ex-30-11-2020.cpp

```cpp
#include<iostream>
using namespace std;

class A {
private:
  void h() {cout<<" A::h ";}
public:
  virtual void g() {cout <<" A::g ";}
  virtual void f() {cout <<" A::f "; g(); h();}
  void m() {cout <<" A::m "; g(); h();}
  virtual void k() {cout <<" A::k "; g(); h(); m(); }
  A* n() {cout <<" A::n "; return this;}
};

class B: public A {
private:
  void h() {cout <<" B::h ";}
public:
  virtual void g() {cout <<" B::g ";}
  void m() {cout <<" B::m "; g(); h();}
  void k() {cout <<" B::k "; g(); h(); m();}
  B* n() {cout <<" B::n "; return this;}
};

B* b = new B(); A* a = new B();

// COMPILA? ERRORE RUN-TIME? COSA STAMPA?
```

```
B* b = new B(); A* a = new B();

int main() {
  (a->n())->m(); // A::n A::m B::g A::h

   (a->n())->g(); // A::n B::g,   A* a->n(), TD(a->n()) = B*
  (b->n())->g(); // B::n B::g
  (b->n())->n()->g(); // B::n B::n B::g


  b->f(); // A::f B::g A::h
  b->m(); // B::m B::g B::h
  b->k(); // B::k B::g B::h B::m B::g B::h

  a->f(); // A::f B::g A::h
  a->m(); // A::m B::g A::h
  a->k(); // B::k B::g B::h B::m B::g B::h

}
```


Wrote /Volumes/GoogleDrive/My Drive/GD-MacBookPro/dida/20-21/pogg/slides/ex20-21/ex-30-11-2020.cpp

```
class clonable {
  public:
    virtual ~clonable() {}
    virtual clonable* clone() const = 0;
};

class Base : public clonable {
  public:
     Base* clone() const override { return new Base( *this ); }
};

class Derived : public Base {
  public:
     Derived* clone() const override { return new Derived( *this ); }
};

void copy_me(const Base& b) {
    Base* clone = b.clone();
    ...
    // delete clone;
};
```

```
siderati 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) inseri*
sce 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.

*/
#include<string>

class FileAudio {
private:
  std::string titolo;
  double size;
public:
  virtual FileAudio* clone() const = 0;
  virtual bool qualita() const =0;
  virtual ~FileAudio() {}
};

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;}
};
const unsigned int Mp3::sogliaQualita = 192;


class WAV: public FileAudio {
```

```
};

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;}
};
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;}
};
const unsigned int WAV::sogliaQualita = 96;

class iZod {


};
```

```
  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;}
};
const unsigned int WAV::sogliaQualita = 96;

class iZod {
private:
  class Brano {
  public:
    FileAudio* ptr; // puntatore (super)polimorfo
    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;} // OK, perche' -FileAudio() e' virtuale
    }
  }

public:

};
```