Item 43: Know how to access names in templatized base classes

فرض کنید که قرار است ما یک برنامه‌ای بنویسیم که بتواند به چندین شرکت مختلف پیام ارسال کند. پیام‌ها را می‌توان هم به صورت رمز شده، و هم به صورت متن ساده ارسال کرد. اگر ما در هنگام کامپایل اطلاعات لازم را برای مشخص کردن این که چه پیامی به چه شرکتی ارسال می‌شود را داشته باشیم، می‌توانیم از روش template-based استفاده کنیم:

```
class CompanyA{
public:
    void sendCleartext(const std::string& msg);
    void sendEncrypted(const std::string& msg);
};
class CompanyB
{
public:
    void sendCleartext(const std::string& msg);
    void sendEncrypted(const std::string& msg);
};

class MsgInfo{};   //class for holding information used to create message

template<typename Company>
class MsgSender{
  void sendClear(const MsgInfo& info)
  {
     std::string msg;
     //create msg from info
     Company C;
     C.sendCleartext(msg);
  }

  void sendSecret(const MsgInfo& info)
  {
     //similar to sendClear,except calls C.sendEncrypted
  }
};
```

تا اینجا این کد به درستی کار می‌کند، اما فرض کنید که ما می‌خواهیم هر بار که پیامی ارسال شد، log هم بگیریم. یک کلاس مشتق شده می‌تواند به راحتی این قابلیت را به کد اضافه کند، در زیر یک راه منطقی برای انجام آن را آورده‌ایم.

```
template <typename Company>
class LoggingMsgSender : public MsgSender<Company>
{
public:
   void sendClearMsg(const MsgInfo& info)
   {
      //write "before sending" info to the log
      sendClear(info);    //class base class function this code will not compile!
      //write "after  sending" info to the log
   }
};
```

توجه کنید که چطور تابع message-sending در کلاس مشتق شده دارای یک نام متفاوت (sendClearMsg) از کلاس پایه آن است (که در آنجا آن sendClear نامیده می‌شد.) . این یک طراحی خوب است، چرا که مساله‌ی مخفی کردن اسامی به ارث برده شده ندارد(آیتم ۳۳ را ببینید)، همچنین مشکل دوباره‌نویسی یک تابع non-virtual را ندارد(آیتم ۳۶ را ببینید). ولی کد بالا کامپایل نخواهد شد، حداقل در مورد کامپایلرهایی که من می‌شناسم به این صورت است. این کامپایلرها در مورد این که sendClear وجود ندارد به ما ایراد خواهند گرفت. ما sendClear را می‌توانیم در کلاس پایه ببینیم، ولی کامپایلرها آنجا را نگاه نخواهند کرد. برای رفع این مشکل نیاز داریم بدانیم که چرا کامپایلرها اینگونه رفتار می‌کنند.

مشکل اینجاست که وقتی کامپایلرها با تعریف کلاس LoggingMsgSender روبه‌رو می‌شوند، نمی‌توانند بفهمند که از چه کلاسی مشتق شده است. البته که، ما می‌دانیم که آن از MsgSender<Company> مشتق شده است، ولی Company یک پارامتر template می‌باشد، این پارامتر تا موقعی که LoggingMsgSender یک نمونه ازش ساخته نشه نامشخص است، و هیچ راهی نیست که بتوان فهمید که کلاس MsgSender<Company> 1 شبیه چه خواهد بود. به طور خاص، هیچ راهی نیست که بتوان فهمید که آیا تابعی به نام sendClear دارد یا نه.

برای این که مشکل را عمیق‌تر بررسی کنیم، فرض کنید که ما یک کلاس به نام CompanyZ داریم که اصرار دارد که ارتباطات به صورت رمز شده باشد:

```
class CompanyZ{          //this class offers no sendCleartext function
public:
   void sendEncrypted(const std::string& msg);
};
```

حالت کلی MsgSender برای CompanyZ مناسب نیست، چرا که این template یک تابع به نام sendClear نیز دارد که در مورد اشیاء از نوع CompanyZ این ایراد دارد. برای اصلاح کردن این ایراد، می‌توانیم یک ورژن خاص از MsgSender برای CompanyZ بنویسیم:

```
template <>                          // a total specialization of MsgSender; the same as the
class MsgSender<CompanyZ>{ //general template, except sendClear is omitted
public:
   void sendSecret(const MsgInfo& info);
};
```

توجه داشته باشید که

```
template <>
```

در ابتدای تعریف به این معنی است که این تعریف نه در مورد template و نه در مورد کلاس است. در عوض، آن یک ورژن خاص از MsgSender است که به عنوان آرگومان template مربوط به CompanyZ استفاده می‌شود. ما این را به عنوان یک شخصی‌سازی template می‌شناسیم: MsgSender برای نوع CompanyZ شخصی‌سازی شده است. در واقع وقتی CompanyZ به عنوان پارامتر شناسایی شود، هیچکدام از سایر پارامترهای template قابل تغییر نیستند.

با توجه به این که MsgSender برای CompanyZ شخصی‌سازی شده، دوباره کلاس مشتق شده‌ی LoggingMsgSender را دوباره با هم ببینیم:

```
template <typename Company>
class LoggingMsgSender : public MsgSender<Company>
{
public:
   void sendClearMsg(const MsgInfo& info)
   {
      //write "before sending" info to the log
      sendClear(info);    // if Company == CompanyZ, this function doesn't exist!
      //write "after  sending" info to the log
   }
};
```

همچنانکه که comment اشاره می‌کند، این کد وقتی که کلاس پایه MsgSender<CompanyZ> باشد ، درست نیست چرا که این کلاس هیچ تابعی به نام sendClear ندارد. به همین دلیل C++ این فراخوانی را رد می‌کند: C++ میفهمد که template کلاس پایه ممکن است که شخصی سازی شده باشد و چنین شخصی سازی‌ای ممکن است که interface یکسانی با template عادی نداشته باشد. در نتیجه، C++ به طور کلی نگاه‌کردن به کلاس پایه template شده، را رد می کند. در برخی موارد، وقتی ما از شیءگرایی C++ به template C++ می‌رویم(آیتم یک را ببینید)، وراثت ممکن است که از کار کردن بیفتد.

برای این که کاری کنیم که این کد کار کند، ما باید به طریقی این رفتار C++ را که کلاس‌های پایه template شده را نگاه نمی‌کند را از کار بیندازیم. سه روش برای انجام این کار وجود دارد. اولی این است که فراخوانی به کلاس پایه را با استفاده از

```
this→
```

انجام دهیم.

```
template <typename Company>
class LoggingMsgSender : public MsgSender<Company>
{
public:
  void sendClearMsg(const MsgInfo& info)
  {
    //write "before sending" info to the log
    this->sendClear(info); //okay, assumes that sendClear will be inherited
    //write "after  sending" info to the log
  }
};
```

دومین روش، می‌توانید از اعلان using استفاده کنید، روشی که شما باید با توجه به این که آیتم ۳۳ را خوانده‌اید، با آن آشنا باشید. در این آیتم دیدیم که چطور این اعلان می‌تواند نام‌های مخفی کلاس پایه در ناحیه دید کلاس مشتق شده قرار میگیرد. بنابراین می‌توانیم sendClearMsg را به صورت زیر بنویسیم.

```
template <typename Company>
class LoggingMsgSender : public MsgSender<Company>
{
public:
  using MsgSender<Company>::sendClear;      //tell compilers to assume
                                            //that sendClear is in the base class
```

```
    void sendClearMsg(const MsgInfo& info)
    {
      //write "before sending" info to the log
      sendClear(info); //okay, assumes that sendClear will be inherited
      //write "after  sending" info to the log
    }
};
```

با وجود این که استفاده از اعلان using هم در اینجا و هم در آیتم ۳۳ کار کرد، ولی روش حل مشکل در این دو با هم متفاوت است. در اینجا، مساله این نبود که نام‌های کلاس پایه در کلاس مشتق شده مخفی شده بود، بلکه مشکل این بود که کامپایلر در ناحیه‌ی دید کلاس پلیه جستجویی انجام نمی‌دهد مگر این که شما این را بگویید.

آخرین راه این است که کدتان را به گونه‌ای کامپایل کنید که در آن صریحا تابع در داخل کلاس پایه فراخوانی شده باشد.

```
template <typename Company>
class LoggingMsgSender : public MsgSender<Company>
{
public:
                                  //that sendClear is in the base class
    void sendClearMsg(const MsgInfo& info)
    {
      MsgSender<Company>::sendClear(info); //okay, assumes that sendClear will be
inherited
    }
};
```

این راهی است که ما کمترین اشتیاق را برای حل کردن مشکلمان با آن داریم، چرا که اگر تابع به صورت virtual فراخوانی شده باشد، این فراخوانی صریح منجر به از بین رفتن رفتار virtual binding خواهد شد.

از نقطه نظر ، قابل رویت بودن نام‌ها، هر کدام از این روش‌ها یک کار یکسان را انجام می‌دهند: آن‌ها به کامپایلر می‌گویند که هر شخصی‌سازی از کلاس پایه template با استفاده از template کلی از interface پشتیبانی می‌کند. این موضوع وقتی که کلاس پایه در حالت تجزیه شدن است برای همه‌ی کامپایلرها مورد نیاز است.