Virtual Member Functions

- Virtual member functions are used to adjust the behavior of objects but keeping their interface stable. ➡ Separation of interface and implementation
- To override a member function, is must be declared virtual. For documentation purpose, the overriding member function should also be declared as virtual.

```
struct Account{
  virtual void withdraw(double amt){ balance -= amt; }
...

struct BankAccount: Account{
  void withdraw(double amt) override {
    if ((balance - amt) > 0.0) balance -= amt;
...
```

Virtual Member Functions

- Which version of a virtual member function is called depends on the dynamic type of the object. To apply virtuality an indirection such as a pointer or a reference is required.

```
BankAccount bankAccount(100.0);

Account* aPtr = &bankAccount;
aPtr->withdraw(50.0);

Account& aRef = bankAccount;
aRef.withdraw(50.0);
```



Polymorphie:

- The run-time decides which member function should be used.
- Is often called dynamic or late binding.

Virtual Member Functions

- Rules:
  1. Constructor cannot be virtual.
  2. Virtual member functions do not have to be overriden.
  3. When you declare a member function virtual it stays virtual in the hierarchy.
  4. The parameters, return type, or constness in the overriding member function must be identical to the parameters in the virtual member function.
  5. When you don't use the specifier virtual, you will not get polymorphic behavior.
  6. You can override a private member function in the base.



Distinguish between override and overload.

`memberFunctionVirtual.cpp`

Virtual Destructors

- When you destruct an object using its interface (base class pointer or reference), the destructor must be virtual.

```
Account* aptr;

Account* aptr = new BankAccount(100.0);
delete aptr;

Account& aRef = BankAccount(200.0);
```