Лабораторна робота № 4
Класи та інтерфейси

Мета роботи: розробити просту ієрархію класів для моделювання роботи комп’ютерного домену.

Теоретична частина

Нехай потрібно розробити деяку множину інтерфейсів та класів, які їх реалізують, для управління інформацією що до комп’ютерного домену.
На рис. 4.1 представлена спрощена схема успадкування.

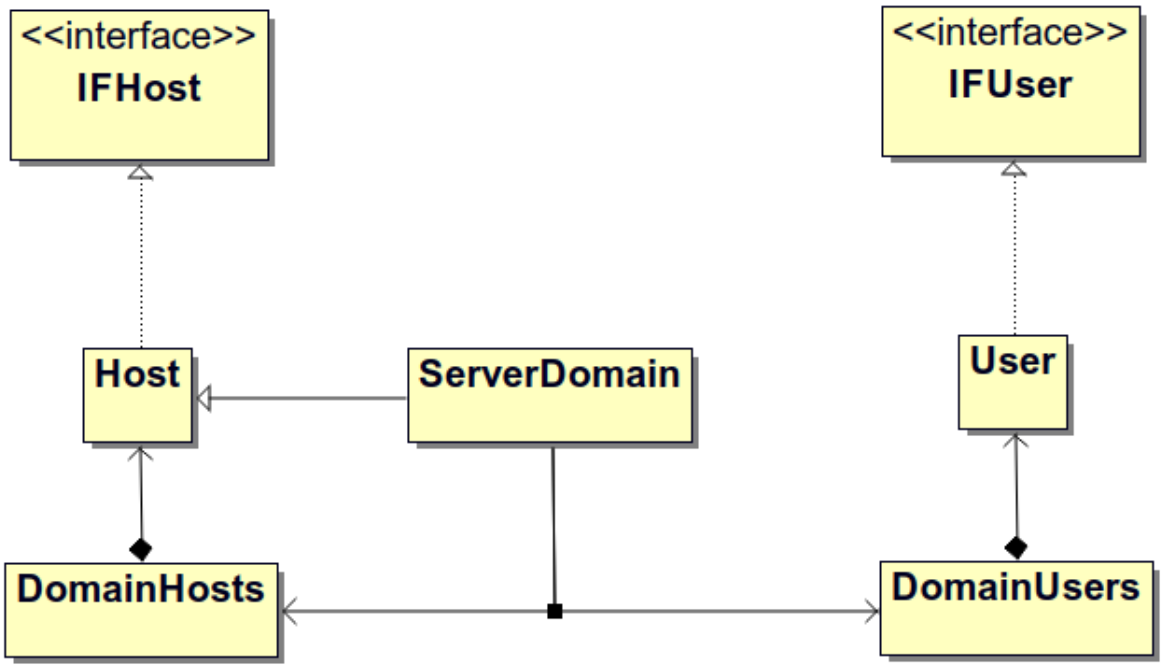

Рис. 4.1. Діаграма класів проекту

Інтерфейс *IFHost* обов’язково повинен описувати методи:
– призначення імені комп’ютера;
– призначення IP-адреси четвертої версії;
– виводу на консоль інформацію про ім’я та IP-адресу комп’ютеру;
– повернення імені комп’ютера;
– повернення IP-адреси.
Інтерфейс *IFUser* обов’язково повинен описувати методи:
– призначення імені користувача для входу в мережу;
– призначення повного імені користувача;
– призначення паролю користувача;
– задання часу входу в домен;
– повернення імені користувача для входу в домен;
– повернення повного імені користувача;

– повернення часу останнього входу в домен;

– перевірки співпадіння введеного паролю з існуючим паролем користувача.

Як бачимо з рис. 4.1, класи *Host* та *User* повинні реалізовувати відповідні інтерфейси.

Класи *DomainHosts* та *DomainUsers* агрегують відповідно динамічні списку комп'ютерів домену та користувачів. Для агрегації використовують, наприклад, клас *java.util.ArrayList*. В цих класах розміщуються методи, які спрощують управляння списками користувачів та комп'ютерів.

Основний клас *ServerDomain* виконує операції по управлінню користувачами та комп'ютерами домену – є спадкоємцем класу *Host*. Тобто, він містить всі необхідні методи, які дозволяють виконувати дії на рівні класів *Host* та *User*. Клас *ServerDomain* обов'язково повинен виконувати додавання та видалення користувачів та комп'ютерів домену, виводити назву домену (тобто інкапсулює її), виводити списки як комп'ютерів домену, так й користувачів, виводити інформацію по конкретним користувачам та комп'ютерам.

Для збереження дати та часу використовуйте відповідні класи (рис. 4.2).

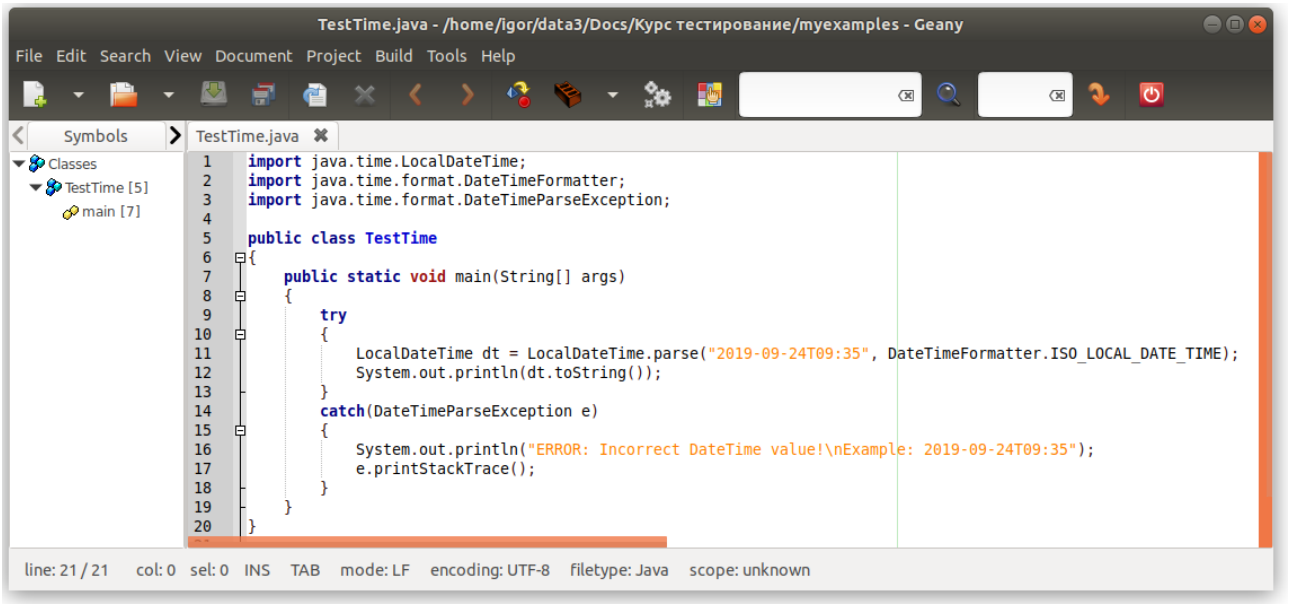

Рис. 4.2. Приклад збереження інформації про дату та час,
та її виводу на консоль

Зауважимо, що перед додаванням нового користувача, або комп'ютера в домен, повинні проводитися перевірки на присутність вже існуючих в домені подібних користувачів та комп'ютерів.

Наприклад:

```
. . .
public ArrayList<Host> hosts = new ArrayList<>();
. . .
public boolean containsHost(Host newHost)
{
   for(Host host : hosts)
```

```
   {
      if(
        host.getHostName().equalsIgnoreCase( newHost.getHostName() ) ||
        host.getHostAddressIPv4().equalsIgnoreCase( newHost.getHostAddressIPv4() )
      ) return true;
   }
   return false;
}
. . .
public boolean addHostToDomain(Host newHost)
{
   if( !containsHost(newHost) )
   {
      hosts.add(newHost);
      return true;
   }
   return false;
}
. . .
```

Таким чином зауважте, що хоча клас *ArrayList* вже містить метод перевірки входження в список певного об’єкту (стандартний метод *contains( )*), але в даному випадку потрібне складне порівняння, оскільки не повинно співпадати ні ім’я комп’ютеру, ні IP-адреса.

Практична частина

Використовуючи матеріал лекцій, в середовищі IntelliJ IDEA створити пакет *org.network.domain*, який буде містити java-модулі інтерфейсів та класів з рис. 4.1, враховуючи опис теоретичної частини.

Розробити окрему програму *TestDomain*, яка використовує клас *org.network.domain.ServerDomain* та виконує повне тестування роботи всіх розроблених методів, перевіряє вірне функціонування класу.

Якщо окрему програму створювати в середовищі IntelliJ IDEA, то можна не вказувати ім’я пакету при створені програми *TestDomain*, а після створення перейменувати як модуль *Main*, так й клас в ньому на *TestDomain*.

В звіті представити коди модулів з коментарями та результати тестування.