

CS105

Introduction to Object-Oriented Programming

Prof. Dr. Nizamettin AYDIN

naydin@itu.edu.tr

nizamettin.aydin@ozyegin.edu.tr

Extending Bank Account Example

Outline

- Primitive types
- Object types
- Memory Allocation
- Heaps
- Member Functions
- Memory Model
- Class Instances
- Standard Streams
- Arrays

Bank Account

- Lets implement a bank account program
- What type of information do we need for a bank account?
 - Account ID (int)
 - Balance (double)
 - Currency (String)

```
public class AccountTest {  
  
    public static void main(String [] args) {  
  
        int account1ID = 1;  
        double account1Balance = 1000;  
        String account1Currency="TL";  
  
    }  
  
}
```

Bank Account

```
public class AccountTest {  
  
    public static void main(String [] args) {  
  
        int account1ID = 1;  
        double account1Balance = 1000;  
        String account1Currency="TL";  
  
    }  
}
```

- **int** and **double** are primitive types
- **String** is an object type
- What is primitive type? What is object type?

Primitive types

- 8 types
 - byte
 - short (16 bit signed)
 - int (32 bit signed)
 - long (64 bit)
 - float (32 bit floating point)
 - double (64 bit floating point)
 - boolean
 - char

Object types

- Everything else that is not primitive
 - Arrays
 - All other user defined classes
- An object can be created with the **new** keyword
 - `int [] myArray = new int [10];`
- When **new** keyword is used, some space to store this object is allocated from the memory.
- Where in memory?

Bank Account

```
public class AccountTest {  
  
    public static void main(String [] args) {  
  
        int accountID = 1;  
        double accountBalance = 1000;  
        String accountCurrency="TL";  
  
    }  
}
```

- **int** and **double** are primitive types
- **String** is an object type
- How are they represented in memory?

Bank Account

```
public class AccountTest {  
  
    public static void main(String [] args) {  
  
        int accountID = 1;  
        double accountBalance = 1000;  
        String accountCurrency="TL";  
  
    }  
}
```

- Primitive types are stored in **Stack**
- Objects are stored in **Heap**
- What is **Stack** and **Heap**?

Memory Allocation

- When you declare a variable in a program, Java allocates space for that variable from one of several memory regions:
- **Heap**
 - Holds objects created in the program
- **Stack**
 - Used during the execution of the program
 - Stack holds
 - short lived objects (local primitive types)
 - When a function is called a block of memory (stack frame) is allocated to hold the local variables.
 - It is removed when the execution of function finishes
 - references to other objects in the heap

Memory Allocation

- Heap vs. Stack
 - Heap holds the objects where Stack holds reference to these objects
- Objects
 - When new keyword is used, some space to store this object is allocated from the heap memory.
- **Variable declaration:**
 - Primitive type
 - `int myInt;`
 - Object type
 - `String myString;`

myInt

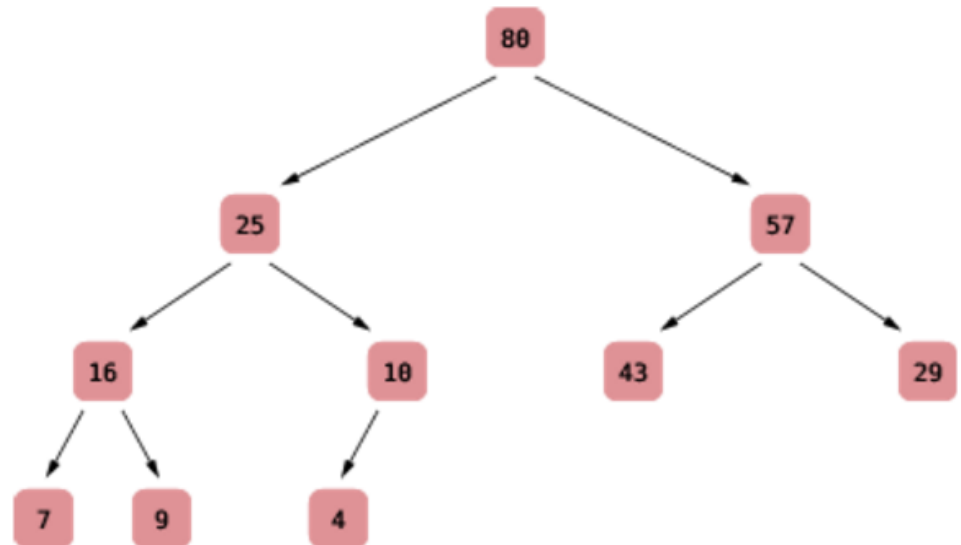


myString



Heaps

- A **heap** (or, for greater clarity, max-heap) is a binary tree that:
 - is almost complete: all nodes are filled except the last level may have some missing toward the right.
 - all nodes store values that are at least as large as the values stored in their descendants.
- The heap property ensures that the tree's largest element is stored in the root
- The shape of a heap is very regular
- In a heap, the left and right subtrees both store elements that are smaller than the root element



Memory Allocation

- **Variable assignment:**

- **Primitive type**

- `int myInt;`
 - `myInt = 5;`

`myInt`

`5`

- **Object type**

- `String myString;`
 - `myString = new String("Text");`

`myString`

`1234`

`1234`

`"Text"`

Memory Allocation

- **Variable assignment:**

- **Primitive type**

- `int myInt;`
 - `myInt = 5;`

myInt



- **Object type**

- `String myString;`
 - `myString = new String("Text");`

myString



- Instead of showing the address, we will use an arrow

Bank Account

```
public class AccountTest {  
  
    public static void main(String [] args) {  
  
        int account1ID = 1;  
        double account1Balance = 1000;  
        String account1Currency="TL";  
  
    }  
}
```

- Primitive types are stored in [Stack](#)

account1ID

1

account1Balance

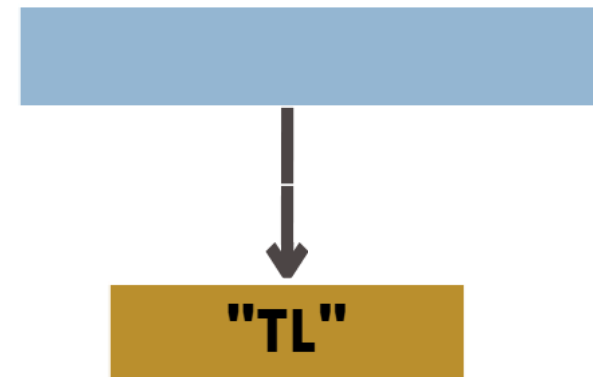
1000

Bank Account

```
public class AccountTest {  
  
    public static void main(String [] args) {  
  
        int accountID = 1;  
        double accountBalance = 1000;  
        String accountCurrency="TL";  
  
    }  
}
```

- Objects are stored in **Heap**
- Their reference is stored in **Stack**

accountCurrency



Bank Account

```
int account1ID = 1;  
double account1Balance = 1000;  
String account1Currency="TL";
```

account1ID

1

account1 Balance

1000

account1 Currency

"TL"

```
int account2ID = 2;  
double account2Balance = 800;  
String account2Currency="US";
```

account2ID

2

account2Balance

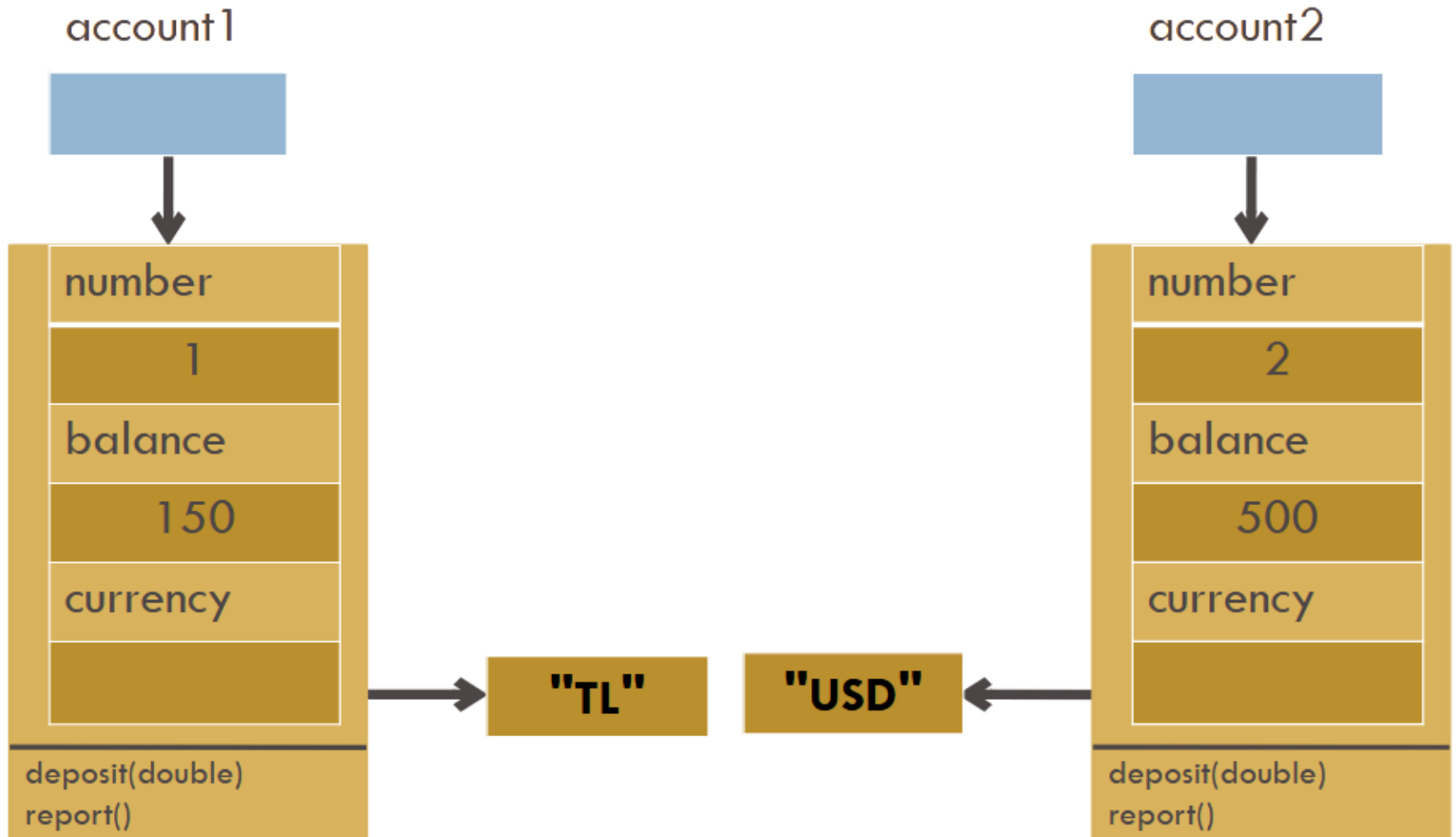
800

account2Currency

"US"

Member Functions

- Member functions can also be represented in memory diagrams



Current Account Class (Version 15)

```
public class Account {
    private int number;
    private double balance;
    private String currency;

    public Account(int number, double balance, String currency) {}
    public Account(int number, String currency) {}
    public Account(int number) {}

    public int getNumber() {}
    public double getBalance() {}
    public String getCurrency() {}

    public void setCurrency(String currency) {}

    private void checkSetCurrency (String c) {}

    public void deposit(double d) {}
    public void withdraw(double d) {}

    public void report() {}

    public String toString() {}
}
```

Another class

- Lets add another class

- Customer object

- Name
 - Account

```
private String name;  
private Account account;  
  
public Customer(String name, Account account) {  
    this.name = name;  
    this.account = account;  
}
```

Customer Class

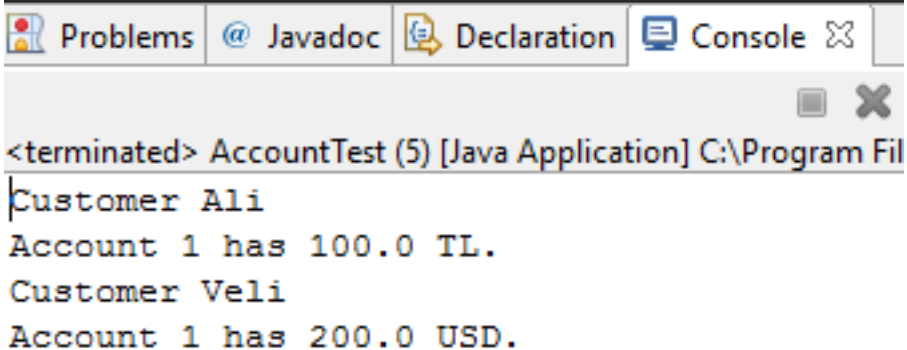
```
public String getName() {
    return this.name;
}
public Account getAccount() {
    return this.account;
}

public void deposit(double amount) {
    this.account.deposit(amount);
}
public void withdraw(double amount) {
    this.account.withdraw(amount);
}

public void report() {
    System.out.println("Customer " + this.name + " ");
    this.account.report();
}
```

Using Customer Class

```
public static void main(String[] args) {  
  
    Account account1 = new Account(1, 100, "TL");  
    Customer customer1 = new Customer("Ali", account1);  
  
    Account account2 = new Account(1, 200, "USD");  
    Customer customer2 = new Customer("Veli", account2);  
  
    customer1.report();  
    customer2.report();  
}
```

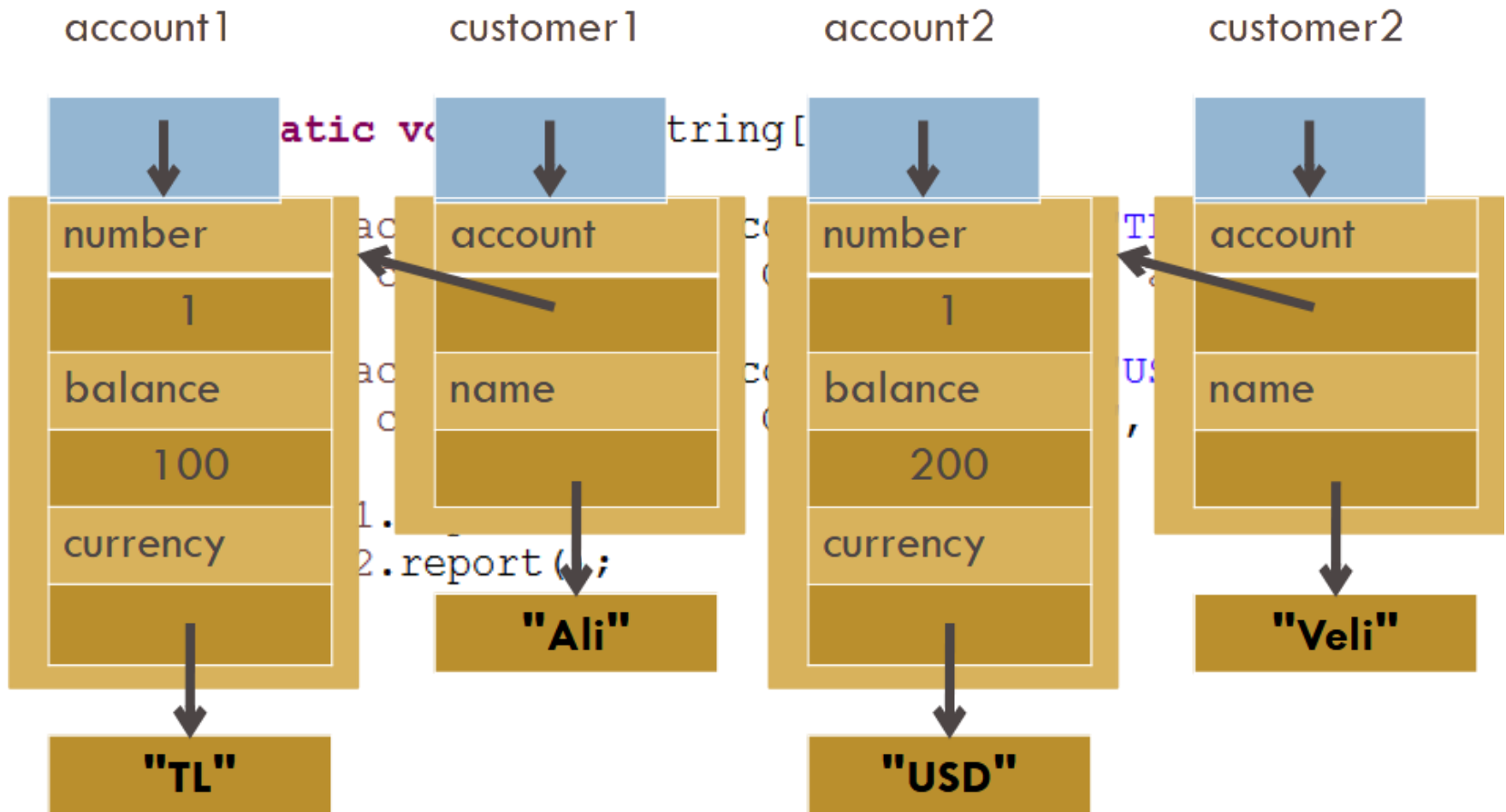


The screenshot shows a console window titled "<terminated> AccountTest (5) [Java Application] C:\Program Fil". The console output is as follows:

```
Customer Ali  
Account 1 has 100.0 TL.  
Customer Veli  
Account 1 has 200.0 USD.
```

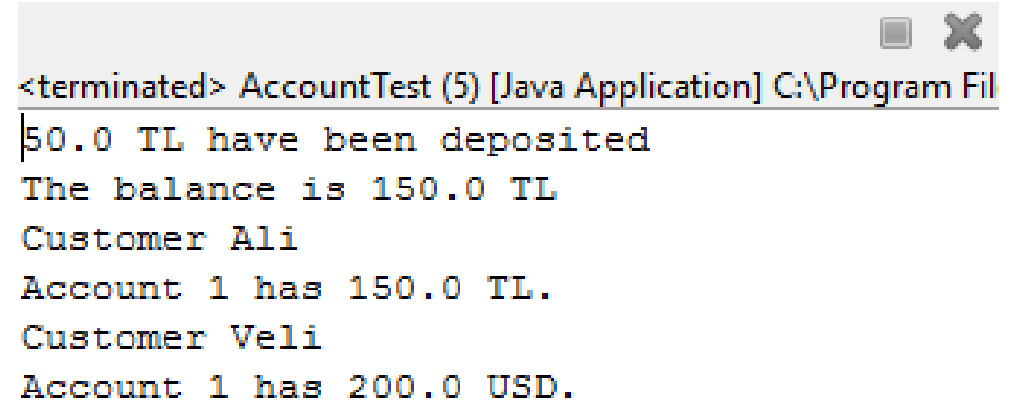
- Draw the Memory Model

Draw the Memory Model



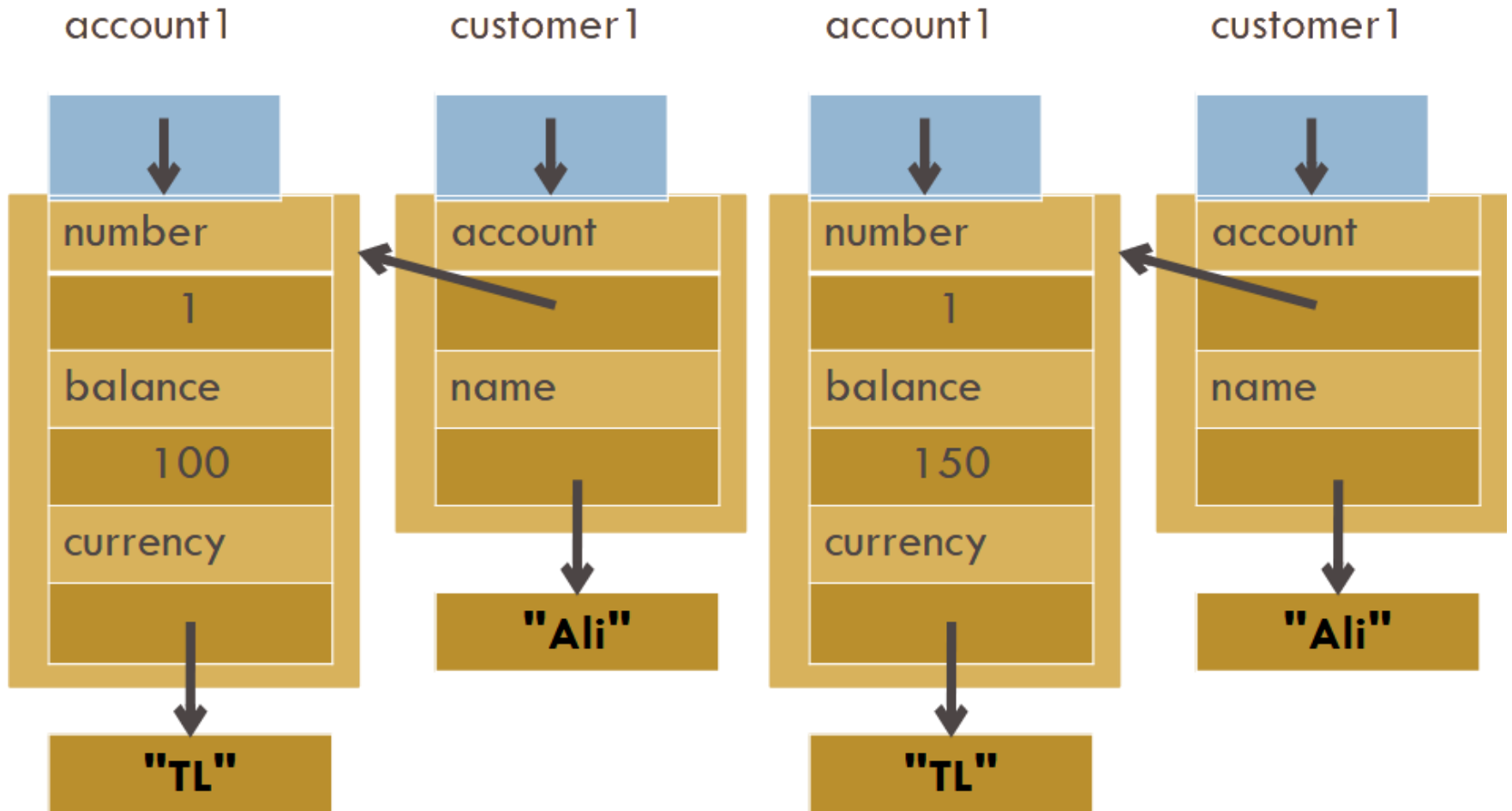
What is the output?

```
public static void main(String[] args) {  
  
    Account account1 = new Account(1, 100, "TL");  
    Customer customer1 = new Customer("Ali", account1);  
  
    Account account2 = new Account(1, 200, "USD");  
    Customer customer2 = new Customer("Veli", account2);  
  
    customer1.deposit(50);  
  
    customer1.report();  
    customer2.report();  
}
```



<terminated> AccountTest (5) [Java Application] C:\Program Fil
50.0 TL have been deposited
The balance is 150.0 TL
Customer Ali
Account 1 has 150.0 TL.
Customer Veli
Account 1 has 200.0 USD.

Before and After deposit



What is the output?

```
Account account1 = new Account(1, 100, "TL");
Customer customer1 = new Customer("Ali", account1);

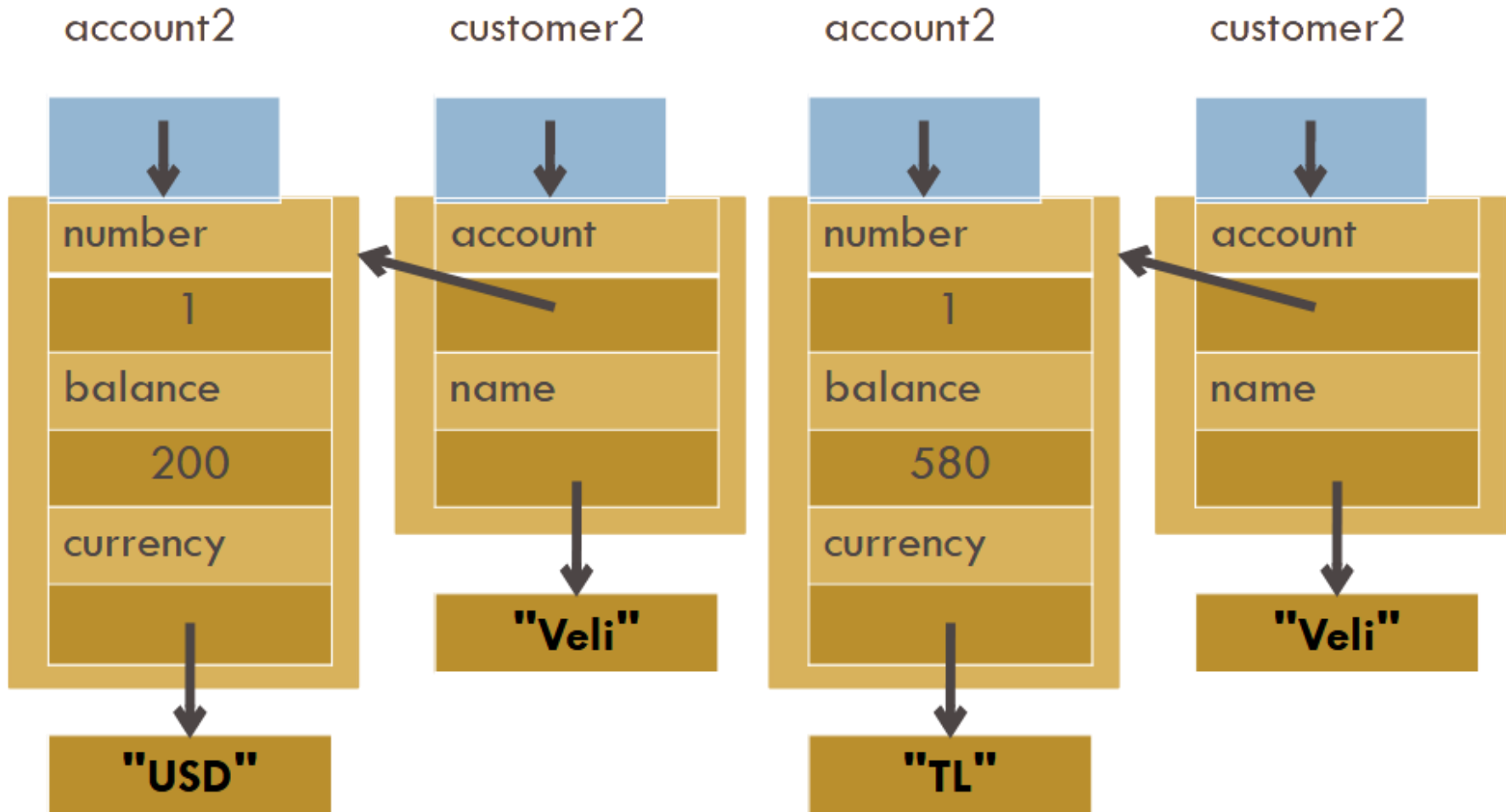
Account account2 = new Account(1, 200, "USD");
Customer customer2 = new Customer("Veli", account2);

customer1.deposit(50);
customer2.getAccount().setCurrency("TL");

customer1.report();
customer2.report();
```

```
<terminated> AccountTest (5) [Java Application]
|50.0 TL have been deposited
|The balance is 150.0 TL
|Customer Ali
|Account 1 has 150.0 TL.
|Customer Veli
|Account 1 has 580.0 TL.
```

Before and After setCurrency



What is the final memory model?

```
Account account1 = new Account(1, 100, "TL");
Customer customer1 = new Customer("Ali", account1);

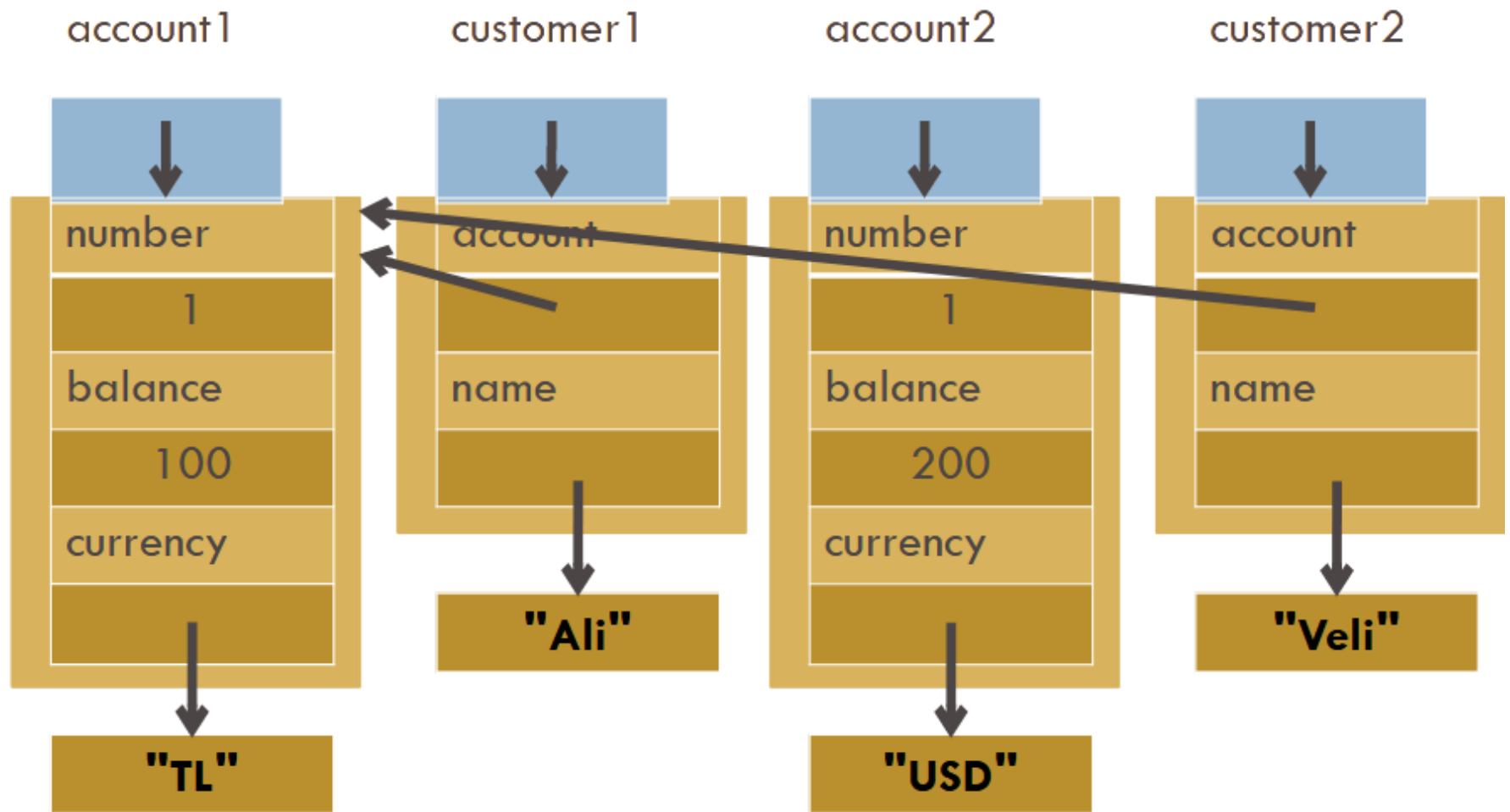
Account account2 = new Account(1, 200, "USD");
Customer customer2 = new Customer("Veli", account1);

customer1.deposit(50);
customer2.deposit(500);

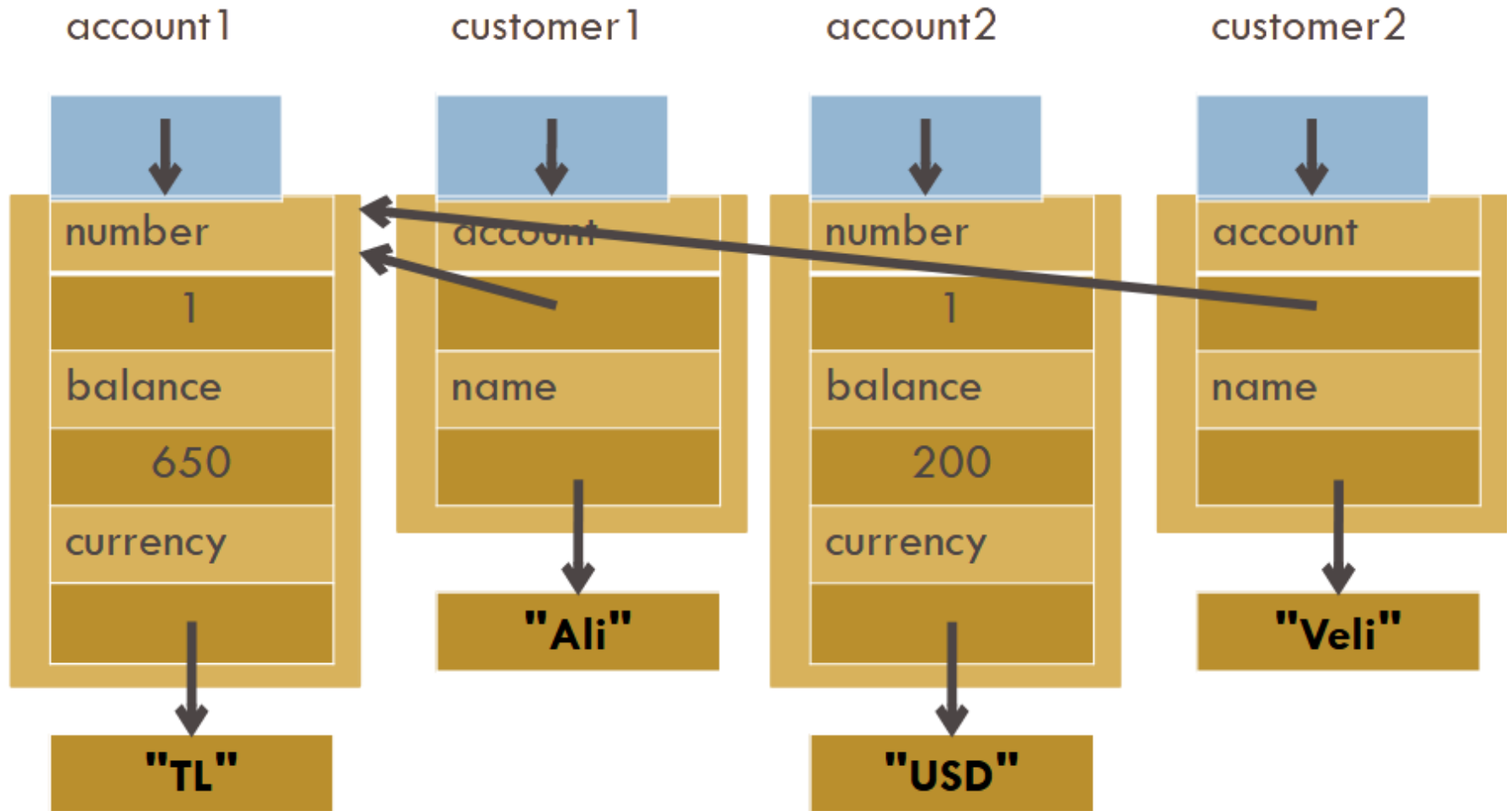
account1.withdraw(100);
account2.withdraw(200);

customer1.report();
customer2.report();
```

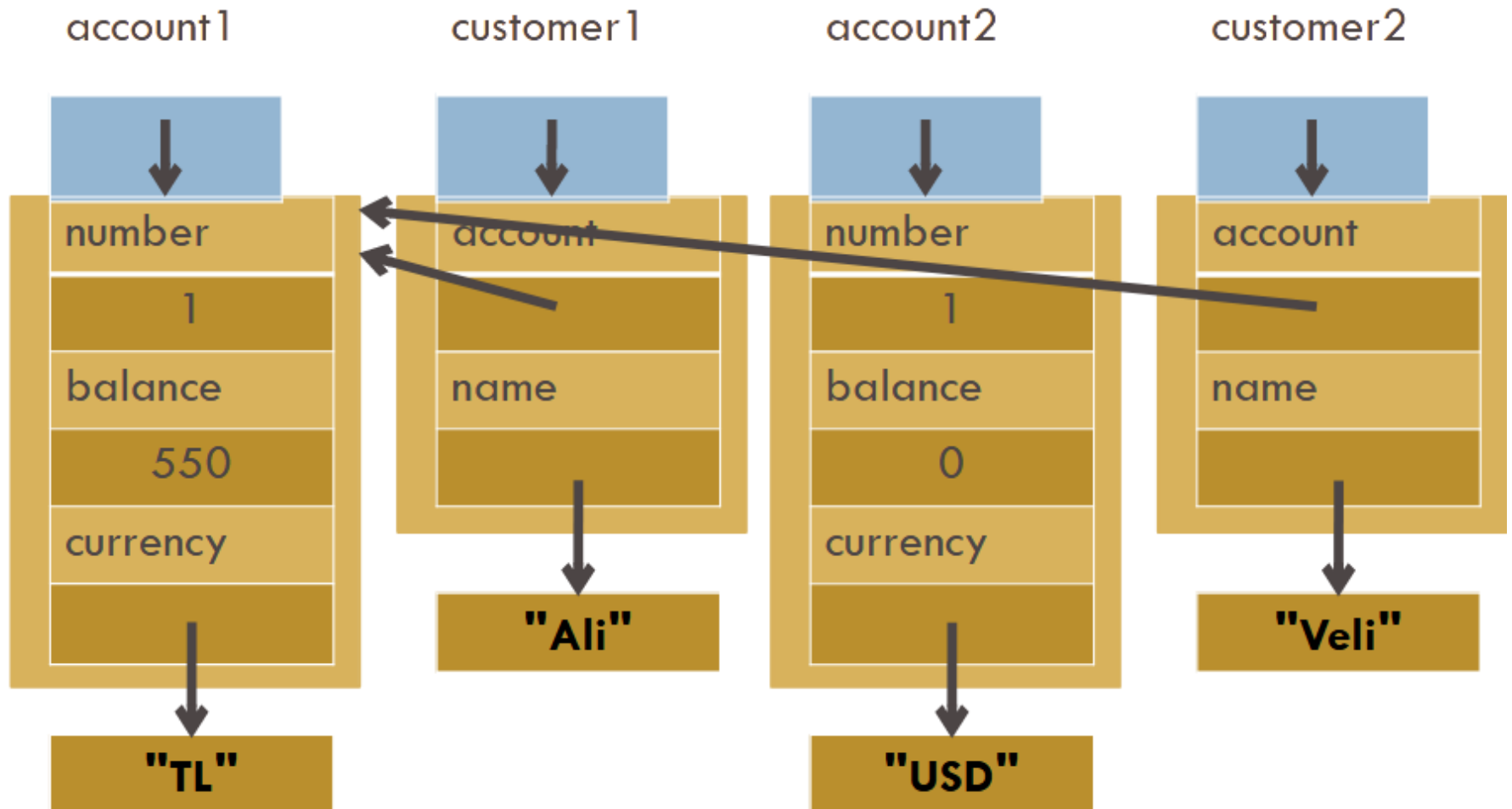
```
Account account1 = new Account(1, 100, "TL");  
Customer customer1 = new Customer("Ali", account1);  
  
Account account2 = new Account(1, 200, "USD");  
Customer customer2 = new Customer("Veli", account1);
```



```
customer1.deposit(50);  
customer2.deposit(500);
```



```
account1.withdraw(100);  
account2.withdraw(200);
```



Additional Classes

- We have customer and account, lets have a bank then.
- A bank has a name and customers.

- **Bank Class – Class Instances:**

- Only one name but multiple customers.

- name (String)

- customers (array)

```
public class Bank {  
    private String name;  
    private Customer[] customers;
```

- How many customers?
 - Need to know in advance, why?

Bank Class – Class Instances

- Lets say a bank can have at most 3 customers.
- Create an array of size 3
- But you don't have to use all 3 customers.
 - It can be less.
 - Therefore keep the number of customers value in a variable.

```
public class Bank {  
    private String name;  
    private Customer[] customers;  
    private int numCustomers;
```

Bank Class - Constructor

- Initially banks have no customers.
- What should be the constructor arguments?

```
public Bank(String n) {  
    name = n;  
    customers = new Customer[3];  
    numCustomers = 0;  
}
```

Bank Class – Adding Customers

- An addCustomer method to add customers.
- This method takes one customer as an argument.
- It updates the array and the numCustomers value.

```
public void addCustomer(Customer c) {  
    customers[numCustomers] = c;  
    numCustomers++;  
}
```

Bank Class – Other Functions

```
public String getName() {
    return name;
}

public void setName(String n) {
    name = n;
}

public void display() {
    System.out.println("---- "+name+" ----");
    for(int i=0; i < numCustomers; i++) {
        customers[i].report();
    }
    System.out.println("-----");
}
```

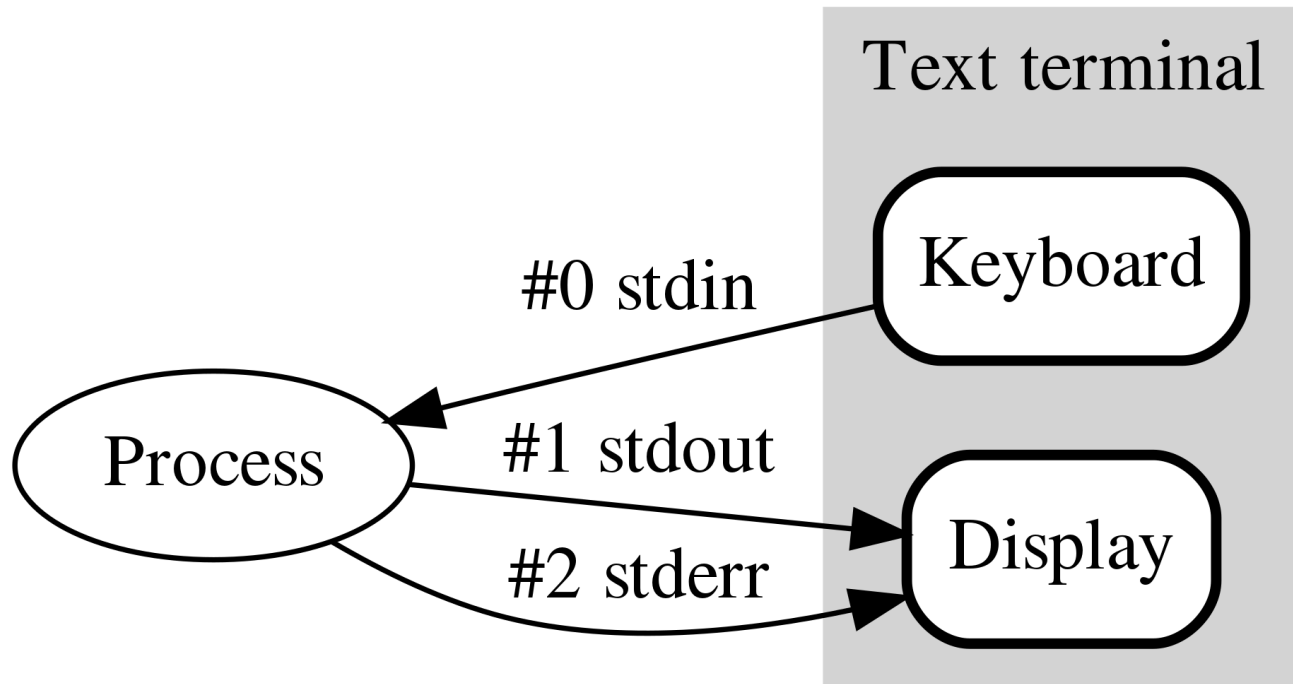
Bank Application

- Assume that we have an application which takes customer information in runtime from users.
- We need to use Scanner in order to read the input from the console.

```
import java.util.Scanner;

public class AccountTest {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
```

Standard Streams



Bank Application

- For each customer, what kind of information do we need?
 - Name
 - Account
 - Balance
 - Currency
 - Number?
 - The system can assign the next available account number to the account.
 - Need to keep a counter for account number.

Bank Application

```
public static void main(String[] args) {
    Scanner input = new Scanner(System.in);

    Bank bank = new Bank("TrustBank");
    int accountNo = 1;

    System.out.println("Welcome to " + bank.getName());
    while(true) {
        System.out.print("Enter customer name (empty to quit): ");
        String customerName = input.nextLine();
        if(customerName.equals(""))
            break;

        System.out.print("Enter currency: ");
        String curr = input.nextLine();

        System.out.print("Enter initial balance: ");
        double balance = Double.parseDouble(input.nextLine());

        bank.addCustomer(new Customer(customerName,
            new Account(accountNo, balance, curr)));
        accountNo++;
        bank.display();
    }
    System.out.println("Bye!");
}
```


Memory Model?

- What will be the memory model after user enters
 - “Ali” for customer name
 - “TL” for account’s currency
 - 100 for initial balance

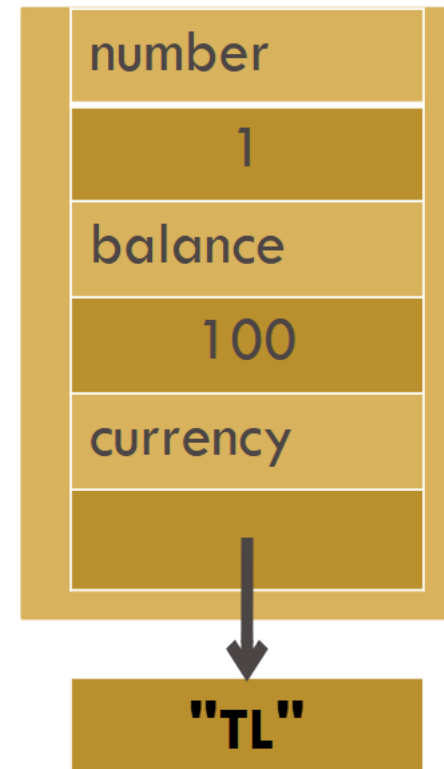
- How many objects are we going to create?

```
bank.addCustomer(new Customer(customerName,  
                             new Account(accountNo, balance, curr)));
```

Memory Model

```
bank.addCustomer(new Customer(customerName,  
    new Account(accountNo, balance, curr)));
```

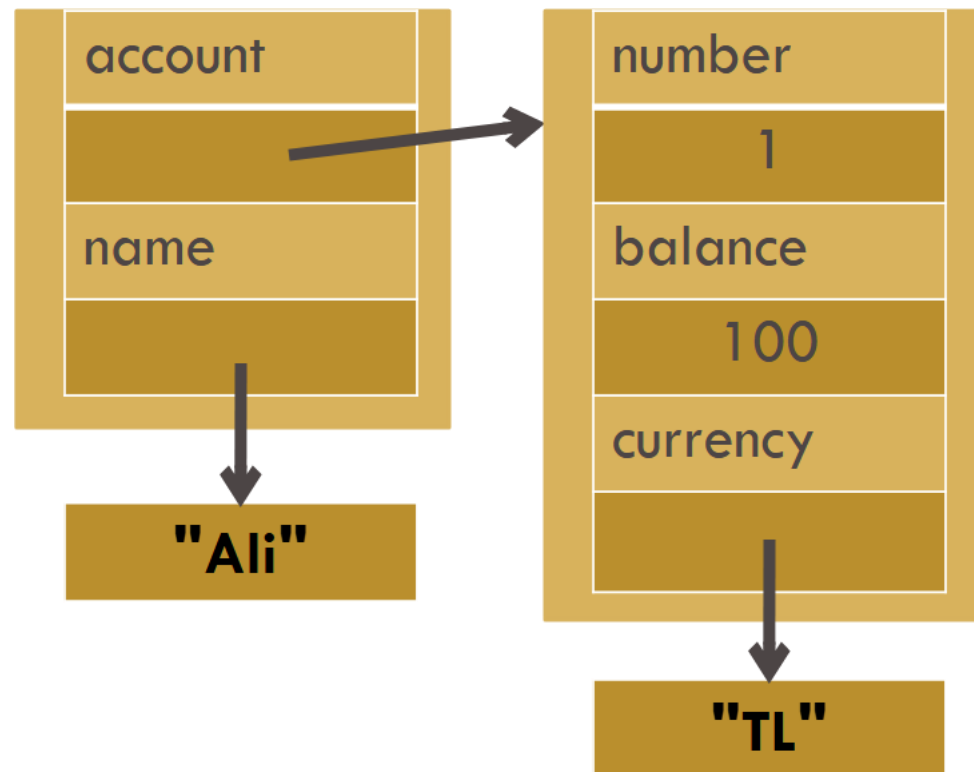
- The path is from inside out.
- User entered
 - "TL" for account's currency
 - 100 for initial balance
- Return its reference to
- Customer constructor.



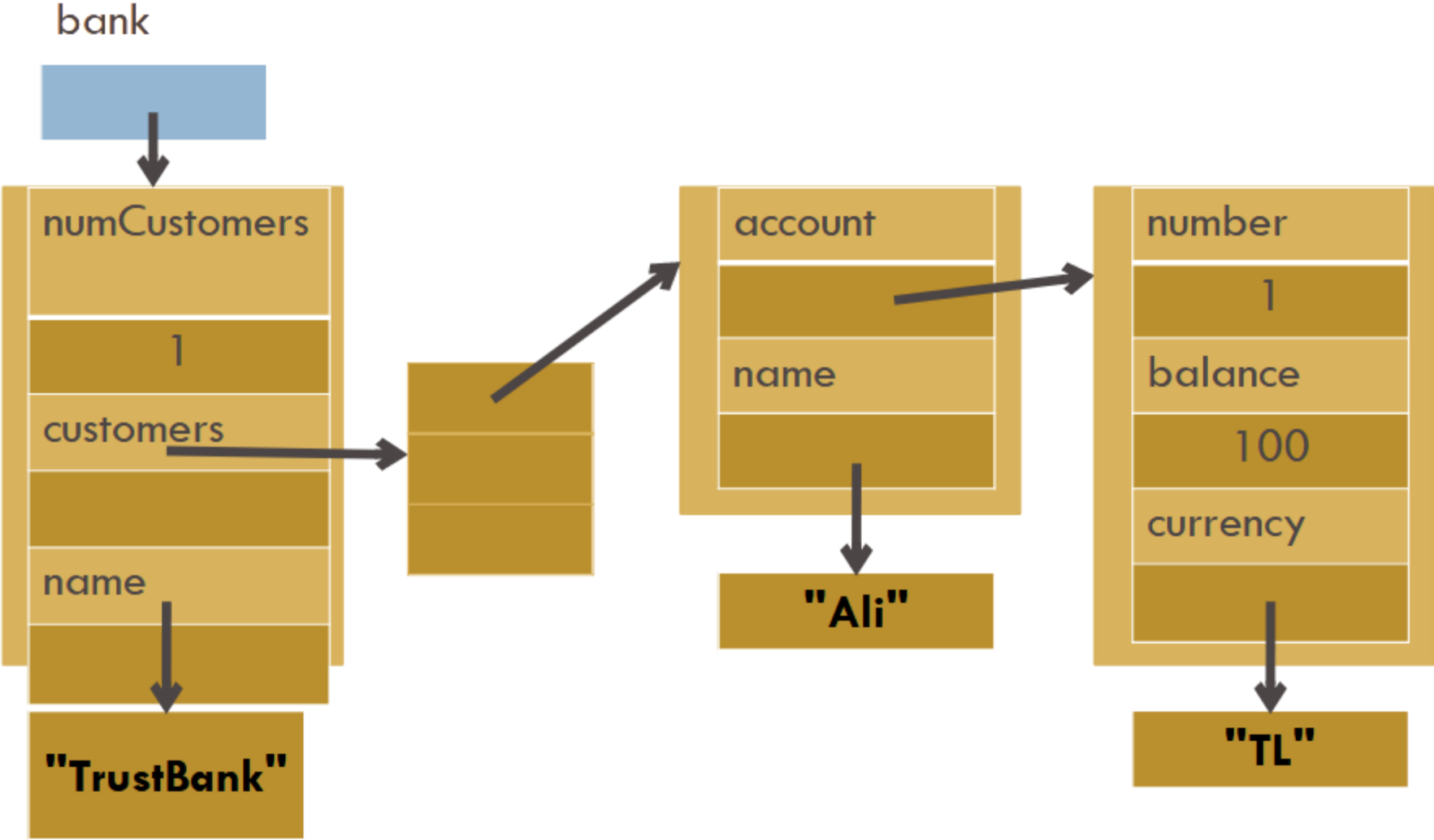
Memory Model

```
bank.addCustomer(new Customer(customerName,  
    new Account(accountNo, balance, curr)));
```

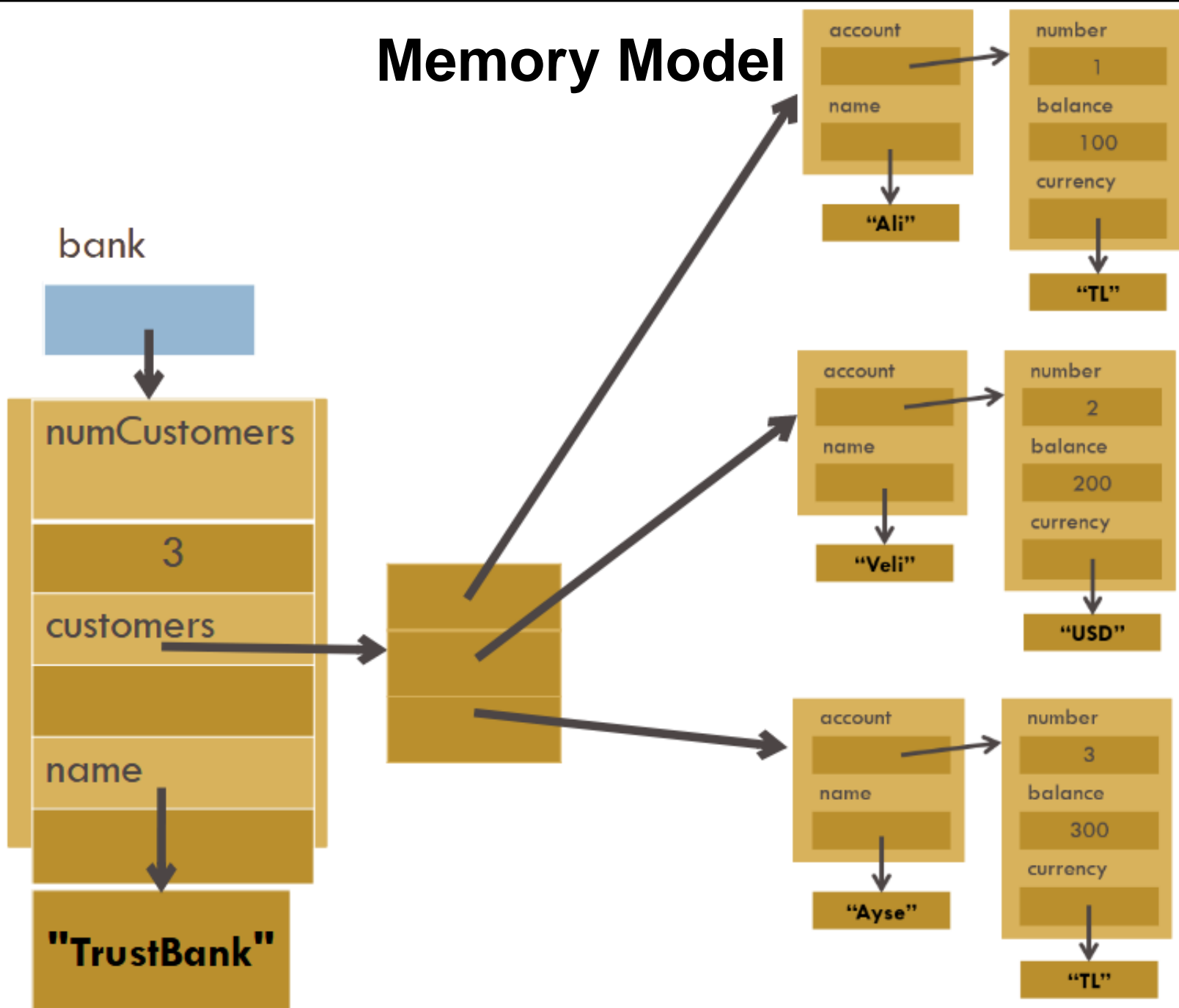
- Save customer's address at bank's customer array



Memory Model



Memory Model



Arrays

- Arrays are fixed length.
- We need a data structure that can be resized.
 - ArrayList

- ArrayList
 - Dynamic in size
 - See ArrayList slides ...

Bank Class

- with ArrayList
- We don't need numCustomers anymore.

```
private String name;
private ArrayList<Customer> customers;

public Bank(String n) {
    name = n;
    customers = new ArrayList<Customer>();
}

public String getName() {
    return name;
}
public void setName(String n) {
    name = n;
}

public void addCustomer(Customer customer) {
    customers.add(customer);
}

public void display() {
    System.out.println("---- "+name+" ----");
    for(Customer customer: customers) {
        customer.report();
    }
    System.out.println("-----");
}
```

Any Questions?