gMock
(GKxB_INTM006)

Dr. Hatwagner F. Miklós

Széchenyi István Egyetem, Győr

https://github.com/wajzy/GKxB_INTM006.git

2023. november 2.

Különféle tevékenységek céljai:

Egységtesztelés
: nyilvános interfészen elérhető szolgáltatások ellenőrzése bemenet-elvárt kimenet párokkal

Mockolás (kb. *utánzás*)
: objektumok közötti interfészek működésének ellenőrzése (gyakran még nem teljeskörűen implementált objektumokkal, ld. TDD):

- Milyen metódusokat kell hívni?
- Milyen paraméterekkel?
- Mit kell visszaadniuk?
- Milyen sorrendben?
- Hányszor?

Tesztelési célból egy valódi objektumot helyettesítő objektum (test double) lehet:

Dummy
: Soha nem használt objektum, jellemzően csak azért hozzák létre, hogy pl. egy metódusnak formálisan átadhassák. Pl. üres `Person` osztály.

Stub
: Előre bekészített válaszokat ad a tesztek során feltett kérdésekre, másra nem képes. Pl. a `Person` osztály példányai mindig ugyanazt a vezetéknevet adják vissza.

Spy
: Olyan *stub*, ami rögzíti a hívás módját, mennyiségét. Pl. olyan osztály, ami rögzíti, melyik metódusát hányszor hívták.

Mock
: A hívásokkal kapcsolatban beprogramozott, specifikáció szerinti elvárásokat támaszt (pl. mennyiség, paraméterezés, sorrend), valamennyi tervezett szolgáltatással kapcsolatban. Pl. a személyes adatok kiírásához valamennyi *getter* metódust hívni kell.

Fake
: Teljesen funkcionális kód, de bizonyos egyszerűsítésekkel, amik nem teszik alkalmassá a termelésben történő használatra. Pl. in-memory adatbázisba menti a személyek adatait a valódi helyett.

gMock

C++-ban mockolni nehéz:

- Nehéz a mock osztályokat elkészíteni, és sok a hibalehetőség.
- Minőségük változó
- Az egyik mock használatával szerzett tapasztalat nehezen alkalmazható a továbbiakra.

jMock, EasyMock $\rightarrow$ gMock

Mikor vehetjük hasznák a gMock-nak?

- Prototípusok készítése hatékonyabb megoldások keresésére; C++-ban ezt nem lehet elég gyorsan kivitelezni.
- A tesztek lassúak, mert pl. sok könyvtártól függenek, vagy drága erőforrásokat (pl. adatbázis) használnak.
- A tesztek törékenyek, mert bizonyos erőforrások megbízhatatlanok (pl. hálózat).
- Le szeretnénk ellenőrizni egy forgatókönyv esetén a szoftver viselkedését, de nem könnyű ilyen helyzetet előidézni.
- A modulok közötti kommunikációt szeretné megfigyelni.
- Különféle függőségek viselkedését kellene utánozni, de ezek „kézi” megvalósítása nehézkes.

Mire lehet használni a gMock-ot?

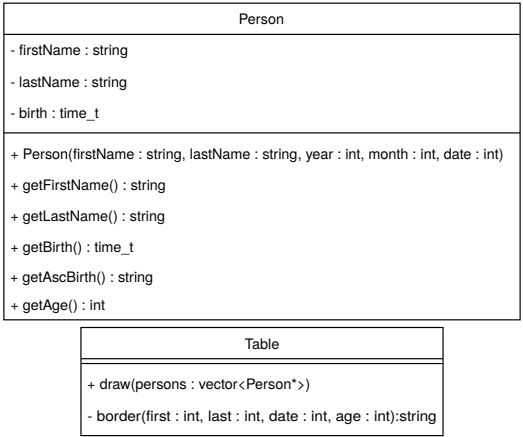
Tervezési segédeszközként

Könnyen, gyakran lehet kísérletezni az interfészek terveivel.

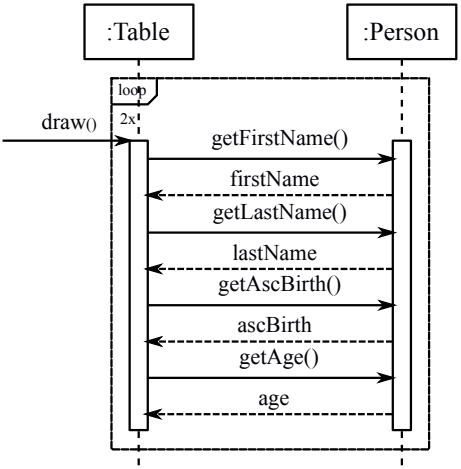
Tesztelési segédeszközként

Tesztek külső függőségeinek csökkentésére, modulok közti együttműködés tesztelésére.

Készítsünk egy osztályt, mely táblázatosan megjeleníti személyek adatait!



Hogyan működnek együtt az objektumok?



15/Person.h

```
 8  class Person {
 9      public:
10          Person(std::string firstName,
11              std::string lastName,
12              int year, int month, int date) {
13              this->firstName = firstName;
14              this->lastName = lastName;
15              struct tm birthTM = {0, 0, 0, date, month-1, year
                    ↪ -1900, 0, 0, -1};
16              this->birth = mktime(&birthTM);
17          }
```

15/Person.h

```
18          virtual std::string getFirstName();
19          virtual std::string getLastName();
20          virtual time_t getBirth();
21          virtual std::string getAscBirth();
22          virtual int getAge();
23          virtual ~Person() {}
24      protected:
25          std::string firstName;
26          std::string lastName;
27          time_t birth;
28  };
```

15/Person.cpp

```
11 time_t Person::getBirth() {
12     return this->birth;
13 }
14
15 std::string Person::getAscBirth() {
16     struct tm *birth = localtime(&this->birth);
17     return std::to_string(birth->tm_mday) + "/" + std::to_string(
           ↪ birth->tm_mon+1) + "/" + std::to_string(birth->tm_year
           ↪ +1900);
18 }
```

15/Person.cpp

```
20 int Person::getAge() {
21     struct tm birthTM = *localtime(&birth);
22     time_t now = time(nullptr);
23     struct tm nowTM = *localtime(&now);
24
25     int age = nowTM.tm_year - birthTM.tm_year;
26     if(nowTM.tm_mon < birthTM.tm_mon) {
27         age--;
28     } else if(nowTM.tm_mon==birthTM.tm_mon && nowTM.tm_mday<
       ↪ birthTM.tm_mday) {
29         age--;
30     }
31     return age;
32 }
```

15/Table.h

```
10 class Table {
11     public:
12         virtual void draw(std::vector<Person*> persons);
13     private:
14         virtual std::string border(int first, int last, int
               ↪ date, int age);
15 };
```

15/Table.cpp

```
3  void Table::draw(std::vector<Person*> persons) {
4      int firstW=0, lastW=0, dateW=0, ageW=0;
5      int length;
6      for(auto p : persons) {
7          if((length = p->getFirstName().length()) > firstW) {
8              firstW = length;
9          }
10         if((length = p->getLastName().length()) > lastW) {
11             lastW = length;
12         }
13         if((length = p->getAscBirth().length()) > dateW) {
14             dateW = length;
15         }
16         if((length = std::to_string(p->getAge()).length()) > ageW) {
17             ageW = length;
18         }
19     }
```

15/Table.cpp

```
20      std::cout << border(firstW, lastW, dateW, ageW) << std::endl;
21      for(auto p : persons) {
22          std::cout << "| " << std::setw(firstW) << p->getFirstName()
23                    << " " << std::setw(lastW) << p->getLastName()
24                    << " | " << std::setw(dateW) << p->getAscBirth()
25                    << " | " << std::setw(ageW) << p->getAge()
26                    << " |" << std::endl;
27      }
28      std::cout << border(firstW, lastW, dateW, ageW) << std::endl;
29  }
```

15/Table.cpp

```
31 std::string Table::border(int first, int last, int date, int age) {
32     std::string s = "+";
33     s.append(first + last + 3, '-');
34     s += '+';
35     s.append(date + 2, '-');
36     s += '+';
37     s.append(age + 2, '-');
38     s += '+';
39     return s;
40 }
```

15/main.cpp

```
5  int main() {
6      std::vector<Person*> persons = {
7          new Person("Andras", "Arato", 1921, 3, 9),
8          new Person("Bela", "Bokor", 1978, 12, 2),
9          new Person("Cecilia", "Cudar", 2013, 5, 21)
10     };
11     Table t;
12     t.draw(persons);
13     for(auto p : persons) {
14         delete p;
15     }
16     return 0;
17 }
```

Kimenet

```
g++ -std=c++11 -o main Person.cpp Table.cpp main.cpp && ./main
```

```
+---------------+-----------+-----+
|  Andras Arato |  9/3/1921 | 101 |
|    Bela Bokor | 2/12/1978 |  43 |
| Cecilia Cudar | 21/5/2013 |   9 |
+---------------+-----------+-----+
```

Mock-olás előkészítése

- Mock osztály származtatása a *hívott* osztályból (`MockPerson`)
- Lehetőleg a virtuális függvényeit mock-oljuk (egyszerűbb)
- Legyen a destruktora is virtuális; a hívások számát az objektum törlésekor ellenőrzi a gMock
- A származtatott (mock) osztály *nyilvános* részében jelöljük meg a mock-olandó függvényeket (`MOCK_METHOD()`)
- A makró paraméterei a visszatérési érték típusa, a függvény neve, paraméterlistája.
- Negyedikként állhat `const` a konstans függvényeknél, és `override` a virtuállis függvények felüldefiniálásánál.

15/MockPerson.h

```
1  #include "gmock/gmock.h"
2  #include "Person.h"
3
4  class MockPerson : public Person {
5      public:
6          MockPerson(std::string firstName,
7              std::string lastName,
8              int year, int month, int date) : Person(firstName, lastName
                   ↪ , year, month, date) {}
9          MOCK_METHOD(std::string, getFirstName, (), (override));
10         MOCK_METHOD(std::string, getLastName, (), (override));
11         MOCK_METHOD(std::string, getAscBirth, (), (override));
12         MOCK_METHOD(int, getAge, (), (override));
13 };
```

Teszt osztály előkészítése

- Tegyük ismertté (`using`) a gMock névterében lévő, felhasználandó azonosítókat (pl. `Return`)!
- Hozzunk létre teszt eseteket a már ismert módon!
- Példányosítsuk a mock osztályt, majd definiáljuk az elvárásokat (`EXPECT_CALL`).
- Csak ezután kezdeményezzük a hívásokat! (Különben definiálatlan működés.)
- Szabadítsuk fel az objektumokat!

15/PersonTest.cpp

```
1  #include "MockPerson.h"
2  #include "Table.h"
3  #include "gmock/gmock.h"
4  #include "gtest/gtest.h"
5
6  using ::testing::Return;
7
8  TEST(PersonTest, getFirstName) {
9      MockPerson* andras = new MockPerson("Andras", "Arato", 1921, 3, 9);
10     std::vector<Person*> persons = { andras };
11     Table t;
12     EXPECT_CALL(*andras, getFirstName())
13         .Times(2)
14         .WillRepeatedly(Return("Andras"));
```

15/PersonTest.cpp

```
15      EXPECT_CALL(*andras, getLastName())
16          .Times(2)
17          .WillRepeatedly(Return("Arato"));
18      EXPECT_CALL(*andras, getAscBirth())
19          .Times(2)
20          .WillRepeatedly(Return("9/3/1921"));
21      EXPECT_CALL(*andras, getAge())
22          .Times(2)
23          .WillRepeatedly(Return(101));
24      t.draw(persons);
25      delete andras;
26  }
```

Konfiguráljuk a cmake-et!

15/CMakeLists.txt

```
19 add_executable(
20   person_mock
21   Table.cpp
22   Person.cpp
23   PersonTest.cpp
24 )
25 target_link_libraries(
26   person_mock
27   GTest::gtest_main
28   GTest::gmock_main
29 )
```

```
cmake -S . -B build
cmake --build build
cd build && ./person_mock
```

Kimenet

```
Running main() from /.../15/build/_deps/googletest-src/googletest/src/gtest_main.cc
[==========] Running 1 test from 1 test suite.
[----------] Global test environment set-up.
[----------] 1 test from PersonTest
[ RUN      ] PersonTest.getFirstName
+--------------+----------+-----+
| Andras Arato | 9/3/1921 | 101 |
+--------------+----------+-----+
[       OK ] PersonTest.getFirstName (0 ms)
[----------] 1 test from PersonTest (0 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test suite ran. (0 ms total)
[  PASSED  ] 1 test.
```

Hogyan értesülünk a hibákról?

15/PersonTest_hiba1.cpp

```
12      EXPECT_CALL(*andras, getFirstName())
13          .Times(1)
14          .WillRepeatedly(Return("Andras"));
```

Kimenet

```
Mock function called more times than expected - returning default value.
    Function call: getFirstName()
          Returns: ""
         Expected: to be called once
           Actual: called twice - over-saturated and active
```

A mock objektum nem *teszteli* a visszatérési értéket, hanem *injektálja* azt! → Az ős osztály állhat akár csupa pure virtual függvényből.

15/PersonTest_hiba2.cpp

```
21      EXPECT_CALL(*andras, getAge())
22          .Times(3)
23          .WillRepeatedly(Return(-1234));
24      EXPECT_EQ(101, andras->getAge());
```

Kimenet

```
Expected equality of these values:
  101
  andras->getAge()
    Which is: -1234
```

Elvárások megadásának általános formája

EXPECT_CALL(*mock_objektum*, *tagfüggvény(illesztők)*)
.Times(*számosság*)
.WillOnce(*tevékenység*)
.WillRepeatedly(*tevékenység*);

Az elvárások könnyen olvashatóak:

```
EXPECT_CALL(*andras, getAge())
    .Times(5)
    .WillOnce(99)
    .WillOnce(100)
    .WillRepeatedly(101);
```

Azaz azt várjuk, hogy 5× hívják majd az `andras` címen lévő objektum `getAge()` tagfüggvényét, ami először 99, aztán 100, majd minden további alkalommal a 101 értékkel tér vissza.

A hívott tagfüggvény paraméterének elvárt értéke

- `EXPECT_CALL(*andras, setFirstName("Bela"));` → pontosan ezt a paramétert kell kapnia
- `EXPECT_CALL(*andras, setFirstName(_));` → nem érdekel, *milyen értékű* paramétert kap
- `EXPECT_CALL(*andras, setFirstName);` → nem érdekel, *milyen értékű és hány darab* paramétert kap; a felültöltött változatok közül vajon melyik hívását várjuk el?

Az `_` a használható *illesztők* egyike. Néhány példa:

`Eq(érték)` egyezés vizsgálat

`Ge(érték)` a paraméter nagyobb, vagy egyenlő mint az *érték*

`IsTrue()` a paraméter kifejezés értéke *igaz*

Akár saját illesztők is készíthetők.

Hívások elvárt mennyiségének kifejezése

`Times(n)` pontosan n ismétlés

`Times(0)` speciálisan: egyszer sem szabad hívni a tagfüggvényt

`Times(AtLeast(n))` legalább n ismétlés; további lehetőségek

Az elvárt hívási mennyiséget a `WillOnce()` és `WillRepeatedly()` hívások számából is kitalálhatja:

- ha egyetlen ilyen záradék sem fordul elő → `Times(1)`
- egymást követi n `WillOnce()`, de nincs egyetlen `WillRepeatedly()` sem → `Times(`n`)`
- egymást követi n `WillOnce()` és egyetlen `WillRepeatedly()` → `Times(AtLeast(`n`))`

Tevékenységek

- Ha nem mondunk mást (`Return()`), a mock fv. visszatérési értékének típusa beépített típus vagy mutató → *alapértelmezett tevékenység*, pl. `void` visszatér érték nélkül, `bool` mindig `false`-szal tér vissza, mások 0-val.
  C++11 és újabbak: ha van alapértelmezett konstruktor, akkor az általa előállított értéket szolgáltatják.
- Ha nincs alapértelmezett tevékenység, vagy nem megfelelő, felülírhatjuk azt (pl. `Return(-1234)`)
- Mi történik, ha a hívások száma nagyobb, mint a megadott tevékenységek száma? → alapértelmezett tevékenység
- Hogyan lehet referenciát visszaadni? → `ReturnRef(`*változó*`)`
- További lehetőségek

Minden `Return()`-be írt kifejezés egyszer lesz kiértékelve → mellékhatások elmaradnak!

Mindig 100-at ad vissza!

```
int n = 100;
EXPECT_CALL(*andras, getAge())
    .Times(4)
    .WillRepeatedly(Return(n++));
```

Ha szeretnénk mellékhatásokat, saját tevékenységet kell készíteni.

Elvárások illesztése fordított sorrendben (új szabály felülírja a régit)

Mi történik, ha az elvárások az alábbiak:

```
EXPECT_CALL(*andras, setFirstName(_));
EXPECT_CALL(*andras, setFirstName("Daniel"))
    .Times(2);
```

majd

- `setFirstName("Daniel")`-t 3× hívják? → hiba
- `setFirstName("Daniel")`-t 2×, majd `setFirstName("Endre")`-t 1×? → OK

Általános elvárások előre (pl. bármilyen paraméterrel, `Times(AnyNumber())` → váratlan hívások kerülése), speciálisak hátulra!

Az elvárt hívások tetszőleges sorrendben megtörténhetnek.

Hívások pontos sorrendjének előírása

```
TEST(PersonTest, pontosSorrend) {
    ...
    {
        InSequence sorrend;
        EXPECT_CALL(*andras, getFirstName());
        EXPECT_CALL(*andras, getLastName());
        EXPECT_CALL(*andras, getAscBirth());
        EXPECT_CALL(*andras, getAge());
    }
    t.draw(persons);
    ...
}
```

- Az `InSequence` objektum neve nem érdekes; a konstruktora teszi a dolgát.
- Részlegesen rendezett hívási sorrend is megadható.

Ragadós (sticky) elvárások

```
for (int i = n; i > 0; i--) {
    EXPECT_CALL(*andras, getFirstName())
        .WillOnce(Return("Andras" + to_string(i)));
}
```

Az illeszkedő elvárások aktívak maradnak → "Andras5", "Andras4", ... helyett a második hívás után hiba!

Megoldás

```
{
    InSequence sorrend;
    for (int i = n; i > 0; i--) {
        EXPECT_CALL(*andras, getFirstName())
            .WillOnce(Return("Andras" + to_string(i)))
            .RetiresOnSaturation();
    }
}
```

Érdektelen hívások

- Ha nem érdekes, mi történik egy függvénnyel → nem mondunk róla semmit
- Ha meghívják → figyelmeztető üzenet, de nem hiba (Naggy, kb. zsémbes)
- A viselkedés megváltoztatható