Rešitev 4. domače naloge (Skladovnica)

Atributi

Katere podatke o posameznih skladovnicah naj vzdržujemo? Prav gotovo kapaciteto prvega kupa (K_1) in prirast kapacitete vsakega naslednjega kupa (ΔK):

```
public class Skladovnica {
    private int kapacitetaPrvega;  // K_1
    private int prirast;           // ΔK
    ...
}
```

Ne glede na število škatel na skladovnici so neprazni kupi od prvega do predzadnjega vsi do konca napolnjeni. Njihova zasedenost je enaka kapaciteti, zato nam je ni treba hraniti (kapaciteto določenega kupa lahko namreč izračunamo s preprosto formulo). Vzdrževati moramo torej le zasedenost zadnjega nepraznega kupa in njegovo zaporedno številko, pa je stanje skladovnice opisano v celoti.

Hranili bomo tudi skupno število škatel na skladovnici. Tega atributa sicer ne bi nujno potrebovali, saj ga lahko izračunamo na podlagi zaporedne številke zadnjega nepraznega kupa in njegove zasedenosti, kljub temu pa nam bo olajšal marsikatero metodo. Poleg tega ga ni težko vzdrževati, saj ga je treba le ustrezno povečevati oz. zmanjševati ob dodajanju oz. odvzemanju škatel.

```
public class Skladovnica {
    ...
    private int stevilkaZadnjega;    // zaporedna številka zadnjega nepraznega kupa
    private int zasedenostZadnjega;  // zasedenost zadnjega nepraznega kupa
    private int skupnoStSkatel;      // skupno število škatel na skladovnici this
    ...
}
```

Konstruktor

Konstruktor prekopira vrednosti svojih parametrov (`kapacitetaPrvega` in `prirast`) v pripadajoča atributa pravkar ustvarjenega objekta (`this.kapacitetaPrvega` in `this.prirast`), inicializira pa tudi ostale atribute. Številko zadnjega nepraznega kupa bi sicer načeloma morali nastaviti na 0, vendar pa si bomo prihranili kak pogojni stavek, če jo bomo postavili na 1. Zasedenost zadnjega nepraznega kupa je seveda enaka 0.

```
public class Skladovnica {
    ...
    public Skladovnica(int kapacitetaPrvega, int prirast) {
        this.kapacitetaPrvega = kapacitetaPrvega;
        this.prirast = prirast;

        this.stevilkaZadnjega = 1;
        this.zasedenostZadnjega = 0;
```

```
        this.skupnoStSkatel = 0;
    }
    ...
}
```

Metoda kapacitetaKupa

Prvi kup lahko vsebuje največ K_1 škatel, drugi $K_1 + \Delta K$, tretji $K_1 + 2\Delta K$, n-ti pa $K_1 + (n-1)\Delta K$.

```
public class Skladovnica {
    ...
    public int kapacitetaKupa(int kup) {
        return this.kapacitetaPrvega + (kup - 1) * this.prirast;
    }
    ...
}
```

Metoda skupnoSteviloSkatel

Ker skupno število škatel vzdržujemo v atributu, je ta metoda trivialna.

```
public class Skladovnica {
    ...
    public int skupnoSteviloSkatel() {
        return this.skupnoStSkatel;
    }
    ...
}
```

Metoda dodaj

Denimo, da želimo na skladovnico dodati s škatel. Na zadnji neprazni kup jih bomo postavili $\min(s, K^\star - Z^\star)$, kjer je $K^\star$ kapaciteta, $Z^\star$ pa trenutna zasedenost zadnjega nepraznega kupa. Razlika $(K^\star - Z^\star)$ potemtakem podaja število škatel, ki manjkajo do vrha zadnjega nepraznega kupa. Če nam še kaj škatel ostane, pričnemo polniti naslednji kup (ta postane naš novi zadnji neprazni kup). Postopek ponavljamo, dokler ne postavimo vseh škatel.

```
public class Skladovnica {
    ...
    public void dodaj(int stSkatel) {
        // stPreostalihSkatel: število škatel, ki jih moramo še postaviti
        int stPreostalihSkatel = stSkatel;
        int kapaciteta = this.kapacitetaKupa(this.stevilkaZadnjega);
        int stDodanih = Math.min(stPreostalihSkatel,
                                 kapaciteta - this.zasedenostZadnjega);
```

```
        this.zasedenostZadnjega += stDodanih;
        stPreostalihSkatel -= stDodanih;

        while (stPreostalihSkatel > 0) {  // dokler ne postavimo vseh škatel
            this.stevilkaZadnjega++;    // začnemo polniti naslednji kup
            kapaciteta = this.kapacitetaKupa(this.stevilkaZadnjega);
            stDodanih = Math.min(stPreostalihSkatel, kapaciteta);
            this.zasedenostZadnjega = stDodanih;
            stPreostalihSkatel -= stDodanih;
        }

        this.skupnoStSkatel += stSkatel;
    }
    ...
}
```

Nalogo bi lahko rešili tudi brez zanke, saj je število dodanih kupov mogoče neposredno izračunati. Podobno velja za nadaljnje metode.

Metoda `zasedenostKupa`

Kupi desno od zadnjega nepraznega so prazni, kupi levo od njega pa polni — vsebujejo toliko škatel, kot znaša njihova kapaciteta. Zadnji neprazni kup vsebuje `this.zasedenostZadnjega` škatel.

```
public class Skladovnica {
    ...
    public int zasedenostKupa(int kup) {
        if (kup > this.stevilkaZadnjega) {
            return 0;
        }
        if (kup == this.stevilkaZadnjega) {
            return this.zasedenostZadnjega;
        }
        return this.kapacitetaKupa(kup);
    }
    ...
}
```

Metoda `odvzemi`

Recimo, da želimo s skladovnice odstraniti s škatel. Če je s večji od trenutnega števila škatel na skladovnici, pri priči vrnemo `false`. V nasprotnem primeru pa najprej odstranimo $\min(s,\ Z^\star)$ škatel, kjer je $Z^\star$ trenutna zasedenost zadnjega kupa. Če s tem še nismo odstranili vseh s škatel, pričnemo prazniti predhodni kup (ta postane naš novi zadnji neprazni kup). Postopek ponavljamo, dokler naloge ne opravimo.

```
public class Skladovnica {
    ...
```

```
    public boolean odvzemi(int stSkatel) {
        if (stSkatel > this.skupnoStSkatel) {
            return false;
        }
        int stPreostalihSkatel = stSkatel;
        int stOdvzetih = Math.min(stPreostalihSkatel, this.zasedenostZadnjega);
        this.zasedenostZadnjega -= stOdvzetih;
        stPreostalihSkatel -= stOdvzetih;

        while (stPreostalihSkatel > 0) {
            this.stevilkaZadnjega--;
            int kapaciteta = this.kapacitetaKupa(this.stevilkaZadnjega);
            stOdvzetih = Math.min(stPreostalihSkatel, kapaciteta);
            this.zasedenostZadnjega = kapaciteta - stOdvzetih;
            stPreostalihSkatel -= stOdvzetih;
        }

        this.skupnoStSkatel -= stSkatel;
        return true;
    }
    ...
}
```

Metoda `poisciKup`

Če je podana številka škatle večja od števila škatel na skladovnici, metoda vrne -1, saj iskana škatla očitno ne obstaja. V nasprotnem primeru pa po vrsti pregledujemo kupe, dokler ne naletimo na prvega, pri katerem je številka vrhnje škatle najmanj enaka številki iskane škatle.

```
public class Skladovnica {
    ...
    public int poisciKup(int skatla) {
        if (skatla > this.skupnoStSkatel) {
            return -1;
        }
        int kup = 1;
        int vrhnjaSkatla = this.kapacitetaKupa(kup);
        while (vrhnjaSkatla < skatla) {
            kup++;
            vrhnjaSkatla += this.kapacitetaKupa(kup);
        }
        return kup;
    }
    ...
}
```

Metoda `prestavi`

Najprej ustvarimo novo skladovnico s podano kapaciteto in prirastom kapacitete, nato nanjo dodamo toliko škatel, kot jih je trenutno na skladovnici `this`, nazadnje pa skladovnico `this` izpraznimo. Metoda vrne referenco na novo skladovnico.

```
public class Skladovnica {
    ...
    public Skladovnica prestavi(int kapacitetaPrvega, int prirast) {
        Skladovnica nova = new Skladovnica(kapacitetaPrvega, prirast);
        nova.dodaj(this.skupnoStSkatel);
        this.odvzemi(this.skupnoStSkatel);
        return nova;
    }
    ...
}
```