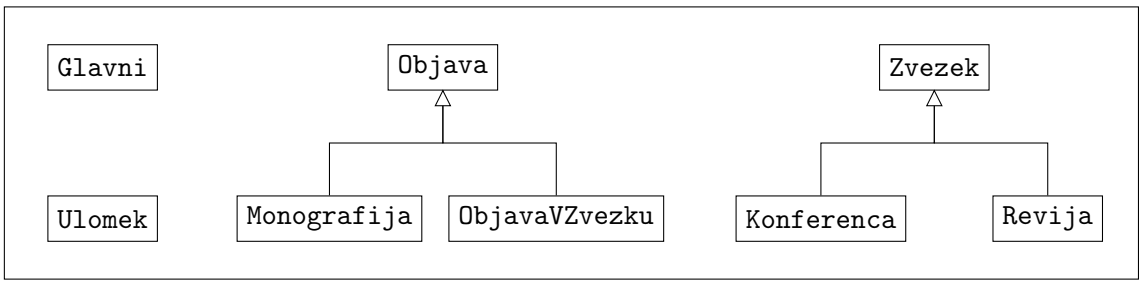
Rešitev osme domače naloge (Bibliografija)

Zgradba rešitve

Ker obravnavamo tri vrste objav, se ponuja rešitev z abstraktnim razredom `Objava` in neposredno izpeljanimi razredi `Monografija`, `Referat` in `Clanek`. S takšnim pristopom seveda ni nič narobe, kljub temu pa bomo prikazali rešitev, ki skupne lastnosti različnih tipov publikacij še močneje izkorišča. Rešitev temelji na opažanju, da lahko referate in članke združimo pod skupno streho.

Iz abstraktnega razreda `Objava` bomo izpeljali razreda `Monografija` in `ObjavaVZvezku`. Objekt razreda `ObjavaVZvezku` lahko predstavlja referat ali članek, saj sta oba tipa prispevkov objavljena v *zvezku* — bodisi v konferenčnem zborniku bodisi v reviji. Zvezek, v katerem je objavljen referat ali članek, bomo predstavili kot objekt abstraktnega razreda `Zvezek` oziroma njegovih podrazredov `Konferenca` in `Revija`.

Poleg hierarhije za predstavitev objav in hierarhije za predstavitev zvezkov bomo napisali tudi izvršilni razred `Glavni` in pomožni razred `Ulomek`. Razredi in hierarhični odnosi med njimi so prikazani na sliki 1.



Slika 1: *Razredi, ki tvorijo rešitev, in hierarhični odnosi med njimi.*

Razred `Glavni`

V metodi `main` najprej preberemo objave s standardnega vhoda v tabelo tipa `Objava[]`, nato pa tabelo uredimo po padajočih točkah objav in izpišemo njeno vsebino.

```
import java.util.Scanner;

public class Glavni {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String referencniAvtor = sc.next();
        int stObjav = sc.nextInt();

        Objava[] objave = new Objava[stObjav];
        for (int i = 0;  i < stObjav;  i++) {
            objave[i] = preberiObjavo(sc, referencniAvtor);
        }
        urediObjave(objave);

        for (int i = 0;  i < stObjav;  i++) {
            Ulomek tocke = objave[i].tocke();
            System.out.printf("%s | %s%n", objave[i].toString(), tocke.toString());
        }
```

```
    }
    ...
}
```

Metoda `preberiObjavo` prebere vrsto objave, seznam avtorjev, naslov objave in podatke, specifične za posamezne vrste objav. Metoda ustvari in vrne objekt tipa `Monografija` (če gre za monografijo) ali `ObjavaVZvezku` (če gre za referat ali članek). Če je objava konferenčni referat, se objektu tipa `ObjavaVZvezku` ob izdelavi posreduje objekt tipa `Konferenca` s podatki o konferenci. Če je objava članek v reviji, pa se posreduje objekt tipa `Revija` s podatki o reviji.

```
public class Glavni {
    ...
    private static Objava preberiObjavo(Scanner sc, String referencniAvtor) {
        String vrsta = sc.next();
        String[] avtorji = preberiAvtorje(sc, referencniAvtor);
        String naslov = sc.next();

        switch (vrsta) {
            case "monografija":
                return new Monografija(avtorji, naslov,
                                       sc.next(), sc.nextInt(), sc.next());

            case "referat":
                return new ObjavaVZvezku(avtorji, naslov,
                        new Konferenca(sc.next(), sc.nextBoolean()),
                        sc.nextInt(), sc.nextInt());

            default:
                return new ObjavaVZvezku(avtorji, naslov,
                        new Revija(sc.next(), sc.nextInt(), sc.nextInt(),
                                   sc.nextInt(), sc.nextInt(), sc.nextInt()),
                        sc.nextInt(), sc.nextInt());
        }
    }

    // Vrne tabelo avtorjev, ki jih prebere s standardnega vhoda.
    // Znak # se nadomesti z referenčnim avtorjem.
    private static String[] preberiAvtorje(Scanner sc, String referencniAvtor) {
        int stAvtorjev = sc.nextInt();
        String[] avtorji = new String[stAvtorjev];
        for (int i = 0;  i < stAvtorjev;  i++) {
            String avtor = sc.next();
            if (avtor.equals("#")) {
                avtor = referencniAvtor;
            }
            avtorji[i] = avtor;
        }
        return avtorji;
    }
    ...
}
```

Metoda `urediObjave` uredi podano tabelo objav po padajočem številu točk. Tabelo uredimo z algoritmom navadnega vstavljanja, saj ta med seboj ne zamenjuje elementov, ki so po urejevalnem kriteriju med seboj enaki. Zato bodo objave z enakim številom točk v urejeni tabeli nanizane v istem medsebojnem vrstnem redu kot na vhodu.

Ker so točke objav ulomki (objekti tipa `Ulomek`), jih ne moremo primerjati z navadnimi primerjalnimi operatorji, pač pa si pomagamo z metodo `primerjaj` iz razreda `Ulomek`.

```
public class Glavni {
    ...
    private static void urediObjave(Objava[] objave) {
        for (int i = 1;  i < objave.length;  i++) {
            Objava obj = objave[i];
            Ulomek tocke = obj.tocke();
            int j = i - 1;
            while (j >= 0 && objave[j].tocke().primerjaj(tocke) < 0) {
                objave[j + 1] = objave[j];
                j--;
            }
            objave[j + 1] = obj;
        }
    }
}
```

Razred `Objava`

Z razredom `Objava` zajamemo skupne lastnosti monografij, referatov in člankov. Za vse tipe objav hranimo seznam avtorjev in naslov. Točke se pri vseh tipih izračunajo kot količnik med točkovno osnovo in številom avtorjev, razlike so le pri računanju točkovne osnove (zato je metoda `tockovnaOsnova` abstraktna). Metoda `toString` vrne del izpisa, ki je skupen vsem tipom objav. Ta del tvorijo z vejico ločen seznam avtorjev, dvopičje, presledek, naslov objave in pika.

```
public abstract class Objava {

    private String[] avtorji;
    private String naslov;

    protected Objava(String[] avtorji, String naslov) {
        this.avtorji = avtorji;
        this.naslov = naslov;
    }

    public Ulomek tocke() {
        return new Ulomek(this.tockovnaOsnova(), this.avtorji.length);
    }

    public abstract int tockovnaOsnova();

    @Override
    public String toString() {
        return String.format("%s: %s.", lociZVejico(this.avtorji), this.naslov);
    }

    private static String lociZVejico(String[] elementi) {
        return String.join(", ", elementi);
    }
}
```

Konstruktor je namenjen izključno temu, da ga s pomočjo konstrukta `super(...)` kličemo iz konstruktorjev podrazredov, objektov tipa `Objava` pa seveda ni mogoče izdelovati.

Razred `Monografija`

Razred `Monografija` je podrazred razreda `Objava`, zato v njem definiramo samo atribute, specifične za monografije. Metoda `tockovnaOsnova` je enostavna, saj je točkovna osnova za monografijo vedno enaka 10. Izpis je sestavljen iz izpisa, skupnega vsem tipom objav, ter iz založbe, leta objave in kode ISBN.

```
public class Monografija extends Objava {

    private String zalozba;
    private int leto;
    private String isbn;

    public Monografija(String[] avtorji, String naslov,
            String zalozba, int leto, String isbn) {

        super(avtorji, naslov);
        this.zalozba = zalozba;
        this.leto = leto;
        this.isbn = isbn;
    }

    @Override
    public int tockovnaOsnova() {
        return 10;
    }

    @Override
    public String toString() {
        return String.format("%s %s %d, ISBN %s",
                super.toString(), this.zalozba, this.leto, this.isbn);
    }
}
```

Razred `ObjavaVZvezku`

Razred `ObjavaVZvezku` je podoben razredu `Monografija`. Točkovna osnova objave v zvezku je enaka točkovni osnovi zvezka; objava na »dobri« konferenci ali v »dobri« reviji samodejno velja za »dobro« in obratno. Izpis za objavo v zvezku tvorijo izpis, ki je skupen vsem tipom objav, izpis podatkov o zvezku (naziv konference oziroma naziv, letnik in številka revije) ter začetna in končna stran, pri objavi v reviji pa sledi še leto objave. Metoda `izpisLeta` bo zato vrnila prazen niz v primeru konference in niz, ki podaja leto objave, v primeru revije.

```
public class ObjavaVZvezku extends Objava {

    private Zvezek zvezek;
    private int zacetnaStran;
    private int koncnaStran;

    public ObjavaVZvezku(String[] avtorji, String naslov,
            Zvezek zvezek, int zacetnaStran, int koncnaStran) {

        super(avtorji, naslov);
        this.zvezek = zvezek;
        this.zacetnaStran = zacetnaStran;
```

```
        this.koncnaStran = koncnaStran;
    }

    @Override
    public int tockovnaOsnova() {
        return this.zvezek.tockovnaOsnova();
    }

    @Override
    public String toString() {
        return String.format("%s %s: %d-%d%s",
                super.toString(), this.zvezek.toString(),
                this.zacetnaStran, this.koncnaStran, this.zvezek.izpisLeta());
    }
}
```

Razred `Zvezek`

Razred `Zvezek` zajema tisto, kar je skupno konferencam in revijam. To je pravzaprav samo naziv. Metoda `tockovnaOsnova` bo definirana posebej za konference in revije, zato je v razredu `Zvezek` abstraktna. Metoda `toString` bo v razredu `Konferenca` podedovana, v razredu `Revija` pa redefinirana, saj je pri konferencah izpis podatkov sestavljen samo iz naziva, pri revijah pa iz naziva, letnika in številke. Podobno velja za metodo `izpisLeta`: izpis za revijo se zaključi z letom objave, izpis za konferenco pa tega podatka ne vsebuje.

```
public abstract class Zvezek {

    private String naziv;

    protected Zvezek(String naziv) {
        this.naziv = naziv;
    }

    public abstract int tockovnaOsnova();

    @Override
    public String toString() {
        return this.naziv;
    }

    public String izpisLeta() {
        return "";
    }
}
```

Razred `Konferenca`

V razredu `Konferenca` definiramo metodo `tockovnaOsnova` (točkovna osnova je odvisna od tega, ali je konferenca domača ali mednarodna), metodi `toString` in `izpisLeta` pa se samo podedujeta.

```
public class Konferenca extends Zvezek {

    private boolean jeMednarodna;
```

```
    public Konferenca(String naziv, boolean jeMednarodna) {
        super(naziv);
        this.jeMednarodna = jeMednarodna;
    }

    @Override
    public int tockovnaOsnova() {
        return (this.jeMednarodna ? 3 : 1);
    }
}
```

Razred `Revija`

Podatki o reviji zajemajo poleg naziva še letnik, številko in leto objave. Izračun točkovne osnove revije je nekoliko bolj zapleten, zato se nam ga splača izračunati enkrat za vselej in shraniti v atribut. To seveda ne bi bilo nujno; točkovno osnovo bi lahko izračunali enostavno ob vsakem klicu metode `tockovnaOsnova`.

```
public class Revija extends Zvezek {

    private int letnik;
    private int stevilka;
    private int leto;
    private int tocke;

    public Revija(String naziv, int letnik, int stevilka, int leto,
            int mesto, int stPomembnihRevij) {

        super(naziv);
        this.letnik = letnik;
        this.stevilka = stevilka;
        this.leto = leto;
        this.tocke = izracunajTocke(mesto, stPomembnihRevij);
    }
    ...
}
```

Točkovno osnovo izračunamo tako, da poiščemo najmanjši $i \in \{1, 2, 3, 4\}$, pri katerem velja $\frac{z}{k} \leqslant \frac{i}{4}$. Točkovna osnova je potem enaka $12 - 2i$.

```
public class Revija extends Zvezek {
    ...
    private static int izracunajTocke(int mesto, int stPomembnihRevij) {
        Ulomek pozicija = new Ulomek(mesto, stPomembnihRevij);
        int i = 1;
        while (i <= 4 && pozicija.primerjaj(new Ulomek(i, 4)) > 0) {
            i++;
        }
        return (12 - 2 * i);
    }
    ...
}
```

Primera $z = k + 1$ nam ni treba posebej obravnavati. Ker velja $\frac{z}{k} > \frac{4}{4}$, bo spremenljivka `i` po koncu zanke imela vrednost 5, rezultat vrnjenega izraza pa bo (skladno z navodili) enak $12 - 2 \cdot 5 = 2$.

Ker smo točkovno osnovo izračunali že v konstruktorju in jo shranili v atribut `tocke`, metoda `tockovnaOsnova` vrne kar vrednost tega atributa. Metoda `toString` sestavi izpis podatkov o reviji iz izpisa, skupnega obema tipoma zvezkov, ter iz presledka, letnika in številke v oklepajih. Metoda `izpisLeta` vrne niz, sestavljen iz presledka in leta objave v oklepajih.

```
public class Revija extends Zvezek {
    ...
    @Override
    public int tockovnaOsnova() {
        return this.tocke;
    }

    @Override
    public String toString() {
        return String.format("%s %d(%d)",
                             super.toString(), this.letnik, this.stevilka);
    }

    @Override
    public String izpisLeta() {
        return String.format(" (%d)", this.leto);
    }
}
```

Razred `Ulomek`

Objekt razreda `Ulomek` predstavlja *okrajšan* ulomek.

```
public class Ulomek {

    private int stevec;
    private int imenovalec;

    public Ulomek(int stevec, int imenovalec) {
        // ulomek okrajšamo in zagotovimo, da bo morebiten negativni predznak samo v števcu
        int predznak = Integer.signum(imenovalec);
        int g = gcd(Math.abs(stevec), Math.abs(imenovalec));
        this.stevec = predznak * stevec / g;
        this.imenovalec = predznak * imenovalec / g;
    }

    // Vrne razliko ulomkov this in u.
    public Ulomek razlika(Ulomek u) {
        return new Ulomek(
            this.stevec * u.imenovalec - this.imenovalec * u.stevec,
            this.imenovalec * u.imenovalec
        );
    }

    // Vrne 1, če je ulomek this večji od ulomka u,
    // −1, če je ulomek this manjši od ulomka u,
    // in 0, če sta ulomka enaka.
    public int primerjaj(Ulomek u) {
        return Integer.signum(this.razlika(u).stevec);
    }
```

```
    // Vrne izpis ulomka this.
    @Override
    public String toString() {
        if (this.imenovalec == 1) {
            // celoštevilski ulomek
            return Integer.toString(this.stevec);
        }
        return String.format("%d+%d/%d",
            this.stevec / this.imenovalec,
            this.stevec % this.imenovalec,
            this.imenovalec
        );
    }

    // Vrne največji skupni delitelj števil a in b.
    private static int gcd(int a, int b) {
        while (b > 0) {
            int t = a;
            a = b;
            b = t % b;
        }
        return a;
    }
}
```