The state diagram of an object

- The internal state of an object is represented by its "properties" (fields).
- Fields are usually protected.

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

There are 3 types of services (methods, functions, procedures):

- **Queries (accessors)**
  - Return the state of the object without changing it
  - E.g.: retrieve balance

- **Commands (transformers, mutators)**
  - Operate on the internal state
  - E.g.: transfer money

- **Constructors**
  - Create a new object
  - E.g.: create a new account

There is a need for access to data through methods.

- Why?
  - Not all fields are visible (private).
  - Creating automatic services for each field contradicts the principle of information hiding.

```java
public double getBalance() {
    ...
    return balance;
}
```

In conclusion, the methods and fields are structured as follows:

```
BankAccount:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getBalance</td>
<td>Return the balance</td>
</tr>
<tr>
<td>getAccountNumber</td>
<td>Return the account number</td>
</tr>
<tr>
<td>getOwner</td>
<td>Return the owner</td>
</tr>
</tbody>
</table>
```

Getter/Setter

- The method `get` is used to access private fields.
- A field declared as private should have both `get` and `set` methods.

```java
public double getBalance() {
    return balance;
}
```

The `BankAccount` class:

```java
public class BankAccount {
    public double getBalance() {
        return balance;
    }
    public long getAccountNumber() {
        ...
    }
    public Customer getOwner() {
        ...
    }
    ...
    private double balance;
    private long accountNumber;
    private Customer owner;
}
```
פקודת ה-"להפקיד"

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

פקודת ה-"למשוך"

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

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);
}

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

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;
}

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

```java
/**
 * 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;
}
```

**Customer**

```java
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;
}
```
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, 2984);
        account1.deposit(100);
        account1.transferTo(100, account2);
        System.out.println("account1 has " + account1.getBalance());
        System.out.println("account2 has " + account2.getBalance());
    }
}

public class Bank {
    public static void main(String[] args) {
        Customer customer1 = new Customer("Rita Stein", "025285244");
        Customer customer2 = new Customer("Avi Cohen", "024847638");
        BankAccount account1 = new BankAccount(customer1, 2984);
        BankAccount account2 = new BankAccount(customer2, 1234);
        account1.deposit(100);
        account1.transferTo(100, account2);
        System.out.println("account1 has " + account1.getBalance());
        System.out.println("account2 has " + account2.getBalance());
    }
}
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("account 1 has " + account1.getBalance());
        System.out.println("account 2 has " + account2.getBalance());
    }
}