Laboratory 1: Ticketmachine
Informatik 1, Group 1, Tuesday

Pavel Tvyatkov, Lena Zellin

April 29, 2019

Contents

1 Lab plan

For this exercise we, both group members, agreed on taking this task slowly, since this was the first time we were instructed to write our own bits of code. We would answer every question one by one and work on an answer to the question at hand before going to the next one. After we agreed we downloaded the TicketMachine zip file off of Moodle, opened it in BlueJ, double clicked the class to open the source code and started working on our assignment.

2 TicketMachine assignment

Our project consists of only one class called Ticketmachine. For this laboratory exercise we were asked to take a look at the code, change things and experience how Java works by trying out bits of code. By doing so we found answers to the question which each be answered following this section. Every numbered question and task was given to us by Prof. Weber-Wullf[1]

2.1 1. Download the Ticket-Machine project and open it in BlueJ. Experiment with it like we did in the lecture and record your observations.

To experiment with TicketMachine we needed to initialize an object of said class. After our object was created and we set its price to be 250 cents, we right clicked on it to see all the methods we could call to alter our object or interact with it. At first we looked at the method getPrice, and clicking that indeed showed us our set price of 250 cents a ticket. Next we called the insertMoney method and put 250 cents in our ticket machine. We proceeded with calling the getTicket method and opened the BlueJ terminal - a ticket for 250 cents was printed. At last we called the showBalance method and as expected our balance was 0. Everything worked as intended for us and we got more familiar with the methods through this.

2.2 2. Replace the constructor with the constructor from the pre-lab P1 and try it out. Were you right? What about the change given in P3 and P4? What happens? Record your results in your report.

After replacing the constructor of our ticket machine we asked each other what we guessed in our pre lab work. We both guessed the same - the code would compile without a problem, however the price when creating a ticketMachine object would always be zero, since we initialize another local variable, int price, in our constructor. By adding the int in front of it, Java thinks this price and the field int price are different things and treats them as such. After discussing this we went ahead and initialized a ticketMachine object with the price 250 cents, however clicking on getPrice revealed we were right as it showed the value 0.
For P3 and P4 we also put down the same guess. Both of these variations have the same output when executed: # price cents.
We both understood that this happens, because price is a String in both of these statement and not a field. The only difference is that in P3 all three parts of the sentence are individual Strings and in P4 all three parts are one String. This doesn't change the output that's generated and when calling this method our guess was confirmed.

2.3 3. Is it always necessary to have a semicolon at the end of a field declaration? Experiment via the editor and record your results.

To experiment what would happen in case of a missing semicolon, we decided to delete the semicolon at the end of private int total(;). The result was a red squiggly line at the end of the word total and a red block over the line number indication something is wrong. When trying to compile we got an error massage stating " ';' expected". We concluded that it is necessary to have a semicolon at the end of a field declaration.

2.4 4. If the name of getBalance is changed to getAmount, does the return statement in the body of the method also need to be changed for the code to compile? Try it out within BlueJ. What does this tell you about the name of an accessor method and the name of the field associated with it?

After we changed the name of getBalance to getAmount, nothing happened. No red markings, no error message, it compiled without a problem. We tried initializing an object ticketMachine and called some methods, seeing that everything worked perfectly fine. We understood that the name of an accessor method is not important to Java. It doesn't understand the names that we give things and it's more of a help for whoever has to work with the code when the names correspond to something within the code it contains. As long as there is a name for the method it could be anything, really, however what Java does care for is the name of the field associated with it, since this has a value assigned to it, necessary for executing the code correctly. It doesn't need to be part of the method name but it's useful.

```
/**
 * Return The amount of money already inserted for the
 * next ticket.
 */
public int getAmount()
{
    return balance;
}
```

2.5 5. Write an accessor method getTotal in the TicketMachine class. The new method should return the value of the total field.

For this task we knew that we have to return total so at first we came up with:

```
public int getTotal()
{
      return total;
}
```

but for some reason we thought this didn't work correctly when creating a ticketMachine object and trying it out, although it did. We don't know how we came to think it was wrong. Unfortunalety, because of that we tried to come up with another method where we oriented ourselves on refundBalance:

```
public int getTotal()
{
      int showTotal;
      showTotal = balance;
      return showTotal
}
```

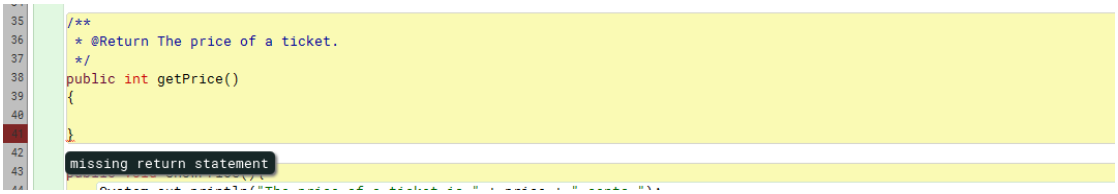
```
}

public int getTotal() {
    int showTotal;
    showTotal = balance;
    return showTotal;
}
```

This was unnecessary and because we constantly looked at refundBalance to construct our method the same way we made a mistake while writing getTotal, as we didn't write showTotal = total, but showTotal = balance! This was a grave mistake. However we didn't notice, until the very end while trying out our empty method that something was wrong.

Two problems combined themselves and made it a little harder to find the mistake at first glance. Firstly, the method getTotal was called get**Total** which made us skip over the fact that it wasn't actually returning the value of the field total, but balance. This again demonstrated to us that the method name is really unimportant to Java but of great importance to the people working with the code. Secondly, whatever we did to total was working, however when clicking on methods like empty, as shown below, and then calling getTotal, which was actually returning balance, made us think that our code just wasn't working.
At the end, we did read over the code individually again in private and found the error. After we changed balance to total, we realized that we wrote three lines of code that did the exact same thing as the one liner return total; , so in the end we changed it back to our first implementation and everything worked perfectly.

2.6 6. Try removing the return statement from the body of getPrice. What error message do you see now when you try compiling the classes?

When we tried to compile with a missing return statement in the method getPrice, we received the error message: missing return statement. This showed us that every method which has a return type other than void needs a return something at the end of the method.



The error message.

2.7 7. Complete the following method, whose purpose is to subtract the value of its parameter from a field named price.

For this task we came up with:

```
public void discount (int amount)
{
        price -= amount;
}
```

```
/**
 * Reduce price by the given amount.
 */
public void discount(int amount)
{
     price -= amount;
}
```

After we compiled it without a problem, we proceeded to create a ticketMachine object with a price of 300 cents a ticket. Next we clicked on our new method discount which then asked us to input an integer. We decided to deduct 50 cents of the ticket price to make it 250 cents a ticket. We clicked on getPrice next and it showed us the now reduced price of 250 cents as expected.

2.8 8. Add a method called prompt to the TicketMachine class. This should have a void return type and take no parameters. The body of the method should print the following single line of output: Please insert the correct amount of money.

Following the given instructions we wrote this method:

```
public void prompt()
{
    System.out.println("Please insert the correct amount of money.");
}
```

```
}

public void prompt()
{
    System.out.println("Please insert the correct amount of money.");
}
```

When clicking on this method after initializing a ticketMachine object we now recieved an output in the terminal of the sentence given above.

2.9 9. Add the possibility to count the number of tickets sold. Include a method for outputting how many tickets have been sold, like we did in class.

Reading this question for the first time left us slightly confused, since we haven't done anything similiar in class up to this point, however after asking Prof. Weber-Wullf she assisted us and gave us some tips with which we could write the following bits of code.
At first we declared a field that would keep track of all the tickets sold: private int count; .
We named it count, because it counts all the sold tickets. We then initialized it with the value 0 in our constructor TicketMachine: count = 0; .

```
public class TicketMachine
{
    // The price of a ticket from this machine.
    private int price;
    // The amount of money entered by a customer so far.
    private int balance;
    // The total amount of money collected by this machine.
    private int total;
    // The amount of tickets sold
    private int count;

    /**
     * Create a machine that issues tickets of the given price.
     */
    public TicketMachine(int cost)
    {
        price = cost;
        balance = 0;
        total = 0;
        count = 0;
    }

    /**
     * @Return The price of a ticket.
     */
    public int getPrice()
    {
        return price;
```

After that we asked ourselves where it would be useful to increase count. Scrolling to the printTicket method we thought the only logical thing would be to increase the number of tickets

sold by one **after** a ticket was actually sold. With this in mind we wrote: count = count + 1; after the if statement and before the else statement of printTicket. Now the only thing left to do was writing a method that would let us see how many tickets were actually sold. For this we put this together below the printTicket method:

```
public int getCount()
{
      int showCount;
      showCount = count;
      return showCount;
}
```

However we realized this was a way more complicated way to simply write:

```
public int getCount()
{
      return count;
}
```

```
public void printTicket()
{
    if(balance >= price) {
        // Simulate the printing of a ticket.
        System.out.println("##################");
        System.out.println("# The BlueJ Line");
        System.out.println("# Ticket");
        System.out.println("# " + price + " cents.");
        System.out.println("##################");
        System.out.println();

        // Update the total collected with the price.
        total = total + price;
        // Reduce the balance by the prince.
        balance = balance - price;
        //increase the amount of tickts sold
        count = count + 1;
    }
    else {
        System.out.println("You must insert at least: " +
                           (price - balance) + " more cents.");

    }
}
//shows how many tickets were sold
public int getCount()
{
    return count;
}
```

After we wrote this method and compiled everything without a problem we proceeded to test our method by creating a new ticketMachine object with a price of 250 cents, inserted 500 cents and printed two tickets. After we made sure everything worked without a problem we clicked on getCount. This returned the value of our field count which was 2, assuring us that our method worked as intended.

2.10 10. Add a showPrice method to the TicketMachine class. This should have a void return type and take no parameters. The body of the method should print: The price of a ticket is xyz cents. Where xyz should be replaced by the value held in the price field when the method is called. Now create two ticket machines with differently priced tickets. Do calls to their showPrice methods show the same output, or different? How do you explain this effect?

Our method for this task looks as following:

```
public void showPrice()
{
```

System.out.println("The price of a ticket is " + price + " cents.");
}

```
}
//prints a sentence in the terminal about the price of a ticket
public void showPrice(){
    System.out.println("The price of a ticket is " + price + " cents.");
}

/**
```

With the help of System.out.println we print a line of what is inide the parameter. We put in a String "The price of a ticket is " leaving a blank at the end and concatenating it with price. This is not a String in our case but the value of our field price. Then we concatenate that with the String " cents." leaving a blank at the beginning. We use the blanks as spaces and, when calling this method, get the output in our terminal: The price of a ticket is (whatever price we set when we initialized a new ticketmachine) cents.
Next we created a second object ticketMachine with a different price of 300, our first ticketMachine had a value of 250 as its price. We called the method showPrice on both objects and then looked at the BlueJ terminal. There our desired output was displayed with 250 cents for ticketMaschine number one and with 300 cents for ticketMachine number two. The output is different, because price is a field that is initilazied in the constructor of our class. This constructor is called when we create a new object of class TicketMachine, asking us to input a value for price. This happens every time when we create an object ticketMachine which leads to every object having its own price. Even if we set two ticketMachines to the same price they would still be object specific and not the same as in sharing the same price.

2.11 11. Implement a method, empty, that simulates the effect of removing all money from the machine. This method should have a void return type, and its body should simply set the total field to zero. Does this method need to take any parameters? Test your method by creating a machine, inserting some money, printing some tickets, checking the total, and then emptying the machine. Is the empty method a mutator or an accessor?

All we put down for this task was this bit of code:

```
public void empty()
{
    total=0;
}
```

```
//empties out the ticket machine
public void empty()
{
    total = 0;
}

/**
```

It simulates someone taking the money from an actual ticket machine like an employee of the BVG for example.
As already stated before we had trouble with writing this method, as we had made a mistake in another method prior to this. Our empty method worked just fine however we couldn't see that as we were returning the value for balance in getTotal and not total unfortunately. As already mentioned above once we changed getTotal everything worked as intendet when we tried it out

by initializing a new object ticketMachine, inserting money, buying tickets and then calling our empty method. After that we called getTotal a last time to see that the money collected in the machine was gone, or rather zero.

3 After the lab

Before we had to finish our work at the lab we agreed that one group member would start writing a PDF with the notes we took in a .txt file and then send the notes and the PDF to the other one until it was completed and satisfactory for both group members. We also talked about looking at the code again to find the mistake already mentioned to complete this exercises tasks and questions.

4 What I learned from this assignment

4.1 Pavel

I learned that in Java it is always necessary to have a semicolon at the end of a field declaration. We can change the name of an accessor method and we don't have to change the name of the field associated with it for the code to compile. I learned how to add the possibility for the ticket machine to count how many tickets have been sold. We had to first create a new "count" field of type int in the main class and then initialize it in the constructor.

4.2 Lena

I learned that it's necessary to give methods good and useful names to quickly decipher what their purpose is. After we changed getBalance to getAmount i had a really hard time finding the method that returns balance, because I orientate myself mostly by method names. I also learned that one should check a method twice for accuracy after writing it to avoid mistakes that in the end take longer to solve.

5 Appendix

5.1 Citation

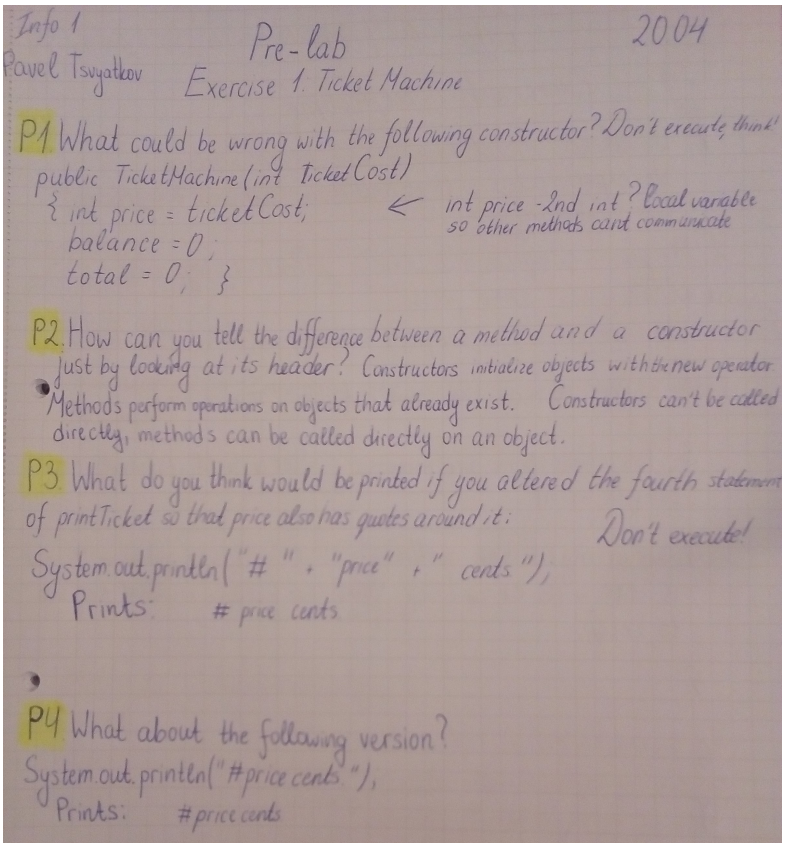
References

[1] HTW Berlin, Prof. Dr. Debora Weber-Wullf
`https://people.f4.htw-berlin.de/~weberwu/info1/Labs/Lab1.shtml`
last accessed: 28.04.2019

5.2 Pre lab

We decided to add pictures of both our pre labs to document our work.

5.2.1 Pavel



Info 1 2004
Pavel Tsvyatkov
Pre-lab
Exercise 1. Ticket Machine

P1 What could be wrong with the following constructor? Don't execute, think!

```
public TicketMachine(int TicketCost)
{ int price = ticketCost;
  balance = 0;
  total = 0; }
```

← int price - 2nd int? local variable so other methods can't communicate

P2. How can you tell the difference between a method and a constructor just by looking at its header? Constructors initialize objects with the new operator. Methods perform operations on objects that already exist. Constructors can't be called directly, methods can be called directly on an object.

P3 What do you think would be printed if you altered the fourth statement of printTicket so that price also has quotes around it:

Don't execute!

```
System.out.println("# " + "price" + " cents.");
```

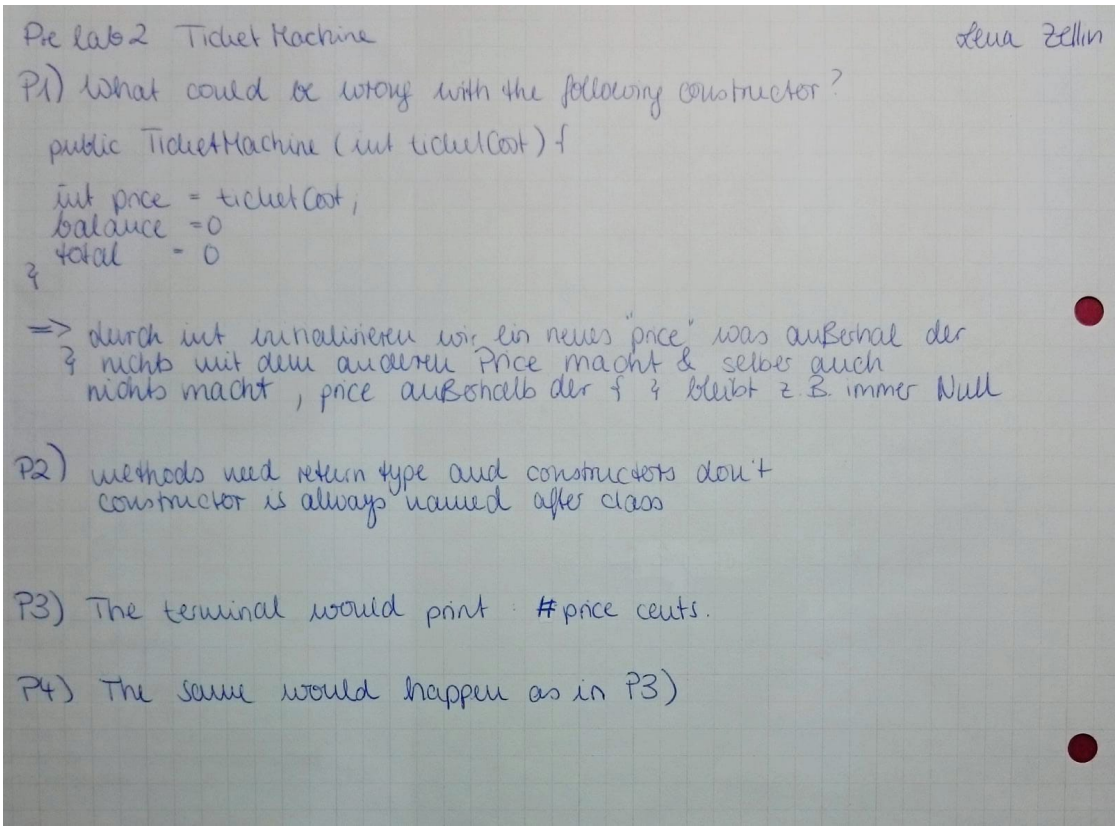
Prints: # price cents

P4 What about the following version?

```
System.out.println("#price cents.");
```

Prints: #price cents

5.2.2 Lena



5.3 Our Code

```
/**
* TicketMachine models a ticket machine that issues
* flat-fare tickets.
* The price of a ticket is specified via the constructor.
* Instances will check to ensure that a user only enters
* sensible amounts of money, and will only print a ticket
* if enough money has been input.
*
* @author David J. Barnes and Michael Kölling
* @version 2011.07.31
*/
public class TicketMachine
{
    // The price of a ticket from this machine.
    private int price;
    // The amount of money entered by a customer so far.
    private int balance;
    // The total amount of money collected by this machine.
    private int total;
    // The amount of tickets sold
    private int count;

    /**
    * Create a machine that issues tickets of the given price.
    */
    public TicketMachine(int cost)
```

```
{
    price = cost;
    balance = 0;
    total = 0;
    count = 0;
}

/**
* @Return The price of a ticket.
*/
public int getPrice()
{
    return price;
}

//prints a sentence in the terminal about the price of a ticket
public void showPrice(){
    System.out.println("The price of a ticket is " + price + " cents.");
}

/**
* Return The amount of money already inserted for the
* next ticket.
*/
public int getAmount()//changed from getBalance
{
    return balance;
}

/**
* Receive an amount of money from a customer.
* Check that the amount is sensible.
*/
public void insertMoney(int amount)
{
    if(amount > 0) {
        balance = balance + amount;
    }
    else {
        System.out.println("Use a positive amount rather than: " +
        amount);
    }
}

/**
* Print a ticket if enough money has been inserted, and
* reduce the current balance by the ticket price. Print
* an error message if more money is required.
*/
public void printTicket()
{
    if(balance >= price) {
        // Simulate the printing of a ticket.
        System.out.println("##################");
        System.out.println("# The BlueJ Line");
        System.out.println("# Ticket");
        System.out.println("# " + price + " cents.");
        System.out.println("##################");
```

```
            System.out.println();

            // Update the total collected with the price.
            total = total + price;
            // Reduce the balance by the prince.
            balance = balance - price;
            //increase the amount of tickets sold
            count = count + 1;
        }
        else {
            System.out.println("You must insert at least: " +
            (price - balance) + " more cents.");

        }

    }

    //shows how many tickets were sold
    public int getCount()
    {
        return count;
    }

    /**
    * Return the money in the balance.
    * The balance is cleared.
    */
    public int refundBalance()
    {
        int amountToRefund;
        amountToRefund = balance;
        balance = 0;
        return amountToRefund;
    }

    //returns how much money has been put into the ticket machine up to now
    public int getTotal()
    {
        return total;
    }

    //empties out the ticket machine
    public void empty()
    {
        total = 0;
    }

    /**
    * Reduce price by the given amount.
    */
    public void discount(int amount)
    {
        price -= amount;
    }

    //prints a prompt in the terminal about inserting the correct amount of money, regardless of
    public void prompt()
    {
```

```
            System.out.println("Please insert the correct amount of money.");
        }

    }
```