Rešitev 8. domače naloge (Banka)

Zgradba rešitve

Rešitev bomo sestavili iz petih razredov:

- Razred `Glavni` vsebuje metodo `main`, ki v zanki bere in izvršuje ukaze, zapisane na standardnem vhodu.
- Objekt razreda `Banka` predstavlja banko. Objekt hrani seznam bančnih računov in obrestni meri za pozitivno in negativno stanje. V razredu `Banka` so med drugim definirane metode za odpiranje (ustvarjanje) novega računa, polaganje denarja na izbrani račun in dvigovanje denarja z izbranega računa.
- Objekt razreda `Racun` predstavlja posamezen račun. Objekt hrani naziv računa, trenutno stanje na računu, omejitve, ki veljajo za račun, in atribute, potrebne za preverjanje omejitev in obračunavanje obresti. V razredu `Racun` sta med drugim definirani metodi za polaganje in dvigovanje denarja.
- Ker bomo veliko delali z datumi, nam bo koristil poseben razred za predstavitev tega koncepta. Objekt razreda `Datum` predstavlja datum, opredeljen z dnevom, mesecem in letom.
- Razred `Odziv` je namenjen poenoteni obravnavi odzivov na ukaze.

Razred `Glavni`

V razredu `Glavni` preberemo podatke o obrestih, »ustanovimo« banko (objekt tipa `Banka`) ter v zanki beremo in izvršujemo ukaze. Datum, sestavni del vsakega ukaza, predstavimo kot objekt tipa `Datum`.

```
import java.util.Scanner;

public class Glavni {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Banka banka = new Banka(sc.nextInt(), sc.nextInt());
        int stUkazov = sc.nextInt();

        for (int i = 0;  i < stUkazov;  i++) {
            String ukaz = sc.next();
            Datum datum = new Datum(sc.nextInt(), sc.nextInt(), sc.nextInt());

            switch (ukaz) {
                case "r": {
                    banka.odpriRacun(datum, sc.next(),
                                     sc.nextInt(), sc.nextInt(), sc.nextInt());
                    System.out.println("OK");
                    break;
                }

                case "+": {
                    Odziv odziv = banka.polozi(datum, sc.next(), sc.nextInt());
                    System.out.println(odziv.vrni());
```

```
                    break;
                }

                case "-": {
                    Odziv odziv = banka.dvigni(datum, sc.next(), sc.nextInt());
                    System.out.println(odziv.vrni());
                    break;
                }
            }
        }
    }
}
```

Razred Banka

Banka hrani podatke o računih. Ker število računov ne more biti večje od 1000 (`MAKS_ST_RACUNOV`), si pripravimo tabelo te velikosti (`racuni`), poleg nje pa še spremenljivko `stRacunov`, ki podaja dejansko število računov v tabeli. Objekt tipa `Banka` hrani tudi obrestni meri za pozitivno in negativno stanje.

```
public class Banka {

    private static final int MAKS_ST_RACUNOV = 1000;

    private Racun[] racuni;
    private int stRacunov;
    private int obrestiPlus;
    private int obrestiMinus;
    ...
}
```

V konstruktorju inicializiramo vse atribute. Obrestni meri posredujemo razredu `Racun`. Ta ju bo shranil kot statična atributa, saj sta za vse račune enaki.

```
public class Banka {
    ...
    public Banka(int obrestiPlus, int obrestiMinus) {
        this.racuni = new Racun[MAKS_ST_RACUNOV];
        this.stRacunov = 0;
        this.obrestiPlus = obrestiPlus;
        this.obrestiMinus = obrestiMinus;
        Racun.nastaviObresti(obrestiPlus, obrestiMinus);
    }
    ...
}
```

Metoda `odpriRacun` ustvari nov objekt tipa `Racun` in ga doda v tabelo `racuni`.

```
public class Banka {
    ...
    public void odpriRacun(Datum datum, String naziv,
            int maksMinus, int maksDnevniDvig, int maksMesecniDvig) {

        this.racuni[this.stRacunov] = new Racun(datum, naziv,
```

```
                maksMinus, maksDnevniDvig, maksMesecniDvig);
        this.stRacunov++;
    }
    ...
}
```

Metodi `polozi` in `dvigni` najprej v tabeli `racuni` poiščeta račun s podanim nazivom, nato pa nad dobljenim objektom pokličeta ustrezno metodo.

```
public class Banka {
    ...
    public Odziv polozi(Datum datum, String naziv, int znesek) {
        return this.poisciRacun(naziv).polozi(datum, znesek);
    }

    public Odziv dvigni(Datum datum, String naziv, int znesek) {
        return this.poisciRacun(naziv).dvigni(datum, znesek);
    }
    ...
}
```

Metoda `poisciRacun` se sprehodi po tabeli računov in poišče račun z iskanim nazivom. Če računa ne najde, vrne `null`. (Če metoda dejansko vrne `null`, je to znak, da se nam je v program prikradel hrošč.)

```
public class Banka {
    ...
    private Racun poisciRacun(String naziv) {
        for (int i = 0;  i < this.racuni.length;  i++) {
            if (this.racuni[i].vrniNaziv().equals(naziv)) {
                return this.racuni[i];
            }
        }
        return null;
    }
}
```

Obstajajo tudi učinkovitejši načini za iskanje objektov po nazivu, vendar pa presegajo snov pri predmetu Programiranje 1.

Razred `Racun`

Kot smo že povedali, sta obrestni meri v razredu `Racun` shranjeni kot statična atributa, saj sta enaki za vse račune.

```
public class Racun {

    private static int s_obrestiPlus;
    private static int s_obrestiMinus;
    ...
}
```

Za vsak račun hranimo njegov naziv, omejitve (maksimalni minus, maksimalni dnevni dvig in maksimalni mesečni dvig) in trenutno stanje (količino denarja na računu).

```
public class Racun {
    ...
    private String naziv;
    private int maksMinus;
    private int maksDnevniDvig;
    private int maksMesecniDvig;
    private int stanje;
    ...
}
```

Vendar pa to še ni dovolj. Da bomo lahko preverjali omejitev maksimalnega dnevnega oziroma mesečnega dviga, hranimo skupni dvig v trenutnem dnevu (`dnevniDvig`) oziroma mesecu (`mesecniDvig`). Da bomo lahko ugotovili, ali se trenutni dvig izvrši v istem dnevu oziroma mesecu kot prejšnji, hranimo datum zadnjega dviga (`datumZadnjegaDviga`). Pred vsakim dvigom in pologom bomo izračunali obresti, ki so se nabrale v vmesnem času, in posodobili stanje računa. Da bomo lahko ugotovili, koliko mesecev je preteklo od zadnjega obračuna obresti, bomo vzdrževali datum, ko smo to naredili (`datumZadnjegaObracuna-Obresti`).

```
public class Racun {
    ...
    private int dnevniDvig;
    private int mesecniDvig;
    private Datum datumZadnjegaDviga;
    private Datum datumZadnjegaObracunaObresti;
    ...
}
```

Statična metoda `nastaviObresti` nastavi obrestni meri za pozitivno in negativno stanje.

```
public class Racun {
    ...
    public static void nastaviObresti(int obrestiPlus, int obrestiMinus) {
        s_obrestiPlus = obrestiPlus;
        s_obrestiMinus = obrestiMinus;
    }
    ...
}
```

Konstruktor nastavi začetne vrednosti vseh nestatičnih atributov. Metoda `vrniNaziv` vrne naziv računa `this`. To metodo potrebujemo pri iskanju računa v razredu `Banka`.

```
public class Racun {
    ...
    public Racun(Datum datum, String naziv,
            int maksMinus, int maksDnevniDvig, int maksMesecniDvig) {

        this.naziv = naziv;
        this.maksMinus = maksMinus;
        this.maksDnevniDvig = maksDnevniDvig;
        this.maksMesecniDvig = maksMesecniDvig;

        this.stanje = 0;
        this.dnevniDvig = 0;
        this.mesecniDvig = 0;
```

```
        this.datumZadnjegaObracunaObresti = datum;
        this.datumZadnjegaDviga = new Datum(0, 0, 0);
    }

    public String vrniNaziv() {
        return this.naziv;
    }
    ...
}
```

Metoda `polozi` se prične s klicem metode `posodobi`, ki po potrebi posodobi atributa `dnevniDvig` in `mesecniDvig` in obračuna morebitne obresti, ki so se nabrale v času od zadnjega obračuna. Metoda `polozi` nato posodobi stanje računa, kot odziv pa vrne novo stanje.

```
public class Racun {
    ...
    public Odziv polozi(Datum datum, int znesek) {
        this.posodobi(datum);
        this.stanje += znesek;
        return new Odziv(Odziv.BREZ_NAPAKE, this.stanje);
    }
    ...
}
```

Pri metodi `dvigni` imamo nekaj več dela, saj je treba najprej preveriti, ali je dvig sploh mogoče izvršiti. Preveriti moramo prekoračitev maksimalnega negativnega stanja, maksimalnega dnevnega dviga in maksimalnega mesečnega dviga (v tem vrstnem redu).

```
public class Racun {
    ...
    public Odziv dvigni(Datum datum, int znesek) {
        this.posodobi(datum);

        // preverimo, ali je znesek mogoče dvigniti
        if (this.maksMinus >= 0 && this.stanje - znesek < -this.maksMinus) {
            return new Odziv(Odziv.PREKORACITEV_MINUSA);
        }
        if (this.maksDnevniDvig >= 0 && this.dnevniDvig + znesek > this.maksDnevniDvig) {
            return new Odziv(Odziv.DNEVNA_PREKORACITEV);
        }
        if (this.maksMesecniDvig >= 0 && this.mesecniDvig + znesek > this.maksMesecniDvig) {
            return new Odziv(Odziv.MESECNA_PREKORACITEV);
        }

        // znesek lahko dvignemo
        this.dnevniDvig += znesek;
        this.mesecniDvig += znesek;
        this.datumZadnjegaDviga = datum;
        this.stanje -= znesek;

        return new Odziv(Odziv.BREZ_NAPAKE, this.stanje);
    }
    ...
}
```

Metoda `posodobi` najprej preveri, ali je podani datum različen od datuma zadnjega dviga. Če je, ponastavi atribut `dnevniDvig` na 0. Nato preveri, ali je mesec podanega datuma različen od meseca zadnjega dviga. Če je, ponastavi atribut `mesecniDvig` na 0. Nazadnje ugotovi, kdaj so se obresti nazadnje obračunale, in trenutnemu stanju prišteje obresti za vsak prelom meseca v vmesnem obdobju.

```
public class Racun {
    ...
    private void posodobi(Datum datum) {
        if (!this.datumZadnjegaDviga.istiDanKot(datum)) {
            this.dnevniDvig = 0;
        }

        if (!this.datumZadnjegaDviga.istiMesecKot(datum)) {
            this.mesecniDvig = 0;
        }

        int mesecnaRazlika = datum.mesecnaRazlika(this.datumZadnjegaObracunaObresti);

        for (int i = 0;  i < mesecnaRazlika;  i++) {
            if (this.stanje > 0) {
                this.stanje += s_obrestiPlus * this.stanje / 1000;
            } else {
                this.stanje -= s_obrestiMinus * (-this.stanje) / 1000;
            }
        }
        this.datumZadnjegaObracunaObresti = datum;
    }
}
```

Razred `Datum`

Razred `Datum` je namenjen predstavitvi posameznih datumov. Poleg konstruktorja vsebuje še metode za primerjanje datumov in izračun razlike v mesecih.

```
public class Datum {

    private int dan;     // zaporedna številka dneva (1–31)
    private int mesec;   // zaporedna številka meseca (1–12)
    private int leto;

    public Datum(int dan, int mesec, int leto) {
        this.dan = dan;
        this.mesec = mesec;
        this.leto = leto;
    }

    // Vrne true natanko v primeru, če imata datuma this in datum isti dan, mesec in leto.
    public boolean istiDanKot(Datum datum) {
        return (this.dan == datum.dan && this.istiMesecKot(datum));
    }

    // Vrne true natanko v primeru, če imata datuma this in datum isti mesec in leto.
    public boolean istiMesecKot(Datum datum) {
        return (this.mesec == datum.mesec && this.leto == datum.leto);
    }
```

```
    // Vrne število prelomov meseca med datumom datum in datumom this.
    public int mesecnaRazlika(Datum datum) {
        return 12 * (this.leto - datum.leto) + (this.mesec - datum.mesec);
    }
}
```

Razred `Odziv`

Objekt razreda `Odziv` predstavlja odziv na ukaz. Odziv je lahko bodisi ena od oznak napake (`N`, `D` ali `M`) ali pa podatek o novem stanju računa. Napako hranimo v obliki številske kode ($0 \mapsto$ `N`, $1 \mapsto$ `D`, $2 \mapsto$ `M`), ki obenem služi kot indeks v tabelo oznak napak. Kodo napake in podatek o stanju računa hranimo kot dva ločena atributa, čeprav je v vsakem trenutku smiseln samo eden (če je prišlo do napake, nas stanje ne zanima, in obratno). Zato bi bilo bolje izdelati razreda `OdzivNapaka` (za odziv v obliki oznake napake) in `OdzivStanje` (za odziv v obliki podatka o novem stanju) in ju izpeljati iz abstraktnega razreda `Odziv`.

```
public class Odziv {

    public static final int BREZ_NAPAKE = -1;
    public static final int PREKORACITEV_MINUSA = 0;     // N
    public static final int DNEVNA_PREKORACITEV = 1;     // D
    public static final int MESECNA_PREKORACITEV = 2;    // M

    private static final String[] OZNAKA_NAPAKE = {"N", "D", "M"};

    private int kodaNapake;
    private int stanje;

    public Odziv(int kodaNapake) {
        this(kodaNapake, 0);
    }

    public Odziv(int kodaNapake, int stanje) {
        this.kodaNapake = kodaNapake;
        this.stanje = stanje;
    }

    public String vrni() {
        if (this.kodaNapake == BREZ_NAPAKE) {
            return Integer.toString(this.stanje);
        }
        return OZNAKA_NAPAKE[this.kodaNapake];
    }
}
```