```java
/*
 * Testtreiberklasse, welche das Bankensystem testet.
 * Es werden neue Bankkonten angelegt und Geld deponiert,
 * abgehoben und transferiert.
 */
public class Main {

	/*
	 * Main Methode fuehrt bestimmte Testfaelle aus.
	 */
	public static void main(String[] args) {
		Bank b = new Bank();
		int c1, s2, s3, c4, c5;
		// Verschiedene Konten erzeugen
		c1 = b.getNewCurrentAccount();
		s2 = b.getNewSavingsAccount();
		s3 = b.getNewSavingsAccount();
		c4 = b.getNewCurrentAccount();
		c5 = b.getNewCurrentAccount();

		// Geld auf den Konten einzahlen
		b.depositMoney(c1, 200f);
		b.depositMoney(s2, 300f);
		b.depositMoney(s3, 400f);
		b.withdrawMoney(c4, 500f);
		b.depositMoney(c5, 1200);

		// Zeigt den insgesamt einbezahlten Betrag auf der Bank.
		// Muss 1600 GE sein.
		System.out.println("Current Bank balance: " + b.getBankBalance());

		// transferiert von Konto c5 nach Konto c4 600 Geldeinheiten
		// Danach:
		// c5 --> 600
		// c4 --> 100
		b.transferAmount(c5, c4, 600);

		// Zeigt alle Konten und deren Saldi an.
		// Muss zeigen:
		//
		b.summarizeBank();

		// Sollte zeigen:
		//Account 5 has balance of 600.0
		//Account 4 has balance of 100.0
		//Account 3 has balance of 400.0
		//Account 2 has balance of 300.0
		//Account 1 has balance of 200.0
		//Total amount deposited in bank:1600.0


		// Geld abheben von Konto c5 (500 GE)
		b.withdrawMoney(c5, 700);

		// Nun sind -100 GE auf Konto c5

		try {
			// Geld abheben von s2 funktioniert --> danach 75 GE auf Konto
			b.withdrawMoney(s2, 225f);
			// Geld abheben von s2 --> danach 0 GE auf dem Konto
			b.withdrawMoney(s2, 75);
			// Geld abheben von s3 --> geht nicht, da Sparkonto und nur 440 GE
			// auf Konto --> Exception
			b.withdrawMoney(s3, 475f); // gibt eine Exception
		} catch (Exception ex) {
			// Exception info
			ex.printStackTrace();
		}

		//java.lang.RuntimeException: Balance of 400.0 to low for withdrawing amount of 475.0
		//at SavingsAccount.withdraw(SavingsAccount.java:46)
		//at Bank.withdrawMoney(Bank.java:140)
		//at Main.main(Main.java:65)

		// Aktueller Bankinhalt anzeigen.
		b.summarizeBank();

		//Sollte zeigen:
		//Account 5 has balance of -100.0
		//Account 4 has balance of 100.0
		//Account 3 has balance of 400.0
		//Account 2 has balance of 0.0
		//Account 1 has balance of 200.0
		//Total amount deposited in bank:600.0
		b.removeAccount(c5);

		System.out.println("Account " + c5 + " existing: " + b.accountExists(c5));

		//Account 5 existing: false
		//Account 4 has balance of 100.0
```

```
                //Account 3 has balance of 400.0
                //Account 2 has balance of 0.0
                //Account 1 has balance of 200.0
                //Total amount deposited in bank:700.0
                b.summarizeBank();

                int c6 = b.getNewCurrentAccount();

                b.withdrawMoney(c6, 100);

                b.summarizeBank();
                //Account 6 has balance of -100.0
                //Account 4 has balance of 100.0
                //Account 3 has balance of 400.0
                //Account 2 has balance of 0.0
                //Account 1 has balance of 200.0
                //Total amount deposited in bank:600.0

                // Zinsen berechnen für alle Bankkonten
                b.accumulateInterests();

                b.summarizeBank();
                //Account 6 has balance of -109.0
                //Account 4 has balance of 100.025
                //Account 3 has balance of 405.0
                //Account 2 has balance of 0.0
                //Account 1 has balance of 200.05
                //Total amount deposited in bank:596.075
                }
}



import java.util.Collections;
import java.util.Enumeration;
import java.util.Observable;
import java.util.Observer;
import java.util.Hashtable;

/**
 * Stellt eine virtuelle Bank dar.
 * Mit dieser Bank koennen Bankkonten erstellt werden,
 * Betraege eingezahlt, transferiert und abgehoben werden.
 * Es gibt zwei Arten von Bankkonten. Dies sind Sparkonten
 * und Kontokorrentkonten. Beide unterscheiden sich im
 * Habenzins und im Betrag den sie ueberzogen werden koennen.
 * Die Bank stellt eine Uebersicht ueber alle Konten und
 * den Gesamtbetrag, der in der Bank eingelagert ist, zur
 * Verfuegung.
 */
public class Bank extends Hashtable implements Observer {
        /**
         * Enthaelt zu jeder Zeit die Summe aller Bankontosalden.
         * Dieser Gesamtbetrag wird bei Veraenderungen auf einem
         * Konto nachgefuehrt.
         */
        private float currentAccountBalance;

        /**
         * Enthaelt die aktiven Bankkonten.
         */
        private Hashtable accounts;

        /**
         * Enthaelt die naechste freie Bankkontonummer.
         */
        private int nextAccountNo;

        /**
         * Konstruktor
         * PRE:  true
         * POST: true
         */
        public Bank() {
                currentAccountBalance = 0.0f;
                nextAccountNo = 1;
                accounts = new Hashtable();
        }

        /**
         * Fuegt ein neues Bankkonto hinzu.
         * Das Konto darf noch nicht existieren.
         */
        void addBankAccount(Account a) {
                currentAccountBalance += a.balance;
                a.addObserver(this);
                accounts.put(new Integer(a.getAccountNo()), a);
        }

        /**
```

```
 * Gibt die Summe der Saldi aller registrierten Konti zurueck.
 *
 * PRE:  true
 * POST: result >= 0
 */
public float getBankBalance() {
        return currentAccountBalance;
}

/**
 * Entfernt ein Bankkonto. Der Saldo der Bank wird dabei nachgefuehrt.
 */
public void removeAccount(Account a) {
        accounts.remove(new Integer(a.getAccountNo()));
        currentAccountBalance -= a.balance;
}

/**
 * Entfernt ein Konto, welches durch die Kontonummer accountNo
 * identifiziert wird.
 */
public void removeAccount(int accountNo) {
        removeAccount(getAccount(accountNo));
}

/**
 * Transferiert den Betrag amount von Konto srcAccount zu Konto
 * destAccount.
 */
public void transferAmount(int srcAccount, int destAccount, float amount) {
        Account a, b;
        a = getAccount(srcAccount);
        b = getAccount(destAccount);
        a.withdraw(amount);
        b.deposit(amount);
}

/**
 * Zahlt einen Betrag amount auf das Konto mit der Nummer accountNo ein.
 */
public void depositMoney(int accountNo, float amount) {
        getAccount(accountNo).deposit(amount);
}

/**
 * Hebt von einem bestimmten Bankkonto mit der Kontonummer accountNo einen
 * Betrag amount ab.
 */
public void withdrawMoney(int accountNo, float amount) {
        getAccount(accountNo).withdraw(amount);
}

/**
 * Gibt ein bestimmtes Bankkonto-Objekt mit der Nummer accountNo zurueck.
 * PRE: accountExists(accountNo)
 * POST: result != null
 */
Account getAccount(int accountNo) {
        return (Account) accounts.get(new Integer(accountNo));
}

/**
 * Gibt den Saldo eines bestimmten Kontos aus.
 * PRE:  accountExists(accountNo)
 * POST: result == getAccount(accountNo).balance
 */
public float getBalance(int accountNo) {
        return ((Account) accounts.get(new Integer(accountNo))).balance;
}

/**
 * Gibt true zurueck, falls ein Bankkonto mit der gegebenen Nummer accountNo
 * existiert. Sonst false.
 *
 * PRE: true
 * POST: if (exists x : Account in
 *           accounts | x.getAccountNo() == accountNo)
 *       then true
 *       else false
 */
public boolean accountExists(int accountNo) {
        return (accounts.get(new Integer(accountNo)) != null);
}

/**
 * Gibt true zurueck, falls das Bankkonto bereits der Bank zugeordnet ist, ansonsten
 * false.
 * PRE: a != null
 * POST: if (exists x : Account in accounts  | x == a) then true else false
 */
```

```
boolean accountExists(Account a) {
        return (accountExists(a.getAccountNo()));
}

/**
 * Rueckrufschnittstelle eines Kontobeobachters. Konten, deren Saldo geaendert werden,
 * melden den alten Saldo ueber diese Methode. Anschliessend wird mittels dem
 * neuen und dem alten Saldo der neue Gesamtsaldo der Bank berechnet.
 *
 * PRE:  o != null and arg != null and
 *       o instanceof Account and arg instanceof Float
 * POST: currentAccountBalance =
 *         currentAccountBalance@PRE -
 *         ((Float)arg).floatValue() +
 *         ((Account)o).balance
 */
public void update(Observable o, Object arg) {
        Float f = (Float) arg;
        currentAccountBalance -= f.floatValue();
        currentAccountBalance += ((Account) o).balance;
}

/**
 * Akkumuliert den aufgelaufenen Zinsbetrag auf allen Konten, ohne
 * Berueksichtigung der Kontenbewegungen.
 * PRE:  true
 * POST: true
 */
public void accumulateInterests() {
        Enumeration e = Collections.enumeration(accounts.values());
        while (e.hasMoreElements()) {
                ((Account) e.nextElement()).accumulateInterests();
        }
}

/**
 * Gibt eine Uebersicht ueber alle Konten aus.
 *
 * PRE: true
 * POST: true
 */
public void summarizeBank() {
        Enumeration e = Collections.enumeration(accounts.values());
        while (e.hasMoreElements()) {
                System.out.println(e.nextElement());
        }
        System.out.println("Total amount deposited in bank:" +
                        this.currentAccountBalance);
        System.out.println("*****");
        System.out.println();
}

/**
 * Erzeugt ein neues Kontokorrentkonto und fuegt dieses der Menge von Konten
 * hinzu. Die Bankkontonummer wird dem Aufrufer zurueckgegeben.
 *
 * PRE: true
 * POST: result == nextAccountNo@PRE + 1 and accountExists(result)
 */
public int getNewSavingsAccount() {
        SavingsAccount s = newSavingsAccount();
        addBankAccount(s);
        return s.getAccountNo();
}

/**
 * Erzeugt ein neues Sparkonto und fuegt dieses der Menge von Konten hinzu.
 * Die Bankkontonummer wird dem Aufrufer zurueckgegeben.
 *
 * PRE: true
 * POST: result == nextAccountNo@PRE + 1 and
 *       accountExists(result)
 */
public int getNewCurrentAccount() {
        CurrentAccount c = newCurrentAccount();
        addBankAccount(c);
        return c.getAccountNo();
}

/**
 * Hilfsmethode zum Erzeugen von neuen Sparkonten.
 */
private SavingsAccount newSavingsAccount() {
        SavingsAccount s = new SavingsAccount(nextAccountNo);

        nextAccountNo++;
        return s;
}

/**
```

```
         * Hilfsmethode zum Erzeugen von Bankkontokorrenten.
         * PRE:  true
         * POST: result != null
         */
        private CurrentAccount newCurrentAccount() {
                CurrentAccount c = new CurrentAccount(nextAccountNo);

                nextAccountNo++;
                return c;
        }
}



import java.util.Observable;

/**
 * Die Klasse Bankkonto stellt die Funktionalitaet eines vereinfachten
 * Bankkontos bereit. Diese beinhaltet das Einzahlen (deposit), das Auszahlen
 * (withdraw) sowie die Berechnung der Zinsen (ohne Berueecksichtigung der
 * Kontobewegungen). Beobachter koennen sich beim Konto anmelden, um Rueckmeldungen
 * ueber Saldoaenderungen zu bekommen.
 */
abstract class Account extends Observable {
        /**
         * Aktueller Saldo des Bankkontos
         */
        public float balance;

        /**
         * Kontonummer des Bankkontos
         */
        private int accountNo;

        /**
         * Konstruktor
         * PRE:  true
         * POST: true
         */
        public Account(int accountNo) {
                this.accountNo = accountNo;
        }

        /**
         * Diese Methode berechnet den aufgelaufenen Zins und fuegt diesen dem
         * aktuellen Saldo hinzu. Die Berechnung berücksichtigt die
         * Kontenbewegungen nicht. Anschliessend werden alle Beobachter dieses Konto
         * ueber die Veraenderung des Saldos informiert.
         *
         * PRE   true
         * POST  balance == balance@PRE + calculateInterests()@PRE
         */
        public void accumulateInterests() {
                float oldBalance = balance;
                balance += calculateInterests();
                // Beobachter benachrichtigen
                setChanged();
                // sende den Beobachtern den alten Betrag des Kontos
                notifyObservers(new Float(oldBalance));
        }

        /**
         * Zahlt auf dem Konto einen gewissen Betrag ein.
         * Allfaellige Beobachter werden ueber die Saldoaenderung
         * informiert. Der Betrag darf nicht kleiner sein als null.
         *
         * PRE:  amount >= 0
         * POST: balance == balance@PRE + amount
         */
        public void deposit(float amount) {
                float oldBalance = balance;
                balance += amount;
                // Beobachter benachrichtigen
                setChanged();
                notifyObservers(new Float(oldBalance));
        }

        /**
         * Gibt die Kontonummer zurueck.
         * PRE: true
         * POST: result == accountNo
         */
        public int getAccountNo() {
                return accountNo;
        }

        /**
         * Gibt den den Hashcode dieses Objektes zurueck. Dieser
         * ergibt sich aus der (eindeutigen) Kontonummer.
         * PRE: true
```

```
         * POST: result == accountNo
         */
        public int hashCode() {
                return accountNo;
        }

        /**
         * Gibt die Informationen (Kontonummer und Saldo) ueber das Konto als String
         * zurueck.
         * PRE: true
         * POST: result != null and !result.equals("")
         */
        public String toString() {
                return "Account " + accountNo + " has balance of " + balance;
        }

        /**
         * Der angegebene Betrag amount wird vom Saldo abgezogen. Der Betrag darf nicht
         * kleiner sein als null. Allfaellige Beobachter des Kontos werden ueber
         * die Saldoaenderung informiert.
         *
         * PRE:  amount >= 0
         * POST: balance == balance@PRE - amount
         */
        public abstract void withdraw(float amount);

        /**
         * Berechnet die aufgelaufenen Zins ohne die Beruecksichtigung der
         * Kontenbewegungen.
         * PRE: true
         * POST: true
         */
        public abstract float calculateInterests();
}



/**
 * Der Bankkontokorrent kann ueberzogen werden, d.h. der Saldo kann kleiner als 0 sein
 * (balance < 0). Ein negativer Banksaldo ergibt eine Zinsbelastung. Der
 * Sollzins bei einem negativen Saldo ist wesentlich hoeher als der Habenzins bei
 * einem positivem Saldo.
 */
class CurrentAccount extends Account {

        /**
         * Habenzinssatz
         */
        protected static float POSITIVE_INTEREST_RATE = 0.025f;

        /**
         * Sollzinssatz
         */
        protected static float NEGATIVE_INTEREST_RATE = 9;

        /**
         * Konstruktor, initialisiert das Konto.
         */
        public CurrentAccount(int accountNo) {
                super(accountNo);
        }

        /**
         * Hebt Geld vom Konto ab und benachrichtigt allfaellige Beobachter von der
         * Aenderung des Saldos.
         * PRE: true
         * POST: balance == balance@PRE - amount
         */
        public void withdraw(float amount) {
                float oldBalance = balance;
                balance -= amount;

                // Beobachter benachrichtigen
                this.setChanged();
                notifyObservers(new Float(oldBalance));
        }

        /**
         * Berechnet den aufgelaufenen Zins seit der letzten Abrechnung, ohne
         * Beruecksichtiung der Kontenbewegungen.
         * PRE: true
         * POST: if (balance < 0)
         *         then result == NEGATIVE_INTEREST_RATE * balance
         *         else result == POSITIVE_INTEREST_RATE * balance
         */
        public float calculateInterests() {
                float retVal = ((balance < 0) ? NEGATIVE_INTEREST_RATE
                                : POSITIVE_INTEREST_RATE)
                                * balance / 100.0f;
                return retVal;
```

```
        }
}



/**
 * Ein Sparkonto muss immer einen positiven Saldo aufweisen
 * (balance >= 0). Ein Sparkonto hat dafuer einen groesseren Habenzins
 * als ein Bankkontokorrent.
 *
 * INV: balance >= 0
 */
class SavingsAccount extends Account {
        /**
         * Habenzinssatz
         */
        protected static float INTEREST_RATE = 1.25f;

        /**
         * Konstruktor
         * PRE: true
         * POST: true
         */
        public SavingsAccount(int accountNo) {
                super(accountNo);
        }

        /**
         * Berechnet den Habenzins seit der letzten Zinsverbuchung.
         * PRE: true
         * POST: result == balance * INTEREST_RATE / 100.0f
         */
        public float calculateInterests() {
                return balance * INTEREST_RATE / 100.0f;
        }

        /**
         * Hebt Geld vom Konto ab. Dabei darf der Saldo nicht kleiner werden als 0.
         * Ist der abzuhebende Betrag zu gross, so
         * wird eine RuntimeException geworfen.
         * PRE: true
         * POST: if (balance - amount < 0) then
         *         exception RuntimeException
         *       else
         *         balance == balance@PRE - amount
         * OBLIGATION: Fangen einer allfaelligen
         *       java.lang.RuntimeException, welche bei einem einem Sparkontosaldo beim
         *       Unterschreiten von 0 ausgloest wird.
         */
        public void withdraw(float amount) {
                float oldBalance = balance;
                if (balance - amount < 0) {
                        throw new RuntimeException("Balance of " + balance
                                        + " to low for withdrawing amount of " + amount);
                } else {
                        balance -= amount;
                }
                // Beobachter benachrichtigen
                setChanged();
                notifyObservers(new Float(oldBalance));
        }
}
```