

Übungsblatt 06

2P

Aufgabe 26+27

```
// package blatt06;

public class A2627_main {

    // -----

    public static void main(String[] args) {

        // SAH: Create Account
        BankAccount ba1 = new BankAccount("Max Mustermann", "DE0123456789");

        // SAH: add Balance
        ba1.addBalance(20);
        ba1.addBalance(-13);

        // SAH: Print
        System.out.println("Bankkonto von " + ba1.getAccountHolderName() + " mit Kontonummer "
            + ba1.getAccountNumber() + " am " + ba1.getOpeningDate() + " eröffnet.");
        System.out.println("Aktueller Kontostand: " + ba1.getBalance());
        System.out.println("Zeichenketten-Darstellung des Kontos:");
        System.out.println(ba1);

        // SAH: Try to create account
        try {
            BankAccount ba2 = new BankAccount("Falscher Test1 naMe", "5R9S3B2N0Q7");
            System.out.println("Ueberpruefung des Namen fehlgeschlagen!");
        }

        catch (IllegalArgumentException e) {
            System.out.println("Falscher Name führt zu folgender Fehlermeldung: " + e.getMessage());
        }

        // SAH: Try to create account
        try {
            BankAccount ba2 = new BankAccount("Gueltiger Name", "5R9S3a2N0Q7");
            System.out.println("Ueberpruefung der Kontonummer fehlgeschlagen!");
        }

        catch (IllegalArgumentException e) {
            System.out.println("Falsche Kontonummer führt folgender Fehlermeldung: " + e.getMessage());
        }
    }
}
```

```
// -----  
}
```

Informatik II – Tutorium 13 – Gruppe 7
Gruppenmitglieder: Hinterreiter, Gallinger, Stein

```
// Aufgabenstellung:

// 1. Bankkonten haben den Namen des Kontohalters, eine Kontonummer, ein Eröffnungsdatum sowie einen Kontowert.

// 2.a Der Name des Kontoinhabers ist eine Zeichenkette und muss aus mindestens einem Wort bestehen, das mit
einem lateinischen Großbuchstaben beginnt, gefolgt von beliebig vielen lateinischen Kleinbuchstaben.
// 2.b Der Name des Kontoinhabers kann aus mehreren Worten, getrennt durch Leerzeichen bestehen.
// 2.c Der Name des Kontoinhabers kann jederzeit geändert und gelesen werden.

// 3.a Die Kontonummer ist eine Zeichenkette, die aus genau 12 Großbuchstaben und Ziffern besteht, z.B.
"A12B34C56D78".
// 3.b Die Kontonummer wird beim Erstellen des Bankkontos festgelegt und kann danach nicht mehr geändert werden.

// 4. Wird für den Namen oder die Kontonummer ein ungültiger Wert übergeben, so wird eine geeignete
ungeprüfte Ausnahme geworfen.

// 5. Das Eröffnungsdatum entspricht dem Tag, an dem das Bankkonto erstellt wurde und kann nicht geändert
werden.

// 6. Der Kontostand ist eine reelle Zahl und ist zu Beginn 0.
// 7. Danach kann der Kontostand sowohl erhöht als auch verringert werden.

// 8. Alle Attribute des Bankkontos sind öffentlich lesbar.

// 9. Zwei Bankkonten sind genau dann gleich, wenn ihre Kontonummern gleich sind.

// 10. Wählen Sie eine geeignete Darstellung des Bankkontos als Zeichenkette.
//

// Necessary packages/classes:
import java.time.LocalDate;
import java.util.regex.Pattern;
//

public class BankAccount {

    // -----

    // Attribute:
    // 1. Bankkonten haben den Namen des Kontohalters, eine Kontonummer, ein Eröffnungsdatum sowie einen Kontowert.

    // 2.a Der Name des Kontoinhabers ist eine Zeichenkette und muss aus mindestens einem Wort bestehen, das mit
    einem lateinischen Großbuchstaben beginnt, gefolgt von beliebig vielen lateinischen Kleinbuchstaben.
    // 2.b Der Name des Kontoinhabers kann aus mehreren Worten, getrennt durch Leerzeichen bestehen.
    // 2.c Der Name des Kontoinhabers kann jederzeit geändert und gelesen werden.
    private String accountHolderName;
    private static final Pattern NAME_PATTERN = Pattern.compile("[A-Z][a-z]*(\\s[A-Z][a-z]*)*");
```

Informatik II – Tutorium 13 – Gruppe 7
Gruppenmitglieder: Hinterreiter, Gallinger, Stein

```
// 3.a Die Kontonummer ist eine Zeichenkette, die aus genau 12 Großbuchstaben und Ziffern besteht, z.B.
"A12B34C56D78".

// 3.b Die Kontonummer wird beim Erstellen des Bankkontos festgelegt und kann danach nicht mehr geändert werden.
private final String accountNumber;
private static final Pattern ACCOUNT_NUMBER_PATTERN = Pattern.compile("[A-Z0-9]{12}");

// 1. ein Eröffnungsdatum
private final LocalDate openingDate;

// 1. sowie einen Kontowert.
private double balance;

// -----

// Konstruktor:
public BankAccount(String accountHolderName, String accountNumber) {

    // 4. Wird für den Namen (oder die Kontonummer) ein ungültiger Wert übergeben, so wird eine geeignete
    // ungeprüfte Ausnahme geworfen. (SAH: Siehe setAccountHolderName methode)
    setAccountHolderName(accountHolderName);

    // 4. Wird für (den Namen oder) die Kontonummer ein ungültiger Wert übergeben, so wird eine geeignete
    // ungeprüfte Ausnahme geworfen.
    if (!ACCOUNT_NUMBER_PATTERN.matcher(accountNumber).matches()) {
        throw new IllegalArgumentException("Ungültige Kontonummer.");
    }

    this.accountHolderName = accountHolderName;
    this.accountNumber = accountNumber;

    // 5. Das Eröffnungsdatum entspricht dem Tag, an dem das Bankkonto erstellt wurde und kann nicht
    // geändert werden.
    this.openingDate = LocalDate.now();

    // 6. Der Kontostand ist eine reelle Zahl und ist zu Beginn 0.
    this.balance = 0.0;
}

// -----

// 8. Alle Attribute des Bankkontos sind öffentlich lesbar. (SAH: Siehe Getter unten)

// 8.a get
public String getAccountHolderName() {
    return accountHolderName;
}
```

Informatik II – Tutorium 13 – Gruppe 7
Gruppenmitglieder: Hinterreiter, Gallinger, Stein

```
// -----

// 8.b get
    public String getAccountNumber() {
        return accountNumber;
    }

// -----

// 8.c get
    public LocalDate getOpeningDate() {
        return openingDate;
    }

// -----

// 8.d get
    public double getBalance() {
        return balance;
    }

// -----

// 2.c Der Name des Kontoinhabers kann jederzeit ge"andert und gelesen werden. (SAH: wird auch von Konstruktor
verwendet)
    public void setAccountHolderName(String accountHolderName) {

        // 4. Wird fur den Namen (oder die Kontonummer) ein ung "ultiger Wert " ubergeben, so wird "eine geeignete
ungeprüfte Ausnahme geworfen.
        if (!NAME_PATTERN.matcher(accountHolderName).matches()) {
            throw new IllegalArgumentException("Ungültiger Kontoinhabername.");
        }
        this.accountHolderName = accountHolderName;
    }

// -----

// 7. Danach kann der Kontostand sowohl erh"oht als auch verringert werden.
    public void addBalance(double amount) {
        this.balance = this.balance + amount;
    }

// -----

// 9. Zwei Bankkonten sind genau dann gleich, wenn ihre Kontonummern gleich sind.
    @Override
    public boolean equals(Object obj) {
```

Informatik II – Tutorium 13 – Gruppe 7
Gruppenmitglieder: Hinterreiter, Gallinger, Stein

```
// First step: Check for memory reference equality (Not needed but can improve performance)
//      "In general, it's good practice to use this as a quick initial check in the equals() method.
//      It's a low-cost operation that can sometimes avoid a potentially more expensive comparison
process."

    if (this == obj) {
        return true;
    }

    // Second step: Check if object is of type BankAccount
    //      Also avoids problem of potentially casting object that is not BankAccount below in third step
    if (!(obj instanceof BankAccount)) {
        return false;
    }

    // Third step: Check if accountNumber is identical
    //      and return
    BankAccount other = (BankAccount) obj;
    return accountNumber.equals(other.accountNumber);
}

// -----


// "It is generally a good practice to override the hashCode method when you override the equals method, and vice
versa.
// This is because the contract of hashCode is designed to work in conjunction with equals."
// SAH: Not overriding this could mean that two bankAccounts are identical according to equals() method but have
different hashes because default uses memory reference for hash -> Not consistent

@Override
public int hashCode() {
    return accountNumber.hashCode();
}

// -----

// 10. Wählen Sie eine geeignete Darstellung des Bankkontos als Zeichenkette.
@Override
public String toString() {
    return "BankAccount (Kontoinhabername = '" + accountHolderName + '\'' + ", Kontonummer = '" + accountNumber +
'\'' + ", Eröffnungsdatum = '" + openingDate + "', Kontostand = '" + balance + '\'' + ')';
}

// -----
}
```



1P

Aufgabe 28

```
// package blatt06;

import java.util.Scanner;
import java.util.regex.Pattern;
//

public class A28_main {

    // -----

    public static void main(String[] args) {

        // SAH: Create scanner
        Scanner input = new Scanner(System.in);
        input.useDelimiter(Pattern.compile("[\\r\\n]+"));

        // SAH: Create Bank
        Bank bank = new Bank();

        // SAH: Try create Bank accounts
        try {
            // SAH: Create Bank Accounts
            bank.createBankAccount("Dagobert Duck", "ENTENHAUSEN1");
            bank.createBankAccount("Dagobert Duck", "ENTENHAUSEN2");
            bank.createBankAccount("Dagobert Duck", "ENTENHAUSEN3");

            bank.createBankAccount("Donald Duck", "ENTENHAUSEN4");
            bank.createBankAccount("Donald Duck", "PHANTOMIAS12");

            bank.createBankAccount("Klaas Klever", "ENTENHAUSEN5");

            // SAH: initialize int i
            int i = 0;

            // SAH: Add Balance
            bank.addBalance("ENTENHAUSEN2", -100);
            bank.addBalance("PHANTOMIAS12", 200);

            // SAH: While loop addBalance
            while (i < 5) {
                bank.addBalance("ENTENHAUSEN2", -150);
                bank.addBalance("ENTENHAUSEN4", -200);
                bank.addBalance("ENTENHAUSEN3", -500);
                i++;
            }

            // SAH: While loop addBalance
```

Informatik II – Tutorium 13 – Gruppe 7
Gruppenmitglieder: Hinterreiter, Gallinger, Stein

```
        while (i < 50) {
            bank.addBalance("ENTENHAUSEN1", 500);
            bank.addBalance("ENTENHAUSEN5", 250);
            i++;
        }
    }

    catch (IllegalBankingException e) {
        throw new RuntimeException(e);
    }

    // SAH: Print out
    System.out.println("Does Dagobert Duck have more money than Klaas Klever? " +
        (bank.getBalanceByHolder("Dagobert Duck") > bank.getBalanceByHolder("Klaas Klever")));

    // SAH: Try to addBalance
    try {
        bank.addBalance("ENTENHAUSEN4", -1000);
    }

    catch (IllegalBankingException e) {
        System.out.println("Donald Duck can't withdraw 1000 Euros");
    }

}

// -----
}
```

```
// Aufgabenstellung:

// 1. Eine Bank verwaltet Bankkonten, die einzigartige Kontennummern, einen Inhaber sowie einen Kontostand haben.

// 2. Eine Bank kann neue Konten eröffnen lassen, dazu bekommt Sie den Namen des Inhabers sowie die gewünschte
Kontonummer gegeben.

// 2.a Ist bereits ein Konto mit dieser Kontonummer bei der Bank angelegt worden, so wird eine
IllegalBankingException geworfen (Die Ausnahme wird in der nächsten Teilaufgabe implementiert).

// 3. Die Bank stellt eine Möglichkeit zur Verfügung, die Summe aller Kontostände eines Inhabers zu
erhalten.

// 3.a Wird versucht, die Summe aller Kontostände für einen Inhaber zu erhalten, der kein Konto in dieser Bank
hat, so wird die Summe 0 zurückgegeben.

// 4. Außerdem kann von bzw. auf einem Konto mit einer gegebenen Konto-Nummer Geld abgehoben bzw. eingezahlt
werden.
```


Informatik II – Tutorium 13 – Gruppe 7
Gruppenmitglieder: Hinterreiter, Gallinger, Stein

```
// 4.a Wenn versucht wird, mehr als 500 Euro abzuheben oder einzuzahlen, oder eine nicht existente Kontonummer
verwendet wird, so wird eine IllegalBankingException geworfen.

// 5. Erzeugen Sie für die Berechnung der Summe aller Kontostände einen geeigneten Stream.
//

// Necessary packages/classes:
import java.util.*;

//

public class Bank {

    // Attribute:
    // 1. Eine Bank verwaltet Bankkonten, die einzigartige Kontennummern, einen Inhaber sowie einen Kontostand haben.
    private Map<String, BankAccount> accountsByNumber;
    private Map<String, List<BankAccount>> accountsByHolder;

    // -----

    // Konstruktor:
    // Not really needed because no specific logic to construct class necessary. HashMaps could be created above in
    "Attribute" instead.
    public Bank() {
        accountsByNumber = new HashMap<>();
        accountsByHolder = new HashMap<>();
    }

    // -----

    // 2. Eine Bank kann neue Konten eröffnen lassen, dazu bekommt Sie den Namen des Inhabers sowie die gewünschte
    Kontonummer gegeben.
    // createBankAccount
    public void createBankAccount(String accountHolderName, String accountNumber) throws IllegalBankingException {

        // 2.a Ist bereits ein Konto mit dieser Kontonummer bei der Bank angelegt worden,
        // so wird eine IllegalBankingException geworfen (Die Ausnahme wird in der nächsten Teilaufgabe
        implementiert).
        if (accountsByNumber.containsKey(accountNumber)) {
            throw new IllegalBankingException("Konto mit dieser Kontonummer existiert bereits.");
        }

        // SAH: Call BankAccount to create new account
        BankAccount newAccount = new BankAccount(accountHolderName, accountNumber);

        // by Number:
        // SAH: Place memory reference to newly created account in HashMap with accountNumber as key
        accountsByNumber.put(accountNumber, newAccount);
    }
}
```

Informatik II – Tutorium 13 – Gruppe 7
Gruppenmitglieder: Hinterreiter, Gallinger, Stein

```
// byHolder:
//      SAH: If holder not yet present in HashMap, create new ArrayList to store associated
accountNumbers in it
    accountsByHolder.putIfAbsent(accountHolderName, new ArrayList<>());
//      SAH: adds newAccount to associated key for accountHolderName
    accountsByHolder.get(accountHolderName).add(newAccount);
}

// -----

// 3. Die Bank stellt eine Möglichkeit zur Verfügung, die Summe aller Kontostände eines Inhabers zu
erhalten.
public double getBalanceByHolder(String accountHolderName) {

    // 5. Erzeugen Sie für die Berechnung der Summe aller Kontostände einen geeigneten Stream.
    // 3.a Wird versucht, die Summe aller Kontostände für einen Inhaber zu erhalten, der kein Konto in dieser
Bank hat, so wird die Summe 0 zurückgegeben.

    return accountsByHolder.getDefault(accountHolderName, Collections.emptyList()) // gets me the
accountHolderName list or an emptylist if no mapping
        .stream()
        .mapToDouble(BankAccount::getBalance)
        .sum();
}

// -----

// 4. Außerdem kann von bzw. auf einem Konto mit einer gegebenen Konto-Nummer Geld abgehoben bzw. eingezahlt
werden.
public void addBalance(String accountNumber, double amount) throws IllegalBankingException {


    // 4.a Wenn versucht wird, mehr als 500 Euro abzuheben oder einzuzahlen, oder eine nicht existente Kontonummer
verwendet wird, so wird eine IllegalBankingException geworfen.
    if (Math.abs(amount) > 500) {
        throw new IllegalBankingException("Der Betrag darf 500 Euro nicht übersteigen.");
    }

    // Holt sich reference zu passendem Account
    BankAccount account = accountsByNumber.get(accountNumber);

    if (account == null) {
        throw new IllegalBankingException("Konto mit dieser Kontonummer existiert nicht.");
    }

    // Added balance to account über addBalance
    account.addBalance(amount);
}


// -----
```



}

Informatik II – Tutorium 13 – Gruppe 7
Gruppenmitglieder: Hinterreiter, Gallinger, Stein

```
public class IllegalBankingException extends Exception {  
  
    public IllegalBankingException(String message) {  
        super(message);  
    }  
}
```



1P

Aufgabe 29

```
// Aufgabenstellung:
// 1. Es soll den Titel Aufgabe29a haben.

// 2. Es soll im Norden von links nach rechts angeordnet die Beschriftung Kontonummer und ein aktiviertes
einzeiliges Textfeld der Breite 20 anzeigen.
// 2.a Beschriftung und Textfeld sollen beide dieselbe Schrift mit Schriftgröße 20 haben.

// 3. Es soll im Süden von links nach rechts angeordnet zwei Schaltflächen mit dem Text Okay und Abbrechen
anzeigen.
// 4. Der Nord- und der Südbereich sollen unterschiedliche Graufarben als Hintergrundfarbe haben.
// 5. Durch Klicken auf das Fensterkreuz soll die Anwendung beendet werden.
//

// Necessary packages/classes:
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JButton;
import java.awt.Color;
import java.awt.Font;
import java.awt.BorderLayout;
//

public class A29a_main extends JFrame {

    // -----

    // Attribute:
    private JPanel northPanel;
    private JLabel label;
    private JTextField textField;

    private JPanel southPanel;
    private JButton okButton;
    private JButton cancelButton;

    // -----

    // Konstruktor:
    public A29a_main() {

        // ++++++

        // Fenstertitel
```

Informatik II – Tutorium 13 – Gruppe 7
Gruppenmitglieder: Hinterreiter, Gallinger, Stein

```
// 1. Es soll den Titel Aufgabe29a haben.
setTitle("Aufgabe29a");

// ++++++

// Nordbereich
northPanel = new JPanel();

// 4. Der Nord- und der Sudbereich sollen unterschiedliche Graufarben als Hintergrundfarbe haben.
northPanel.setBackground(Color.LIGHT_GRAY);

label = new JLabel("Kontonummer");
textField = new JTextField(20);

// 2.a Beschriftung und Textfeld sollen beide dieselbe Schrift mit Schriftgröße 20 haben.
label.setFont(new Font("Arial", Font.PLAIN, 20));
textField.setFont(new Font("Arial", Font.PLAIN, 20));

// 2. Es soll im Norden von links nach rechts angeordnet die Beschriftung Kontonummer und ein aktiviertes
// einzelnes Textfeld der Breite 20 anzeigen.
northPanel.add(label);
northPanel.add(textField);

add(northPanel, BorderLayout.NORTH);

// ++++++

// Südbereich
southPanel = new JPanel();

// 4. Der Nord- und der Sudbereich sollen unterschiedliche Graufarben als Hintergrundfarbe haben.
southPanel.setBackground(Color.DARK_GRAY);

okButton = new JButton("Okay");
cancelButton = new JButton("Abbrechen");

// 3. Es soll im Süden von links nach rechts angeordnet zwei Schaltflächen mit dem Text Okay und
// Abbrechen anzeigen.
southPanel.add(okButton);
southPanel.add(cancelButton);

add(southPanel, BorderLayout.SOUTH);

// ++++++

// Fenster schließen
// 5. Durch Klicken auf das Fensterkreuz soll die Anwendung beendet werden.
```

Informatik II – Tutorium 13 – Gruppe 7
Gruppenmitglieder: Hinterreiter, Gallinger, Stein

```
setDefaultCloseOperation(EXIT_ON_CLOSE);


// ++++++

// Fenstergröße und Sichtbarkeit
setSize(500, 200); pack();
setLocationRelativeTo(null);
setVisible(true);
}

// -----

// main
public static void main(String[] args) {
    new A29a_main();
}

// -----
}
```



```
// Necessary packages/classes:
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.JButton;
import java.awt.GridLayout;
import java.awt.Font;
import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
//

public class A29b_main extends JFrame {

    // -----

    // Attribute:
    private JPanel inputPanel;
    private JTextField inputField;

    private JPanel buttonPanel;
    private JButton[] numberButtons;
    private JButton clearButton;
    private JButton okayButton;
```

Informatik II – Tutorium 13 – Gruppe 7
Gruppenmitglieder: Hinterreiter, Gallinger, Stein

```
// -----  
  
// Konstruktor:  
public A29b_main() {  
  
    // ++++++  
  
    // Fenstertitel  
    setTitle("A29b");  
  
    // ++++++  
  
    // A. Weißes Feld für Input  
    inputPanel = new JPanel();  
    inputPanel.setLayout(new BorderLayout());  
  
    inputField = new JTextField();  
    inputField.setFont(new Font("Arial", Font.PLAIN, 20));  
    inputField.setEditable(false);  
    inputPanel.add(inputField, BorderLayout.CENTER);  
  
    add(inputPanel, BorderLayout.NORTH);  
  
    // ++++++  
  
    // B. Buttons  
    buttonPanel = new JPanel();  
    buttonPanel.setLayout(new GridLayout(4, 3));  
  
    numberButtons = new JButton[10]; // 0 to 9  
  
    // Add buttons 7, 8, 9  
    for (int i = 7; i <= 9; i = i + 1) {  
        numberButtons[i-1] = createNumberButton(String.valueOf(i));  
        buttonPanel.add(numberButtons[i-1]);  
    }  
  
    // Add buttons 4, 5, 6  
    for (int i = 4; i <= 6; i = i + 1) {  
        numberButtons[i-1] = createNumberButton(String.valueOf(i));  
        buttonPanel.add(numberButtons[i-1]);  
    }  
  
    // Add buttons 1, 2, 3  
    for (int i = 1; i <= 3; i = i + 1) {  
        numberButtons[i-1] = createNumberButton(String.valueOf(i));  
        buttonPanel.add(numberButtons[i-1]);  
    }  
}
```


Informatik II – Tutorium 13 – Gruppe 7
Gruppenmitglieder: Hinterreiter, Gallinger, Stein

```
}

// Add 0, Clear, Okay buttons
numberButtons[0] = createNumberButton("0");
buttonPanel.add(numberButtons[0]);

clearButton = new JButton("Clear");
clearButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        inputField.setText("");
    }
});
buttonPanel.add(clearButton);

okayButton = new JButton("Okay");
buttonPanel.add(okayButton);

add(buttonPanel, BorderLayout.CENTER);

// ++++++

// Fenster schließen
setDefaultCloseOperation(EXIT_ON_CLOSE);

// ++++++

// Fenstergröße und Sichtbarkeit
pack();
setLocationRelativeTo(null);
setVisible(true);
}

// -----

private JButton createNumberButton(String text) {

    JButton button = new JButton(text);

    button.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            inputField.setText(inputField.getText() + text);
        }
    });
    return button;
}

// -----
```

```
public static void main(String[] args) {  
    new A29b_main();  
}  
  
// -----  
  
}
```

Super und sogar weit über
die Aufgabenstellung hinaus!



Σ 4P sehr schön!

H