

תוכנה 1

תרגול מס' 6
מחלקות ומימושן על פי מפרט
דיאגרמות

המצב הפנימי של אובייקט

- המצב הפנימי של עצם מיוצג ע"י נתוניו (שדותיו)
- שדות עצם יהיו לרוב עם הרשאת גישה פרטית
- במקרה של חשבון בנק:
 - מצב פנימי: מכיל בין היתר שדה לייצוג היתרה
 - מאיזה טיפוס?

```
public class BankAccount {  
    ...  
    private double balance;  
}
```

שרותי המחלקה

ישנם 3 סוגי שירותים (מתודות, פונקציות, פרוצדורות):

■ שאילתות (queries, accessors)

- מחזירות ערך ללא שינוי המצב הפנימי
- כגון: בירור יתרה

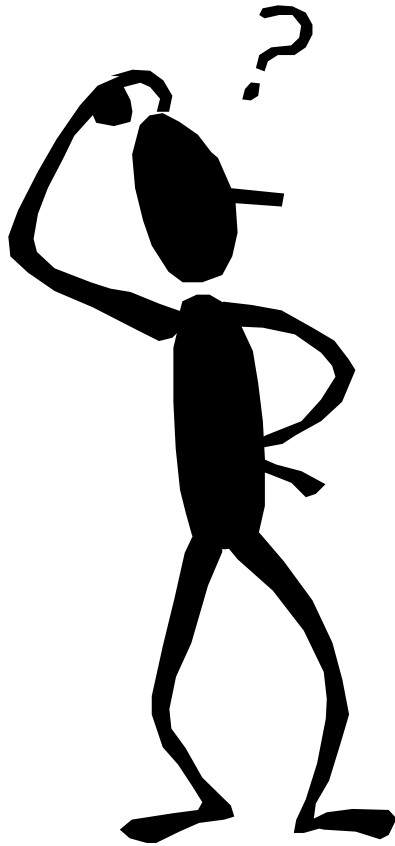
■ פקודות (commands, transformers, mutators)

- מבצעות שינוי במצב הפנימי של העצם
- כגון: משיכה, הפקדה

■ בנאים (constructors)

- יצירת עצם חדש
- כגון: יצירת חשבון חדש

שאלות BankAccount



■ ברור יתרה:

- ארגומנטים?
- מה טיפוס הערך המוחזר?
- חוזה? (תנאי קדם? תנאי אחר?)

■ פרטים על החשבון:

- מספר חשבון?
- פרטים על בעל החשבון?
 - ❖ תעודת זהות?
 - ❖ גיל?

שאלות BankAccount

```
public class BankAccount {  
    public double getBalance() {  
        ...;  
    }  
  
    public long getAccountNumber() {  
        ...;  
    }  
  
    public Customer getOwner () {  
        ...;  
    }  
}
```

שאלות

מצב
פנימי

```
private double balance;  
private long accountNumber;  
private Customer owner;
```

- מוסכמה: הגישה לשדה field תעשה בעזרת המתודה `.getField()`
- שמירה על מוסכמה זו הכרחית בסביבות JavaBeans ו- GUI Builders

getter/setter

- יש חשיבות לגישה לנתונים דרך מתודות. מדוע?
- לא כל שדה עם נראות פרטית (`private`) צריך `getter/setter` ציבורי
- יצירה אוטומטית של שרותים אלו עבור כל שדה פוגמת בעקרון הסתרת המידע
- למשל: עבור השדה `balance`
 - האם דרוש `?getter`
 - כן, זהו חלק מהממשק של חשבון בנק
 - האם דרוש `?setter`
 - לא בהכרח, פעולות של משיכה או הפקדה אמנם משפיעות על היתרה, אבל פעולה של שינוי יתרה במנותק מהן אינה חלק מהממשק

פקודת ה-'להפקיד'

■ המתודה: deposit

■ סכום הכסף המופקד מתווסף ליתרה בחשבון

■ ארגומנטים?

■ ערך מוחזר?

■ חוזה? (תנאי קדם?, תנאי אחר?)

מוסכמה: שמות פקודות
הם שמות פועל



פקודת ה-'להפקיד'

```
/**  
 * Makes a deposit to the account  
 * @pre ?????????????????????????????????????????????????????????  
 * @post ?????????????????????????????????????????????????????????  
 */  
public void deposit(double amount) {  
    ...;  
}
```


פקודת ה-'להפקיד'

```
/**  
 * Makes a deposit to the account  
 * @pre amount > 0  
 * @post ??????????????????????????????????????????????????????????????  
 */  
public void deposit(double amount) {  
    ...;  
}
```

בגישת תכנות לפי חוזה תנאיי הקדם לא יבדקו בגוף המתודה. אחרת: תכנות מתגונן.

פקודת ה-'להפקיד'

```
/**  
 * Makes a deposit to the account  
 * @pre  amount > 0  
 * @post getBalance() == $prev(getBalance()) + amount  
 */  
public void deposit(double amount) {  
    ...;  
}
```

מימוש ה-'להפקיד'

```
/**  
 * Makes a deposit to the account  
 * @pre  amount > 0  
 * @post getBalance() == $prev(getBalance()) + amount  
 */  
public void deposit(double amount) {  
    balance += amount;  
}
```

פקודת ה-'למשוך'

■ המתודה: withdraw

■ סכום הכסף המבוקש יורד מיתרת החשבון.
משיכת יתר (אוברדרפט) אינו אפשרי.

■ ארגומנטים?

■ ערך מוחזר?

■ חוזה? (תנאי קדם? תנאי אחר?)



פקודת ה-'למשוך'

```
/**  
 * Withdraw amount from the account  
 * @pre 0 < amount <= getBalance()  
 * @post getBalance() == $prev(getBalance()) - amount  
 */  
public void withdraw(double amount) {  
    balance -= amount;  
}
```

דיון – העברה בנקאית

מספר חלופות למימוש העברת סכום מחשבון לחשבון:
■ אפשרות א: מתודה סטטית שתקבל שני חשבונות
בנק ותבצע ביניהם העברה:

```
/**  
 * Makes a transfer of amount from one account to the other  
 * @pre 0 < amount <= from.getBalance()  
 * @post to.getBalance() == $prev(to.getBalance()) + amount  
 * @post from.getBalance() == $prev(from.getBalance()) - amount  
 */  
public static void transfer(double amount,  
                             BankAccount from,  
                             BankAccount to) {  
    from.withdraw(amount);  
    to.deposit(amount);  
}
```

דיון – העברה בנקאית

אפשרות ב:

```
/**  
 * Makes a transfer of amount from the current account to  
 * the other one  
 */  
public void transferTo(double amount,  
                       BankAccount other) {  
    other.deposit(amount);  
    balance -= amount;  
}
```

דיון – העברה בנקאית

אפשרות ג: העמסת withdraw ו/או deposit שיקבלו שני ארגומנטים (סכום והפנייה לחשבון נוסף):

```
/**
 * Makes a transfer of amount from other to the current account
 * @pre 0 < amount <= other.getBalance()
 * @post getBalance() == $prev(getBalance()) + amount
 * @post other.getBalance() == $prev(other.getBalance()) - amount
 */
public void deposit(double amount, BankAccount other) {
    other.withdraw(amount);
    balance += amount;
}
```


שמורת המחלקה (Class Invariant)

■ צריכה להתקיים "תמיד"

■ לפני ואחרי ביצוע כל מתודה ציבורית

■ אחרי הבנאי

■ במחלקה חשבון בנק:

■ חשבון חייב להיות עם יתרה אי שלילית

■ לכל חשבון קיים מספר מזהה במערכת

■ לכל חשבון יש בעלים

שמורת BankAccount

```
/**
 * @inv getBalance() >= 0
 * @inv getAccountNumber() > 0
 * @inv getOwner() != null
 */
public class BankAccount {
    ...
}
```

בנאי

■ תפקיד: ליצור עצם חדש המקיים את שמורת המחלקה

■ בנאי לא אמור לכלול לוגיקה נוספת פרט לכך

■ במחלקה `BankAccount`:

■ בנאי ברירת המחדל יוצר עצם שאינו מקיים את השמורה!

■ יש דברים שאינם באחריות המחלקה. למשל:

■ מי דואג לתקינות מספרי חשבון? (למשל שיהיו שונים)

■ מי מנהל את מאגר הלקוחות?

בנאי BankAccount

```
/**
 * Constructs a new account and sets its owner and identifier
 * @pre id > 0
 * @pre customer != null
 * @post getOwner() == customer
 * @post getAccountNumber() == id
 * @post getBalance() == 0
 */
public BankAccount(Customer customer, long id) {
    accountNumber = id;
    owner = customer;
}
```

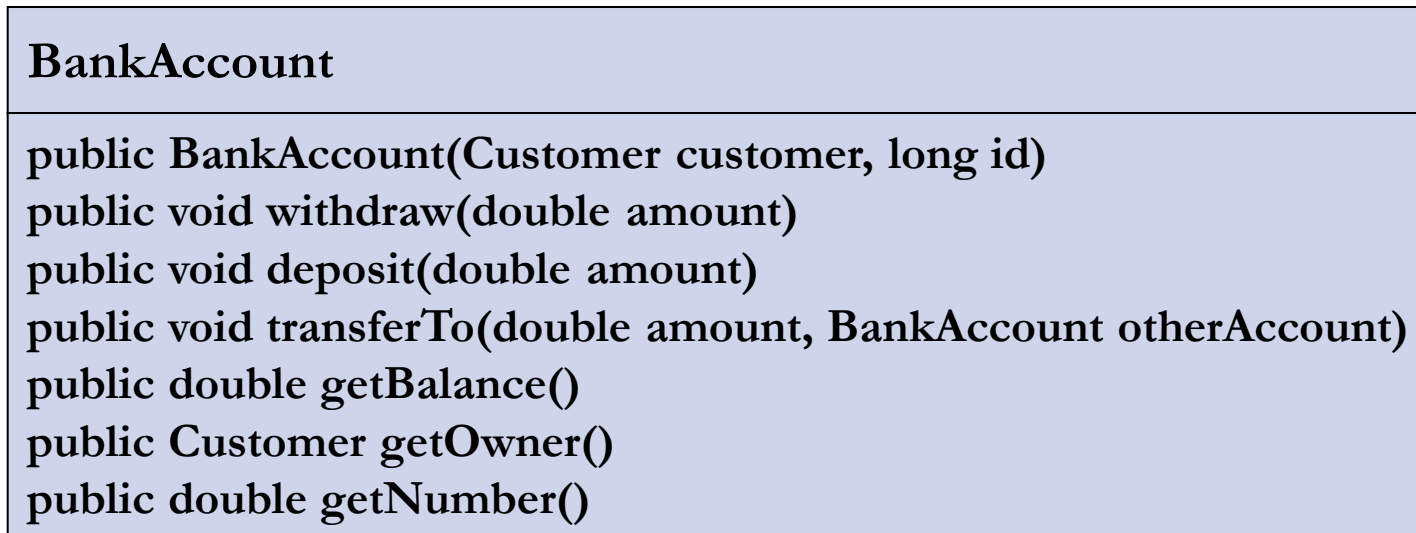
פ'אכרנות

המערכת הבנקאית

- נתאר את מערכת התוכנה שלנו בעזרת דיאגרמות
- דיאגרמות סטטיות:
- תיאור היחסים בין המחלקות השונות במערכת
- דיאגרמות דינאמיות:
- תיאור ההתנהגות של המערכת בזמן ריצה
 - מצב האובייקטים
 - תיאור של תרחיש

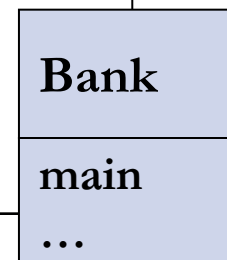
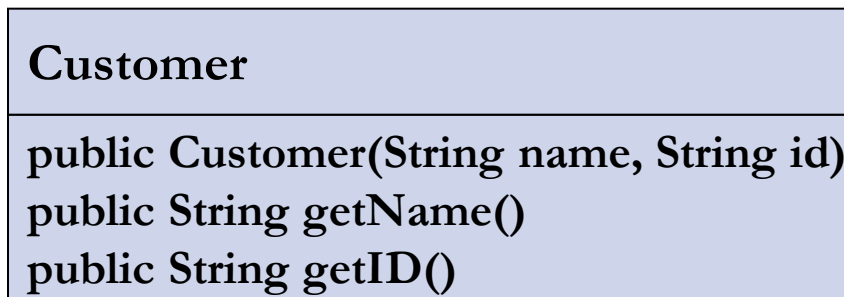


Class Diagram



קשר לוגי של שייכות (למשל,
ב- BankAccount יש שדה
Customer), לא בלעדית

**Aggregation
(has-a)**



Association

קשר כללי בין
מחלקות, למשל,
אחת משתמשת
בשנייה באחת
המתודות

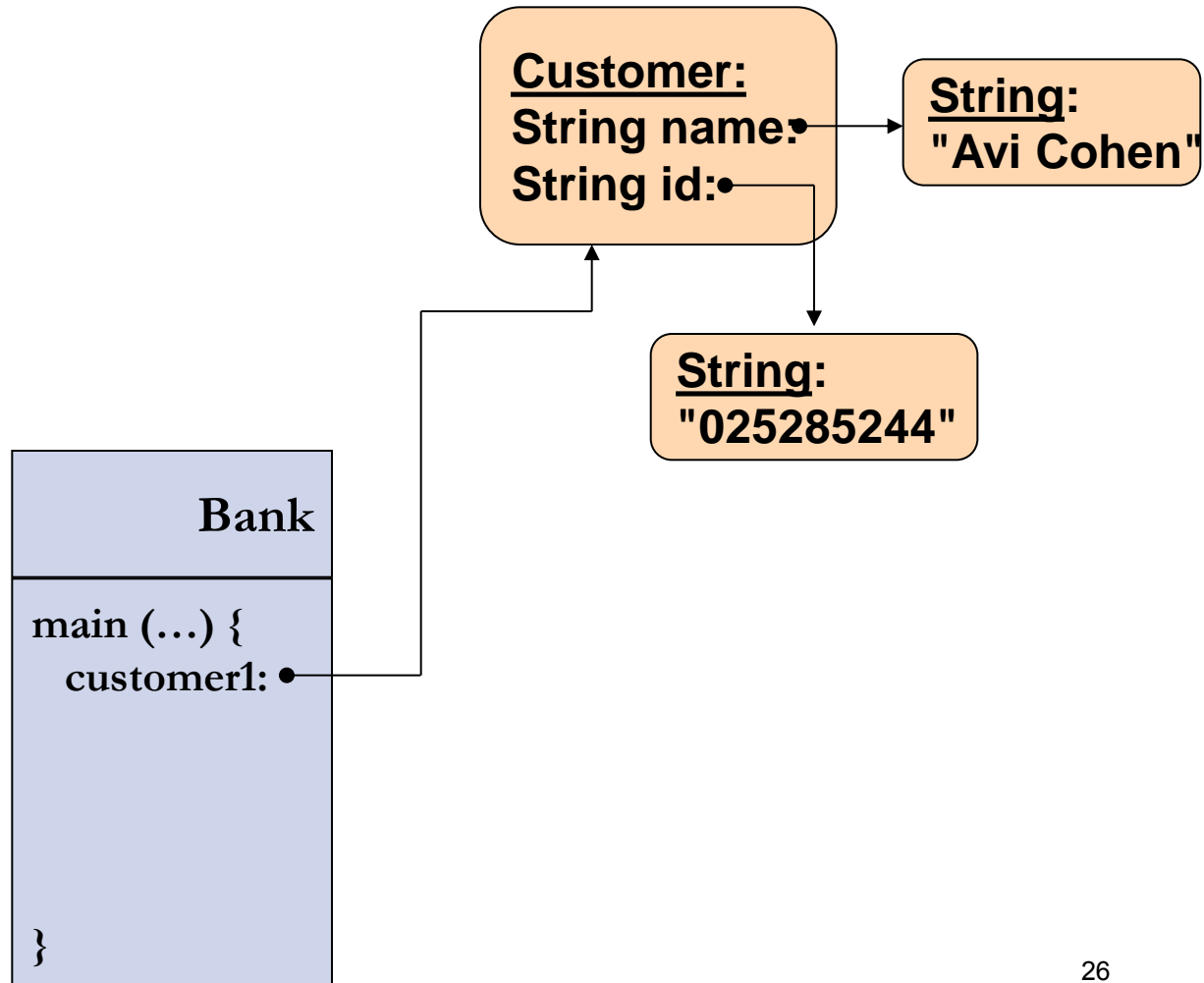
המחלקה Customer

```
public class Customer {  
    public Customer(String name, String id) {  
        this.name = name;  
        this.id = id;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public String getID() {  
        return id;  
    }  
  
    private String name;  
    private String id;  
}
```


Toy Bank Program

```
public class Bank {  
    public static void main(String[] args) {  
        → Customer customer1 = new Customer("Avi Cohen", "025285244");  
        Customer customer2 = new Customer("Rita Stein", "024847638");  
  
        BankAccount account1 = new BankAccount(customer1, 1234);  
        BankAccount account2 = new BankAccount(customer2, 5678);  
        BankAccount account3 = new BankAccount(customer1, 2984);  
  
        account1.deposit(1000);  
        account2.deposit(500);  
        account1.transferTo(100, account3);  
        account2.withdraw(300);  
  
        System.out.println("account1 has " + account1.getBalance());  
        System.out.println("account2 has " + account2.getBalance());  
    }  
}
```

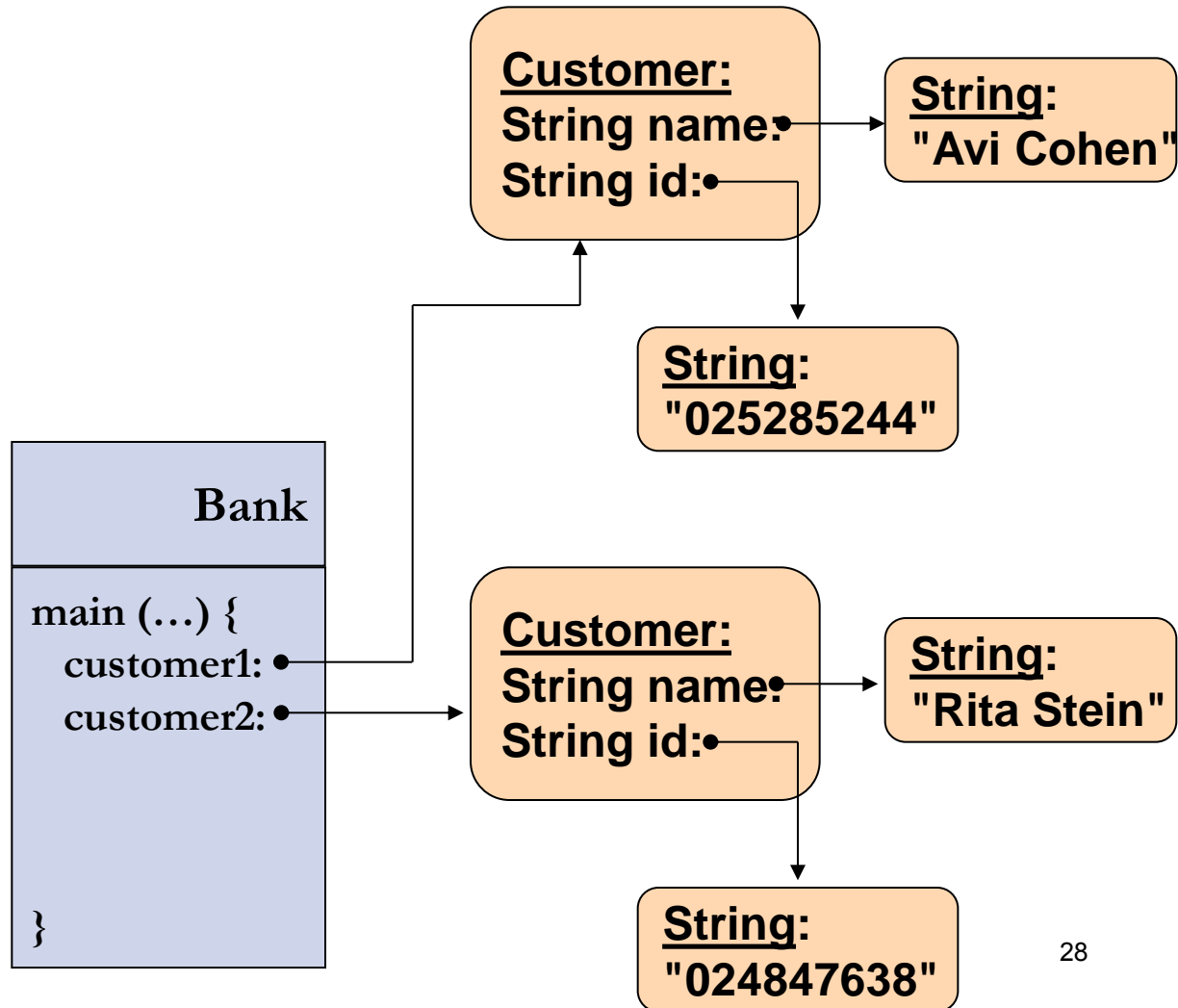
Object Diagram



Toy Bank Program

```
public class Bank {  
    public static void main(String[] args) {  
        Customer customer1 = new Customer("Avi Cohen", "025285244");  
        → Customer customer2 = new Customer("Rita Stein", "024847638");  
  
        BankAccount account1 = new BankAccount(customer1, 1234);  
        BankAccount account2 = new BankAccount(customer2, 5678);  
        BankAccount account3 = new BankAccount(customer1, 2984);  
  
        account1.deposit(1000);  
        account2.deposit(500);  
        account1.transferTo(100, account3);  
        account2.withdraw(300);  
  
        System.out.println("account1 has " + account1.getBalance());  
        System.out.println("account2 has " + account2.getBalance());  
    }  
}
```

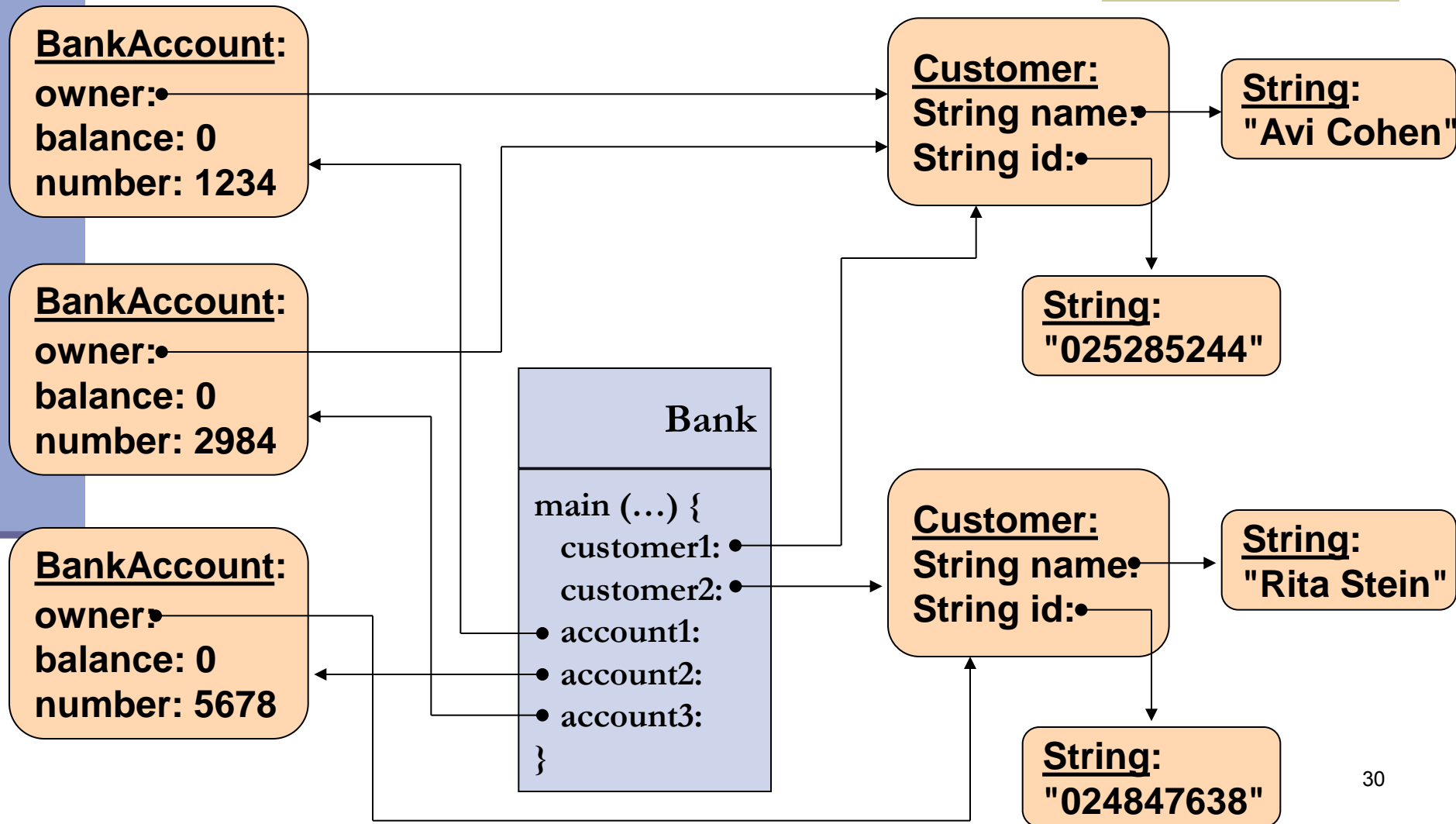
Object Diagram



Toy Bank Program

```
public class Bank {  
    public static void main(String[] args) {  
        Customer customer1 = new Customer("Avi Cohen", "025285244");  
        Customer customer2 = new Customer("Rita Stein", "024847638");  
        → BankAccount account1 = new BankAccount(customer1, 1234);  
        BankAccount account2 = new BankAccount(customer2, 5678);  
        BankAccount account3 = new BankAccount(customer1, 2984);  
  
        account1.deposit(1000);  
        account2.deposit(500);  
        account1.transferTo(100, account3);  
        account2.withdraw(300);  
  
        System.out.println("account1 has " + account1.getBalance());  
        System.out.println("account2 has " + account2.getBalance());  
    }  
}
```

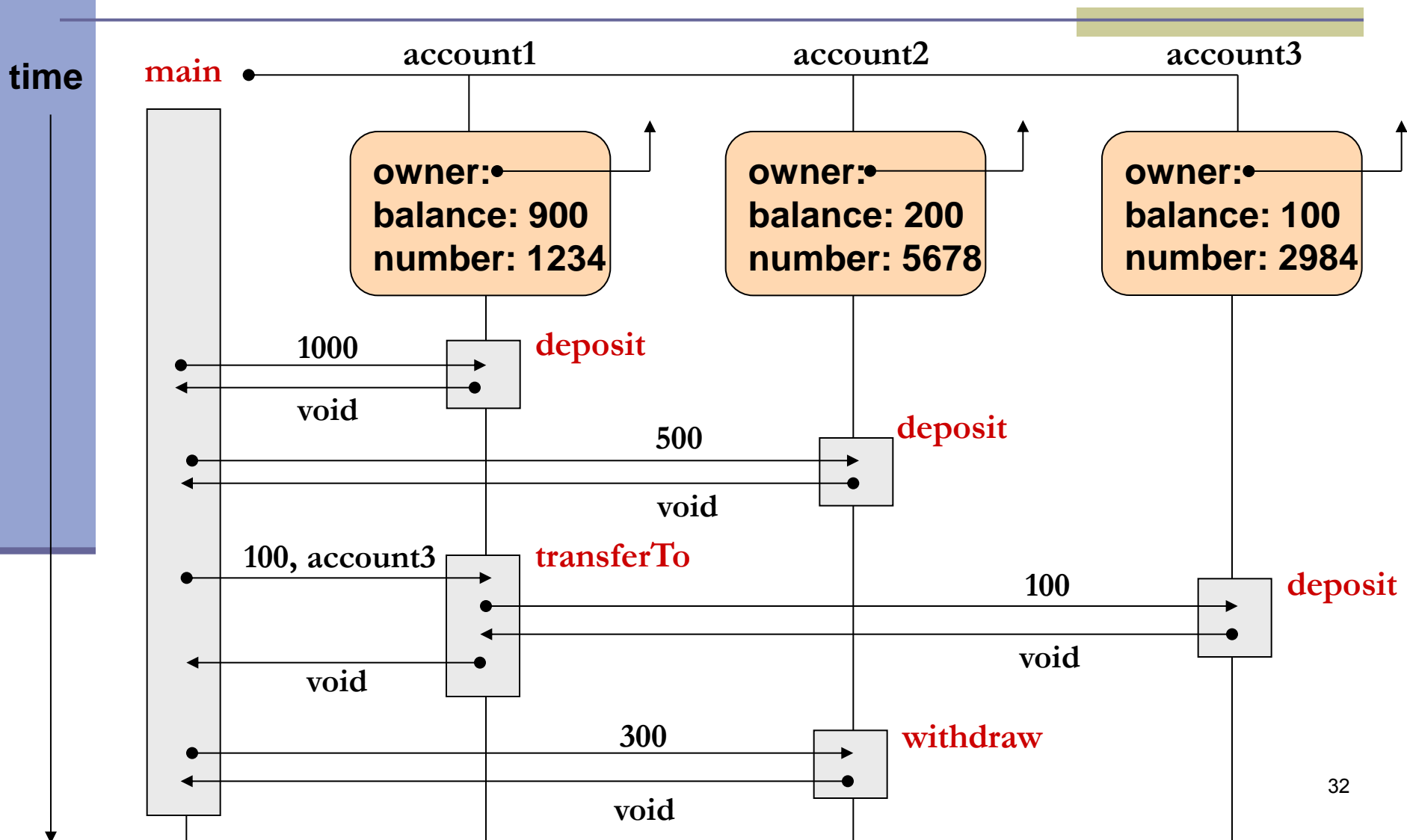
Object Diagram



Message Sequence Chart

```
public class Bank {  
    public static void main(String[] args) {  
        Customer customer1 = new Customer("Avi Cohen", "025285244");  
        Customer customer2 = new Customer("Rita Stein", "024847638");  
  
        BankAccount account1 = new BankAccount(customer1, 1234);  
        BankAccount account2 = new BankAccount(customer2, 5678);  
        BankAccount account3 = new BankAccount(customer1, 2984);  
  
        account1.deposit(1000);  
        account2.deposit(500);  
        account1.transferTo(100, account3);  
        account2.withdraw(300);  
  
        System.out.println("account1 has " + account1.getBalance());  
        System.out.println("account2 has " + account2.getBalance());  
    }  
}
```

Message Sequence Chart



Output

```
public class Bank {  
    public static void main(String[] args) {  
        Customer customer1 = new Customer("Avi Cohen", "025285244");  
        Customer customer2 = new Customer("Rita Stein", "024847638");  
  
        BankAccount account1 = new BankAccount(customer1, 1234);  
        BankAccount account2 = new BankAccount(customer2, 5678);  
        BankAccount account3 = new BankAccount(customer1, 2984);  
  
        account1.deposit(1000);  
        account2.deposit(500);  
        account1.transferTo(100, account3);  
        account2.withdraw(300);  
  
        System.out.println("account1 has " + account1.getBalance());  
        System.out.println("account2 has " + account2.getBalance());  
    }  
}
```

output: account1 has 900.0
account2 has 200.0