Table of Contents

Item 9: Never call virtual functions during construction or destruction

نباید توابع **virtual** رو طی **construction** و یا **destruction** فراخوانی کنیم، چرا که نمیشه فهمید منظور شما از این فراخوانی چی بوده، و حتی اگر این کارو بتونه انجام بده، باز هم از نتیجه ناراضی خواهیم بود.

فرض کنید که شما یک مدل برای تراکنش‌های انبارداری نوشتید، یعنی سفارشات خرید، سفارشات فروش و غیره. این خیلی مهمه که چنین تراکنشاتی قابل حسابرسی باشه، بنابراین هر زمان که یک شیء تراکنش ایجاد میشه، یک **log** مناسب باید برای حسابرسی ساخته بشه. در این صورت یک روش معقول برای انجام این کار کد زیر است:

```
class Transaction
{
public:
   Transaction();
   virtual void logTransaction() const=0;  //make type-dependent log entry
};
Transaction::Transaction()
{
   logTransaction();
}
class BuyTransaction: public Transaction{   //derived class
public:
   virtual void logTransaction() const;  //how to log transactions of this type
};
class SellTransaction: public Transaction{   //derived class
public:
   virtual void logTranstion() const;   //how to log transactions of this type
};
```

ببینید چه اتفاقی میفتد وقتی کد زیر رو اجرا می‌کنید.

```
BuyTransaction b;
```

مشخصا سازنده‌ی کلاس **BuyTransaction** در این مورد فراخوانی خواهد شد، اما قبل از اون، باید سازنده کلاس **Transaction** صدا زده شود. قاعده‌ی کلی به این صورت است که قسمت‌های **base class**

مربوط به کلاس‌های مشتق شده زودتر فراخوانی می‌شوند. خط آخر از سازنده کلاس **Transaction** منجر به فراخوانی ***logTransaction*** خواهد شد که **virtual** هست، اینجا جایی است که ممکنه در موردش تعجب کنید.

ورژن **logTransaction** ای که فراخوانی می‌شود مربوط به کلاس **Transaction** می‌باشد نه ورژنی که در **BuyTransaction** هست(حتی اگه شیء از **Buytransaction** ساخته شده باشه). طی ایجاد کلاس **base** ، توابع **virtual** هرگز وارد کلاس‌های مشتق شده نمی‌شوند. به جای این کار، شیء به نحوی رفتار می‌کند که انگار همان کلاس **base** هست. به بیان ساده‌تر، وقتی که کلاس **base** در حال ایجاد و ساخته شدن هست، توابع **virtual** وجود ندارند.

دلیل این رفتار هم خیلی منطقی به نظر میرسه چون سازنده کلاس **base** قبل از سازنده کلاس‌های مشتق شده اجرا می‌شود، در این صورت وقتی که سازنده‌ی کلاس **base** در حال اجرا هست داده‌های عضو مربوط به کلاس‌های مشتق شده هنوز مقداردهی اولیه نشده‌اند. اگر هنگام اجرای سازنده‌ی کلاس **base** یک تابع **virtual** اجرا بشه، در این صورت ما نمی‌توانیم وارد کلاس‌های مشتق شده بشیم، چون اگر این کار رو بکنیم داریم به اعضای داده‌ی کلاس نیز اشاره می‌کنیم در حالی که این اعضاء هنوز **initialized** نشده‌اند. اگر چنین اشتباهی را انجام بدید، ممکنه ساعت‌ها صرف دیباگ کردن کدتون بکنید. در این صورت، فراخوانی اعضای پایین دستی از یک شیء که هنوز **initialized** نشده‌اند عمل خیلی خطرناکیه، به همین خاطر **C++** راهی برای انجام دادن چنین کاری رو پیشنهاد نداده.

در واقع این موضوع بنیادی‌تر از این حرفاست. در طی ساختن کلاس **base** از یک کلاس مشتق شده، نوع شیء همان کلاس **base** هست. در واقع این نقطه نظر تنها از دیدگاه توابع **virtual** نیست، بلکه برخی از قسمت‌های زبان که از اطلاعات **runtime** استفاده می‌کنند مثل **dynamic_cast** (آیتم ۲۷) و **tipeid** شیء را از نوع کلاس **base** می‌بینند. در مورد مثال ما، وقتی که سازنده کلاس **Transaction** در حال اجرا برای **initialize** کردن قسمت **buytransaction** ، نوع شیء همان **Transaction** می‌باشد. این نحوه‌ی برخورد زبان **C++** با آن است که منطقی هم به نظر میرسد: چرا که هنوز قسمت‌هایی از **BuyTransaction** هنوز **initialize** نشده‌اند، پس امن‌ترین راه برای برخورد با چنین موردی این هست که فرض کنیم اصلا چنین چیزی وجود خارجی ندارد. یک شیء تا وقتی که سازنده‌ی کلاس مشتق شده اجرا نشود به کلاس مشتق شده تبدیل نمی‌شود و **base** کلاس محسوب می‌شود.

همین منطق در هنگام اجرای مخرب کلاس نیز پا برجاست. وقتی که مخرب یک کلاس مشتق شده اجرا می‌شود، عضو داده شیءای که از یک کلاس مشتق شده ساخته شده، تعریف نشده فرض می‌شوند، بنابراین **C++** به نحوی با آن‌ها برخورد می‌کند که انگار وجود ندارند. و به محض این‌که وارد مخرب کلاس **base** می‌شوند، شیء تبدیل به شیء کلاس **base** شده، و همه‌ی قسمت‌های مختلف زبان **C++** (توابع **virtual** و **dynamic_cast** و غیره) به همین نحو برخورد می‌کنند.

در مثال بالا، سازنده کلاس **Transaction** یک فراخوانی مستقیم به تابع **virtual** خواهد داشت، که مشاهده‌ی اشتباهی که انجام شده در این مثال نیز ساده هست، برخی از کامپایلرها در این مورد به ما یک هشدار میدهند(و برخی دیگر این هشدار را نمیدهند، آیتم ۵۳ رو برای این موضوع ببینید). حتی اگه چنین خطایی را نمیگیرفتیم، مشکل قبل از **runtime** نیز قطعی است، چرا که تابع **logTransaction** یک تابع **pure** هست. مگر این که تعریفش کرده باشیم(بله می‌تونیم حتی چنین کاری رو نیز انجام بدهیم آیتم ۳۴ رو ببینید)، در این صورت برنامه نمیتونه **link** بشه، چون که لینکر نمی‌تونه پیاده سازی مناسب رو برای **Transaction::logTransaction** پیدا کنه.

در برخی مواقع تشخیص فراخوانی به توابع **virtual** هنگام تخریب و یا اجرای سازنده چندان هم ساده نیست. اگر **Transaction** چندین سازنده داشته باشد، که هر کدام میبایست یک کاری رو انجام بدهند، به لحاظ طراحی بهتره کد رو طوری بنویسیم که از دوباره کاری توی کد بپرهیزیم، این کار رو می‌تونیم با کد یکسان **initialization** و موارد دیگر مرتفع کنیم، فرض کنید برای این مورد یک تابع به نام **init** نوشته باشیم:

```
class Transaction
{
public:
   Transaction()
   {
      init();    //call to non-virtual
   }
   virtual void logTransaction() const=0;
private:
   void init()
   {
      logTransaction();   //that calls a virtual
   }
};
```

این کد به لحاظ مفهومی مشابه همون کد اول هست، ولی یه مقدار پیچیده تره، چون معمولا کامپایل میشه و بدون مشکلی **link** میشه. در این مورد، چون **logTransaction** یک **pure virtual** در کلاس **Transaction** هست، بیشتر **runtime system** ها برنامه را هنگام فراخوانی **pure virtual** متوقف و یا **abort** خواهند کرد. خطایی که در سیستم عامل **linux** به من داده به این صورت هست.

```
pure virtual method called
terminate called without an active exception
```

حالا اگر **logTransaction** یک تابع **virtual** عادی بود(یعنی **pure** نبود) که یک پیاده سازی هم در **Transaction** داشت، این ورژن فراخوانی میشد.(در کامپایلرهای جدید اجازه‌ی چنین کامپایلی به ما داده نمیشه). تنها راه برای حل این مشکل این هست که اطمینان پیدا کنیم که هیچکدام از سازنده‌ها و مخرب‌های شما نمی‌توانند توابع **virtual** رو فراخوانی بکنند.

ولی چطور می‌تونید اطمینان حاصل کنید که ورژن درستی از **logTransaction** در موقع ساخت شیء فراخوانی شده است یا نه؟ واضح است که فراخوانی یک تابع **virtual** بر روی شیء از سازنده‌ی کلاس **Transaction** یک راه اشتباه برای انجام این کار است.

راه‌های متفاوتی برای مرتفع کردن این مشکل وجود دارد. یک راه تبدیل **logTransaction** به یک تابع **non-virtual** در **Transaction** هست، سپس نیازه که سازنده‌ی کلاس‌های مشتق شده اطلاعات لازم **log** رو برای سازنده‌ی **Transaction** بفرستند. این تابع می‌تواند به صورت امنی **non-virtual** **logTransaction** رو فراخوانی کنه. مثل این مورد:

```
class Transaction
{
public:
   explicit Transaction(const std::string& logInfo);
   void logTransaction(const std::string& logInfo) const; //now a non-virtual function

};
Transaction::Transaction(const std::string& logInfo)
{
   logTransaction(logInfo);
}
class BuyTransaction: public Transaction
{
public:
   BuyTransaction(parameters)
      :Transaction(createLogString(parameters))
private:
   static std::string createLogString(parameters);
};
```

به عبارت دیگر، از اونجایی که شما نمی‌تونید از توابع **virtual** پایین دستی در کلاس **base** استفاده کنید(در هنگام ساختن کلاس **base**)، در این صورت می‌تونید این مورد رو این طوری جبران کنید که اطلاعات لازم رو از کلاس‌های مشتق شده به کلاس **base** بفرستید.

در این مثال، به استفاده از تابع **createLogString** که به صورت **private static** هست توجه کنید. استفاده از یک تابع کمکی برای ساخت و فرستادن مقدار به کلاس سازنده معمولا راه حل مناسب‌تری است. با **static** کردن تابع، دیگر خطر اشاره به اشیاء **BuyTransaction** که هنوز **initalize** نشده‌اند وجود ندارد. این خیلی مهم است، به خاطر این واقعیت که این اعضاء داده‌ای در حالت **undefined** قرار دارند، و این همان دلیلی است که چرا فراخوانی یک تابع **virtual** در سازنده‌ی کلاس **base** نمی‌تواند وارد کلاس‌های مشتق شده شود.