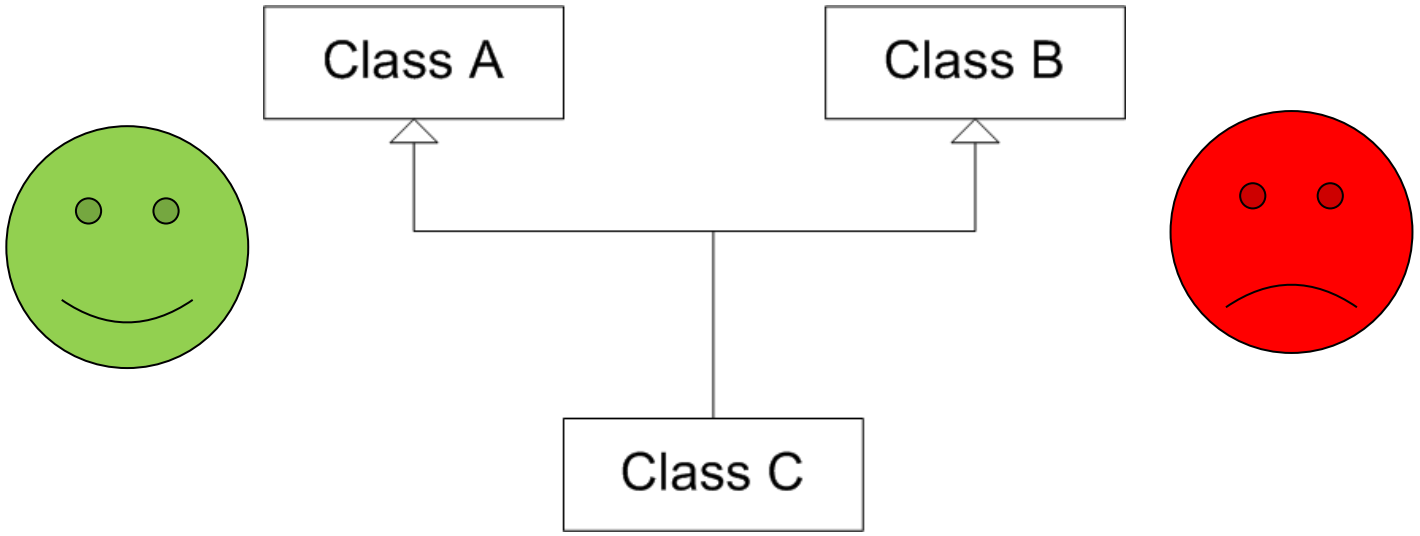
8/11/2010

Multiple Inheritance (MI)

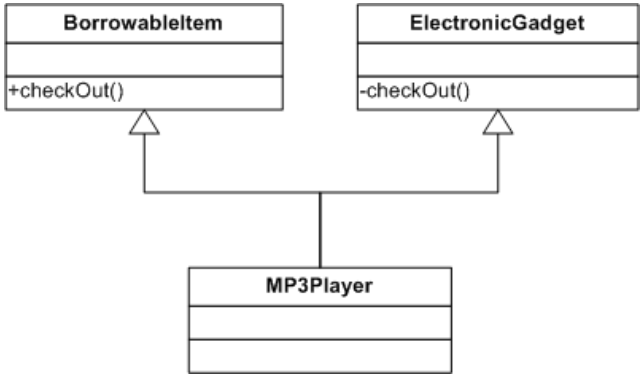
Alexander Seitel

dkfz.
DEUTSCHES
KREBSFORSCHUNGSZENTRUM
IN DER HELMHOLTZ-GEMEINSCHAFT

Recommendable?



Ambiguity problem (1/2)




```
MP3Player mp;
mp.checkOut();
```
what happens here?

1. Search for the best fitting function `checkOut()`
2. Check the accessibility of the so found function

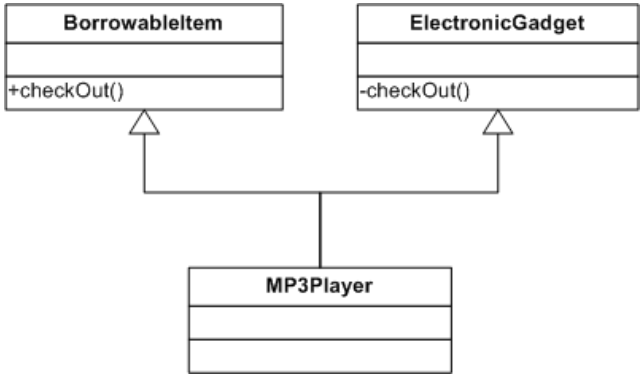
Problems:

- There is no best fitting function `checkOut()`

Ambiguity error

- `ElectronicGadget::checkOut()` will never be checked for accessibility

Ambiguity problem (2/2)



Correct access:

```
mp.BorrowableItem::checkOut();
```

On the other hand…

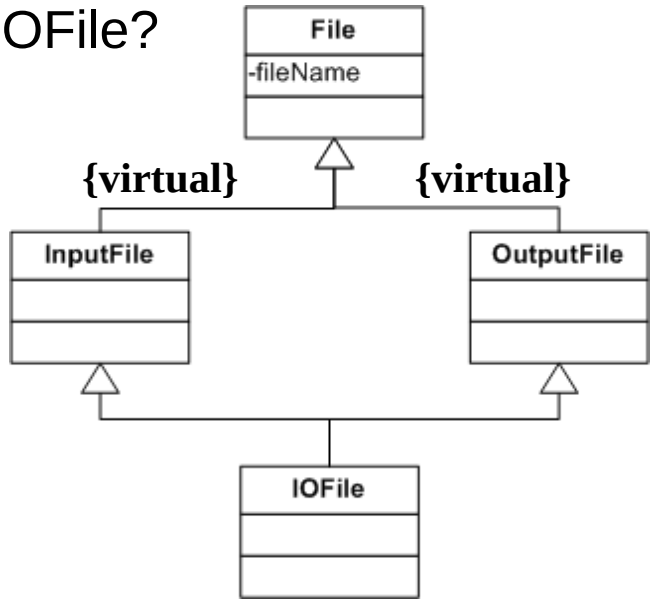
```
mp.ElectronicGadget::checkOut();
```

-> ambiguity error replaced by „you try to call a private member“ error

Virtual inheritance

How many copies of `fileName` in IOFile?

-> 2, one for every base class!



But: correct (logic) would be 1

How to achieve that?

-> **virtual Inheritance**

Always virtual inheritance? **No!**

- Instantiated objects bigger
- Access to elements of base class slower

- Don‘t use virtual inheritance if you don‘t necessarily need it!
- If you need virtual inheritance avoid to save data in the base class

Interface + implementation (1/2)

Interface class for persons

```
class IPerson {
public:
  virtual ~IPerson();
  virtual std::string name() const = 0;
  virtual std::string birthDate() const = 0;
}
```

Some database class `class DatabaseID{…}`

Database based implementation

```
class PersonInfo{
public:
  explicit PersonInfo(DatabaseID pid);
  virtual ~PersonInfo();
  virtual const char *theName() const;
  virtual const char *theBirthDate() const;
  virtual const char *valueDelimOpen() const;
  virtual const char *valueDelimClose() const;
  …
}
```

Interface + implementation (2/2)

Multiple inheritance to create concrete implementation

```
class CPerson: public IPerson, private PersonInfo {
public:
  explicit Cperson(DatabaseID pid): PersonInfo(pid){}
  virtual std::string name() const
        {return PersonInfo::theName();}
  virtual std::string birthDate() const
        {return PersonInfo::theBirthDate();}
private:
  const char* valueDelimOpen() const {return "";}
  const char* valueDelimClose() const {return "";}
  …
}
```

Private inheritance instead of composition because of new definition of virtual functions

Take home message

- MI is more complex and may lead to ambiguities
- MI sometimes causes the need of virtual inheritance
- There are some applications where the use of MI is reasonable, e.g. the combination of interface and implementation.

Try to avoid multiple inheritance as long there exist a proper way to achieve the same result without it.