```cpp
#ifndef _BENEFICIARY_HPP_
#define _BENEFICIARY_HPP_
#include <bits/stdc++.h>
using namespace std;
typedef unsigned long long ull;

class Beneficiary {
  friend class DonationManager;
  static unsigned long long currId; //serve para garantir que todos os beneficiarios têm um id diferente
  string name;
protected:
  ull id;
  double value=0;
public:
  Beneficiary(const string& name) {
    id = this->currId;
    this->currId = (1103515245*this->currId + 12345)%(1ULL<<31); //Isto nao e melhor forma de o fazer, mas damos a cada Beneficiario um id unico.
    //Com a seed atual, e possivel haver K benificiarios diferentes
    this->name = name;
  }
  double getValue() {
    return value;
  }
  virtual void receiveDonation(double don) {
    value += don;
  }
  void addValue(double d) {
    value += d;
  }

  virtual int contains(ull id) {
    return this->id==id;
  }
  friend std::ostream & operator <<( std::ostream &os, const Beneficiary &b ) {
    os << "[" << b.id << "]\t" << b.name << ":\t" << b.value;
    return os;
  }
};

class Individual : public Beneficiary{
  friend class DonationManager;
public:
   Individual(const string& name) : Beneficiary(name) {}
 };

class Population : public Beneficiary{
  friend class DonationManager;
protected:
  vector<Beneficiary *> subBeneficiary;
public:
  Population(const string& name) :Beneficiary(name){}
  void addBeneficiary(Individual& ind) {
    subBeneficiary.push_back(&ind);
  }
  virtual int contains(ull id) {
    for (std::vector<Beneficiary *>::iterator it=subBeneficiary.begin(); it!=subBeneficiary.end(); it++) {
      if ((*it)->contains(id)) return 1;
    }
    return this->id==id;
```

```
60      }
        virtual void receiveDonation(double don) {
          addValue(don);
          for (std::vector<Beneficiary *>::iterator it=subBeneficiary.begin(); it!=
    subBeneficiary.end(); it++) {
            (*it)->receiveDonation(don/subBeneficiary.size());
65        }
        }
    };

    class Region : public Population {
70    friend class DonationManager;
    public:
        Region(const string& name) : Population(name) {}
        void addBeneficiary(Individual& ind) {
          Population::addBeneficiary(ind);
75      }
        void addBeneficiary(Population& pop) {
          Population::subBeneficiary.push_back(&pop);
        }
    };
80
    /*Esta classe nao faz a gestao de memoria, outra classe que trate disso*/
    class DonationManager {
    public:
        vector<Beneficiary *> ben;
85      void registerBeneficiary(Beneficiary &b) {
          ben.push_back(&b);
        }
        void addBeneficiary(Population &p, Individual &i) {
          p.addBeneficiary(i);
90      }
        void addBeneficiary(Region &r, Individual &i) {
          r.addBeneficiary(i);
        }
        void addBeneficiary(Region &r, Population &p) {
95        r.addBeneficiary(p);
        }
        void giveDonation(Beneficiary &b, double ammount) {
          b.receiveDonation(ammount);
          for (std::vector<Beneficiary *>::iterator it=ben.begin(); it!= ben.end(); it+
    +) {
100         if ((*it)->id != b.id && (*it)->contains(b.id))
              (*it)->addValue(ammount);
          }
        }
        ~Beneficiary(){
105       for (std::vector<Beneficiary *>::iterator it=ben.begin(); it!= ben.end(); it+
    +) {
            delete (*it);
          }
        }
    };
110 #endif
```