// Esercizio 11.18 del libro

/*
Si consideri il seguente modello di realtà concernente i file audio memorizzati in un riproduttore audio digitale iZod.

(A) Definire la seguente gerarchia di classi.

1. Definire una classe base polimorfa astratta FileAudio i cui oggetti rappresentano un file audio memorizzabile in un iZod. Ogni FileAudio  caratterizzato dal titolo (una stringa) e dalla propria dimensione in MB. La classe è astratta in quanto prevede i seguenti metodi virtuali puri:

• un metodo di “clonazione”: FileAudio* clone().

• un metodo bool qualita() con il seguente contratto: f->qualita() ritorna true se il file audio *f è considerato di
qualità, altrimenti ritorna false.

2. Definire una classe concreta Mp3 derivata da FileAudio i cui oggetti rappresentano un file audio in formato mp3. Ogni oggetto Mp3 è caratterizzato dal proprio bitrate espresso in Kbit/s. La classe Mp3 implementa i metodi virtuali puri di FileAudio come segue:

• per ogni puntatore p a Mp3, p->clone() ritorna un puntatore ad un oggetto Mp3 che è una copia di *p.

• per ogni puntatore p a Mp3, p->qualita() ritorna true se il bitrate di *p è ≥ 192 Kbit/s, altrimenti ritorna false.

3. Definire una classe concreta WAV derivata da FileAudio i cui oggetti rappresentano un file audio in formato WAV. Ogni oggetto WAV è caratterizzato dalla propria frequenza di campionamento espressa in kHz e dall’essere lossless oppure no (cioè con compressione senza perdita oppure con perdita). La classe WAV implementa i metodi virtuali puri di FileAudio come segue:

• per ogni puntatore p a WAV, p->clone() ritorna un puntatore ad un oggetto WAV che è una copia di *p.

• per ogni puntatore p a WAV, p->qualita() ritorna true se la frequenza di campionamento di *p è ≥ 96 kHz, altrimenti
ritorna false.

(B) Definire una classe iZod i cui oggetti rappresentano i brani memorizzati in un iZod. La classe iZod deve soddisfare le seguenti specifiche:
`
1. E definita una classe annidata Brano i cui oggetti rappresentano un brano memorizzato nell’iZod. Ogni oggetto Brano è
rappresentato da un puntatore polimorfo ad un FileAudio.

• La classe Brano deve essere dotata di un opportuno costruttore Brano(FileAudio*) con il seguente comportamento:
Brano(p) costruisce un oggetto Brano il cui puntatore polimorfo punta ad una copia dell’oggetto *p.

• La classe Brano ridefinisce costruttore di copia profonda, assegnazione profonda e distruttore profondo.

2. Un oggetto di iZod è quindi caratterizzato da un vector di oggetti di tipo Brano che contiene tutti i brani memorizzati nell’iZod.

3. La classe iZod rende disponibili i seguenti metodi:

• Un metodo vector<Mp3> mp3(double, int) con il seguente comportamento: una invocazione iz.mp3(dim,br) ritorna un vector di oggetti Mp3 contenente tutti e soli i file audio in formato mp3 memorizzati nell’iZod iz che: (i) hanno una dimensione ≥ dim e (ii) hanno un bitrate ≥ br.

• Un metodo vector<FileAudio*> braniQual() con il seguente comportamento: una invocazione iz.braniQual() ritorna il vector dei puntatori ai FileAudio memorizzati nell’iZod iz che: (i) sono considerati di qualità e (ii) se sono dei file audio WAV allora devono essere lossless.

• Un metodo void insert(Mp3*) con il seguente comportamento: una invocazione iz.insert(p) inserisce il nuovo oggetto Brano(p) nel vector dei brani memorizzati nell’iZod iz se il file audio mp3 *p non è già memorizzato in iz, mentre se il file audio *p risulta già memorizzato non provoca alcun effetto.

*/

```
/* COCCO's questions:

g++ -Weverything (tutti i possibili warning):

(1) Definition of implicit copy constructor for [ClassName] is deprecated because it has a user-declared
destructor

The implicit definition of a copy constructor as defaulted is deprecated if the class has a user-declared
copy assignment operator or a user-declared destructor. The implicit definition of a copy assignment
operator as defaulted is deprecated if the class has a user-declared copy constructor or a user-declared
destructor (15.4, 15.8). In a future revision of this International Standard, these implicit definitions
could become deleted (11.4).

=> The rationale behind this text is the well-known Rule of three.

(2)[ClassName] has no out-of-line virtual method definitions; its vtable will be emitted in every
translation unit

Clang almeno cerca di scrivere la vtable nell'implementazione (nel senso di non inline) di un metodo
virtuale per evitare di doverla scrivere per ogni unità di compilazione, se tutti i metodi virtuali sono
inline la deve scrivere per ogni unità di compilazione

Si tratta di dettaglio implementativo, out-of-standard, non di primo interesse al
livello di astrazione di pertinenza del programmatore.
Comunque la regola universale da seguire e':
separare dichiarazioni e definizioni in diversi file.
Negli esercizi usiamo metodi inline per comodita'
*/

#include<string>
#include<typeinfo>

class FileAudio {
private:
  std::string titolo;
  double size;
public:
  virtual FileAudio* clone() const = 0;
  virtual bool qualita() const =0;
  virtual ~FileAudio() {}
  double getSize() const {return size;}
  virtual bool operator==(const FileAudio& f) const {
    return typeid(*this) == typeid(f) && titolo == f.titolo && size == f.size;
  }
};

class Mp3: public FileAudio {
private:
  unsigned int Kbits;
public:
  static const unsigned int sogliaQualita;
  Mp3* clone() const override {
    return new Mp3(*this);
  }
  bool qualita() const override {return Kbits>= sogliaQualita;}
  unsigned int getBitrate() const {return Kbits;}
  bool operator==(const FileAudio& f) const override {
    return FileAudio::operator==(f)
      && Kbits == static_cast<const Mp3&>(f).Kbits;
  }
};
const unsigned int Mp3::sogliaQualita = 192;

class WAV: public FileAudio {
private:
  unsigned int frequenza;
  bool lossLess;
public:
  static const unsigned int sogliaQualita;
  WAV* clone() const override {
    return new WAV(*this);
```

```
  }
  bool qualita() const override {return frequenza >= sogliaQualita;}
  bool getLossLess() const {return lossLess;}
  bool operator==(const FileAudio& f) const override {
    return FileAudio::operator==(f)
      && frequenza == static_cast<const WAV&>(f).frequenza &&
      lossLess == static_cast<const WAV&>(f).lossLess;
  }
};
const unsigned int WAV::sogliaQualita = 96;

#include<vector>

class iZod {
private:
  class Brano {
  public:
    FileAudio* ptr; // puntatore (super)polimorfo
    // conversione FileAudio* => Brano
    Brano(FileAudio* p): ptr(p->clone()) {}
    Brano(const Brano& b): ptr(b.ptr->clone()) {}
    Brano& operator=(const Brano& b) {
      if(this != &b) {
        delete ptr;
        ptr = b.ptr->clone();
      }
      return *this;
    }
    ~Brano() {delete ptr;}
  };

  std::vector<Brano> brani;

public:
  std::vector<Mp3> mp3(double dim, int br) const {
    std::vector<Mp3> v;
    for(std::vector<Brano>::const_iterator cit = brani.begin(); cit != brani.end(); ++cit) {
      Mp3* p = dynamic_cast<Mp3*> (cit->ptr);
      if(p != nullptr && p->getSize() >= dim && p->getBitrate() >= br)
        v.push_back(*p);
    }
    return v;
  }

  std::vector<FileAudio*> braniQual() const {
    std::vector<FileAudio*> v;
    for(auto cit = brani.begin(); cit != brani.end(); cit++)
      if((cit->ptr)->qualita() &&
         (dynamic_cast<WAV*>(cit->ptr) == nullptr ||
          static_cast<WAV*>(cit->ptr)->getLossLess()))
      v.push_back(cit->ptr);
    return v;
  }

  void insert(Mp3* p) {
    bool found = false;
    for(auto it = brani.begin(); !found && it != brani.end(); ++it)
      if(*(it->ptr) == *p) found = true;
    if(!found) brani.push_back(p);
  }
};

int main() {

}
```