Lab Report Anh Pham Viet / Dennis Bröcker – 23.04.19

1. **Semester / IMI -** WH C 579

Professor	Name of Exercise
Dr. Prof. Debora Weber-Wulff	Exercise 1

General Information

Index

Lab Structure

- Checking the Pre-Lab
- Explaining the assignments step by step
- Choosing lab partner for this week by memory cards
- Working on the tasks for the lab report
- Saving data on WebDrive
- Logging Out

Material

- Computer
- Logbook / Pen

Assignment

1. Download the **Ticket-Machine** project and open it in BlueJ. Experiment with it like we did in the lecture and record your observations.
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.
3. Is it always necessary to have a semicolon at the end of a field declaration? Experiment via the editor and record your results.
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?
5. Write an accessor method getTotal in the TicketMachine class. The new method should return the value of the total field.
6. Try removing the return statement from the body of getPrice. What error message do you see now when you try compiling the classes?
7. Complete the following method, whose purpose is to subtract the value of its parameter from a field named price.
   /**

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

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.
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.
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?
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?
12. (For the bored) Can you make the better-ticket-machine give proper change with a minimal amount of Euro coins?

Lab Report

Expectation & Goals

Anh Pham Viet:

Since our lectures are easily 2-3 sessions ahead of our laboratory sessions and it for me personally was kind of hard getting a real "grasp" without actual practice, so I was really looking forward to deepen my understanding through working through all the tasks with my partner.

Dennis Bröcker:

My expectations in that lab after the Figures one, was more about deepen my knowledge but also refreshing it in the calculation topic and creating methods.

Assignment

We had the usal "driver" & the "navigator" role-switching during the laboratory session.

1. Task

We created an object called ticketMa1 (was the default name given by the program) by clicking on the class " Ticket Machine". A new window appeared where we could type in a int parameter for the ticket price. Typing any kind of string did result in an error as we tried "hello". We chose 210 as ticket price. Afterwards we chose the method "void insertMoney(int amount)" and typed in 210 and checked the balance via "int getBalance()" and 210 was shown as balance. Finally to print the ticket "void printTicket()" was chosen. Everything worked smoothly as expected since this was the "better-Ticket-Machine".

2. Task

The code from P3 and p4 delivered the same result. Thus it showed price as a part of the string output and does not try to get the value of price.

3. Task

It seems to be necessary since we removed a semicolon after the field “private int price” and “private int total” but BlueJ refused to compile without semicolons at the end of a field declaration.

4. Task

We changed “getBalance” to “getAmount” and compiled the code. Created an instance with 210 as ticket costs and inserted money. Finally we chose the new method “int getAmount()” and it returned 210.

The name of the accessor method is quite irrelevant to the return statement. It is just a name which needs to be properly chosen but does not affect the code. The name of the field is the important one for the code. If it would be declared to, let us say, “return Amount” it would not be possible since no variable or parameter is assigned to “Amount” within the body or in the fields. Also that experiment tells us that the lifetime / scope of a field is accessible within the whole code, not just the current instance.

5. Task

We wrote the new accessor-method getTotal() as a result of task 5 and put it under the other getter-Methods for a better overview of the code. At the end it showed us our desired result: it returned the value of the field “total”.

```
/**
 * Return the amount of Total
 */
public int getTotal()
{
    return total;
}
```

6. Task

The code won`t compile and delivers “not a statement” as a error message. Since we executed the getter-method, we have to return a value. We remembered from the Info 1 lecture that without the return statement, the programm does not know what to do with the value of price.

7. Task

The description in the given assignment told us the calculation we had to do and we worked according to the text. “Reduce price by the given amount” means price = price - amount . It worked as we thought but with only that we added a big error. People are now able to enter values bigger than the actual price leading to a negative price at the end.

As a solution Dennis came up with an if-Statement, forcing the user to enter a amount less than the actual price. You are still able to get the price to 0 but well it is a designer choice in case the fictional company wants to create a event with free tickets! Otherwise we would use “amount < price”. Anyways, if the amount is bigger than the price, it will output a text, telling you to use lower value.

```
/**
 * Reduce price by the given amount.
 */

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

Screenshot 1: Former Code

```
/**
* Reduce the price by given amount.
*/
public void discount ( int amount)
{
    if (amount <= price) {
    price = price - amount;
}
else {
    System.out.println("Pleaser enter a value, which is less than the value of price!");
}
}
```

Screenshot 2: Refactored Code

8. Task

We added the method called prompt into the TicketMachine class, called the method “System.outprint()” and wrote the text inside the bracket as a string. It worked out smoothly.

```
/**
* Task 8
*/
public void prompt()
{
    System.out.print("Please insert the correct amount of money.");
}
```

9. Task

First of all we initialized a new field called “private int sold” and put the value of sold in the constructor to 0. In the method printTicket() we added a counter using “sold = sold+1”. Now everytime we print a ticket, sold counts one value up. For the output method we created a simple getter which returns sold.

```
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;
    // Total amount of tickets sold.
    private int sold;


    public TicketMachine(int cost)
    {
        if ( cost >= 0 && cost <= 500) {
        price = cost;
    }
        else {
        price = 500;
    }
        balance = 0;
        total = 0;
        sold = 0;
    }
```

1: Fields and Constructor

```
public int getSold()
{
    return sold;
}
```

Screenshot 2: Getter-Method

```
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("###################");

        // Update the total collected with the price.
        total = total + price;
        // Reduce the balance by the price.
        balance = balance - price;
        // Update the number of tickets sold.
        sold = sold + 1;
```

Screenshot 3: CounterScreenshot

10. Task

With void we are not able to use a return statement, that's why we used System.out.println("The price of a ticket is " + price + " cents."). The difference between showPrice() and getPrice() is that the output of showPrice() is shown in the console and getPrice() as a method-result.

Basically "return" just returns the value while "System.out.println" is a text-based method, which is showing up in the console.

```
/**
 * Show price method.
 */
public void showPrice()
{
    System.out.print("The price of a ticket is " + price +" cents.");
}
```

11. Task

No, the method has not to take any parameters, since we only want to zero out the total field. However it is a mutator because it sets the value of the field to a other value. Accessor are used to return a value of a field.

```
public void setEmpty()
{ total = 0;

}
```

12. Task

If we have not misunderstood the task, we should change the method to return euros and should not be high numbers. If so, we created a new method called refundEuro() and did not touch the existing

method refundBalance(). In that method we used the data type double. Basically it is the same like the refundBalance(). We initialized “double refundEuro” and refundEuro gets the value of balance but the problem there is that the value gets 210.0 because the amount is still in cent(if cost entered is 210). In order to get it to euros we calculated refundEuro /= 100; which changes the value to 2.10 .

After that we set the balance to 0 and return the value which is 2.10 .

Moreover in order to not get high prices for minimal amount but also not being able to enter negative prices we added a if statement in the constructor “if (cost >= 0 && cost <= 500)” . That forces the userto enter a cost bigger than 0 but also less than 500. If not, the value will automatically set to 500 which can be lowered using the discount() method.

```
public TicketMachine(int cost)
{
    if ( cost >= 0 && cost <= 500) {
    price = cost;
}
    else {
    price = 500;
}
    balance = 0;
    total = 0;
    sold = 0;
}
```

Screenshot 1: New Constructor

```
/**
 * Return money in Euro amount
 * Balance also get cleared
 */
public double refundEuro()
{
    double refundEuro;
    refundEuro = balance;
    refundEuro /= 100;
    balance = 0;
    return refundEuro;
}
```

Screenshot 2: refundEuro-Method

Reflection and Summary

Anh Pham Viet:

First of all I would like to thank Dennis for being a great Partner. He is already relatively well acquainted with Java and was patient enough to explain everything we did step by step.
This lab session definitely deepened my understanding about the java syntax and how the code looks behind the scenes of the representation canvas.I am now able to use setters and mutators for different purposes. Also it was interesting to print out a concatenation string in task 10.
I feel like I can slowly see the patterns and signatures when looking at all the methods in the codes over and over again and this definitely helps with understanding.

Dennis Bröcker:

Comparing to the lesson before, that lab was more interesting and exciting. It felt like to work on something bigger and I was able to refresh a lot of Java knowledge, which I acquired in school. Moreover Anh was a excellent partner, because we helped each other and he was also able to hold me back in form of rushing the tasks. We first discussed how we would like to start the task, what we need, what we have to do. By rushing tasks, we would have added only more errors. That is one attitude which I have to learn over the time because in school we only worked on our own code without a partner. Thus I am very thankful to Anh, showing me what I need to change."

```
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;
    // Total amount of tickets sold.
    private int sold;



    public TicketMachine(int cost)
    {
        if ( cost >= 0 && cost <= 500) {
        price = cost;
}
        else {
        price = 500;
}
        balance = 0;
        total = 0;
        sold = 0;
    }

    /**
     * @Return The price of a ticket.
     */
    public int getPrice()
    {
        return price;
    }
    /**
    * * Return the total money
    */
    public int getTotal()
    { return total;
    }
```

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

public int getSold()
{
    return sold;
}

/**
 * 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);
    }
}

/**
* Task 8
*/
public void prompt()
{
    System.out.print("Please insert the correct amount of money.");
}
```

```
/**
* Reduce the price by given amount.
*/
public void discount ( int amount)
{
    if (amount <= price) {
    price = price - amount;
}
else {
    System.out.println("Pleaser enter a value, which is less than the value of price!");
}
}
/**
* Show price method.
*/
public void showPrice()
{
    System.out.println("The price of a ticket is " + price + " cents.");
}
/**
* Zero out the total field.
*/
public void setEmpty()
{ total = 0;

}
```

```java
/**
 * 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("##################");

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

    }
}

/**
 * Return money in Euro amount
 * Balance also get cleared
 */
public double refundEuro()
{
    double refundEuro;
    refundEuro = balance;
    refundEuro /= 100;
    balance = 0;
    return refundEuro;
}
```

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

Class TicketMachine

java.lang.Object
    TicketMachine

public class **TicketMachine**
extends java.lang.Object

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.

Version:
2011.07.31

Author:
David J. Barnes and Michael Kölling

Constructor Summary

Constructors

Constructor	Description
`TicketMachine(int cost)`	

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method	Description
void	`discount(int amount)`	Reduce the price by given amount.
int	`getAmount()`	Return The amount of money already inserted for the next ticket.
int	`getPrice()`	
int	`getSold()`	
int	`getTotal()`	* Return the total money
void	`insertMoney(int amount)`	Receive an amount of money from a customer.
void	`printTicket()`	Print a ticket if enough money has been inserted, and reduce the current balance by the ticket price.
void	`prompt()`	Task 8

int	refundBalance()	Return the money in the balance.
double	refundEuro()	Return money in Euro amount Balance also get cleared
void	setEmpty()	Zero out the total field.
void	showPrice()	Show price method.

Methods inherited from class java.lang.Object

clone, equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

TicketMachine

```
public TicketMachine(int cost)
```

Method Detail

getPrice

```
public int getPrice()
```

getTotal

```
public int getTotal()
```

* Return the total money

showPrice

```
public void showPrice()
```

Show price method.

setEmpty

```
public void setEmpty()
```

Zero out the total field.

printTicket

```
public void printTicket()
```

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.

refundEuro

```
public double refundEuro()
```

Return money in Euro amount Balance also get cleared

refundBalance

```
public int refundBalance()
```

Return the money in the balance. The balance is cleared.

getAmount

```
public int getAmount()
```

Return The amount of money already inserted for the next ticket.

getSold

```
public int getSold()
```

insertMoney

```
public void insertMoney(int amount)
```

Receive an amount of money from a customer. Check that the amount is sensible.

prompt

```
public void prompt()
```

Task 8

discount

```
public void discount(int amount)
```

Reduce the price by given amount.

showPrice

```
public void showPrice()
```

Show price method.

htw Hochschule für Technik und Wirtschaft Berlin
University of Applied Sciences

HTW Berlin
Fachbereich 4
Internationaler Studiengang
Internationale Medieninformatik (Bachelor)
Info 1: Informatik I
Summer Term 2019

Laboratory 1: TicketMachine

This week's lab work is intended to acquaint you with the BlueJ environment.

Pre-lab

What to Bring to Lab

Please bring these exercises printed out or written out with you to lab. Please have your name on your page.

P1. What could be wrong with the following constructor? Don't execute it, think about it in your head.

Maybe you don't need to work out the data type int when doing assignment operator

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

P2. How can you tell the difference between a method and a constructor just by looking at its header? method - object ~~met~~ a dot is added between method and object
constructor: "new" is the signal word

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

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

Don't execute this, just write down your expectations.

Backslash \"price" to have the quotation mark printed

P4. What about the following version?

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

The same as P3

Post-Lab, AKA What To Turn In

Your completed assignment, submitted in Moodle as a pdf, should include:

- a description of what you did during the lab, including a record of your experiments with the `TicketMachine`, and
- the names and roles of any collaborators in any parts of the exercise.

Lab assignments are due the night before your next lab at 22.00, I want you to get some sleep the night before lab. They may, of course, be turned in earlier. You hand them in by preparing the report in PDF and submitting it to Moodle.

Pre Lab Dennis Bröcker Info 1: Ticket Machine

P1. What could be wrong with the following constructor? Don't execute it, think about it in your head.

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

Int price is a new field with a lifetime just in that constructor because it gets initizialized there

P2. How can you tell the difference between a method and a constructor just by looking at its header?

A constructor equals the class name while a method can have any name and often has return or void in front of it.

P3. What do you think would be printed if you altered the fourth statement of printTicket so that price also has quotes around it, as follows:
`System.out.println("# " + "price" + " cents.");`
Don't execute this, just write down your expectations.

The output would be "#Price cents.". Price will get a word and the program will not try to ask the fields for the value of price in order to show it.

P4. What about the following version?
`System.out.println("# price cents.");`

Same output. Price will be written as a word