bank account

- state
  - has a number
  - has an owner
  - has a balance

- operations (behavior)
  - creation of an account with given number and owner (initial balance will be 0)
  - get number, owner, or balance
  - deposit an amount in the account
  - withdraw an amount from the account
  - transfer an amount to another account
public class Account {
  // Fields
  private int number;  // account number
  private String owner;  // account owner
  private int balance;  // account balance

  // instance variables
  // each instance object of the class has its own instance variables
  // determine the state of the object
  // in general declared private (or protected) for encapsulation
```java
public Account(int nr, String o) {
    number = nr;
    owner = o;
    balance = 0;
}
```

- constructor with parameters for initialization
- called when a new instance object of class Account is created
  ```java
  Account acc;
  acc = new Account(1, "Jan");
  ```
- disables standard default constructor Account() which would initialize `number` and `balance` to 0 and `owner` to `null`
instance methods (functions, queries)

inspection of the state of the instance object the method is called on (e.g. acc.getBalance() returns the balance of the instance object acc refers to)
/ Commands
---

```java
public void deposit(int amount) {
    balance = balance + amount;
}
```

```java
public void withdraw(int amount) {
    balance = balance - amount;
}
```

```java
public void transfer(Account acc, int amount) {
    withdraw(amount);
    acc.deposit(amount);
}
```

- **instance methods (operations, commands)**
- change the state of the instance object the method is called on (and possibly the state of other instance objects this object has references to)
- have access to instance variables and methods of the object
- the object itself can be referenced by using `this`
- within the bodies of the methods the following statements are equivalent

```java
withdraw(100); this.withdraw(100);
```
public class BankApp {
    public static void main(String[] args) {
        Bank bank = new Bank();
        bank.run();
    }
}

- class containing static method \textit{main}
- no instance object of this class will be created
- \textit{main} is executed within a unique object representing the class
```java
public class Bank {
    private Account acc1;
    private Account acc2;

    public Bank() {
        acc1 = new Account(1,"Jan");
        acc2 = new Account(2,"Piet");
    }

    public void run() {
        acc1.deposit(200);
        acc1.transfer(acc2, 100);
    }
}
```
public static void main(String[] args) {
    Bank bank = new Bank();
    bank.run();
}

public Bank() {
    acc1 = new Account("Jan");
    acc2 = new Account("Piet");
}
```java
public static void main(String[] args) {
    Bank bank = new Bank();
    bank.run();
}
```
public static void main(String[] args) {
    Bank bank = new Bank();
    bank.run();
}

void run()
    acc1.deposit(200);
    acc1.transfer(acc2, 100);

void transfer(Account acc, int amount) {
    acc.deposit(amount);
    acc.withdraw(amount);
}

void withdraw(int amount) {
    balance = balance - amount;
}
```java
public class BankApp {
    public static void main(String[] args) {
        Bank bank = new Bank();
        bank.run();
    }
}
```
void main(String[] args) {
    StackOfInt_List stack = new StackOfInt_List();
    stack.push(1);
    stack.push(2);
    stack.push(3);
    stack.pop();
    stack.push(4);
    stack.push(5);
    }

void push(int x) {
    ifTop = new IntNode(x, fTop);
    fCount = fCount + 1;
    }

void pop() {
    fTop = fTop.next;
    fCount = fCount - 1;
    }

int fTop;
int fCount;

boolean $assertionsDisabled true

class IntNode {
    int item;
    IntNode next;

    IntNode(int item, IntNode next) {
        this.item = item;
        this.next = next;
    }
}

class StackOfInt_List {
    IntNode fTop;

    public int size() {
        return fCount;
    }

    public IntNode top() {
        return fTop;
    }

    public boolean empty() {
        return fCount == 0;
    }

    public void push(int x) {
        fTop = new IntNode(x, fTop);
        fCount = fCount + 1;
    }

    public int pop() {
        if (fCount == 0) {
            throw new EmptyStackException();
        }
        int x = fTop.item;
        fTop = fTop.next;
        fCount = fCount - 1;
        return x;
    }

    public int peek() {
        if (fCount == 0) {
            throw new EmptyStackException();
        }
        return fTop.item;
    }

    public int popAt(int index) {
        int x = fTop.item;
        IntNode prev = fTop;
        for (int i = 0; i < index - 1; i++) {
            prev = prev.next;
        }
        IntNode next = prev.next;
        prev.next = next;
a savings account is an account that also satisfies

- gives interest at some rate
- balance may not be negative (being in the red not allowed)

a savings account has

- extended state: interest rate is added
- extended behavior: setting and getting the interest rate, adding interest to the balance
- changed (overridden) behavior: withdrawing an amount only allowed if the balance does not become negative (overdrawing not allowed)
Inheritance

```java
public class SavingsAccount extends Account {
    // Additional field
    protected int rate;
    // Constructor
    public SavingsAccount(int nr, String o, int rate) {
        super(nr, o); // call constructor of super class Account
        this.rate = rate;
    }
}
```

- (sub)class `SavingsAccount` is derived from (super)class `Account` inheriting all members (fields and methods) of its superclass; it is indicated by `extends Account` at the end of its class header.
- class `SavingsAccount` introduces an additional instance field `rate` extending the state represented by instance objects.
- class `SavingsAccount` does not inherit constructors from its superclass; constructors of the superclass can be invoked using `super` with an appropriate argument list.
// Additional query
public int getRate() {
    return rate;
}

// Additional commands
public void setRate(int rate) {
    this.rate = rate;
}

- extension of behavior in the form of an additional getter and setter
```java
public void addInterest() {
    deposit((int) (getBalance() * (rate / 100.0)));
}
```

- extension of behavior in the form of an additional command
- field `balance` is declared `private` in superclass `Account` so it is not directly accessible in subclass `SavingsAccount`; the public methods `getBalance` and `deposit` of superclass `Account` are used to inspect `balance` and to change it
- if the declaration of field `balance` in superclass `Account` is changed into
  ```java
  protected int balance;
  ```
  it becomes directly accessible in subclass `SavingsAccount` and the body of `addInterest` could be replaced by
  ```java
  balance = (int) (balance * (1 + rate / 100.0));
  ```
// Changed command
@Override
public void withdraw(int amount) {
    if (getBalance() - amount > 0) {
        super.withdraw(amount);
    }
}

▶ change of behavior by redefinition of a command of the superclass
   (overriding)
▶ behavior of the overriding method in the subclass should in
   accordance with the behavior of the overridden method in the
   superclass
▶ method withdraw as defined in Account can be invoked in subclass
   SavingsAccount by calling super.withdraw
▶ if o is a reference to an object of type Account (object is an
   instance of class Account) then o.withdraw(100) results in invoking
   the method withdraw on object o as defined in class Account
▶ likewise if o is a reference to an object of type SavingsAccount