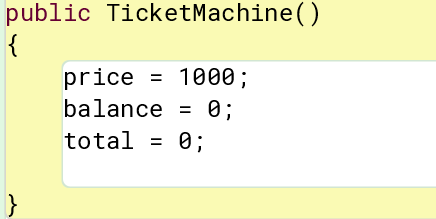
**2.41**



**2.42**

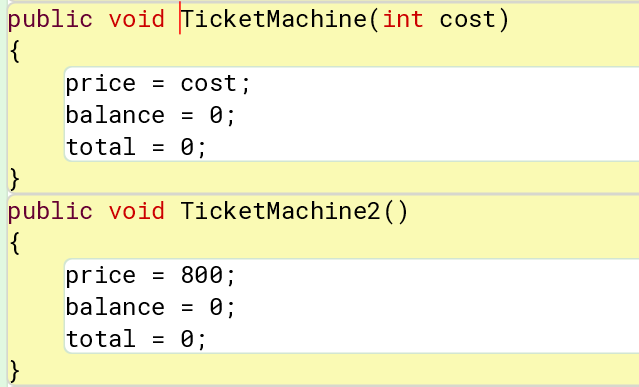
The showPrice methods displays the same output depending on the set price of the respective tickets.

**2.43**

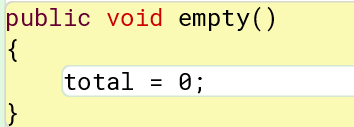
****

When you construct ticket-machine objects within BlueJ, the price is always fixed at 1000 cents and no longer gives you the option for input to specify your own ticket amount.

**2.44**



**2.45**

****

Parameters are not required for this method, it works just fine as is. The method is a mutator because it changes the value of the total field to 0.

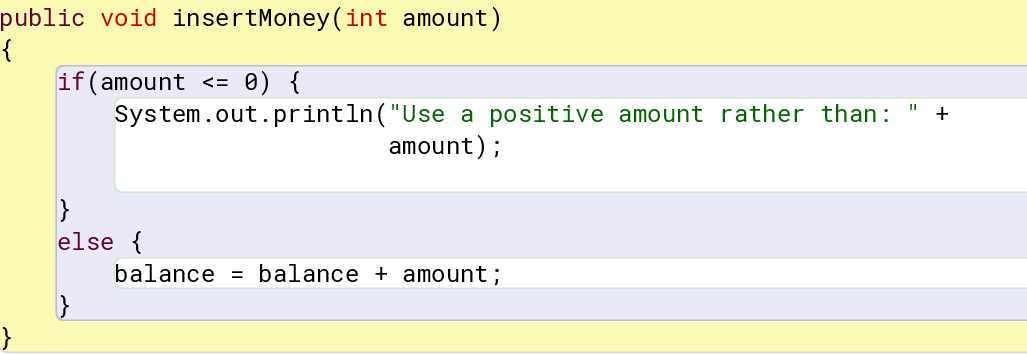
**2.46**

The machine works properly until you enter weird values such as 0. When 0 is entered, the machine does not give an error of any sort and accepts the 0.

**2.47**

Due to the symbol in the if statement being greater than or equal to 0, 0 is accepted as a valid value which does not make sense based on the machine’s purpose.

**2.48**

****

**2.49**

The boolean field used in the figures project was used for the visibility of the circle, it was well suited to be controlled by a type with only two values because it only had two functions; make a circle visible or invisible.

**2.50**

In code 2.1, the total is first updated to be collected with the balance, and the balance is then cleared or set to 0. In code 2.8, the total is then updated to have been added with the value of the price, and then the balance is reduced by the price.

**2.51**

The code will compile without the else part of the if-statement, however when you attempt to insert money into the machine its as if nothing was inserted, and the printed ticket says it still needs the specified amount of money.

**2.52**

With the if and else-statements in place, the value in the balance field could never be set to a negative value as the statements prevent such a thing from happening.

**2.53**

* multiplication (\*)
* division (/)
* modulus, or remainder after division (%)

**2.54**

saving = price \* discount

**2.55**

mean = total / count;

**2.56**

if(price > budget)

{

System.out.println("Too expensive.");

}

else {

System.out.println("Just right.");

}

**2.57**

if(price > budget)

{

System.out.println("Too expensive. Your budget is only: " + budget);

}

else {

System.out.println("Just right.");

}

**2.58**

It does not give the same results because its just returning a balance of 0.

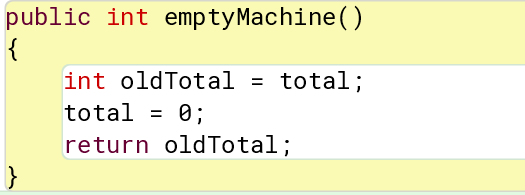
**2.59**

If you try to compile that version of refundBalance, you get an error of “unreachable statement”. This is because the return is before the value of what it’s returning is set, so it does not make sense. Returns should always be last.

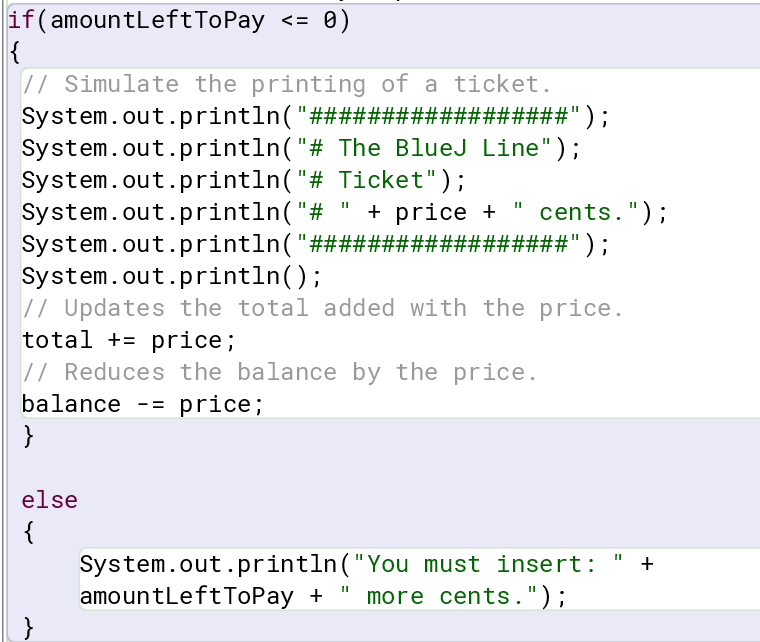
**2.60**

When that version of the constructor is used, the getPrice method returns a value of 0 no matter what is set as the price. This is because the two values of int price and cost do not go together.

**2.61**



**2.62**



**2.63**

The only way this would work is if different ticket prices for say, children, adults, and seniors were stored in fields. Then you would have to make a method that allows the user to choose what age demographic they are. If and else statements can be then utilized to result in different ticket printouts based on what the user chose.