Zadatak 1.

U Javi napisati klasu `Fraction` kojom se modelira razlomak čiji je brojnik i nazivnik cijeli broj. Jednom postavljeni razlomak ne može se mijenjati.

Potrebno je implementirati metode kojim se razlomci mogu zbrajati, oduzimati, množiti, dijeliti te pronaći recipročan razlomak i razlomak suprotnog predznaka. Svaka od navedenih metoda ne mijenja objekt nad kojim je pozvana, već vraćaju novi razlomak.

Za pronalazak najvećeg zajedničkog djeljitelja dva pozitivna broja koristite sljedeću metodu te ju po potrebi upotrijebite u ostalim metodama:

```
private static int gcd(int x, int y) {
      return (y == 0) ? x : gcd(y, x % y);
}
```

Pretpostavite da se dijeljenje s nulom neće nikada dogoditi. Primjer programskog koda metode `main` je dan u nastavku:

```
Fraction f1 = new Fraction(1, 4);
Fraction f2 = new Fraction(2, 3);

Fraction f = f1.add(f2);
System.out.println(f);//ispisuje: 11 / 12

f = f1.subtract(f2);
System.out.println(f);//ispisuje: -5 / 12

System.out.println(f2.invert().negate()); // -3/2

f = f1.multiply(f2);
System.out.println(f);//ispisuje: 1 / 6

f = f1.divide(f2);
System.out.println(f);//ispisuje: 3 / 8
```

Zadatak 2. Napravite klase iz dijagrama klasa. One predstavljaju sustav ispate plaća. Nalozi za plaćanje vrše se preko klase `BankManager`.

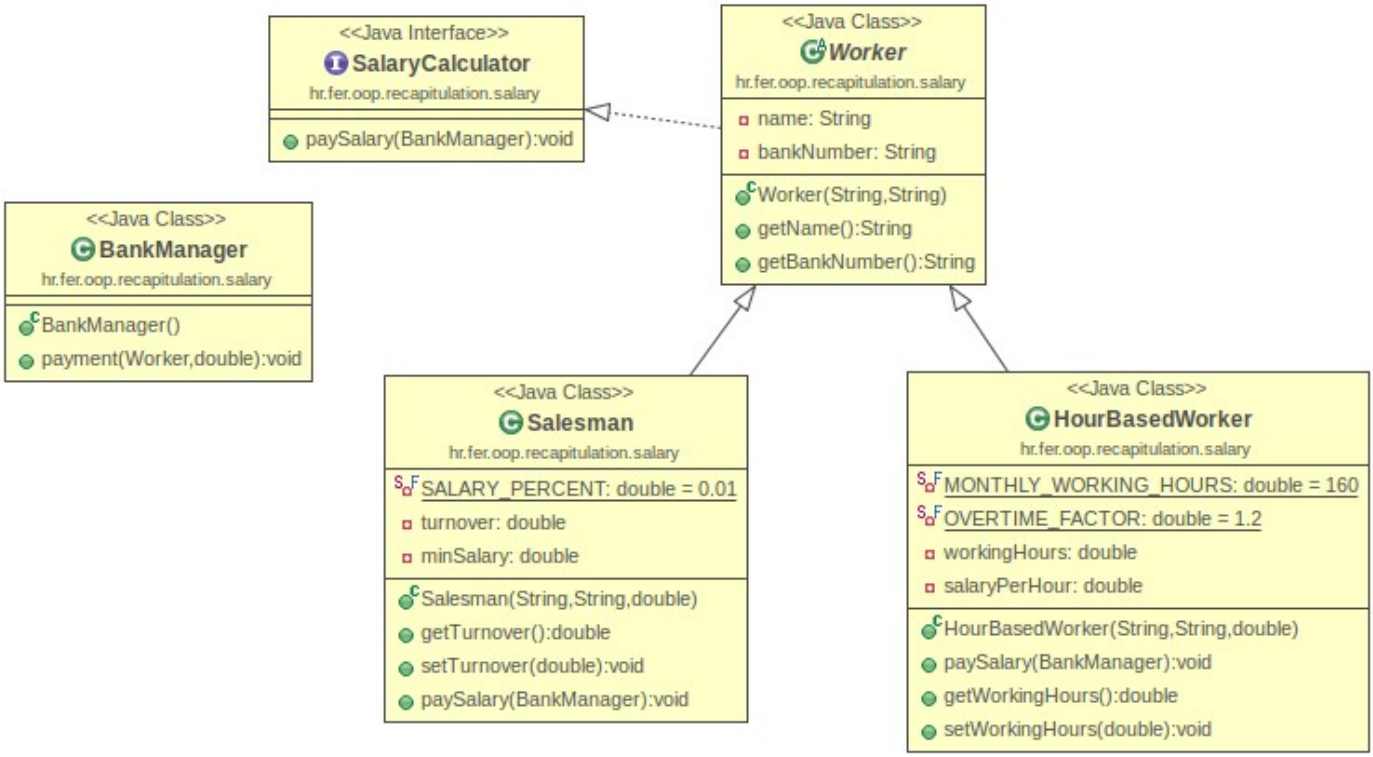
```
public class BankManager {
  public void payment(Worker worker, double amount) {
    System.out.println(worker.getName() + " - " + worker.getBankNumber() + ": " +
      amount);
  }
}
```

Klasa `Worker` je osnovna klasa koja predstavlja radnika kojem se može izračunati plaća i dati nalog za isplatu. Plaća se računa i isplaćuje pozivom metode iz sučelja `SalaryCalculator`. Postoje dvije klase koje nasljeđuju radnika. Klasa `Salesman` predstavlja radnika koji ima minimalnu plaću zagarantiranu (`minSalary`), a postotak (konstanta `SALARY_PERCENT` koja je 1%) od prodaje (`turnover`) se nadodaje na tu plaću. Klasa `HourBasedWorker` predstavlja radnika koji radi na satnicu. Plaća po satu je `salaryPerHour` ako je broj sati manji od konstante `MONTHLY_WORKING_HOUR` (160), a sve preko te satnice se plaća uvećano za faktor (konstanta `OVERTIME_FACTOR` = 1,2). Uz ove klase su zadane klase `Main` i klasa `BankManager` kojoj se šalje nalog za plaćanje (metoda samo ispisuje nalog).

```
public class Main {
  public static void main(String[] args) {
    BankManager bankManager = new BankManager();

    SalaryCalculator employeeList[] = new SalaryCalculator[3];

    Salesman salesman = new Salesman("s1", "s1b", 3000);
    salesman.setTurnover(10000);
    employeeList[0] = salesman;

    HourBasedWorker hourWorker1 = new HourBasedWorker("h1_no", "h1", 50);
    hourWorker1.setWorkingHours(100);
    employeeList[1] = hourWorker1;

    HourBasedWorker hourWorker2 = new HourBasedWorker("h2_overtime", "h2", 50);
    hourWorker2.setWorkingHours(200);
    employeeList[2] = hourWorker2;

    for (SalaryCalculator salaryCalculator : employeeList) {
      salaryCalculator.paySalary(bankManager);
    }
  }
}
```

Ispis:

```
s1 - s1b: 3100.0
h1_no - h1: 5000.0
h2_overtime - h2: 10400.0
```



Zadatak 3.

Potrebno je napraviti vlastitu implementaciju jednostruko povezane liste `MyList` sa sljedećim javnim metodama:

`int addLast(Object o)` – dodaje element na kraj liste, vraća poziciju na koju je element dodan
`void removeAt(int index)` – briše element na određenoj poziciji u listi
`Object elementAt(int index)` – dohvaća element na određenoj poziciji u listi ili vraća `null` ako on ne postoji
`int size()` – vraća broj elemenata liste

Čvor liste modelirajte posebnom klasom `MyListElement`. Primjer programskog koda metode `main` je dan u nastavku:

```
MyList list = new MyList();

list.addLast("first");
list.addLast("second");
list.addLast("third");

System.out.println(list.size());//ispisuje: 3
list.removeAt(1);
System.out.println(list.elementAt(1));//ispisuje: third
System.out.println(list.size());//ispisuje: 2

list.addLast("fourth");
System.out.println(list.elementAt(2));//ispisuje: fourth
```

Zadatak 4.
Potrebno je modelirati klase i sučelja koje se koriste u aplikaciji za logistiku.

Tereti (`Cargo`) imaju masu (`weight`), volumen (`volume`) i jedinstvenu oznaku (`id`). U spremnike (`CargoHolder`) se tereti mogu staviti i iz njih ukloniti. Modelirajte trenutni teret u spremniku korištenjem liste `MyList` iz prethodnog zadatka. Za spremnike je potrebno omogućiti da se dohvati njihova ukupna težina s teretom (`getWeight`). Spremnici mogu imati limitiranu maksimalnu težinu ili volumen tereta kojeg mogu primiti.

Tri su konkretne vrste spremnika: kontejner (`Container`), kamion (`BoxedCargoTruck`) i brod (`ContainerShip`). Kontejner ima definiranu težinu, volumen, maksimalni volumen tereta (`maxCargoVolume`) koji se u njega može staviti te svoj jedinstveni `id`. Pretpostavite da je definirani volumen kontejnera uvijek veći od maksimalnog volumena tereta u njemu.

U kamionu se može prevoziti samo posebna vrsta zapakiranog tereta (`BoxedCargo`). Kamion definira svoju težinu i maksimalnu težinu zapakiranog tereta (`maxCargoWeight`) koji smije prevoziti.

Brod prevozi teret u obliku ograničenog broja kontejnera (`maxContainers`), a definira svoju težinu i maksimalnu težinu tereta koji smije prevoziti.

Primjer programskog koda metode `main` je dan u nastavku:

```
Cargo boxedCargo1 = new BoxedCargo(32.5, 56.7, 0); // w, v, id
Cargo boxedCargo2 = new BoxedCargo(18.9, 23.5, 1);

Container container = new Container(10, 100, 80, 3); // w, v, maxV, id
System.out.println(container.add(boxedCargo1)); // true
System.out.println(container.add(boxedCargo2)); // 56.7 + 23.5 = 80.2 -> false
System.out.println(container.getWeight()); // 10 + 32.5 -> 42.5

System.out.println(container.remove(boxedCargo2)); // false
System.out.println(container.remove(boxedCargo1)); // true
System.out.println(container.getWeight()); // 10

CargoHolder truck = new BoxedCargoTruck(8.2, 60); // w, maxW
System.out.println(truck.add(boxedCargo1)); // true
System.out.println(truck.add(boxedCargo2)); // true
System.out.println(truck.getWeight()); // 8.2 + 32.5 + 18.9 -> 59.6

CargoHolder ship = new ContainerShip(120.3, 567.2, 60); // w, maxW, maxC
System.out.println(ship.add(container)); // true
System.out.println(ship.remove(container)); // true

System.out.println(truck.add(container)); // false
System.out.println(ship.add(boxedCargo2)); // false
```

Zadatak 5.

Stablo je podatkovna struktura u kojoj svaki element (čvor) stabla sadrži podatak i reference na podređene čvorove (tj. djecu) pri čemu među referencama nema duplikata te niti jedna referenca u nekom čvoru ne pokazuje na korijen stabla (prvi čvor u stablu).

Binarno stablo je stablo u kojem svaki čvor može imati maksimalno dvoje djece. Čvorovi bez djece nazivaju se listovima.

Binarno stablo traženja je binarno stablo kod kojeg za svaki čvor u stablu vrijedi da se u njegovom lijevom podstablu (stablo koje počinje s lijevim djetetom promatranog čvora) nalaze manje vrijednosti, a u desnom podstablu veće vrijednosti nego što je vrijednost u promatranom čvoru.

Napisati klasu `BST` koja predstavlja binarno stablo traženja (BST – *Binary Search Tree*) u kojem su pohranjeni cijeli brojevi. Klasa mora imati mogućnost dodavanja članova u stablo (`public void add(int i)`) i ispis članova stabla (`public void print()`).

Nakon toga napisati glavnim program koji će brojeve učitane kao argumente programa pohraniti u stablu i ispisati vrijednosti svih elemenata u stablu. Duplikate treba ignorirati.