



Софийски университет
„Св. Климент Охридски“
Факултет по математика и
информатика

ЗАДАЧИ ЗА ВТОРО ДОМАШНО

курс Обектно-ориентирано програмиране
за специалности Компютърни науки и Информационни системи
летен семестър 2019/2020 г.

Общи изисквания

Решенията трябва да отговарят на следните изисквания и ограничения:

- Отделните елементи на системата да са представени с класове
- Следвайте добрите ООП практики за реализация на тези класове
- В реализираните класове трябва да са добавени подходящи методи, които са необходими за правилното решение на задачата, въпреки че може да не са явно посочени в условието.
- Решение, което не се компилира или грубо нарушава принципите на ООП, ще бъде оценено с 0 точки.
- В условието са посочени минималните характеристики за съответните типове, което означава, че те могат да бъдат разширени при необходимост.
- Демонстрирайте работата на програмата със смислени примери.

Задача

Група ентусиасти са решили да започнат свой бизнес. Особеното в тяхната идея е, че клиентите им няма да бъдат конкретни потребители, а ще се работи само с институции, на чиито служители ще бъдат предоставени въпросните услуги.

Това, от което се нуждаят ентусиастите е система, която поддържа следната функционалност:

- Основната структурна единица е **институцията**.
Институциите имат **уникален идентификатор**.
Идентификаторът е специален и носи и допълнителна информация за **общия брой** създадени институции **към момента**.

Институциите могат да бъдат два различни типа - **група** и **организация**.

- Групите **организируют** множество физически потребители, които не могат да са част от системата самостоятелно. Заради закона за личните данни, групите не съдържат списък от реални потребители, а могат само да разпознаят дали даден потребител е част от групата или не.
- Организацията представлява съвкупност от други (под)организации и/или групи.

Важна част от законодателството е, че всяка институция трябва да има застраховател.

Това е включено в изискванията за съответната система по следния начин:

- Потребителите **могат** да имат или не свой застраховател. Потребителите, които имат застраховател, притежават **номер** на застрахователната си полица (*insurance_id*).
Номерът е съобразен със застрахователя и служи за определяне принадлежността им към дадена група.
- Групата има **единствен** застраховател.
Застрахователят на една група покрива разходите на участниците в нея.
Всяка група има идентификатор (*group_id*), който е различен от идентификатора на институцията, и е специален за застрахователя, който е асоцииран с нея.
- Организациите могат да включват групи и (под)организации с различни застрахователи, но считаме, че имат един **основен**, който се определя по определени критерии.

За целите на системата да се реализират следните класове:

- **Person**, който описва потребител. Той се характеризира поне с:
 - **person_name** - име на потребител;
 - **insurance_id** - цяло число, което описва номер на застрахователната му полица.
- **Payer**, който описва застраховател. Той се характеризира с:
 - **payer_name** - име на застраховател;
 - **payer_member_rule** - *двуместна функция**, с която се описват правилата по съпоставяне на **груповия идентификатор** с този на **потребителската полица**. Функцията връща истина или лъжа в зависимост от това дали номерът на полицата е успешно съпоставен или не. Съпоставянето може да се извършва по различни критерии, които НЕ са предварително дефинирани.
** За реализацията могат да бъдат използвани различни изучени инструменти.*

Застрахователите не могат да се копират!

Считайте, че застрахователите ще бъдат създадени в началото на програмата.

- **Institution**, който представя гореспоменатата институция. Характеризира се с поне следното:
 - **institution_id** - уникален идентификатор;
 - **institution_name** - име на институцията;Да се реализат поне следните методи:
 - **has_member** - който по подаден потребител връща истина, ако той принадлежи към институцията;
 - **payer** - който връща указател към основния застраховател на институцията. Ако такъв няма, методът да връща празен указател;
 - **valid** - който казва дали институцията е валидна
- **Group**, който представя групата. Той се характеризира поне с:
 - **group_id** (unsigned int) - идентификатор на групата, базиран на асоциирания застраховател;
 - **group_payer** - застраховател;В сила са и следните особености:
 - Една група е валидна, **ако има** асоцииран застраховател;
 - За основен застраховател на групата считаме единствения асоцииран;

- Един потребител принадлежи към дадена група, ако номерът на застрахователната му полица може успешно да бъде съпоставен към идентификатора на групата (`group_id`) с помощта на правилата, определени от застрахователя ѝ.
- **Organization**, който представя организацията. Тя се характеризира с:
 - **`institutions_list`** - списък от съдържащи се институции;
 - **`organization_address`** - адрес, който представлява символен низ с променлива дължина.

В сила са и следните особености:

- Основният застраховател на организацията е този с най-много пряко асоциирани групи. При равенство се избира първия по ред;
- Казваме, че две организации са съвместими, ако основните им застрахователи са едни и същи;
- Организация без нито една група се счита за невалидна;
- Един потребител принадлежи към някоя организация, ако той принадлежи към някоя от прилежащите ѝ групи или подорганизации.

Да бъде реализиран поне метод **`void add_institution`**, който добавя институция в края на списъка си съгласно следното правило:

Невалидни институции не се добавят. Ако добавяме организация, то тя трябва да бъде съвместима.

Да се добави и възможността гореспоменатото правило да бъде отменяно и директно да се добавя институция при необходимост.

Забележки:

- **Имате пълна свобода при реализирането на потребителите в групата, както и списъка от институции за организацията.**
- **Уникалността на идентификатора за институциите да бъде запазена при работата с организациите и групите!**
- **Една група/организация може да има копие, но то да се счита за различна институция.**

За така изградената йерархия да се реализират следните външни функции:

- **`find_most_popular_institution`**([подходящи параметри]), която по списък от институции и списък от потребители връща институцията с най-много

асоциирани клиенти. В случай, че се открият две институции с еднакъв брой клиенти, да се върне първата.

- **clear_institutions**([подходящи параметри]), която при подаден списък от институции премахва невалидните като не променя текущия, а връща нов.

Пример:

Важно: Примерът е написан на псевдокод и е с цел да обясни работата на програмата, а не да опише коректно дефиниран тест.

Payer p1 = {name: "Bulstrad", rule: bulstrad_member_rule, other data...}; //По условие застрахователите се създават в началото на програмата.

Payer p2 = {name: "Alianz", rule: alianz_member_rule, other data...};

Group g1 = {group_id: 12345, other data...}; //g1 няма асоцииран застраховател;

Group g2 = {group_id: 878782, payer: p1, other data...}; //g2 има застраховател и това е p1;

Group g3 = {group_id: 5555, other data...}

Group g4 = {group_id: 19870, payer: p2, other data...}

Ogranization o1 = {institution_list: [g2,g3], address: "unknown address", other data...}; //Организацията o1 включва групите g1 и g3;

o1::payer -> p1; // p1 е асоцииран с g1, и няма друг застраховател;

Organization o2 = {institution_list:[], address: "another address", other data...}

o2 няма застраховател, понеже няма институции, които да обхваща.

Person p1 = {name: "John Smith", insurance_id: 87545, other data...};

Person p2 = {name: "Jane Smith", insurance_id: 13645, other data...};

Person p3 = {name: "Erica Smith", insurance_id: 32445, other data...};

Нека:

bulstrad_member_rule(g2.group_id, p1.insurance_id) -> true

bulstrad_member_rule(g2.group_id, p2.insurance_id) -> true

bulstrad_member_rule(g2.group_id, p3.insurance_id) -> true

alianz_member_rule(g4.group_id, p1.insurance_id) -> false

alianz_member_rule(g4.group_id, p2.insurance_id) -> false

alianz_member_rule(g4.group_id, p3.insurance_id) -> false

g2 обхваща p1, p2, p3;

g1 няма застраховател -> тя няма потребители. Същото важи за g3;
g4 също няма потребители, понеже p2 не разпознава нито един от създадените
потребители;

find_most_popular_institution([o1, o2, g2, g4], [p1, p2, p3]) -> o1, понеже тя съдържа
g2, към която принадлежат всички потребители
clear_institutions([o1, o2, g2, g4]) -> [o1, g2, g4]