МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

# JAVA-ПРОГРАМУВАННЯ
## КОМПЮТЕРНИЙ ПРАКТИКУМ

*Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського*
*як навчальний посібник для студентів,*
*які навчаються за спеціальністю 122 «Комп'ютерні науки»,*
*освітньо-професійною програмою «Комп'ютерний моніторинг та геометричне моделювання процесів і систем»*

Київ
КПІ ім. Ігоря Сікорського
2021

Java-програмування: комп’ютерний практикум [Електронний ресурс] : навч. посіб. для студ. спеціальності 122 «Комп’ютерні науки», освітньо-професійної програми «Комп'ютерний моніторинг та геометричне моделювання процесів і систем» / КПІ ім. Ігоря Сікорського ; уклад.: Ю. А. Тарнавський. – Електронні текстові дані (1 файл: 686 Кбайт). – Київ : КПІ ім. Ігоря Сікорського, 2021. – 95 с.

*Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол № 7 від 13.05.2021 р.)*
*за поданням Вченої ради Інституту/Факультету (протокол № 10 від 29.03.2021 р.)*

Електронне мережне навчальне видання

# JAVA-ПРОГРАМУВАННЯ
## КОМПЮТЕРНИЙ ПРАКТИКУМ

Укладач: *Тарнавський Юрій Адамович*, канд. фіз.-мат. наук, доц.

Відповідальний редактор *Шаповалова С. І.,* канд. техн. наук, доц.

Рецензент *Новаківський Є. В.,* канд. техн. наук, доц.

*За редакцією укладача*

Посібник розроблений на підставі робочої програми кредитного модуля «Java-програмування» та призначений для організації та проведення комп’ютерного практикуму для студентів, які навчаються за спеціальністю 122 - «Комп’ютерні науки».

Спрямований на формування у студентів навиків програмування прикладних задач на мові Java засобами Intellij IDEA. Складається з п’яти практикумів базового рівня. Кожний практикум включає короткі теоретичні відомості, варіанти індивідуальних завдань, опис ходу виконання практикуму і контрольні питання. Важливою особливістю практикумів є наявність в кожному з них значної кількості варіантів індивідуальних завдань, достатньої для проведення занять в типовій академічній групі.

# ЗМІСТ

# ВСТУП

Мета вивчення дисципліни “Java-програмування” полягає в ознайомленні студентів з описом технологій Java і життєвого циклу продукту та вивчення мови програмування Java, використання різних конструкцій мови для створення Java додатків, використання умовних конструкцій, циклів і методів керування програмним потоком, реалізації технологій програмування Java і об’єктно-орієнтовані (ОО) концепції в Java-програмах.

Вивченя дисципліни забезпечує здатність програмувати з використання технологій Java для різних типів прикладних задач всіх технологічних рівнів систем автоматизованого управління.

Отримання глибоких навичок з програмування на Java передбачає виконання відповідного комп’ютерного практикуму.

В даному навчальному посібнику представлені п’ять практикумів базового рівня. Кожний практикум включає короткі теоретичні відомості, необхідні для виконання практикуму, варіанти індивідуальних завдань, опис ходу виконання практикуму і контрольні питання.

В теоретичних відомостях окрема увага приділена питанням ефективного використання такої популярної IDE, як Intellij IDEA, наведені покрокові інструкції виконання типових операцій.

Важливою особливістю практикумів є наявність в кожному з них значної кількості варіантів індивідуальних завдань (їх 24), достатньої для проведення занять в типовій академічній групі.

Опис ходу виконання практикуму містить покрокові інструкції, яких рекомендується дотримуватись, щоб досягти результат.

Контрольні запитання охоплюють весь матеріал, викладений в коротких теоретичних відомостях, і можуть використовуватись для оцінювання як готовності до виконання практикуму, так і ступеня засвоєння матеріалу - під час захисту роботи.

# ПРАКТИКУМ 1. УСТАНОВКА JAVA SE І КОМПІЛЯЦІЯ/ЗАПУСК JAVA-ПРОГРАМ В РЕЖИМІ КОМАНДНОГО РЯДКА

## Короткі теоретичні відомості по роботі з Java SE

### Установка пакета Java SE

Установочні файли Java SE можна завантажити з офіційного сайту компанії Oracle (http://oracle,com), пройшовши процедуру реєсстрації.

Для установки пакета необхідні два файли:

1. **jdk-8u271-windows-i586.exe** (або пізніша версія) - містить всі програмні засоби, необхідні для роботи з Java;
2. **jdk-8u271-doc-all.zip** - архівований файл, який містить документацію по пакету (англійською мовою).

Установка пакета починається з запуску програми **jdk-8u271-windows-i586.exe**.

В процесі установки рекомендується використовувати параметри, прийняті за замовчуванням. При виборі встановлюваних компонентів Java рекомендується не включати в установку вихідні тексти бібліотеки Java.

Після установки програмних засобів розархівуйте файл **jdk-8u271-doc-all.zip** в папку, куди був встановлений JDK, наприклад, C: \ Program Files \ Java \ jdk -8u271. В папці з JDK з'явиться папка **docs**.

### Робота з довідковою системою

Відкрийте в папці **docs** файл **index.html**, що містить короткий огляд JDK SE, а також посилання на документацію по Java.

У цьому файлі виберіть гіперпосилання Java SE API на специфікацію інтерфейсу прикладного програмування мови Java.

У лівому верхньому фреймі Web-сторінки виводиться пункт **All Classes** і найменування всіх пакетів мови Java (Рисунок 1.1). У разі вибору опції **All Classes** (він вибирається за замовчуванням при завантаженні Web-сторінки),

то в лівому нижньому фреймі виводяться найменування всіх класів і інтерфейсів всіх пакетів Java (найменування класів виводяться звичайним шрифтом, а інтерфейсів - курсивом). Якщо лівому верхньому фреймі вибрати один з пакетів, в лівому нижньому фреймі виводяться класи і інтерфейси тільки цього пакета.

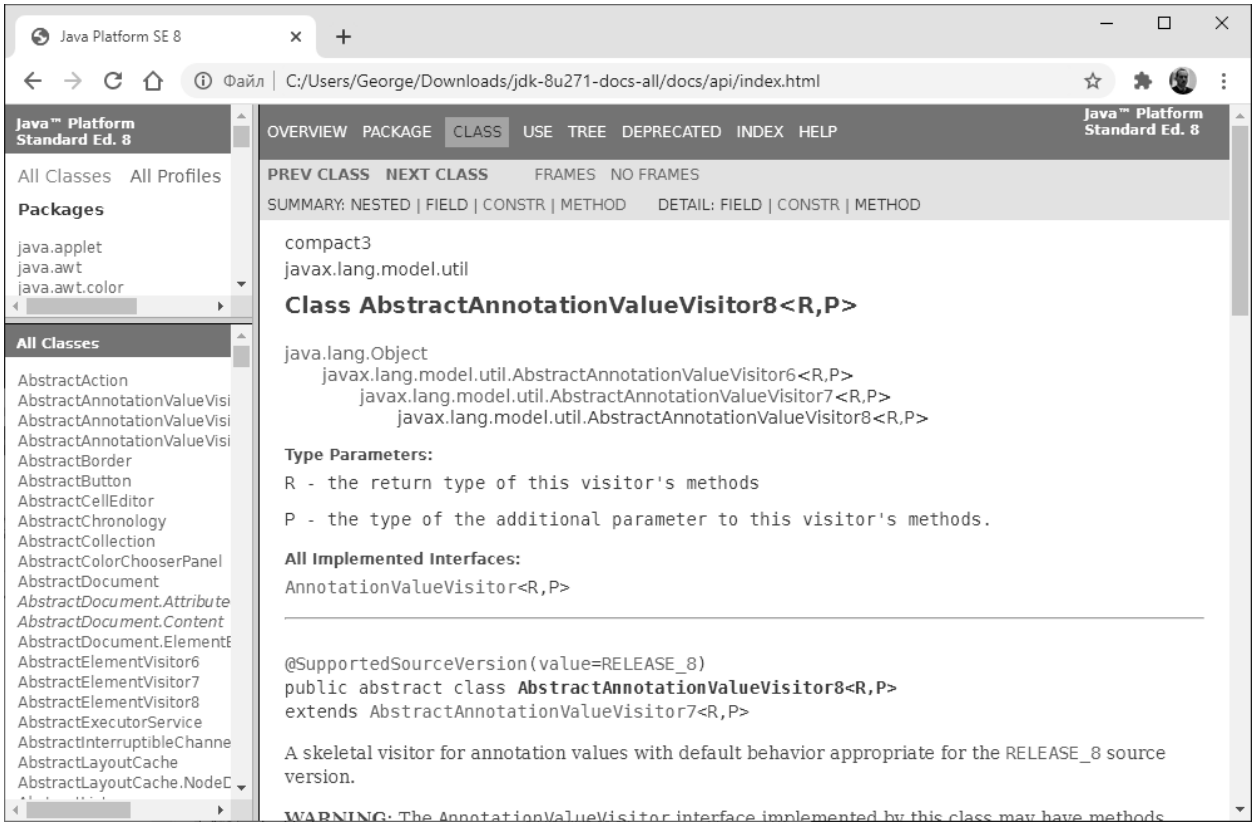


Рис.1.1. Вікно довідкової системи Java SE

Для отримання документації по компоненту в лівому нижньому фреймі необхідно вибрати ім'я цього компонента. Після цього в правому фреймі виводиться Web-сторінка з описом компонента, що містить: ім'я компонента; ієрархію його класів-предків; реалізовані в компоненті інтерфейси; список прямих нащадків даного компонента (якщо вони є); короткий опис компонента; список полів компонента; список полів, успадкованих від компонентів-предків; список конструкторів компонента; список методів компонента; список методів, успадкованих від компонентів-предків; більш докладні описи полів, конструкторів і методів компонента.

Щоб отримати більш докладний опис поля, конструктора або методу, треба в списку клацнути по його імені мишкою.

### Створення та редагування Java-програми

Програма на мові Java - це текстовий файл з розширенням .**java**, що містить опис одного або декількох класів [1-3]. Назва файлу повинна відповідати імені одного з класів в файлі, або співпадати з ім'ям єдиного класу в файлі. Якщо одн з класів файлу містить метод main (), в якості імені файлу має бути вибрано ім'я цього класу.

При програмуванні додатків на мові Java в режимі командного рядка створення і редагування програми можна виконати в будь-якому текстовому редакторі.

### Компіляція і запуск програми

Команда компілятора **javac.exe** знаходиться в папці **bin** каталогу, що містить JDK, і є програмою, що виконується в режимі командного рядка. Щоб відкомпілювати програму, треба зробити поточним папку з файлом програми, а потім набрати і виконати команду:

**javac** [*опції*] *ім'я-файла*.java

Якщо програма містить помилки, то компілятор видасть повідомлення з описом помилки і зазначенням номера рядка файлу, в якій міститься помилка. Після усунення помилок за допомогою редактора треба повторно скомпілювати програму.

При запуску компілятора **javac** можна вказати опції компілятора. Основними опціями компілятора є:

**-classpath** ***каталог-1, каталог-2, ...,*** в якому вказується ім'я або імена каталогів (із зазначенням шляху), з яких беруться файли байт-кодів, необхідних для компіляції програми (якщо опція не задана, передбачається використання бібліотеки Java і файлів байт кодів поточного каталогу);

**-d** ***каталог***, в якому вказується ім'я каталогу (із зазначенням шляху), куди будуть поміщені файли байт-кодів (якщо опція не задана, все створювані класи поміщаються в поточний каталог);

**-verbose** - для докладного виведення результатів компіляції.

Для запуску скомпілюваної програми (.class) треба виконати команду:

**java** [*опції*] *ім'я-файла*

**Приклад 1.** Компіляція програми *HelloWorld.java* з розміщенням скомпільованого файлу в поточному каталозі і запуску програми:

```
javac HelloWorld.java
java -classpath . HelloWorld
```

**Приклад 2.** Нехай в поточному каталозі створені каталоги *src* і *bin*, а вихідний файл програми *HelloWorld.java* розташований в *src*. Його компіляція з розміщенням скомпільованого файлу в поточному каталозі *bin* і запуск програми:

```
javac -d bin src/HelloWorld.java
java -classpath ./bin HelloWorld
```

**Варіанти завдань**

| № | Завдання | № | Завдання | № | Завдання | № | Завдання |
|---|---|---|---|---|---|---|---|
| **1** | 1,6,11,16 | **7** | 7,12,17,18 | **13** | 8,13,14,18 | **19** | 3,9,14,18 |
| **2** | 2,7,12,17 | **8** | 8,13,18,17 | **14** | 2,9,14,19 | **20** | 4,10,15,19 |
| **3** | 3,8,13,18 | **9** | 1,9,14,19 | **15** | 5,10,15,20 | **21** | 2,11,16,20 |
| **4** | 4,9,14,19 | **10** | 5,10,15,20 | **16** | 1,6,11,16 | **22** | 5,12,17,20 |
| **5** | 5,10,15,20 | **11** | 6,11,16,19 | **17** | 3,7,12,16 | **23** | 2,6,13,18 |
| **6** | 6,11,16,19 | **12** | 7,12,17,18 | **18** | 4,8,13,17 | **24** | 5,7,14,19 |

Введення даних від користувача має забезпечуватися за допомогою аргументів командного рядка.

1. Задано 3 цілих числа. Знайти їх середнє арифметичне і найближче до нього ціле.
2. Знайдіть суму цифр тризначного числа, його квадратний корінь і найближче до кореня ціле.
3. У двозначному цілому числі поміняйте цифри місцями, знайдіть його квадратний корінь і найближче до кореня ціле.
4. Визначте число, отримане виписуванням в зворотному порядку цифр заданого тризначного числа. Знайдіть його натуральний логарифм.
5. Нехай дано цілі довжини сторін трикутника. Знайдіть його площу і напівпериметр.
6. Дано 2 цілі довжини сторін трикутника і цілий кут між ними в градусах. Знайдіть площу.
7. Дана площа квадрата. Знайти сторону даного квадрата і мінімальну цілу сторону квадрата, в який вихідний квадрат може бути вписаний.
8. Дано ціле число A. Обчислити $A^8$, використовуючи допоміжну змінну і три операції множення. Для цього послідовно знайдіть $A^2$, $A^4$, $A^8$. Вивести всі знайдені степені числа A.
9. Дано ціле число A. Обчислити $A^{15}$, використовуючи дві допоміжні змінні і п'ять операцій множення. Для цього послідовно знайдіть $A^2$, $A^3$, $A^5$, $A^{10}$, $A^{15}$. Вивести всі знайдені степені числа A.
10. Знайти корені квадратного рівняння $A \cdot x^2 + B \cdot x + C = 0$, заданого своїми цілими коефіцієнтами A, B, C.
11. Дано ціле число, знайти його факторіал.
12. Дано ціле тризначне число. Знайти суму і добуток його цифр.

13. Дано ціле тризначне число. У ньому закреслили першу праворуч цифру і приписали її зліва. Вивести отримане число і його квадратний корінь.
14. Дані катети прямокутного трикутника, знайти висоту і гіпотенузу.
15. Дана ціла сторона ромба і цілий (в градусах) кут при вершині. Знайти площу.
16. Дано сторони прямокутника, знайти його діагональ, периметр.
17. Дано 4 цілих числа. Знайти їх середнє арифметичне і найближче до нього ціле.
18. Квадратне рівняння задано дійсними коефіцієнтами A, B, C. Перевірити, чи має воно цілі корінні.
19. Дано 4 сторони і діагональ опуклого чотирикутника. Знайти його площу (розглянувши 2 трикутника).
20. У чотиризначному числі цифри циклічно зрушити на 1 позицію вправо. Знайти його натуральний логарифм.

## Хід виконання роботи

1. Завантажте та встановіть JDK.
2. В текстовому редакторі введіть наступний текст програми на мові Java:

```
public class TestJava {
        public static void main (String [] args) {
                System.out.println ("Hello, Java!");
        }
}
```

3. Збережіть програму у своїй (індивідуальній) папці у файлі з іменем **TestJava** .**java**.
4. Запустіть програму на компіляцію.

5. Запустіть скомпільовану програму на виконання і переконайтесь в її працездатності.
6. Використовуючи довідкову систему, перегляньте методи класу **Math** з пакету **java.lang** – ними доцільно користуватись при виконанні індивідуальних завдань.
7. Створіть в своєму індивідуальному каталозі підкаталоги **src/bin**.
8. Виконайте індивідуальні завдання, зберігаючи файли вихідних програми в підкаталозі **src**, а скомпільованих – в підкаталозі **bin**.
9. Оформіть звіт про виконання роботи.

## Контрольні запитання

1. Як здійснюється установка Java SE?
2. Яким мінімальним набором інструментів може обмежитись Java-розробник?
3. Як встановлюється довідкова система Java SE? Як вона організована?
4. Яке призначення класу **Math** з пакету **java.lang**?
5. Яке розширення мають файли програм на мові Java?
6. Як виконується компілювання програм на мові Java?
7. Як можна оцінити час компіляції Java-програми?
8. Яке розширення мають виконувані файли програм на Java?
9. Як здійснюється запуск Java-програми?

# ПРАКТИКУМ 2. РОЗРОБКА І ТЕСТУВАННЯ ПРОГРАМ В INTELLIJ IDEA

## Короткі теоретичні відомості по роботі в Intellij IDEA

### Установка IntelliJ IDEA

Перед установкою Intellij IDEA повинен бути встановлений пакет JDK.

Скопіювати інсталяційний пакет можна з сайту розробника компанії JetBrains за посиланням http://www.jetbrains.com/idea/.

Для установки програми треба виконати наступні кроки:

1. Запустити програму установки.

2. Прийняти умови ліцензійної угоди.

3. Вибрати, в якій папці встановити програму.

4. Якщо пакет JDK вже встановлено, програма установки сама його знайде і виведе у вікні розташування пакета.

Після розгортання IDE в зазначеній папці процес установки завершується.

### Основні відомості про Intellij IDEA

Intellij IDEA містить повний набір компонент: редактор, середовище компіляції і виконання, а також відладчик. Пакет працює не з програмами, а з проектами.

Проект - це група файлів програми і байт-кодів, а також установки, за допомогою яких виконується складання, виконання та налагодження цих файлів. Все програмування в Intellij IDEA, навіть якщо програма складається з одного файлу, виконується в рамках проекту. Якщо програма містить велику кількість коду, її рекомендується розбивати на кілька файлів (зазвичай в кожному файлі розміщують один клас, хоча це і не обов'язково).

Вікно Intellij IDEA має стандартний вид (Рис.2.1): зверху панель заголовка, меню і панель інструментів, знизу рядок стану, в середині вікно програми.

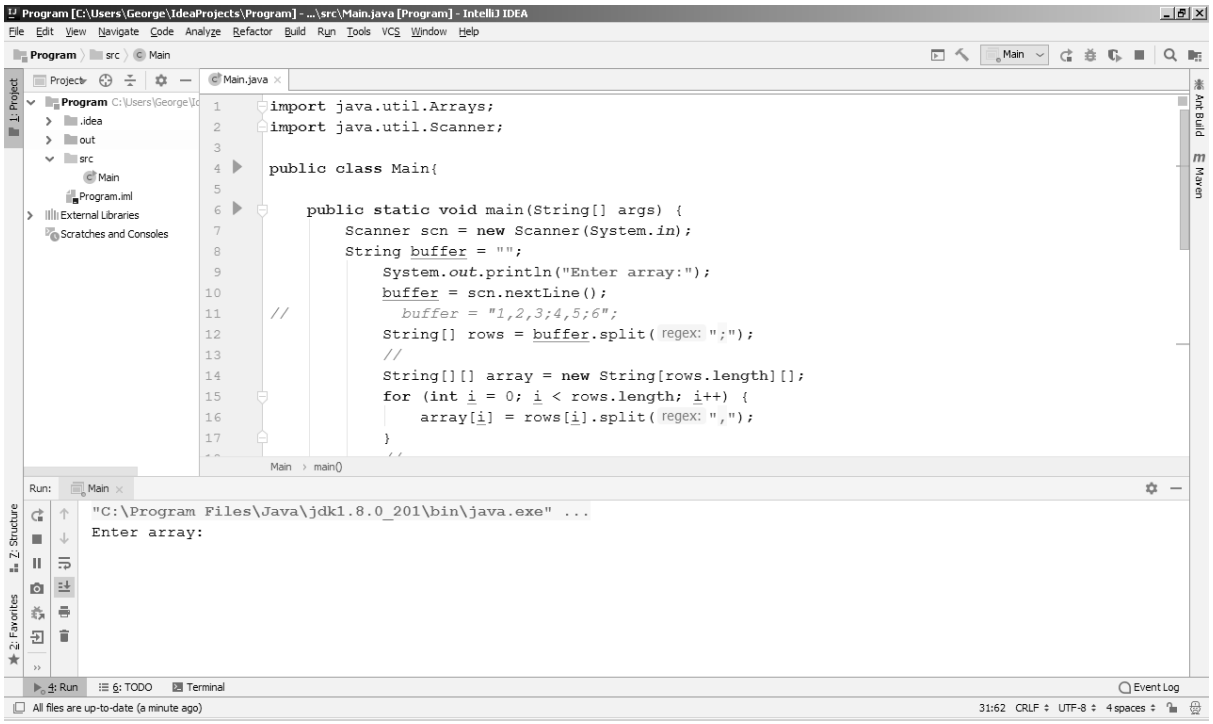


Рис.2.1. Головне вікно Intellij IDEA

Вікно додатка, в свою чергу складається з трьох основних вікон:

- вікно проектів (Projects);
- вікно редактора
- вікно виведення (Output).

У вікні проектів виводиться їх компоненти - класи, а також компоненти класів: поля, конструктори і методи. За допомогою контекстного меню додавати нові класи в проект, а також перейменовувати і видаляти існуючі класи та виконувати деякі інші операції.

Вікно редактора може містити кілька вкладок. Відкриття нової вкладки можна виконати або за допомогою команди **Open** ... меню **File**, або подвійним клацанням миші по імені класу в вікні проектів. Подвійний клік мишею по імені властивості, конструктора або методу підсвічує перший рядок з визначенням властивості, конструктора або методу. Клацання мишею по знаку "×" в найменуванні вкладки закриває вкладку.

Результати виконання програми виводяться у вікні виводу.

### Створення та відкриття проекту

Для створення проекту програми Java необхідно виконати наступні дії:

1. У меню **File** виберіть команду **New-Project**.
2. У вікні **New Project** виберіть тип проекту **Java** і натисніть кнопку **Next**.
3. У наступній вкладці вікна **New Project** виберіть шаблон проекту (не обов'язково) і натисніть кнопку **Next**.
4. У наступній вкладці вікна **New Project** в поле **Project Name** задайте ім'я проекту, в поле **Project Location** задайте ім'я папки, в якій буде знаходитися проект і натисніть кнопку **Finish**.

### Додавання файлу в проект

Якщо проект містить більше одного файлу, то додавання файлу в цей проект потрібно виконати наступні дії:

1. Виконати команду **New** в меню **File**.
2. Вибрати категорією файлу, наприклад, для нового класу - **Java Class**.
3. Після цього виводиться друге вікно запиту на створення нового файлу. Компоненти цього вікна залежать від категорії і типу файлу. У цьому вікні для нового класу досить задати ім'я нового класу (поле **Class Name**).

### Збирання і виконання проекту

Збиранням називається компіляція файлів вихідного проекту і створення файлів, що містять байт-коди класи проекту (це файли з розширенням .class). Засоби управління проектом компілюють тільки ті класи проекту, які були змінені під час останнього редагування.

Збирання проекту виконується або за допомогою команди **Build Project** в меню **Build**, або за допомогою команди **Build Project** в

контекстному меню проекту у вікні **Projects**, або при натисканні клавіші **Ctrl + F9**.

Результати збирання виводяться у вікні **Output**. Якщо файли проекту містять синтаксичні помилки, виводяться повідомлення про помилки.

Виконання проекту здійснюється або за допомогою команди **Run Main** в меню **Run**, або при натисканні клавіші **Shift + F10**.

## Створення unit-тестів в середовищі IntelliJ IDEA

### Організація тестування в JUnit

Unit-тестування - тестування окремих невеликих ділянок коду (юнітів) на відповідність очікуваному поведінки [5-7].

Для unit-тестування на мові Java може використовуватися бібліотека **JUnit**.

Тест в **JUnit** - звичайний метод. Якщо метод завершується успішно, то вважається, що тест пройдено. Тестові методи містяться в тестових класах.

При написанні тестового методу необхідно дотримуватися правила AAA:

- **Arrange** - підготовка до тесту (створення та налаштування тестових об'єктів);
- **Act** - безпосереднє тестування функції;
- **Assert** - перевірка очікуваного і реального результату роботи функції.

**Приклад 1**. Тестовий метод, побудований за правилом AAA.

```
public class MyClassTest {

   @Test
   public void testSquareIntMethod() throws Exception {
      // Arrange - подготовка к тесту
      MyClass myClass = new MyClass();
```

```
        // Act - тестирование функционала
        int result = myClass.squareInt(5);

        // Assert - проверка результата
        assertEquals(25, result);
    }
}
```

**Assert** - твердження про деякий стан об'єкта. Якщо твердження вірне - метод повертає void; якщо твердження невірне - метод викидає виключення.

### Види тверджень в JUnit

**fail** (String) - примусове викидання помилки.

**assertTrue** ([String message], boolean condition) - перевіряє, що логічна умова істинна.

**assertsEquals** ([String message], expected, actual) - перевіряє, що два значення збігаються.

**assertNull** ([String message], object) - перевіряє, що об'єкт є порожнім null.

**assertNotNull** ([String message], object) - перевіряє, що об'єкт не є порожнім null.

**assertSame** ([String message], expected, actual) - перевіряє, що обидві змінні відносяться до одного об'єкту.

**assertNotSame** ([String message], expected, actual) - перевіряє, що обидві змінні відносяться до різних об'єктів.

### Доступні анотації JUnit

Анотація @**Test** визначає що метод є тестовим.

Анотація @**Before** вказує на те, що метод буде виконуватися перед кожним тестованим методом @**Test**.

Анотація @**After** вказує на те що метод буде виконуватися після кожного тестованого методу @**Test**

Анотація @**BeforeClass** вказує на те, що метод буде виконуватися на початку всіх тестів, а точніше в момент запуску тестів (перед усіма тестами @**Test**).

Анотація @**AfterClass** вказує на те, що метод буде виконуватися після всіх тестів.

Анотація @**Ignore** каже, що метод буде проігнорований в момент проведення тестування.

Анотація @**Test** (**expected** = **Exception**.**class**) вказує на те, що в даному тестовому методі ви навмисно очікується Exception.

Анотація @**Test** (**timeout** = 1000) вказує, що тестований метод не повинен займати більше ніж 1 секунду.

### Порядок створення unit-тестів в IntelliJ IDEA

1. Створіть **Maven Project** в Intellij IDEA.
2. Переконайтеся, що в папку **src** проекту додана папка **test**.
3. В папці **src\main\java** розмістіть файли програм (* .java).
4. Відкрийте в редакторі клас, для якого будуть створюватися тести.
5. У контекстному меню виберіть команду **GoTo-Test** (**Ctrl** + **Shift** + **T**).
6. У вікні **Create Test** (Рис.2.2):
    a. Виберіть бібліотеку **JUnit5**, натисніть кнопку **Fix** (щоб додати JUnit в проект).
    b. Виберіть методи для тестування.
    c. Натисніть кнопку **ОК**.

    Відкриється шаблон тестового класу.
7. Внесіть необхідні зміни в шаблон тестового класу.
8. Розпочніть процес виконання тестів, вибравши в контекстному меню тестового класу команду **Run** <*ім'я проекту*>.

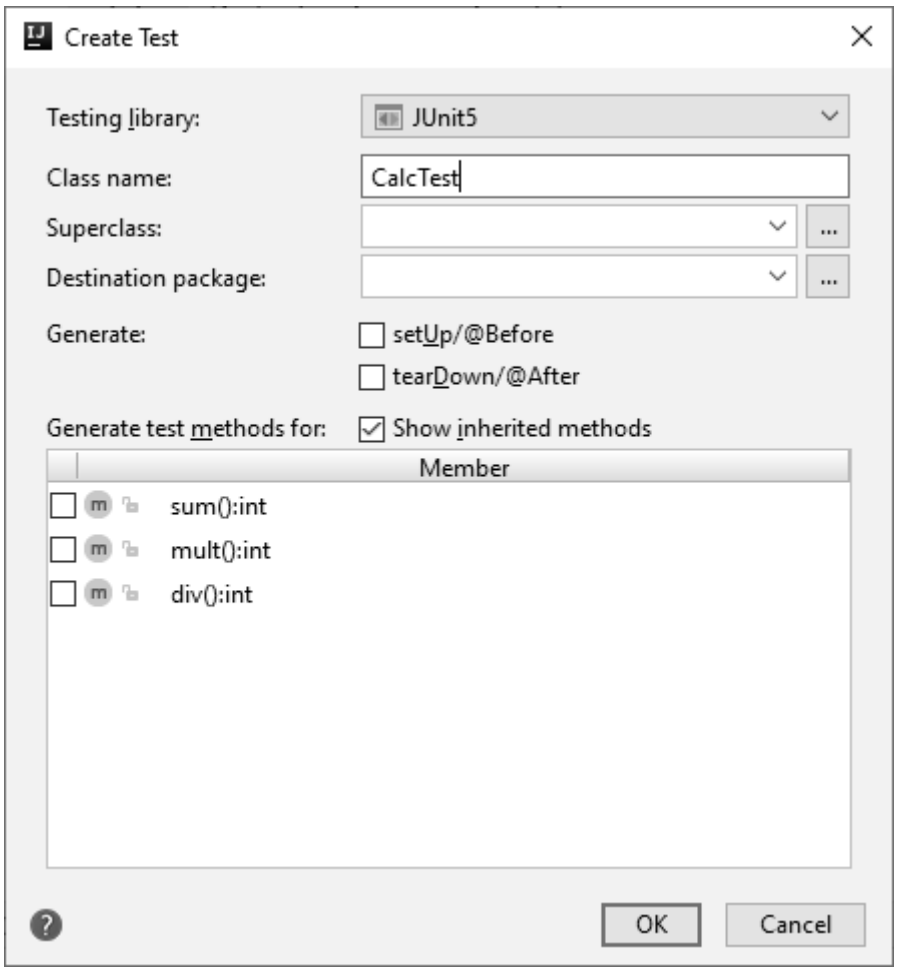


Рис.2.2. Діалог Create Test.

**Приклад 2**. Простий тестовий клас

```
import static org.junit.Assert.*;

class CalculatorTest {
   Calculator calc;
   int x,y;

   @org.junit.Before
   void setUp() {
      calc = new Calculator();
      x=-15;
      y=2;
```

```
    }

    @org.junit.Test
    void getSum() {
        int expected = calc.getSum(x,y);
        int actual = -13;
        assertEquals(expected,actual);
    }

    @org.junit.Test
    void getDivide() {
        int expected = calc.getDivide(x,y);
        int actual = -7;
        assertTrue(expected == actual);
    }

    @org.junit.Test
    void getMultiple() {
        int expected = calc.getMultiple(x,y);
        int actual = -7;
        assertFalse(expected == actual);
    }
}
```

**Варіанти завдань**

| № | Завдання | № | Завдання | № | Завдання | № | Завдання |
|---|---|---|---|---|---|---|---|
| **1** | 1,6,11,16,21 | **7** | 7,12,17,22,27 | 13 | 8,13,18,23,28 | 19 | 9,14,18,24,28 |
| **2** | 2,7,12,17,22 | **8** | 8,13,18,23,28 | 14 | 9,14,19,23,29 | 20 | 10,15,19,25,29 |
| **3** | 3,8,13,18,23 | **9** | 9,14,19,24,29 | 15 | 5,10,15,20,24 | 21 | 4,11,16,20,26 |
| **4** | 4,9,14,19,24 | **10** | 5,10,15,20,25 | 16 | 6,11,16,21,25 | 22 | 5,12,17,21,27 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **5** | 5,10,15,20,25 | **11** | 6,11,16,21,26 | 17 | 7,12,16,22,26 | 23 | 6,13,18,22,28 |
| **6** | 6,11,16,21,26 | **12** | 7,12,17,22,27 | 18 | 8,13,17,23,27 | 24 | 7,14,19,23,29 |

1. Сформувати і вивести на дисплей одновимірний масив **b**, в якому першими елементами є елементи вихідного одновимірного масиву **a** з негативними значеннями (зі збереженням порядку проходження), а потім елементи **a** з нульовими і позитивними значеннями.
2. Визначити значення двох найбільших і різних за значенням елементів вихідного одновимірного масиву **a** і їх індекси (масив може містити елементи з рівними значеннями, тобто необхідно вивести значення і індекси елементів з максимальними значеннями і значення другого за величиною елемента, а також індекси всіх елементів, мають друге за величиною значення).
3. Сформувати одновимірний масив **b** з вихідного одновимірного масиву **a** наступним чином: якщо значення будь-яких двох або більше елементів масиву **a** дорівнюють один одному, на місці всіх цих елементів в масиві **b** виводиться 1, в іншому випадку (якщо і-ий елемент не дорівнює ніякому іншого елементу) в масиві **b** виводиться 0.
4. Сформувати одновимірний масив **b** з вихідного одновимірного масиву **a** шляхом циклічного зсуву елементів **a** на k позицій вправо. Значення k задається як перший аргумент при виклику програми, інші аргументи - елементи масиву.
5. Визначити, чи є всі елементи вихідного одновимірного масиву **a** негативними величинами, чи вони всі позитивні або серед елементів **a** є як позитивні, так і негативні величини, і вивести відповідні повідомлення для кожного випадку.
6. Визначити значення і індекси локальних мінімумів вихідного одновимірного масиву **a** (елемент масиву називається локальним мінімумом, якщо він строго менше своїх сусідів).
7. Визначити абсолютне значення найменшої різниці між двома будь-якими значеннями елементів вихідного одновимірного масиву **a**.

8. Визначити абсолютні значення найбільшою і найменшою різниці між середнім значенням і значеннями елементів вихідного одновимірного масиву **a**.
9. Сформувати масив **b** з вихідного одновимірного масиву **a** за наступним алгоритмом: спочатку йдуть елементи масиву **a** з парними значеннями в порядку їх зростання, потім елементи з непарними значеннями в порядку їх зменшення. Для визначення кількості парних елементів використовуйте оператор взяття модуля "%".
10. Сформувати масив **b** з вихідного одновимірного масиву **a** за наступним алгоритмом: $b_i$ дорівнює кількості елементів із значенням, рівним $a_i$ в масиві **a**.
11. Визначити індекси і значення елементів вихідного одновимірного масиву **a**, величини яких лежать поза нижньою $a_{min}$ і верхньою $a_{max}$ границями ($a_i < a_{min}$ або $a_i > a_{max}$). Значення $a_{min}$ і $a_{max}$ задаються як перші два аргументи при виклику програми, інші аргументи - елементи масиву.
12. Визначити, чи утворюють значення елементів вихідного одновимірного масиву **a**: строго зростаючу послідовність ($a_i < a_{i+1}$), строго спадну послідовність ($a_i > a_{i+1}$) або елементи масиву не впорядковані, і вивести для кожного випадку відповідне повідомлення.
13. Визначити, чи утворюють значення елементів вихідного одновимірного масиву **a**: арифметичну прогресію, тобто $a_i = a_{i-1} + n$, де n - різниця прогресії, і вивести відповідне повідомлення.
14. Перевірте, чи є елементи масиву **a** множиною (для цього серед елементів масиву не повинно бути двох елементів з однаковим значенням).
15. Виведіть на дисплей значення тих елементи масивів **a** і **b**, які є і в тому, і в іншому масиві (передбачається, що і масив **a**, і масив **b** є множинами, тобто кожен з них не містить елементів з однаковими значеннями).
16. Виведіть на дисплей значення тих елементів масивів **a** і **b**, які є тільки в одному з масивів, і відсутні в іншому масиві (передбачається, що і масив **a**, і масив **b** є множинами, тобто кожен з них не містить елементів з однаковими значеннями ).

17. Сформуйте з масиву **a** масив **b** за наступним алгоритмом: елемент масиву **b** дорівнює значенню різниці між максимальним значенням елементів масиву **a** і значенням даного елемента масиву **a**.
18. Виведіть на дисплей розподіл значень елементів масиву **a** по інтервалах. Межі інтервалів задаються у вигляді масиву **b**, причому нульовий елемент визначає нижню межу першого інтервалу, а елементи з 1-го по n-ий - верхні межі інтервалів. В результаті роботи програми на дисплей повинні бути виведені рядки «Інтервал nn - xx» і останній рядок «Поза інтервалами - xx».
19. Виведіть на дисплей значення і індекси тільки тих елементів масиву **a**, значення яких не дорівнюють значенням інших елементів, тобто унікальних елементів масиву.
20. Визначте (у відсотках від загальної кількості елементів), скільки елементів в масиві **a** мають значення менше, ніж середнє значення, скільки елементів - значення, рівне середньому значенню і скільки елементів мають значення, більше, ніж середнє значення.
21. Перевірте, чи не є значення і-их елементів масиву **a** лінійної комбінацією і-их значень елементів масиву **b**, тобто $a_i = k * b_i + c$, де k і c - константи (значення k і c можна визначити зі значень двох перших елементів **a** і **b** як два рівняння з двома невідомими).
22. Сформувати масив **b**, елементами якого є значення індексів елементів вихідного одновимірного масиву **a** в порядку убування значень елементів.
23. Визначити індекси і значення рівних елементів (якщо вони є) вихідного одновимірного масиву ().
24. Визначте номер дня в році по заданому номеру дня в місяці і номеру місяця (вводяться як аргументи при виклику програми). Ознака, чи є рік високосним, задається як булевська змінна. Вказівка: кількість днів до початку цього місяця (НЕ високосний рік): январь 0, лютий - 31, березень - 59, квітень - 90, травень -120, червень - 151, липень - 181, серпень - 212, вересень - 243, жовтень - 273, листопад - 314, грудень - 334 задати у

вигляді масиву. У високосному році, починаючи з березня, до кількості днів додається 1.

25. Визначте номер дня у місяці та номер місяця року по заданому номеру дня в році (вводиться як аргумент при виклику програми). Ознака, чи є рік високосним, задається як булевська змінна. Вказівка: кількість днів до початку цього місяця (НЕ високосний рік): січень - 0, лютий - 31, березень - 59, квітень - 90, травень -120, червень - 151, липень - 181, серпень - 212, вересень - 243, жовтень - 273, листопад - 314, грудень - 334 задати у вигляді масиву. У високосному році, починаючи з березня, до кількості днів додається 1.
26. Сформувати масив **b**, елементами якого є елементи вихідного одновимірного масиву **a**, розташовані в зворотному порядку.
27. Визначити кількість рівних елементів і їх індекси для двох вихідних одновимірних масивів **a** і **b**.
28. Сформувати масив **b** з вихідного одновимірного масиву **a** наступним чином: якщо $a_{min} < b_i < a_{max}$, то $b_i = a_i$; якщо $b_i \leq a_{min}$, то $b_i = a_{min}$; якщо $b_i \geq a_{max}$, то $b_i = a_{max}$ ($a_{max}$ і $a_{min}$ - максимальне і мінімальне значення елементів в масиві).
29. Сформувати масив **b** з масиву **a** наступним чином: масив **b** складається з тих елементів масиву **a**, які повторюються в масиві (по одному значенню для однакових елементів), наприклад, для масиву **a**: 3 7 4 3 8 7 5, масив **b** матиме вигляд: 3 7.

## Хід виконання роботи

1. Встановіть та завантажте IntelliJ IDEA.
2. Створіть проект **Maven**.
3. Розкрийте дерево проекту і переконайтесь в присутності папок **java** – для програмних класів і **test/java** – для тестових класів
4. В дереві проекту виберіть папку **java** і додайте до неї новий файл класу, в якому реалізуйте виконання одного завдання вашого варіанту.

5. Створіть unit-тест для тестування методів розробленого класу.
6. Переконайтесь, що тестовий клас з’явився в папці **test/java** проекту.
7. Запустіть тест на виконання.
8. Якщо тест не пройдений, внесіть необхідні виправлення в код програми і перейдіть до п.7.
9. Якщо тест успішно пройдений, перейдіть до п.10.
10. Повторіть пп.4-9 для інших завдань вашого варіанту.
11. Оформіть звіт про виконання роботи.

## Контрольні запитання

1. Як здійснюється установка Intellij IDEA?
2. Яку структуру має головне вікно Intellij IDEA?
3. Що називається проектом в Intellij IDEA?
4. Як створюється новий проект в Intellij IDEA?
5. Як здійснюється збирання і запуск проекту в Intellij IDEA?
6. Яке призначення unit-тестів?
7. Як реалізується unit-тестування в Intellij IDEA?
8. Яке призначення JUnit?
9. Які види тверджень підтримуються в JUnit?
10. Які анотації доступні в JUnit?
11. Який порядок створення unit-тестів в Intellij IDEA?

# ПРАКТИКУМ 3. РОБОТА З РЯДКАМИ І РЕГУЛЯРНИМИ ВИРАЗАМИ

## Короткі теоретичні відомості про класи String і StringBuffer

### Класи String і StringBuffer

Для зберігання і обробки рядків в Java є два класи [3,8]:

- **String** для незмінних рядків і
- **StringBuffer** - для рядків, які можуть змінюватися.

Обидва класи розширюють клас **Object**. Вони знаходяться в пакеті **java.lang**, тому їх не треба підключати за допомогою оператора **import**.

Рядкові літерали в Java, як і в C, записуються в лапках, наприклад, "abc" задає строковий літерал abc.

Якщо всередині строкового литерала необхідно задати символ апострофа, він задається за допомогою символів \ ', наприклад, "it \' s" задає строковий літерал it's.

### Створення та ініціалізація об'єкта класу String

Ініціалізація об'єкта класу **String** може виконуватися як за допомогою оператора присвоювання змінній класу **String** строкового літерала, наприклад: String str = "Рядок 1"; або при створенні об'єкта за допомогою оператора **new** з використанням одного з конструкторів класу **String**:

- **String**(),
- **String** (**String** рядок),
- **String**(**char**[] масив),
- **String**(**byte**[] масив).

Рядок, створюваний з масиву байт, створюється з використанням кодування на даному комп'ютері за замовчуванням (для української мови в Windows - кодування Windows-1252).

Довжина рядка може бути визначена за допомогою методу **public int length** ().

Єдина операція, яку можна використовувати для рядків, є операція зчеплення (конкатенація) двох або більше рядків - "+". З його допомогою рядки класу **String** можна змінювати, але при кожній зміні довжини рядка створюється новий екземпляр рядка.

### Створення та ініціалізація об'єкта класу StringBuffer

Клас **StringBuffer** схожий на клас **String**, але рядки, створені за допомогою цього класу можна модифікувати, тобто їх вміст і довжина можуть змінюватися. При зміні рядка класу **StringBuffer** програма не створює новий об'єкт рядка, а працює безпосередньо з вихідним рядком, тобто всі методи оперують безпосередньо з буфером, що містить рядок. Тому клас **StringBuffer** зазвичай використовується, коли рядок доводиться часто модифікувати.

Розміщення рядків в об'єкті **StringBuffer** виконується наступним чином. Для об'єкта **StringBuffer** задається розмір або ємність (capacity) буферної пам'яті для рядка. Рядок символів в об'єкті **StringBuffer**, характеризується своєю довжиною, яка може бути менше або дорівнює ємності буфера. Якщо довжина рядка менше ємності буфера, то решта довжина рядка заповнюється символом Unicode "\ u0000". Якщо в результаті модифікації рядки її довжина стане більше ємності буфера, ємність буфера автоматично збільшується.

У класі **StringBuffer** є три конструктора, що дозволяють по-різному створювати об'єкти типу **StringBuffer**:

- **StringBuffer** (),
- **StringBuffer** (**int** довжина),
- **StringBuffer** (**String** рядок).

Перший конструктор створює порожній об'єкт **StringBuffer** з ємністю буферної пам'яті в 16 символів, другий конструктор задає буфера з ємністю заданої довжини для зберігання рядка, а третій конструктор створює об'єкт **StringBuffer** з рядка з ємністю буфера, що дорівнює довжині рядка.

Методи визначення та встановлення характеристик рядка в класі **StringBuffer**:

- **public int length** () - повертає довжину рядка для об'єкта класу.
- **public int capacity** () - повертає поточну ємність буферної області для об'єкта класу.
- **public void ensureCapacity** (**int** ємність) - встановлює ємність буферної області для об'єкта класу.
- **public void setLength** (**int** нова-довжина) - встановлює нову-довжину рядка. Якщо нова довжина більше старої, збільшуються довжини рядка і буфера, при цьому додаткові символи заповнюються нулями. Якщо нова довжина менше старої, символи в кінці рядка відкидаються, а розмір буфера не змінюється.
- **public String toString** () - перетворює рядок **StringBuffer** в рядок **String**.

### Порівняння рядків

Оскільки в Java рядки є об'єктами, для порівняння рядків можна використовувати оператор "==" і метод **public boolean equals** (**Object** об'єкт), що порівнює рядок, для якої викликається метод, з об'єктом. Результат буде true, тільки якщо об'єкт є рядком і значення порівнюваних рядків рівні.

Використання оператора "==" для порівняння рядків може привести до невірного результату, якщо порівнювані рядки - різні об'єкти, тому більш привабливим є використання методу **equals** () (цей метод працює і для рядків **String** і для рядків **StringBuffer**).

Інші методи порівняння рядків, які також повертають булівські значення:

- **equalsIgnoreCase** (**String** рядок-1) - повертає true, коли рядок дорівнює рядку-1 незалежно від регістру символів.
- **startsWith** (**String** префікс) - повертає true, коли рядок рядок починається з префікса.

- **endsWith** (**String** суфікс) - повертає true, коли рядок закінчується рядком суфікс.

Метод **int compareTo** (**String** anotherString) - лексикографічно порівнює два рядки і повертає 0, якщо рядки рівні по довжині і мають однакове значення, менше 0, якщо в першій позиції, в якій символи рядків не рівні, код символу в першому рядку менше коду символу у другому рядку або довжина першого рядка менше довжини другого рядка. Якщо в першій позиції, в якій символи рядків не рівні, код символу в першому рядку більше коду символу в другому рядку або довжина першого рядка більше довжини другого рядка, повертається значення, більше 0.

### Пошук в рядках

Для пошуку символів або послідовностей символів (тільки в рядках класу **String**) використовуються наступні перевантажені методи **indexOf** ():

- **int indexOf** (**int** символ) – повертається перша позиція в рядку, в якій зустрічається символ.
- **public int indexOf** (**int** символ, **int** початковий-індекс) – повертається перша позиція в рядку, починаючи з позиції початковий-індекс, в якій зустрічається символ.
- **int indexOf** (**String** підрядок) – повертається перша позиція в рядку, в якій зустрічається підрядок.
- **public int indexOf** (**String** підрядок, **int** початковий-індекс) – повертається перша позиція в рядку, починаючи з позиції початковий-індекс, в якій зустрічається підрядок.

Для кожного методу **indexOf** () є відповідний метод **lastIndexOf** (), який починає пошук символу або підрядка не з початку, а з кінця рядка. Якщо символ або підрядок не знайдені в рядку, методи **indexOf** () і **lastIndexOf** () повертають значення -1.

### Отримання символів і підрідків з рядка

Отримання символів і підрідків з рядка, а також створення нових рядків на основі існуючих рядків виконується за допомогою таких методів:

- **char charAt** (**int** індекс) – повертається символ рядка в позиції індекс.
- **char** [] **toCharArray** ()– повертається масив символів рядка.
- **String substring** (**int** початковий-індекс) – повертається підрядок початкового рядка, що починається з позиції початковий-індекс початкового рядка.
- **String substring** (**int** початковий-індекс, **int** кінцевий-індекс) – повертається підрядок початкового рядка, що починається в позиції початковий-індекс і закінчується в позиції кінцевий-індекс-1 початкового рядка.
- **void getChars** (**int** початковий-індекс, **int** кінцевий-індекс, **char** [] масив, **int** індекс-вставки) – копіюється частина рядка, починаючи з символу в позиції початковий-індекс і закінчуючи символом в позиції індекс-вставки + (кінцевий-індекс - початковий-індекс) -1 в символьний масив, починаючи з позиції індекс-вставки.

### Модифікація рядків

Створення нових рядків на основі існуючих рядків виконується за допомогою таких методів класу **String**:

- **String concat** (**String** підрядок) – повертає вихідний рядок, в кінець якого додано підрядок.
- **String toLowerCase** () – повертає вихідний рядок в нижньому регістрі.
- **String toUpperCase** () – повертає вихідний рядок в верхньому регістрі.
- **String trim** () – повертає вихідний рядок, з якого виключені початкові і кінцеві пробільні символи.

- **String replace** (**char** символ-1, **char** символ-2) – повертає вихідний рядок, в якому всі символи символ-1 замінені на символ-2.
- **public static String valueOf** (тип ім'я) – повертає рядок, перетворений з примітивних типів даних. Допустимі типи параметра: **boolean**, **char**, **int**, **long**, **float**, **double** або **char** [].
- **public static String valueOf** (**char** [] масив, **int** початковий-індекс, **int** довжина) – повертає рядок, отриманий з фрагмента масиву, який починається в позиції початковий-індекс і заданої довжини.

Методи класу **StringBuffer**, що дозволяють безпосередньо модифікувати рядок:

- **public void setCharAt** (**int** індекс, **char** символ) - розміщує в позиції індекса заданий символ.
- **public void deleteCharAt** (**int** індекс) - видаляє символ в позиції індексу.
- **public StringBuffer replace** (**int** початковий-індекс, **int** кінцевий-індекс, **String** підрядок) - замінює фрагмент рядка, починаючи з позиції початковий-індекс і до позиції кінцевий-індекс підрядком, а потім повертає змінений рядок.
- **public StringBuffer append** (тип ім'я) додає в кінець рядка дані заданого типу з заданим ім'ям і повертає змінений рядок. Допустимі типи параметра: **Object**, **boolean**, **char**, **int**, **long**, **float**, **double**, **String**, **StringBuffer** або **char** [].
- **public StringBuffer insert** (**int** початковий-індекс, тип ім'я) - вставляє в рядок в позиції початковий-індекс дані заданого типу з заданим ім'ям і повертає змінений рядок. Допустимі типи параметра: **Object**, **boolean**, **char**, **int**, **long**, **float**, **double**, **String**, **StringBuffer** або **char** [].
- **public StringBuffer append** (**char** [] масив, **int** початковий-індекс, **int** довжина) - додає в рядок фрагмент символьного масиву, що

починається з позиції початковий-індекс заданої довжини, і повертає змінений рядок.

- **public StringBuffer insert** (**int** початковий-індекс-рядки, **char** [] масив, **int** початковий-індекс-масиву, **int** довжина) - вставляє в рядок в позиції початковий-індекс-рядки фрагмент символьного масиву, що починається з позиції початковий-індекс-масиву заданої довжини, і повертає змінений рядок.

## Регулярні вирази

### Поняття регулярного виразу

Функції роботи з підрядками дозволяють виконати операції пошуку і заміни підрядків в рядку. Однак при роботі з даними часто доводиться виконувати операції пошуку і заміни по досить складним алгоритмам. Хоча такі операції можна виконати, використовуючи функції роботи з рядками, умовні оператори і оператори циклу, в мові Java, як і в інших мовах програмування, існує більш зручний спосіб - використання регулярних виразів.

Регулярні вирази використовуються для вирішення наступних завдань [2-5]:

- перевірки даних на відповідність певному зразку, званого шаблоном (pattern);
- заміни або видалення даних.

Синтаксис регулярних виразів в Java в основному такий же, як і в інших мовах програмування.

### Синтаксис регулярного виразу

Регулярний вираз в мові Java є об'єктом класу **String**, тобто є рядком - послідовністю символів.

Символи в рядку можуть бути наступних типів:

- алфавітно-цифрові символи, включаючи літери кирилиці;
- символ '\\' - зворотна коса риска (зворотний слеш);

- символ '\ 0num' - вісімкове число, де num - одна, дві або три восьмеричні цифри;
- символ '\ xhh' - код символу ASCII, де hh - дві шістнадцяткові цифри;
- символ '\ uhhhh' - код символу Unicode, де hhhh - чотири шістнадцяткові цифри;
- символ табуляції ( '\ t' або '\ u0009');
- символ нового рядка ( '\ n' або '\ u000A');
- символ повернення каретки ( '\ r' або '\ u000D');
- символ переходу до нової сторінки ( '\ f' або '\ u000C');
- символ звукового сигналу ( '\ a' або '\ u0007');
- символ Escape (Esc) ( '\ u001B');
- символ '\ cx' - відповідає керуючому символу x (наприклад, \ cM відповідає символу Ctrl + M або символу повернення каретки).

Деякі символи в регулярних виразах, звані метасимволами, мають особливе значення.

Метасимволами є такі символи: ^ $ () \ | [{? . + *

Саме використання метасимволов забезпечує всю міць і гнучкість регулярних виразів.

Якщо в регулярному виразі необхідно розглядати метасимвол як звичайний символ, перед ним треба поставити символ '\\', наприклад, "\\ (".

### Операція альтернації

У регулярних виразах можна об'єднувати кілька шаблонів, так щоб знайдена рядок відповідала хоча б одному з них. Для вирішення подібної проблеми служить операція альтернації, яка в регулярних виразах задається символом "|". Наприклад шаблон "Internet | Інтернет" означає пошук або рядків "Internet", або рядків "Інтернет".

### Одиночний метасимвол

Метасимвол точка "." всередині регулярного виразу точка відповідає будь-якому одиночному символу, крім символу переведення рядка.

### Квантіфікатори

Квантіфікатори - це метасимволи, використовувані для вказівки кількісних відносин між символами в шаблоні і в шуканому рядку. Квантіфікатор може бути поставлений після одиночного символу або після групи символів.

Найпростішим квантіфікатор є метасимвол "+". Він означає, що вказаний перед ним символ відповідає кільком розташованим підряд таким символам в рядку пошуку. Кількість символів може бути будь-якою, але повинен бути присутнім хоча б один символ.

Дія метасимвола "*" схоже на дію метасимвола "+". Метасимвол "*" вказує, що вказаний перед ним символ зустрічається нуль або більше разів.

Метасимвол "?" вказує, що передуючий йому символ повинен зустрічатися або один раз, або не зустрічатися взагалі.

Якщо необхідно вказати точно кількість повторень символу, можна скористатися конструкцією {n, m}. Тут n - мінімально допустима кількість повторень попереднього символу, m - максимально допустима кількість повторень. Один з параметрів n або m можна опустити.

Фактично квантіфікатори "+", "*" і "?" є окремими випадками конструкції {n, m}: відповідно, {1,}, {0,} і {0,1}.

У регулярних виразах часто використовують поєднання метасимволів ".*". Йому відповідають будь-які символи. За правилами обробки регулярних виразів знаходиться найдовший рядок, який все ще задовольняє шаблоном пошуку.

### Класи символів

Для пошуку в регулярних виразах можна задавати також класи символів, укладені в квадратні дужки. Під час пошуку всі символи в класі розглядаються як один символ. Усередині класу можна задавати діапазон

символів (коли такий діапазон має сенс), поміщаючи дефіс між границями діапазону. Усередині символьних класів більшість метасимволів втрачають свої значення і стають звичайними символами.

Якщо першим символом класу є знак вставки "^", то значення виразу інвертується. Іншими словами, такого класу відповідає будь-який символ, який не входить в клас.

Так як в класах символи "]", "^" і "-" мають спеціальне значення, для їх використання в класі існують певні правила:

- літерал "^" не повинен бути першим символом класу;
- перед літералом "]" повинен стояти символ зворотної косої риски;
- для розміщення в класі символу "-" достатньо або поставити його на першу позицію, або помістити перед ним символ зворотної косої риски.

**Спеціальні символи**

Найбільш поширені класи символів можна задати за допомогою наступних спеціальних символів:

\d - відповідає будь-якому цифровому символу (еквівалентно [0-9]);

\D - відповідає будь-якому нецифровому символу (еквівалентно [^ 0-9]);

\w - відповідає будь-якій латинській букві або цифрі (еквівалентно [A-Za-z0-9]);

\W - відповідає будь-якому небуквеному (латинському) і нецифровому символу (еквівалентно [^ A-Za-z0-9]);

\s - відповідає будь-якому символ пробілу (еквівалентно [\ f \ n \ r \ t \ v]);

\S - відповідає будь-якому непробільному символу (еквівалентно [^ \ f \ n \ r \ t \ v]).

Слід зазначити, що спеціальні символи \w і \W не можна використовувати для букв кирилиці, а також букв західноєвропейських

алфавітів, відмінних від латинських букв. В цьому випадку необхідно безпосередньо задавати діапазон символів, як це робиться для класів символів.

### Анкери

За допомогою анкерів можна вказати, в якому місці рядка має бути знайдено відповідність з шаблоном.

В Java визначені наступні анкери:

^ - відповідає позиції на початку рядка;

$ - відповідає позиції в кінці рядка;

\b - відповідає границі слова, тобто границі між словом і пробільним символом;

\B - відповідає не границі слова.

На жаль, анкери \b і \B діють тільки для рядків, що складаються з латинських букв.

### Угруповання елементів

Операція угруповання елементів, тобто занесення групи елементів в круглі дужки, дозволяє розглядати дану групу елементів як один елемент.

Якщо в регулярних виразах використовуються дужки, частини шуканого рядка, що відповідають фрагментам в дужках, запам'ятовуються в спеціальних змінних $1 (перший фрагмент в дужках), $2 (другий фрагмент в дужках), $3 (третій фрагмент в дужках) і т.д. Така операція називається *захопленням* (capture) змінної.

Для вкладених дужок змінні $1-$9 формуються в порядку появи лівої (відкриваючої) дужки виразу.

Слід зазначити, що змінні модифікуються при кожному успішному пошуку, незалежно від використання в регулярному виразі дужок. Крім того, значення цих змінних встановлюються тоді і тільки тоді, коли рядок повністю відповідає шаблону. В іншому випадку значення цих змінних рівні порожньому рядку.

Змінні $1-$9 можна записувати і в шаблоні у формі \i, де і - номер змінної.

Змінні $1-$9 не завжди необхідно використовувати як результат пошуку відповідності шаблону. У разі, коли необхідно згрупувати будь-які символи шаблону, але не виконувати для них операції по визначенню відповідних змінних (не виконувати для них операцію захоплення), використовується наступна форма групування: (?: Символи).

## Класи Java для роботи з регулярними виразами

Для роботи з регулярними виразами в Java використовуються класи **Pattern**, **Matcher** і **PatternSyntaxException** пакета **java.util.regex** [3,7,9].

### Клас Pattern

Об'єкт класу Pattern є відкомпільованим представленням шаблону регулярного виразу і створюється не за допомогою ключового слова **new**, а за допомогою статичних методів **compile** () класу **Pattern**.

- Метод **public static Pattern compile** (**String** шаблон) повертає об'єкт класу **Pattern** для заданого в параметрі шаблону.
- Метод **public static Pattern compile** (**String** шаблон, **int** прапорці) повертає об'єкт класу **Pattern** для заданого в параметрі шаблону з заданими прапорцями.

Прапорці представлені в Java як статичні поля типу **public static final int** класу **Pattern**:

- **CASE_INSENSITIVE** - включає пошук відповідності без урахування верхнього або нижнього регістру.
- **UNICODE_CASE** - якщо цей прапорець включений разом з прапорцем CASE_INSENSITIVE, то верхній і нижній регістри букв в коді Unicode не враховуються при пошуку відповідності.
- **UNIX_LINES** - при включенні цього прапорця тільки символ "\n" враховується як символ закінчення рядка, в якому виконується пошук відповідності.

- **MULTILINE** - якщо всередині рядка, в якому виконується пошук відповідності, є символи "\n", то вважається, що рядок складається з декількох рядків.
- **LITERAL** - всі символи шаблону, включаючи метасимволи, розглядаються як звичайні символи.
- **DOTALL** - **я**кщо в шаблоні є метасимвол ".", То йому буде відповідати будь-який символ, включаючи символ "\n".
- **COMMENTS** - у рядку шаблону допустимі пробіли і коментарі, що починаються з символу "#" до кінця рядка.
- **CANON_EQ** - **п**ри пошуку відповідності буде враховуватися відповідність між кодом символу і самим символом.

Прапорці можна включати безпосередньо в шаблоні, використовуючи наступну синтаксичну форму:

(? Рядок-символів),

де символи в рядку-символів можуть мати одне з наступних значень:

i - для прапорця CASE_INSENSITIVE;

d - для прапорця UNIX_LINES;

m - для прапорця MULTILINE;

s - для прапорця DOTALL;

u - для прапорця UNICODE_CASE;

x - для прапорця COMMENTS.

Методи класу Pattern:

- **public static boolean matches** (**String** шаблон, **CharSequence** рядок-пошуку) - перевіряє відповідність шаблону рядку-пошуку і повертає значення true, якщо рядок пошуку відповідає шаблону і false - в протилежному випадку.
- **public String pattern** () - повертає рядок шаблону для об'єкта класу **Pattern**.

- **public String** [] **split** (**CharSequence** рядок-пошуку) - створює з рядка-пошуку масив, розділений на елементи за шаблоном, заданому в об'єкті класу **Pattern**.

### Клас Matcher

Клас **Matcher** забезпечує виконання пошуку або заміни відповідності заданого об'єктом класу **Pattern** шаблоном.

Об'єкт класу **Matcher** створюється за допомогою методу **public Matcher matcher** (**CharSequence** рядок-пошуку) класу **Pattern** для рядка-пошуку.

Строкове представлення об'єкту класу **Matcher** можна отримати за допомогою методу **public String toString** ().

Пошук відповідності виконується в підрядку початкового рядка, який називають *регіоном* (region). За замовчуванням регіоном є вся введена послідовність символів.

Методи для операцій з регіонами:

- **public Matcher region** (**int** початковий-індекс, **int** кінцевий-індекс) - встановлює межі регіону і повертає об'єкт класу **Matcher** для підрядка, що починається з початкового-індексу і закінчується індексом, на одиницю меншим, ніж кінцевий-індекс.
- **public int regionStart** () - отримує і повертає індекс початку регіону.
- **public int regionEnd** () - отримує і повертає індекс закінчення регіону.

Методи з пошуку відповідностей:

- **public boolean matches** () - виконує для об'єкта класу **Matcher** пошук на відповідність всього регіону, починаючи з початку регіону. Повертає true, якщо відповідність знайдена і false - в протилежному випадку.
- **public boolean lookingAt** () - виконує для об'єкта класу **Matcher** пошук, починаючи з початку регіону на наявність шаблону в

регіоні, але необов'язкова відповідність всього регіону шаблону. Повертає true, якщо відповідність знайдена і false - в протилежному випадку.

- **public boolean find** () - виконує для об'єкта класу **Matcher** пошук, починаючи з початку регіону або, якщо попередній виклик методу був успішним, з першого символу після знайденого попереднього відповідності. Повертає true, якщо відповідність знайдена і false - в протилежному випадку.
- **public boolean find** (**int** початковий-індекс) - виконується так само, як і метод find () без параметрів, але пошук починається не з початку регіону, а заданого початкового-індексу. Якщо необхідно знайти всі відповідності шаблоном в рядку, починаючи з початкового-індексу, то цей метод можна використовувати тільки для пошуку першого збігу. Всі інші відповідності визначаються за допомогою методу find () без параметрів.

Методи класу **Matcher** дозволяють не тільки виконати пошук в рядку за заданим шаблоном, але і замінити знайдені відповідності заданими послідовностями символів - рядками заміни.

Методи заміни:

- **public String replaceFirst** (**String** рядок-заміни) - замінює тільки першу відповідність в рядку пошуку рядком-заміни і повертає змінений рядок.
- **public String replaceAll** (**String** рядок-заміни) - замінює всі відповідності в рядку пошуку рядком-заміни і повертає змінений рядок.

### Клас PatternSyntaxException

Клас **PatternSyntaxException** кидає виключення, якщо регулярний вираз (шаблон) містить синтаксичну помилку.

Методи класу **PatternSyntaxException**:

- **public String getPattern** () – повертає шаблон, що містить помилку.
- **public String getDescription** () – повертає опис помилки.
- **public int getIndex** () – повертає позицію символу помилки в шаблоні.
- **public String getMessage** () – повертає повідомлення про помилку, що містить всі перераховані вище компоненти: опис помилки і її індекс, шаблон, що містить помилку і візуальну індикацію індексу помилки усередині шаблону.

### Створення регулярних виразів

Робота з регулярними виразами в будь-який Java-програмі починається зі створення об'єкта класу **Pattern**. Для цього необхідно викликати один з двох наявних в класі статичних методів **compile**. наприклад:

```
Pattern p = Pattern.compile ( "java", Pattern.CASE_INSENSITIVE);
```

При створенні об’єкту **Pattern** проводиться синтаксична перевірка регулярного виразу. При наявності помилок в рядку - генерується виключення **PatternSyntaxException**.

**Matcher** - клас, з якого створюється об'єкт для пошуку збігів за шаблоном, це «пошуковик», «движок» регулярних виразів. Для створення об'єкта **Matcher** в класі **Pattern** передбачений метод **public Matcher matcher** (**CharSequence** input). Як аргумент метод приймає об'єкт, що реалізує інтерфейс **CharSequence**, в якому буде проводитись пошук. В якості аргумент можна передати не тільки **String**, а й **StringBuffer**, **StringBuilder** та інше.

Шаблоном для пошуку є об'єкт класу **Pattern**, на якому викликається метод **matcher**:

```
Matcher m = p.matcher ( "Hello Java");
```

Тепер за допомогою нашого «пошукача» ми можемо шукати збіги, дізнаватися позицію збіги в тексті, замінювати текст за допомогою методів класу. Наприклад, перевіримо наявність збігів:

```
while (m.find ()) {
    int start = m.start ();
    int end = m.end ();
  System.out.println ( "Знайдено збіг" + text.substring (start, end) + "з" + start + "по" + (end-1) + "позицію");
    }
```

### Тестування регулярних виразів

Для тестування регулярних виразів можна використовувати онлайн сервіси, наприклад, http://myregexp.com/.

Але якщо ви працюєте в IDEA, то простіше прямо в ній і перевіряти, поставивши плагін **RegExp**.

Також IDEA дає потужний вбудований інструмент по роботі з виразами: поставивши курсор на регулярний вираз в коді натисніть **Alt** + **Enter**, а далі виберіть **Check Regex**.

### Методи класу String для роботи з регулярними виразами

Для роботи з регулярними виразами в класі **String** визначені методи для роботи з регулярними виразами:

- **public boolean matches** (**String** шаблон) - якщо об'єкт класу **String** відповідає шаблону, повертає значення true, в іншому випадку повертає false (діє аналогічно методу **matches**() класу **Pattern**).
- **public String** [] **split** (**String** шаблон) - створює для об'єкта класу **String** масив рядків, розділений на елементи за заданим

шаблоном (діє аналогічно відповідному методу **split**() класу **Pattern**).

- **public String replaceFirst** (**String** шаблон, **String** рядок-заміни) - замінює в об'єкті **String** першу відповідність шаблону на рядок-заміни і повертає змінену рядок (діє аналогічно методу **replaceFirst**() класу **Match**).
- **public String replaceAll** (**String** шаблон, **String** рядок-заміни) - замінює в об'єкті **String** всі відповідності шаблону на рядок-заміни і повертає змінену рядок (діє аналогічно методу **replaceAll**() класу **Match**).

**Варіанти завдань**

| № | Завдання | № | Завдання | № | Завдання | № | Завдання |
|---|---|---|---|---|---|---|---|
| **1** | 1,5,16,40 | **7** | 8,13,18,22 | 13 | 2,14,28,38 | 19 | 8,15,22,34 |
| **2** | 2,13,17,41 | **8** | 9,14,23,43 | 14 | 3,5,29,39 | 20 | 9,13,21,35 |
| **3** | 3,15,18,42 | **9** | 5,10,24,44 | 15 | 4,15,26,30 | 21 | 10,14,20,36 |
| **4** | 4,14,19,43 | **10** | 11,15,25,35 | 16 | 5,13,25,31 | 22 | 5,11,19,37 |
| **5** | 5,6,20,44 | **11** | 12,14,26,36 | 17 | 5,7,24,32 | 23 | 12,15,18,38 |
| **6** | 7,15,17,21 | **12** | 1,13,27,37 | 18 | 14,6,23,33 | 24 | 1,13,17,39 |

1. Вивести таблицю перетворень цілих десяткових чисел в інтервалі від 10 до 100 з кроком 20 в 16-вому представленні.
2. Вивести таблицю перетворень цілих десяткових чисел в інтервалі від min до max з кроком step в 16-вому представленні. Параметри завдання задаються як параметри командного рядка.
3. Вивести таблицю перетворень цілих десяткових чисел в інтервалі від min до max з кроком step в 16-вому представленні. Параметри завдання вводяться користувачем в ході діалогу з програмою за один раз.

4. Забезпечте виведення команд меню у вигляді нумерованого списку і виклик команд по номеру.
5. Створіть програму для шифрування\розшифровки тексту методом Цезаря. У ньому ключем є ціле число, а шифрування\ розшифровка полягає в сумовуванні\відніманні кодів символів відкритого тексту\криптотексту з ключем.
6. Створіть програму для визначення коду введеного символу в стандарті Юнікод. Забезпечте циклічність виконання програми до введення знака рівності, яке призводить до виходу з програми).
7. Створіть програму для визначення коду введеного символу в 16-ном представлені в форматі '\uXXXX ". Забезпечте циклічність виконання програми до введення символу пробіл. Перевірте правильність роботи програми, скориставшись нею для отримання кодів символів в будь-якому слові (наприклад," cat ") і вивівши на екран слово з отриманих кодів.
8. Створіть програму, яка приймає довільний текстовий рядок, а повертає інший, в якому символи розташовуються в зворотному порядку.
9. Створіть програму для перевірки тексту на паліндром. Літери і пропуски не враховуються.
10. Створіть програму для визначення кількості входжень зазначеного символу в заданому тексті. Робота програми припиняється після введення символу "-".
11. Створіть програму для визначення позицій входжень зазначеного символу в заданому тексті. Робота програми припиняється після введення символу "-".
12. Створіть програму, яка приймає текстовий рядок і видає інший, в якому букви переставлені в випадковому порядку.
13. Створіть програму, яка здійснює шифрування/розшифрування методом простої заміни. У ньому кожен символ

незашифрованого тексту з вихідного алфавіту замінюється іншим з алфавіту, символи в якому представлені відповідно до ключа шифрування.

14. Створіть програму, яка здійснює шифрування/розшифрування методом гамування. У ньому ключем є текстовий рядок такої ж довжини, як і відкритий текст, а шифрування/розшифрування полягає в сумуванні/відніманні кодів символів відкритого тексту/ криптотексту з кодами символів ключа.
15. Створіть програму, яка здійснює шифрування/розшифрування модифікованим методом Цезаря. У ньому ключем є гасло, яке багаторазово повторюється до тих пір, щоб досягти довжини відкритого тексту, а шифрування/розшифрування полягає в сумуванні/відніманні кодів символів відкритого тексту/ криптотексту з відповідними кодами символів гасла.
16. Аналіз типів аргументів, що задаються під час запуску програми. Якщо аргумент є правильним цілим числом (шаблон: одна або кілька цифр, першим символом може бути або цифра, або знак "+" або "-"), то тип аргументу "Integer", інакше "String". Програма виводить кількість заданих аргументів і, для кожного аргументу, його тип і значення.
17. Аналіз типів аргументів, що задаються під час запуску програми. Якщо аргумент є десятковим числом з цілою і дробовою частиною (шаблон: складається з однієї або декількох цифр, однієї десяткової точки, яка може бути на початку, в середині або в кінці числа, і, крім того, першим символом числа може бути знак "+ "або" - "), то тип аргументу" Decimal ", інакше" String ". Програма виводить кількість заданих аргументів і, для кожного аргументу, його тип і значення.
18. Аналіз типів аргументів, що задаються під час запуску програми. Якщо аргумент є числом з плаваючою точкою (шаблон: складається з мантиси - одна або кілька цифр, можливо зі знаком

"+" або "-", яка може містити десяткову точку на початку, середині або наприкінці, а також порядку - цілого числа зі знаком "+" або "-" або без знака, роздільником між мантисою і порядком служить символ "e" або символ "E"), то тип аргументу "Float", інакше "String". Програма виводить кількість заданих аргументів і для кожного аргументу його тип і значення.

19. Аналіз типів аргументів, що задаються під час запуску програми. Якщо аргумент є цілим восьмеричним числом (шаблон: складається з цифр від 0 до 7, причому першою цифрою повинен бути 0), то тип аргументу "Octal", інакше "String". Програма виводить кількість заданих аргументів і для кожного аргументу його тип і значення.
20. Аналіз типів аргументів, що задаються під час запуску програми. Якщо аргумент є цілим шістнадцятковим числом (шаблон: складається з цифр від 0 до 9 і букв A (a), B (b), C (c), D (d), E (e), F (f), перед числом повинні стояти символи "0X" або "0x"), то тип аргументу "Hexadecimal", інакше "String". Програма виводить кількість заданих аргументів і для кожного аргументу його тип і значення.
21. Аналіз типів аргументів, що задаються під час запуску програми. Якщо аргумент є цілим двійковим числом (шаблон: одна і більш цифр 0 і 1), то тип аргументу "Binary", інакше "String". Програма виводить кількість заданих аргументів і для кожного аргументу його тип і значення.
22. Аналіз типів аргументів, що задаються під час запуску програми. Якщо аргумент має вигляд "ім'я = значення", то він є ключовим параметром (тип "Keyed"), якщо аргумент має вигляд "-значення" або "/ значення", то він є опцією (тип "Optional") і якщо має вигляд "значення", то є безпосереднім параметром (тип "Immediate"). Шаблон для значення: одна або кілька цифр і букв (включаючи літери кирилиці). Програма виводить кількість

заданих аргументів і для кожного аргументу його тип і значення (для ключових параметрів додатково виводиться ім'я параметра).

23. Аналіз типів аргументів, що задаються під час запуску програми. Якщо аргумент має вигляд "значення", то він є одиночним параметром (тип "Single"), якщо аргумент має вигляд "значення, значення, ...", то він є списком (тип "List"). Програма виводить кількість заданих аргументів і, для кожного аргументу, його тип і значення (для кожного параметра-списку при виведенні його значення перетвориться в масив типу String, кожен елемент якого містить елемент списку і виводиться в циклі за елементами).
24. Аналіз типів аргументів, що задаються під час запуску програми. Якщо аргумент є правильним ідентифікатором Java (шаблон: складається з латинських букв, цифр і символів "$" і "_", що вважаються буквами, і, крім того, перший символ є літерою), то його тип "Identifier", якщо аргумент є ключовим словом Java (для прикладу задати кілька ключових слів Java, "if", "for", "while", "do" і "else"), то його тип "Keyword", інакше його тип вважається "Illegal". Програма виводить кількість заданих аргументів і для кожного аргументу його тип і значення.
25. Аналіз аргументів, що задаються під час запуску програми. Програма шукає найбільший рядок, що міститься у введених аргументах (шаблон аргументу: рядок або латинських букв, або букв кирилиці). Програма виводить кількість заданих аргументів, їх значення і найбільший рядок або повідомлення про те, що такого немає.
26. Аналіз аргументів, що задаються під час запуску програми. Програма визначає, які символи містяться у введених аргументах (наприклад, аргументи "abc", "cf", "bfc" містять символи "abcf"). Шаблон аргументу: рядок або латинських букв, або букв кирилиці. Програма виводить кількість заданих аргументів, значення аргументів і рядок символів, що містяться в аргументах.

27. Аналіз аргументів, що задаються під час запуску програми. Програма видаляє з аргументів все повторювані символи, крім одного (наприклад, аргумент "abcadc", перетворюється в "abcd"). Шаблон аргументу: рядок або латинських букв, або букв кирилиці. Програма виводить кількість заданих аргументів, їх значення і перетворені значення аргументів.
28. Аналіз аргументів, що задаються під час запуску програми. Програма визначає рядок символів, що зустрічаються тільки в одному з аргументів (наприклад, для аргументів "agc", "cf", "bfc" такий рядком буде рядок "gb"). Шаблон аргументу: рядок або латинських букв, або букв кирилиці. Програма виводить кількість заданих аргументів, їх значення і знайдену рядок символів або повідомлення про те, що таких символів немає.
29. Аналіз аргументів, що задаються під час запуску програми. Програма визначає, які з введених аргументів містять рядок, що задається в якості першого параметра. Шаблон аргументу: рядок або цифр, або латинських букв, або букв кирилиці. Програма виводить кількість заданих аргументів (без урахування першого аргументу) і аргументи, що містять задану підрядок або повідомлення про те, що даний рядок не міститься у введених аргументах.
30. Аналіз аргументів, що задаються під час запуску програми. Програма визначає, скільки разів символ зустрічається у введених аргументах (наприклад, для аргументів "agc", "cf", "bfc" символи "a", "g" і "b" зустрічаються один раз, символ "f" - два рази і символ "c" - три рази). Шаблон аргументу: рядок або латинських букв, або букв кирилиці. Програма виводить кількість заданих аргументів, їх значення і для кожного символу - частоту його повторення в аргументах.
31. Аналіз аргументів, що задаються під час запуску програми. Програма сортує введені аргументи (шаблон аргументу: рядок

латинських букв) по першому символу аргументу. Програма виводить кількість заданих параметрів, їх значення і список відсортованих аргументів.

32. Аналіз аргументів, що задаються під час запуску програми. Програма визначає тип аргумент - ціле число або рядок (шаблон: цілим числом вважається рядок, яка містить цифри і, крім того, першим символом рядка може бути цифра, знак "+" або "-"). Потім серед аргументів-чисел шукається максимальне число. Програма виводить кількість заданих аргументів, їх значення, кількість аргументів-чисел і значення максимального числа.
33. Аналіз аргументів, що задаються під час запуску програми. Програма визначає, чи є серед введених аргументів однакові аргументи і число їх повторення. Шаблон аргументу: рядок або цифр, або латинських букв. Програма виводить кількість заданих аргументів, значення повторюваних аргументів і кількість їх повторення або повідомлення про те, що повторюваних аргументів немає.
34. Аналіз аргументів, що задаються під час запуску програми. Програма переставляє введені аргументи в порядку зростання їх довжини. Шаблон аргументу: рядок або цифр, або латинських букв, або букв кирилиці. Програма виводить кількість заданих аргументів, їх значення, а також список значень аргументів в порядку зростання їх довжини.
35. Аналіз аргументів, що задаються під час запуску програми. Програма визначає, які з введених аргументів містять рядок, що задається в якості першого параметра. Шаблон аргументу: рядок або цифр, або латинських букв, або букв кирилиці. Програма виводить кількість заданих аргументів (без урахування першого аргументу) і аргументи, що містять задану підрядок або повідомлення про те, що даний рядок не міститься у введених аргументах.

36. Аналіз аргументів, що задаються під час запуску програми. Програма видаляє з масиву введених аргументів все повторюються аргументи, крім одного (наприклад, з аргументів "ab", "cd", "ab" будуть залишені аргументи "ab" і "cd"). Шаблон аргументу: рядок латинських букв. Програма виводить кількість заданих аргументів, їх значення, а також кількість різних аргументів і їх значення.
37. Аналіз аргументів, що задаються під час запуску програми. Програма визначає тип аргумент - ціле число або рядок (шаблон: цілим числом вважається рядок, яка містить цифри і, крім того, першим символом рядка може бути цифра, знак "+" або "-"). Потім аргументи-числа сортуються в порядку зростання їх значення. Програма виводить кількість заданих аргументів, їх значення, кількість аргументів-чисел і їх значення в порядку зростання.
38. Перетворення аргументів, що задаються під час запуску програми. Програма визначає тип аргументу - двійкове число без знака або рядок (шаблон: двійковим числом без знака вважається рядок, яка містить одну або більше цифр 0 і 1). Введені аргументи-числа перетворюються в шістнадцяткові числа (кожні чотири цифри двійкового числа перетворюються в одне шістнадцяткове, тому, при необхідності, в значення аргументу додаються нулі до довжини, кратні 4). Програма виводить кількість заданих аргументів, їх значення, а також кількість аргументів-чисел і їх шістнадцяткові значення.
39. Перетворення аргументів, що задаються під час запуску програми. Програма визначає тип аргумент - шістнадцяткове число без знака (шаблон: шістнадцятковим числом без знака вважається рядок, яка містить цифри від 0 до 9 і букви A (a), B (b), C (c), D (d), E ( e), F (f)) або рядок. Введені аргументи-числа перетворюються в двійкові числа. Програма виводить кількість

заданих аргументів, їх значення, а також кількість аргументів-чисел і їх виконавчі значення.

40. Перетворення аргументів, що задаються під час запуску програми. Програма перетворює російські та латинські літери в аргументах в верхній регістр (якщо вони є малими). Шаблон аргументу: або рядки латинських букв, або рядки букв кирилиці. Програма виводить кількість заданих аргументів, їх значення, а також нові значення аргументів.
41. Перетворення аргументів, що задаються під час запуску програми. Програма визначає тип аргументу - шістнадцяткове число без знака (шаблон: шістнадцятковим числом без знака вважається рядок, яка містить цифри від 0 до 9 і букви A (a), B (b), C (c), D (d), E ( e), F (f)) або рядок. Введені аргументи-числа перетворюються в десяткові числа (кожна i-а цифра шістнадцятирічного числа перетвориться в десяткове число $N_i$ за формулою $N_i = 16^{n-i}-1$, де n - кількість цифр в числі; i = [0, n] - індекс цифри в числі, шукане число є сумою всіх $N_i$). Програма виводить кількість заданих аргументів, їх значення, а також кількість аргументів-чисел і їх десяткові значення.
42. Аналіз аргументів, що задаються під час запуску програми. Аргумент має такий вигляд: ім'я-типу-або-ім'я-класа.імя-змінної-або-ім'я-методу і являє собою звернення до змінної або виклик методу для об'єкта або класу в Java. Шаблон аргументу: латинські букви, цифри і символи "$" і "_", які вважаються буквами, і, крім того, перший символ є літерою. Якщо перший символ звернення - велика літера, то виводиться тип "Static", якщо мала літера - виводиться "Object". Якщо ім'я змінної або методу починається з малої літери, то якщо ім'я містить символи "(" і ")", то виводиться тип звернення "Method", інакше виводиться тип звернення "Variable". Якщо будь-яка з наведених умов не виконується, то виводиться тип звернення "Illegal".

Програма виводить кількість заданих аргументів і, для кожного аргументу, його тип, тип звернення і значення.

43. Аналіз типів аргументів, що задаються під час запуску програми. Якщо аргумент є правильним ідентифікатором мови С (шаблон: складається з латинських букв, цифр і символу і "_", що вважається літерою, і, крім того, перший символ є літерою), то його тип "Identifier", якщо параметр є ключовим словом С (для прикладу задати кілька ключових слів С, "if", "for", "while", "do" і "else"), то його тип "Keyword", інакше його тип вважається "Illegal". Програма виводить кількість заданих аргументів і, для кожного аргументу, його тип і значення
44. Аналіз типів аргументів, що задаються під час запуску програми. Якщо аргумент є числовим літералом, тобто починається з цифри, то визначається його тип ( "Integer" або "Real"), якщо аргумент укладений в поодинокі апострофи і містить один символ, то його тип - "Character", якщо аргумент укладений в подвійні апострофи, то його тип - "String ". Якщо не одна з умов не виконується, то тип аргументу - "Identifier". Програма виводить кількість заданих аргументів і, для кожного аргументу, його тип і значення.

## Хід виконання роботи

1. Завантажте IntelliJ IDEA.
2. Створіть проект Maven.
3. Розкрийте дерево проекту і перейдіть в папку **java**.
4. Додайте до неї новий файл класу, в якому реалізуйте виконання завдання вашого варіанту.
5. Створіть unit-тест для тестування методів розробленого класу.
6. Запустіть тест на виконання.
7. Якщо тест не пройдений, внесіть необхідні виправлення в код програми і перейдіть до п.6.

8. Якщо тест успішно пройдений, перейдіть до п.9.
9. Повторіть пп.4-8 для інших завдань вашого варіанту.
10. Оформіть звіт про виконання роботи.

## Контрольні запитання

1. Яке призначення класів **String** і **StringBuffer**?
2. Якими способами можна створити об’єкт класу **String**?
3. Якими методами можна скористатись при роботі з об’єктом класу **String**?
4. Які конструктори передбачені в класі **StringBuffer**?
5. Як можна перетворити об’єкт класу **String** в об’єкт класу **StringBuffer** чи навпаки?
6. Яку роль відіграє метод **equals** ()і для рядків **String** і **StringBuffer**?
7. Як здійснюється пошук в рядках класу **String**?
8. Як отримати символ рядка класу **String** за індексом?
9. Чи можуть  рядках класу **String** модифікуватись?
10. Яке призначення регулярних виразів?
11. Які метасимволи використовуються в регулярних виразах? Яке їх призначення?
12. Які класи Java призначені для роботи з регулярними виразами?
13. Як протестувати створений регулярний вираз?
14. Чи підтримує клас **String** роботу з регулярними виразами?

# ПРАКТИКУМ 4. ВИКОРИСТАННЯ КОЛЕКЦІЙ

## Короткі теоретичні відомості про колекції

### Компоненти колекцій

Колекція, іноді звана контейнером, - це об'єкт, який об'єднує кілька елементів в один об'єкт [4-6,10]. Колекції використовуються для зберігання даних, доступу та маніпуляцій з даними, а також для передачі даних від одного методу до іншого.

Масив також можна розглядати як колекцію, яка об'єднує дані одного типу, елементи якої розташовані послідовно в порядку зростання індексу.

Всі інтерфейси і класи, що відносяться до колекцій, знаходяться в пакеті **java.util**.

Схема колекцій (collections framework) - це уніфікована архітектура для подання і маніпулювання колекціями. Всі схеми колекцій містять такі три компоненти:

1. **Інтерфейси** - абстрактні типи даних, що представляють колекції. Інтерфейси дозволяють маніпулювати колекціями незалежно від деталей їх подання. В Java, як і в інших об'єктно-орієнтованих мовах, ці інтерфейси зазвичай утворюють ієрархію.
2. **Реалізації** - конкретні реалізації інтерфейсів колекції. За своєю суттю вони є повторно використовуваними структурами даних.
3. **Алгоритми** - методи, які виконують деякі корисні дії, наприклад, пошук або сортування, над об'єктами, що реалізують інтерфейси колекцій. Ці методи називають поліморфними, так як один і той же метод може використовуватися в багатьох різних реалізаціях відповідного інтерфейсу колекцій. По суті алгоритми забезпечують повторно використовувану функціональність.

## Інтерфейс Collection

Інтерфейс **Collection** представляє групу об'єктів, відомих як його елементи. Пакет SDK не забезпечує прямої реалізації цього інтерфейсу і він використовується для передачі колекцій і маніпулювання ними, коли потрібне максимальне узагальнення.

Інтерфейс **Collection** оголошує методи, які виконують такі операції:

- визначення кількості елементів в колекції (метод **int size** ());
- перевірка, чи містить колекція елементи (метод **boolean isEmpty** ());
- перевірка, чи перебуває даний елемент в колекції (метод **boolean contains** (**Object** element));
- порівняння заданого об'єкта з даної колекцією на рівність (метод **public boolean equals** (**Object** o));
- додавання елемента в колекцію (метод **boolean add** (**Object** element)) і видалення елемента з колекції (метод **boolean remove** (**Object** element)), а також очищення колекції (метод **public void clear** ());
- забезпечення ітераційних операцій над колекцією (метод **Iterator iterator** ());
- забезпечення моста між колекціями і старими інтерфейсами прикладних програм, які передбачали передачу їм параметрів об'єкта у вигляді масиву (методи **Object** [] **toArray** () і **Object** [] **toArray** (Object a [])).

Крім цього, для колекцій визначені наступні групові операції:

- **boolean containsAll** (**Collection** c),
- **boolean addAll** (**Collection** c),
- **boolean removeAll** (**Collection** c),
- **boolean retainAll** (**Collection** c),

які дозволяють перевірити, чи містяться всі елементи колекції c в даній колекції; додати всі елементи колекції c до даної колекції; видалити з даної

колекції всі елементи, що містяться в колекції c або залишити в даній колекції тільки ті елементи, які містяться в колекції c.

Метод **iterator** повертає об'єкт інтерфейсу **Iterator**. Інтерфейс **Iterator** оголошує наступні методи для колекцій:

- **boolean hasNext** (), перевіряючий, чи є наступний елемент,
- **Object next** (), який повертає наступний елемент і
- **void remove** (), що видаляє даний елемент.

Нижче наводиться приклад, як об'єкт **Iterator** використовується для фільтрації колекції, тобто видалення з колекції елементів, що відповідають певним умовам:

```
static void filter (Collection c)
{
    for (Iterator i = c.iterator (); i.hasNext ();)
        if (! умова (i.next ()))
            i.remove ();
}
```

Даний код є поліморфним, тобто буде працювати для будь-якої колекції, що підтримує видалення елементів, незалежно від реалізації.

### Інтерфейс Set

Інтерфейс **Set** є колекцією, яка не повинна містити повторюваних елементів. Інтерфейс розширює інтерфейс **Collection** і містить ті ж методи, що і батьківський інтерфейс. Однак методи в цьому інтерфейсі мають більш конкретний математичний сенс. Так наприклад:

- вираз s1.**containsAll** (s2) повертає true, якщо s2 є підмножиною s1;
- вираз s1.**addAll** (s2) перетворює s1 в об'єднання s1 і s2;
- вираз s1.**retainAll** (s2) перетворює s1 в перетин s1 і s2 (перетин двох множин містить елементи, загальні для обох множин);

- вираз s1.**removeAll** (s2) видаляє з s1 всі елементи, що містяться в s2.

### Інтерфейс List

Інтерфейс **List** є впорядкованою колекцією (іноді званої послідовністю). Колекція **List** може містити впорядковані елементи. На додаток до операцій, успадкованим від **Collection**, інтерфейс оголошує наступні операції:

- отримання значення елемента із заданим індексом (метод **public Object get** (**int** index)) і установка значення елемента із заданим індексом (метод **public Object set** (**int** index, **Object** element));
- пошук елемента в списку і повернення значення його індексу (методи **public int indexOf** (**Object** o) і **public int lastIndexOf** (**Object** o));
- додавання (метод **public void add** (**int** index, **Object** element)) або видалення (метод **public Object remove** (**int** index)) елемента по заданому індексу, а також очищення списку (метод **public void clear** ());
- операції над частиною списку (метод **public List subList** (**int** fromIndex, **int** toIndex));
- ітераційні операції в списку (метод **public ListIterator listIterator** () і **public ListIterator listIterator** (**int** index)).

Інтерфейс **ListIterator** розширює інтерфейс **Iterator** і містить наступні методи перегляду списку:

- **Object next** () - повертає наступний елемент (якщо наступного елемента немає, то викидається виключення типу **NoSuchElementException**);
- **Object previous** () - повертає попередній елемент (якщо попереднього елемента немає, то викидається виключення типу **NoSuchElementException**);

- **boolean hasNext** () - повертає true, якщо існує наступний елемент, інакше повертає false;
- **boolean hasPrevious** () - повертає true, якщо існує попередній елемент, інакше повертає false;
- **int nextIndex** () - повертає індекс наступного елемента (якщо наступного елемента немає, повертає розмір списку);
- **int previousIndex** () - повертає індекс попереднього елемента (якщо попереднього елемента немає, повертає -1);
- **void add** (**Object** obj) - вставляє obj в списку перед елементом, який буде повернений наступним викликом next ();
- **void remove** () - видаляє поточний елемент зі списку (викидається виключення типу **IllegalStateException**, якщо метод **remove** () викликається, перш ніж викликаний метод **next** () або **previous** ());
- **void set** (**Object** obj) - призначає obj на поточний елемент (це останній елемент, повернутий викликом методу **next** () або **previous** ()).

### Інтерфейс Comparable

Інтерфейс **Comparable** дозволяє розробнику класу задати спосіб сортування, який може використовуватися в колекціях об'єктів цього класу.

У багатьох класах інтерфейс **Comparable** вже реалізований, наприклад, в класах **Byte**, **Character**, **String**, **Short** і т.д. Конкретні реалізації цього методу в цих об'єктних розширеннях числових класів дозволяють порівнювати числа за значенням, для рядків - порівнювати рядки в лексикографічному порядку, для дат - в хронологічному порядку. Сортування, заснована на такому порівнянні, називається *природним* порядком порівняння.

В інтерфейсі **Comparable** оголошується єдиний метод - **compareTo** (**Object** obj)). Метод повинен повертати значення 0, якщо порівнювані

об'єкти дорівнюють один одному; значення, менше нуля, якщо приймає об'єкт менше obj; значення, більше нуля, якщо приймає об'єкт більше obj.

Приклад класу з "вбудованої" сортуванням по полю **age**:

```
class Person implements Comparable <Person> {
   String name;
   int age;

   @Override
   public int compareTo (Person o) {
      if (this.age == o.age) {return 0;}
      else if (this.age <o.age) {return -1;}
      else {return 1;}
   }
}
```

Після додавання таких об'єктів в колекцію, скажімо, **TreeSet** <Person> при проходженні за допомогою **foreach** вони виявляться впорядкованими за віком.

Якщо перевантажений метод **compareTo** прибрати з классу, то спроба додати об'єкт в колекцію буде приводити до викидання виключення.

### Інтерфейс Comparator

Але часто необхідно впорядкувати елементи колекції в порядку, відмінному від природного або впорядкувати елементи без реалізації інтерфейсу **Comparable**. В цьому випадку необхідно реалізувати інтерфейс **Comparator**.

В інтерфейсі **Comparator** оголошений метод **compare** (**Object** obj1, **Object** obj2), який дозволяє порівнювати між собою два об'єкти. На виході метод повертає значення 0, якщо об'єкти рівні, позитивне значення або негативне значення, якщо об'єкти не тотожні.

У найпростішому випадку сортування можна виконати методом **sort** самої колекції, наприклад, **ArrayList**.

Для сортування об'єктів в колекції можна використовувати метод **sort** класу **Collections**, який в якості першого вхідного аргументу приймає список об'єктів:

Collections.sort (List <T> arg1, Comparator <? Super T> arg2);

Сортування може виконується також за допомогою класу **Arrays**, у якого є метод **sort**. Даний метод в якості другого аргументу приймає тип компаратора.

Arrays.sort (T [] arg1, Comparator <? Super T> arg2);

Наприклад, для сортування класу **Person** з попереднього прикладу можна створити такий компаратор:

```
class AgeComparator implements Comparator <Person> {
   @Override
   public int compare (Person o1, Person o2) {
      if (o1.age == o2.age) {return 0;}
      else if (o1.age <o2.age) {return -1;}
      else {return 1;}
   }
}
```

Якщо кілька екземплярів класу **Person** додані в колекцію **ArrayList** <Person> persons, то їх відсортований список можна отримати так:

persons.**sort** (**new** AgeComparator ());

або так:

**Collections**.**sort** (persons, **new** AgeComparator ());

### Інтерфейс SortedSet

Інтерфейс **SortedSet** є розширенням інтерфейсу **Set**, елементи якого відсортовані в природному порядку або згідно з порядком, що задається методом інтерфейсу **Comparator**.

Інтерфейс реалізує наступні операції (додатково до операцій інтерфейсу **Set**):

- операції над частиною набору (метод **public SortedSet subSet** (**Object** fromElement, **Object** toElement)), головною частиною набору (метод **public SortedSet headSet** (**Object** toElement)) і хвостовою частиною набору (метод **public SortedSet tailSet** (**Object** fromElement));
- повернення першого (метод **public Object first** ()) і останнього (метод **public Object last** ()) елемента відсортованого набору;
- доступ до інтерфейсу **Comparator** (метод **public Comparator comparator** ()).

### Інтерфейс Map

Інтерфейс **Map** є об'єктом, який ставить у відповідність ключам значення. Ключі повинні бути унікальними і їм повинно відповідати єдине значення. Таку колекцію називають відображенням (map), словником (dictionary) або асоціативним масивом (associative array).

В інтерфейсі **Map** визначені наступні основні операції:

- отримання розміру відображення (метод **public int size** ());
- перевірка відображення на наявність елементів (метод **public boolean isEmpty** ());

- розміщення елемента в відображення (метод **public Object put** (**Object** key, **Object** value)) і отримання значення елемента із заданим ключем (метод **public Object get** (**Object** key));
- порівняння відображення із заданим об'єктом на рівність (метод **public boolean equals** (**Object** o));
- надання ключів колекції у вигляді множини (метод **public Set keySet** ());
- видалення елемента із заданим ключем (метод **public Object remove** (**Object** key)) і очищення відображення (метод **public void clear** ());
- перевірка наявності елемента із заданим ключем (метод **public boolean containsKey** (**Object** key)) або значенням (метод **public boolean containsValue** (**Object** value));
- додавання до даного відображення всіх пар із заданого відображення (метод **public void putAll** (**Map** t));
- представлення всіх значень даного відображення у вигляді колекції (метод **public Collection values** ());
- представлення колекції у вигляді множини, кожен елемент якої - пара з даного відображення, з якою можна працювати методами вкладеного інтерфейсу **Map**.**Entry** (метод **public Set entrySet** ()).

Інтерфейс **Map**.**Entry** описує методи роботи з парами «ключ-значення», отриманими методом **entrySet** ():

- отримання ключа (метод **public Object getKey** ()) і значення (метод **public Object getValue** ());
- зміна значення (метод **public Object setValue** (**Object** value));
- порівняння заданого об'єкта з даною парою «ключ-значення» на рівність (метод **public boolean equals** (**Object** o)).

Можливо також використання об'єктів **Map**, в яких ключам відповідає кілька значень. Це досягається зазвичай, якщо в якості значень задати об'єкти **List**.

### Інтерфейс SortedMap

Інтерфейс **SortedMap** є розширенням **Map**. Елементи для цього інтерфейсу відсортовані в природному порядку або згідно з порядком, що задається методами інтерфейсу **Comparator**.

Інтерфейс реалізує наступні операції (додатково до операцій інтерфейсу **Map**):

- виділення частини відображення (метод **public SortedMap subMap** (**Object** fromKey, **Object** toKey)), головної частини відображення (метод **public SortedMap subMap** (**Object** fromKey, **Object** toKey)) або хвостовій частині відображення (метод **public SortedMap tailMap** (**Object** fromKey));
- отримання першого (метод **public Object firstKey** ()) і останнього (метод **public Object lastKey** ()) елемента відображення;
- доступ до інтерфейсу **Comparator** (метод **public Comparator comparato**r ()).

В цілому, інтерфейс **SortedMap** є аналогом інтерфейсу **SortedSet**.

## Реалізації колекцій і алгоритми

### Клас HashSet

Клас **HashSet** реалізує інтерфейс **Set** с підтримкою хеш-таблиці (зазвичай є реалізацією **HashMap**) і має наступні конструктори:

- **HashSet** () - створює новий, порожній набір з ємністю за замовчуванням і фактором завантаження, рівним 0.75;
- **HashSet** (**Collection** c) - створює новий набір, що містить елементи заданої колекції;
- **HashSet** (**int** initialCapacity) - створює новий порожній набір із заданою ємністю і фактором завантаження, рівним 0.75;
- **HashSet** (**int** initialCapacity, **float** loadFactor) - створює новий порожній набір із заданою ємністю за замовчуванням і заданим фактором завантаження.

У класі **HashSet** реалізовані наступні методи інтерфейсу **Set**: **add** (), **clear** (), **contains** (), **isEmpty** (), **iterator** (), **remove** () і **size** ().

Метод **public Object clone** () дозволяє отримати порожню копію об'єкта **HashSet** (без елементів).

### Клас ArrayList

Клас **ArrayList** забезпечує реалізацію інтерфейсу **Set** для масивів із змінними розмірами. Так само, як об'єкти класу **StringBuffer**, об'єкти **ArrayList** мають дві характеристики - ємність і розмір масиву. Якщо розмір списку перевищить ємність, ємність автоматично збільшується на деяку кількість елементів.

Клас **ArrayList** має наступні конструктори:

- **ArrayList** () - створює порожній список;
- **ArrayList** (**Collection** c) - створює список з заданої колекції в порядку, заданому ітератором даної колекції;
- **ArrayList** (**int** initialCapacity) - створює порожній список із заданою ємністю.

Клас **ArrayList** реалізує наступні методи інтерфейсу **List**: **add** (), **addAll** (), **clear** (), **contains** (), **get** (), **indexOf** (), **isEmpty** (), **lastIndexOf** (), **remove** (), **set** (), **size** () і **toArray** ().

Крім того, клас містить наступні власні методи:

- **public Object clone** () - отримання порожній копії об'єкта **ArrayList** (без елементів);
- **public void ensureCapacity** (**int** minCapacity) - збільшує ємність примірника **ArrayList**, якщо це необхідно, для того, щоб він міг утримувати число елементів, заданий в **minCapacity**;
- **protected void removeRange** (**int** fromIndex, **int** toIndex) - видаляє з елементи в заданому діапазоні індексів;
- **public void trimToSize** () - зменшує ємність об'єкта **ArrayList** до його попереднього значення.

### Клас HashMap

Клас **HashMap** є реалізацією інтерфейсу **Map** з використанням хеш-таблиць.

Клас **HashMap** має наступні конструктори:

- **HashMap** () - створює нове, пусте відображення з ємністю за замовчуванням 16 і фактором завантаження, рівним 0.75;
- **HashMap** (**Map** m) - створює нове відображення, що містить елементи заданого відображення;
- **HashMap** (**int** initialCapacity) - створює нове, пусте відображення із заданою ємністю і фактором завантаження, рівним 0.75;
- **HashMap** (**int** initialCapacity, **float** loadFactor) - створює нове, пусте відображення із заданою ємністю і заданим фактором завантаження.

Клас **HashMap** реалізує наступні методи інтерфейсу **Map**: **clear** (), **containsKey** (), **containsValue** (), **entrySet** (), **get** (), **isEmpty** (), **keySet** (), **put** (), **putAll** (), **remove** (), **size** ( ) і **values** ().

### Клас Collections

Клас **Collections** оперує колекціями або повертає колекції. Клас не містить конструктора, а тільки такі **public static** методи:

- **int binarySearch** (**List** list, **Object** key) - двійковий пошук в заданому списку заданого об'єкта;
- **int binarySearch** (**List** list, **Object** key, **Comparator** c) - двійковий пошук в заданому списку заданого об'єкта з використанням компаратора;
- **void copy** (**List** dest, **List** src) - копіювання списку-джерела в список призначення;
- **void fill** (**List** dest, **Object** o) - заповнення списку заданим об'єктом;

- **Object max** (**Collection** c) і **Object min** (**Collection** c) - повернення максимального або мінімального елемента колекції відповідно до природного порядком порівняння;
- **Object max** (**Collection** c, **Comparator** com) і **Object min** (**Collection** c, **Comparator** com) - повернення максимального або мінімального елемента колекції відповідно до порядку, заданих за допомогою компаратора;
- **void reverse** (**List** l) - переставляє елементи списку в зворотному порядку;
- **void shuffle** (**List** l) - перемішує елементи списку з використанням генератора випадкових чисел за замовчуванням;
- **void shuffle** (**List** l, **Random** rnd) - перемішує елементи списку з використанням заданого генератора випадкових чисел;
- **void sort** (**List** l) і **void sort** (**List** l, **Comparator** c) - сортує список по зростанню відповідно до природного порядке або з використанням компаратора;
- **Collection unmodifiableCollection** (**Collection** c), **List unmodifiableList** (**List** l), **Map unmodifiableMap** (**Map** m), **Set unmodifiableSet** (**Set** s), **SortedSet unmodifiableSortedSet** (**SortedSet** ss) і **SortedMap unmodifiableSortedMap** (**SortedMap** sm) - створюють незмінні екземпляри відповідних типів колекцій.

**Варіанти завдань**

| № | Завдання | № | Завдання | № | Завдання | № | Завдання |
|---|---|---|---|---|---|---|---|
| **1** | 1,5,16,30 | **7** | 8,13,18,22 | 13 | 2,8,14,28 | 19 | 8,15,22,27 |
| **2** | 2,13,17,29 | **8** | 9,14,23,25 | 14 | 3,5,19,29 | 20 | 9,13,21,26 |
| **3** | 3,15,18,28 | **9** | 5,10,24,30 | 15 | 4,15,26,30 | 21 | 10,14,20,25 |
| **4** | 4,14,19,27 | **10** | 11,15,25,29 | 16 | 5,13,17,25 | 22 | 5,11,19,24 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **5** | 5,6,20,26 | **11** | 12,14,26,30 | 17 | 5,7,18,24 | 23 | 12,15,18,28 |
| **6** | 7,15,17,21 | **12** | 1,7,13,27 | 18 | 1,6,14,23 | 24 | 1,13,17,22 |

1. Створіть додаток для додавання абонентів і перегляду списку абонентів телефонної мережі. Записи в списку (5 записів) створюються в програмі і є об'єктами класу **TreeMap**, де ключем є номер телефону (типу **Integer**), а значенням - об'єкт **Abonent**, що містить чотири значення типу **String**: прізвище, ім'я, по батькові та адреса. Передбачити можливість сортування елементів колекції по 2-3 полях, для чого використовувати клас **ArrayList**.
2. Створіть додаток для пошуку абонентів телефонної мережі. Список абонентів (5 записів) створюється в програмі і є об'єктом класу **HashMap**. Ключем записи є номер телефону (типу **String**), а значенням - об'єкт **Abonent**, що містить чотири значення типу **String**: прізвище, ім'я, по батькові та адреса. Передбачити можливість пошуку абонентів колекції по ключу.
3. Створіть додаток для перегляду списку елементів - цілих чисел (типу **Integer**) і зміни елементів списку в заданому діапазоні індексів. Список (10 чисел) є об'єктом класу **ArrayList**. Передбачити можливість сортування елементів колекції по 2-3 полях. Передбачити можливість розрахунку кількості повторів елементів, для чого використовувати **HashSet**.
4. Створіть додаток для перегляду списку елементів і додавання елементів в список. Список (5 елементів типу **Integer**) створюється в програмі і є об'єктом класу **ArrayList**. Передбачити можливість сортування елементів колекції по 2-3 полях. Передбачити можливість перевірки повторів елементів, для чого використовувати **TreeSet**.
5. Створіть додаток для перегляду списку книг і видалення книг в бібліотечному каталозі. Записи в списку (5 записів) є об'єктами

класу **HashMap**, де ключем є індекс ISBN книги (типу **Integer**), а значенням - об'єкт **Book**, що містить найменування книги, прізвище, ім'я та по батькові (ПІБ) учасника, видавництво (всі поля типу **String**), рік видання (типу **int**) і ціну книги (типу **float**). Передбачити можливість сортування книг по 2-3 полях, для чого використовувати **ArrayList**.

6. Створіть додаток для додавання книг і перегляду списку книг в бібліотечному каталозі. Список книг (5 записів) створюється в програмі і є об'єктом класу **HashMap**. Ключем записи є індекс ISBN книги (типу Integer), а значенням - об'єкт **Book**, що містить найменування книги, прізвище, ім'я та по батькові (ПІБ) учасника, видавництво (всі записи типу **String**), рік видання (типу **int**) і ціну книги (типу **float**). Передбачити можливість отримання списку авторів книг, де автори не повторюються для чого використовувати **TreeSet**.
7. Створіть додаток для перегляду списку черговиків. Список (5 записів) створюється в програмі і є об'єктом класу **LinkedList**. Запис списку є об'єктом **QueuePerson**, що містить поля прізвища, імені та по батькові черговика (типу **String**), поле типу **AddressValue** для адреси черговика і пріоритет черговика (типу **int**). У свою чергу, поле типу **AddressValue** містить три поля типу **String**: найменування міста, найменування вулиці та номер будинку, а також поле типу **int** - номер квартири (якщо номер квартири дорівнює 0, будинок, в якому проживає суб'єкт, не має квартир). Записи в черзі мають бути упорядковані відповідно до пріоритету і черговик додається останнім в чергу свого пріоритету. Забезпечити, щоб в черзі не було повторення по ПІБ черговика, для чого використовувати **TreeSet**.
8. Створіть додаток для перегляду списку черговиків і зміни пріоритету черговиків в черзі. Список (5 записів) створюється в програмі і є об'єктом класу **LinkedList**. Запис списку є об'єктом

**QueuePerson**, що містить поля прізвища, імені та по батькові черговика (типу **String**), поле типу **AddressValue** для адреси черговика і пріоритет черговика (типу **int**). У свою чергу, поле типу **AddressValue** містить три поля типу **String**: найменування міста, найменування вулиці та номер будинку, а також поле типу **int** - номер квартири (якщо номер квартири дорівнює 0, будинок, в якому проживає черговик, не має квартир). Записи в черзі упорядковуються відповідно до пріоритету, і черговик додається останнім в чергу свого пріоритету.

9. Створіть додаток для перегляду списку зображень і видалення зображення зі списку зображень. Список (5 зображень) створюється в програмі і є об'єктом класу **HashMap**, де ключем є найменування зображення (типу **String**), а значенням - зображення (об'єкт класу **Image**). Передбачити можливість сортування зображень за допомогою **ArrayList**.
10. Створіть додаток для пошуку зображення в списку зображень. Список (5 зображень) створюється в програмі і є об'єктом класу **HashMap**, де ключем є найменування зображення (типу **String**), а значенням - зображення (об'єкт класу **Image**). За допомогою **TreeSet** забезпечити, щоб зображення додавалися з унікальними іменами.
11. Створіть додаток для зміни значення елементів та перегляду списку елементів у множині цілих чисел типу **Integer**. Множина (10 елементів) створюється в програмі і є об'єктом класу **HashSet** (елементи множини не повинні повторюватись). Передбачити можливість сортування за допомогою **ArrayList**.
12. Створіть додаток для додавання елементів і перегляду списку елементів у множині цілих чисел типу **Integer**. Множина (10 елементів) створюється в програмі і є об'єктом класу **HashSet** (елементи множини не повинні повторюватись). Передбачити можливість сортування за допомогою **TreeSet**.

13. Створіть додаток для пошуку та зміни абонента бібліотечної мережі. Список абонентів (5 записів) створюється в програмі і є об'єктом класу **HashMap**. Ключем запису є номер абонента (типу **Integer**), а значенням - об'єкт **Abonent**, що містить чотири значення типу **String**: прізвище, ім'я, по батькові та адреса. Передбачити можливість сортування на прізвище за допомогою **ArrayList**.
14. Створіть додаток для видалення абонента і перегляду списку абонентів бібліотечної мережі. Список абонентів (5 записів) створюється в програмі і є об'єктом класу **HashMap**. Ключем записи є номер телефону (типу **Integer**), а значенням - об'єкт **Abonent**, що містить чотири значення типу **String**: прізвище, ім'я, по батькові та адреса. Передбачити можливість сортування по прізвищу за допомогою **TreeMap**.
15. Створіть додаток для покупки продуктів в магазині з доставкою. Список продуктів (5 записів) створюється в програмі і є об'єктом класу **HashMap**. Ключем запису є артикул продукту (типу **Integer**), а значенням - об'єкт **Article**, що містить найменування продукту (типу **String**) і ціну товару (типу **float**). Передбачити можливість перевірки унікальності найменування товару за допомогою **TreeSet**.
16. Створіть додаток для перегляду списку товарів і зміни ціни товару в магазині. Список товарів (5 записів) створюється в програмі і є об'єктом класу **TreeMap**. Ключем записи є артикул товару (типу **Integer**), а значенням - об'єкт **Article**, що містить найменування товару (типу **String**) і ціну товару (типу **float**). Передбачити можливість перевірки унікальності найменування товару за допомогою **HashSet**.
17. Створіть додаток для пошуку книг в бібліотечному каталозі по заданому критерію. Список книг (5 записів) створюється в програмі і є об'єктом класу **HashMap**. Ключем записи є індекс

ISBN книги (типу **Integer**), а значенням - об'єкт **Book**, що містить найменування книги, прізвище, ім'я та по батькові (ПІБ) учасника, видавництво (всі записи типу **String**), рік видання (типу **int**) і ціну книги (типу **float**). Передбачити можливість сортування каталогу по ПІБ автора або роком видання, для чого використовувати **ArrayList**. Передбачити можливість перевірки унікальності книги за допомогою **TreeSet**.

18. Створіть додаток для перегляду списку книг в магазинному каталозі і зміни ціни книги. Список книг (5 записів) створюється в програмі і є об'єктом класу **HashMap**. Ключем записи є індекс ISBN книги (типу **Integer**), а значенням - об'єкт **Book**, що містить найменування книги, ПІБ автора, видавництво (типу **String**), рік видання (типу **int**) і ціну книги (типу **float**). Передбачити можливість сортування книг по видавництву або ціною, використовуючи **ArrayList**.
19. Створіть додаток для додавання черговика в чергу і перегляду черги. Список (5 записів) створюється в програмі і є об'єктом класу **LinkedList**. Запис списку є об'єктом **QueuePerson**, що містить поля прізвища, імені та по батькові черговика (всі поля типу **String**), поле типу **AddressValue** для адреси черговика і пріоритет черговика (типу **int**). У свою чергу, поле типу **AddressValue** містить три поля типу **String**: найменування міста, найменування вулиці та номер будинку, а також поле типу int - номер квартири (якщо номер квартири дорівнює 0, будинок, в якому проживає суб'єкт, не має квартир). Записи в черзі упорядковано відповідно до пріоритету і черговик додається останнім в чергу свого пріоритету. Передбачити можливість розрахунку кількості черговиків кожного пріоритету, використовуючи **HashMap**.
20. Створіть додаток для вибірки черговика з черги. Список (5 записів) створюється в програмі і є об'єктом класу **LinkedList**.

Запис списку є об'єктом **QueuePerson**, що містить поля прізвища, імені та по батькові черговика (всі поля типу **String**), поле типу **AddressValue** для адреси черговика і пріоритет черговика (типу **int**). У свою чергу, поле типу **AddressValue** містить три поля типу **String**: найменування міста, найменування вулиці та номер будинку, а також поле типу **int** - номер квартири (якщо номер квартири дорівнює 0, будинок, в якому проживає суб'єкт, не має квартир). Записи в черзі упорядковано відповідно до пріоритету, і черговик додається останнім в черзі свого пріоритету.

21. Створіть додаток для перегляду списку елементів в масиві байтових чисел і видалення елементів списку. Список (5 елементів типу **Byte**) створюється в програмі і є об'єктом класу **ArrayList**.
22. Створіть додаток для перегляду списку елементів і зміни значень елементів списку. Список (5 елементів типу **Byte**) створюється в програмі і є об'єктом класу **ArrayList**.
23. Створіть додаток для додавання елемента в стек з пріоритетами і перегляду стека. Стек (5 записів) створюється в програмі і є об'єктом класу **LinkedList**. Запис в стек є об'єктом **StackMember**, що містить найменування програми (типу **String**), обсяг пам'яті для програми в мегабайтах (типу **int**) і пріоритет програми (типу **int**). Записи в стеці упорядковано відповідно до пріоритету і черговик додається першим в чергу свого пріоритету.
24. Створіть додаток для вибірки елемента з стека з пріоритетами. Стек (5 записів) створюється в програмі і є об'єктом класу **LinkedList**. Запис в стек є об'єктом **StackMember**, що містить найменування програми (типу **String**), обсяг пам'яті для програми в мегабайтах (типу **int**) і пріоритет програми (типу **int**). Записи в стеці упорядковано відповідно до пріоритету і черговик додається першим в чергу свого пріоритету.

25. Створіть додаток для перегляду списку товарів і видалення товару в електронному магазині. Список товарів (5 записів) створюється в програмі і є об'єктом класу **HashMap**. Ключем записи є артикул товару (типу **Integer**), а значенням - об'єкт **Article**, що містить найменування товару (типу **String**) і ціну товару (типу **float**).
26. Створіть додаток для додавання товару і перегляду списку товарів електронного магазину. Список товарів (5 записів) створюється в програмі і є об'єктом класу **HashMap**. Ключем запису є артикул товару (типу **Integer**), а значенням - об'єкт **Article**, що містить найменування товару (типу **String**) і ціну товару (типу **float**).
27. Створіть додаток для перегляду списку студентів і зміни значень елементів списку. Список (5 елементів) створюється в програмі і є об'єктом класу **ArrayList**. Кожен елемент списку містить прізвище, ім'я та по батькові (ПІБ) студента (типу **String**), ім'я групи (типу **String**), дату народження (типу **BirthDate**) і середню оцінку за минулу сесію (типу **float**). Об'єкт класу **BirthDate** в свою чергу містить три елементи типу **int** (рік, номер місяця і день народження).
28. Створіть додаток для перегляду, додавання і видалення елементів списку студентів. Список (5 елементів) створюється в програмі і є об'єктом класу **ArrayList**. Кожен елемент списку містить прізвище, ім'я та по батькові (ПІБ) студента (типу **String**), ім'я групи (типу **String**), дату народження (типу **BirthDate**) і середню оцінку за минулу сесію (типу **float**). Об'єкт класу **BirthDate** в свою чергу містить три елементи типу **int** (рік, номер місяця і день народження).
29. Створіть додаток для перегляду списку файлів і додавання файлу в список. Список (для 5 текстових файлів з розширенням .txt) створюється в програмі і є об'єктом класу **HashMap**, де ключем є

ім'я файлу (типу **String**), а значенням - об'єкт типу **TextFile**, що містить два елементи **String** - абсолютний шлях до файлу (без імені файлу ) і короткий опис вмісту файлу.

30. Створіть додаток для пошуку в списку файлів і видалення файлу зі списку. Список (для 5 текстових файлів з розширенням .txt) створюється в програмі і є об'єктом класу **HashMap**, де ключем є ім'я файлу (типу **String**), а значенням - об'єкт типу **TextFile**, що містить два елементи **String** - абсолютний шлях до файлу (без імені файлу ) і короткий опис вмісту файлу.

## Хід виконання роботи

1. Завантажте IntelliJ IDEA.
2. Створіть проект Maven.
3. Розкрийте дерево проекту і перейдіть в папку **java**.
4. Додайте до неї новий файл класу, в якому реалізуйте виконання завдання вашого варіанту.
5. Створіть unit-тест для тестування методів розробленого класу.
6. Запустіть тест на виконання.
7. Якщо тест не пройдений, внесіть необхідні виправлення в код програми і перейдіть до п.6.
8. Якщо тест успішно пройдений, перейдіть до п.9.
9. Повторіть пп.4-8 для інших завдань вашого варіанту.
10. Оформіть звіт про виконання роботи.

## Контрольні запитання

1. Що називають колекціями в Java? Яке призначення колекцій?
2. Який пакет Java треба імпортувати в програму, щоб отримати можливість користування колекціями?
3. Яке призначення інтерфейсу **Collection**? Які методи в ньому оголошені?
4. Як і для чого використовується ітератор колекції?

5. Яке призначення інтерфейсу **Set**?
6. Який інтерфейс містить опис методів для роботи з упорядкованою колекцією?
7. Які можливості надає інтерфейс **Comparable** для упорядкування елементів колекцій?
8. Як інтерфейс **Comparator** може використовуватись для визначення власного порядку сортування в колекції?
9. Який інтерфейс може бути використаний для створення словника в Java?
10. Які відмінності існують між інтерфейсами **Set** і **SortedSet** (або **Map SortedMap**)?

# ПРАКТИКУМ 5. КЛАСИ І ЇХНЄ ДОКУМЕНТУВАННЯ

## Короткі теоретичні відомості про роботу з класами

### Оголошення класу

Клас визначається за допомогою ключового слова **class**:

***class*** Ім'я_класу {
// Поля
// Методи
}

Для зберігання стану об'єкта в класі застосовуються *поля* або *змінні* класу. Для визначення поведінки об'єкта в класі застосовуються *методи* [1-4].

Крім звичайних методів класи можуть визначати спеціальні методи, які називаються *конструкторами*. Конструктори викликаються при створенні нового об'єкта даного класу. Конструктори виконують ініціалізацію об'єкта:

class ***Ім'я_класу*** {
// Конструктор
***Ім'я_класу*** (параметри) {// ...}
}

Якщо в класі не визначено жодного конструктора, то для цього класу автоматично створюється конструктор без параметрів.

Для створення об'єкта використовується оператор **new**:

Ім'я_класу Ідентифікатор = ***new*** ім'я_класу ().

Ключове слово **this** є посиланням на поточний екземпляр класу. Через це ключове слово ми можемо звертатися до змінних, методів об'єкта, а також викликати його конструктори.

Крім конструктора початкову ініціалізацію об'єкта цілком можна проводити за допомогою *ініціалізатора* об'єкта. Ініціалізатор виконується до будь-якого конструктора:

```
class ім'я_класу {
        / * Початок блоку ініціалізатор * /
    {
        // поле = значення;
        // ...
    }
     / * Кінець блоку ініціалізатор * /

    // Конструктори
    // ...
}
```

## Модифікатори доступу

Java надає ряд модифікаторів доступу, щоб задати рівні доступу для класів, змінних, методів і конструкторів. Існує чотири рівні доступу:

1. Видимий в пакеті (стоїть за умовчанням і модифікатор не потрібні).
2. Видимий тільки для класу (**private**).
3. Видимий для всіх (**public**).
4. Видимий для пакета і всіх підкласів (**protected**).

Модифікатор **private** є найбільш обмежуючим рівнем доступу. Клас і інтерфейси не можуть бути **private**. Змінні, оголошені як **private**, можуть бути доступні поза класом, якщо отримують їх відкриті (**public**) методи присутні в класі. Використання модифікатора **private** в Java є основним способом, щоб приховати дані.

Модифікатор **public** - клас, метод, конструктор, інтерфейс і т.д. можуть бути доступні з будь-якого іншого класу. Проте якщо ми намагаємося отримати доступ до **public** класу з іншого пакету, то його доводиться імпортувати. В Java все публічні методи і змінні класу успадковуються його підкласами.

Модифікатор **protected** - змінні, методи і конструктори в суперкласі можуть бути доступні тільки для підкласів в іншому пакеті або для будь-якого класу в пакеті класу **protected**. Модифікатор доступу **protected** в Java не може бути застосований до класу і інтерфейсу.

### Забезпечення інкапсуляції

Якщо змінна має рівень доступу **private**, до неї неможливо звернутися ззовні класу, в якому вона оголошена. Але все одно необхідний спосіб звернення до **private** змінних з іншого класу, інакше такі ізольовані змінні не матимуть сенсу. Це досягається за допомогою оголошення спеціальних **public** методів. Методи, які повертають значення змінних, називаються *геттери*. Методи, які змінюють значення властивостей, називаються *сеттери*.

Існують правила оголошення таких методів:

- Якщо властивість НЕ типу **boolean**, префікс геттера має бути **get**. Наприклад: *getName* () це коректне ім'я геттера для змінної name.
- Якщо властивість типу **boolean**, префікс імені геттера може бути **get** або **is**. Наприклад, *getPrinted* () або *isPrinted* () обидва є коректними іменами для змінних типу **boolean**.
- Ім'я сетера має починатися з префікса **set**. Наприклад, *setName* () коректне ім'я для змінної *name*.
- Для створення імені геттера або сетера, перша буква властивості повинна бути змінена на велику і додана до відповідного префікса (**set**, **get** або **is**).
- Сетер повинен бути **public**, повертати **void** тип і мати параметр відповідно до типу змінної. Наприклад:

public void *setAge* (int *age*) {
this.*age* = *age*;
}

- Геттер метод повинен бути **public**, не мати параметрів методу, і повертати значення відповідне типу властивості. Наприклад:

public int getAge () {
return *age*;
}

У мові Java при проектуванні класів прийнято обмежувати рівень доступу до змінних за допомогою модифікаторів **private** або **protected** і звертатися до них через геттери і сеттери.

## Наслідування класів

Наслідування (inheritance) - механізм, який дозволяє описати новий клас на основі існуючого (батьківського). При цьому властивості і функціональність батьківського класу запозичуються новим класом.

Щоб оголосити один клас спадкоємцем іншого, треба використовувати після імені наслідуваного класу ключове слово **extends**, після якого йде ім'я базового класу:

```
class Child extends Parent {}
```

У Java немає множинного наслідування класів, але є множинне наслідування інтерфейсів.

Коли підкласу потрібно послатися на його безпосередній суперклас використовується ключове слово **super**. У ключового слова **super** є дві загальні форми:

- Для виклику конструктора суперкласу: **super** (списокАргументів);
- Для звернення до члена суперкласу, приховуваного членом підкласу: **super**.метод (списокАргументів).

Похідний клас має доступ до всіх методів і полів базового класу (навіть якщо базовий клас знаходиться в іншому пакеті) крім тих, які визначені з модифікатором **private**. При цьому похідний клас також може додавати свої поля і методи. При цьому в Java успадковувати тільки нестатичні методи.

Дочірній клас може перевизначити методи екземпляра свого батьківського класу. Це називається *перевизначенням* методу. Перевизначення методу виконується для досягнення поліморфізму під час виконання програми.

## Абстрактні класи

В абстрактному класі також можна визначити поля і методи, в той же час не можна створити об'єкт або екземпляр абстрактного класу. Абстрактні класи покликані надавати базовий функціонал для класів-спадкоємців. А похідні класи вже реалізують цей функціонал.

При визначенні абстрактних класів використовується ключове слово **abstract**.

Крім звичайних методів абстрактний клас може містити абстрактні методи. Такі методи визначаються за допомогою ключового слова **abstract** і не мають ніякого функціоналу.

Похідний клас зобов'язаний перевизначити і реалізувати всі абстрактні методи, які є в базовому абстрактному класі. Також слід враховувати, що якщо клас має хоча б один абстрактний метод, то даний клас повинен бути визначений як абстрактний.

## Інтерфейси

Інтерфейси визначають деякий функціонал без реалізації, який потім реалізують класи, які застосовують ці інтерфейси. І один клас може застосувати кілька інтерфейсів.

Щоб визначити інтерфейс, використовується ключове слово **interface**. наприклад:

```
interface Printable {
   void print ();
}
```

Інтерфейс може визначати константи і методи, які можуть мати, а можуть і не мати реалізації. Методи без реалізації схожі на абстрактні методи абстрактних класів.

Всі методи інтерфейсу не мають модифікаторів доступу, але фактично за замовчуванням доступ **public**, так як мета інтерфейсу - визначення функціоналу для реалізації його класом. Тому весь функціонал повинен бути відкритий для реалізації.

Щоб клас застосував інтерфейс, треба використовувати ключове слово **implements**:

```
доступ class ім'я_класу [extends суперклас]
[Implements інтерфейс [, інтерфейс ...]] {
// тіло класу
}
```

## Документування коду за допомогою JavaDoc

### Коментарі Javadoc

Javadoc - стандартний генератор документації в HTML-форматі з коментарів вихідного коду.

Для створення опису до елементу (поле, клас, метод) використовуються спеціальний коментар, розташований вище цього елемента:

```
/ **
  *
  * /
```

У IntelliJ IDEA, якщо ввести / ** і натиснути return, вона автоматично заповнить решту синтаксису.

Документування здійснюється за допомогою спеціальних тегів (анотацій).

Формат для додавання різних елементів виглядає наступним чином:

/ **

* [Short description]

* <P>

* [Long description]

*

* [Tags]

* [See also]

* /

**Основні теги**

Нижче наведені найбільш поширені теги, які використовуються в Javadoc.

@**author** - людина, яка внесла значний внесок у код. Застосовується тільки на рівні класу або пакета.

@**param** - параметр, який приймає метод або конструктор.

@**deprecated** - цим тегом позначаються клас або метод, які більше не використовуються. Супроводжується тегом @see або {@link}.

@**return** - що повертає метод.

@**see** - створює список "див.також". Використовується в парі з тегом {@link} для зв'язку з вмістом.

{@**Link**} - використовується для створення посилань на інші класи або методи. Приклад: {@link Foo # bar} посилається на метод bar, який належить до класу Foo. Для посилання на метод в тому ж класі, просто додається #bar.

@**since** 2.0 - версія з моменту додавання функції.

@**throws** - вид виключення, яке видає метод. В коді повинно бути вказано виключення, в іншому випадку Javadoc видасть помилку.

@**exception** - альтернатива тегу @throws.

@**Override** - використовується з інтерфейсами і абстрактними класами. Виконує перевірку, щоб побачити, чи є метод перевизначенням.

### Порядок тегів

Oracle пропонує наступний порядок тегів:

- @**author** (classes and interfaces)
- @**version** (classes and interfaces)
- @**param** (methods and constructors)
- @**return** (methods)
- @**throws** (@exception is an older synonym)
- @**see**
- @**since**
- @**serial**
- @**deprecated**

### Приклад документування класу

```
/ ** Клас служить для зберігання об'єктів з властивостями
 * <B> maker </ b> і <b> price </ b>.
 * @Autor Filippov Yakov

 * @version 1.0

* /
class Product {
    / ** Властивість - виробник * /
    private String maker;

    / ** Властивість - ціна * /
    public double price;
```

```
    / ** Створює новий порожній об'єкт
     * @See Product # Product (String, double)
    * /
    Product () {
        setMaker ( "");
        price = 0;
    }

    / ** Створює новий об'єкт із заданими значеннями
     * @Param maker - виробник
     * @Param price - ціна
     * @See Product # Product ()
    * /

    Product (String maker, double price) {
        this.setMaker (maker);
        this.price = price;
    }

    / ** Функція для отримання значення поля {@link Product # maker}
     * @Return Повертає назву виробника
     * /
    public String getMaker () {
        return maker;
    }

    public void setMaker (String maker) {
        this.maker = maker;
    }

}
```

### Налаштування IntelliJ IDEA для роботи з Javadoc

Для відображення підказок Javadoc досить в редакторі коду клацнути по імені класу (методу) і натиснути **Ctrl** + **Q**.

За замовчуванням підказка відображається в спливаючому вікні, але таку поведінку можна змінити і відображати її в окремій панелі. Для цього можна скористатися кнопкою у вигляді шестерінки.

IntelliJ IDEA може бути налаштована так, щоб підказка виникала при переміщенні миші над ім'ям класу (методу). Для цього:

1. Виконайте команду **File-Settings ... (Ctrl + Alt + S)**.
2. У вікні "**Settings**" перейдіть на вкладку **Editor-General** і встановіть прапорець **Show quick documentation on mouse move**.

### Генерація довідки Javadoc

Для генерації файлових довідок у форматі HTML необхідно:

1. Виконайте команду **Tools-Generate JavaDoc**.
2. У вікні "**Spesify Generate JavaDoc Scope**" встановіть необхідні параметри та виберіть папку для збереження файлів.
3. Натисніть **ОК** для запуску генерації довідки.

Після завершення генерації головна сторінка довідки (index.html) відкривається в браузері.

## Побудова діаграм класів у середі IntelliJ IDEA

### Установка плагіна simpleUMLCE

Засіб для побудови діаграми класів у безкоштовній версії IntelliJ IDEA відсутня (хоча присутній в версіях Ultimate). Усунути цю проблему дозволяє встановлення плагіна simpleUMLCE. Для його встановлення необхідно виконати наступні дії:

1. Запустіть IntelliJ IDEA.
2. Виконайте команду **File-Settings… (Ctrl + Alt + S)**.
3. У вікні "**Settings**" перейдіть на вкладку **Plugins** та використовуйте поле пошуку для пошуку плагіни за назвою.

4. Встановіть найденніший плагін, використовуючи кнопку **Install**.

### Побудова діаграм

Після встановлення плагіну simpleUMLCE та перезапуска IntelliJ IDEA можна перейти до побудови діаграми. Для цього:

- У дереві проекту виберіть папку з класами.
- У контекстному меню папки виберіть команду **Add to simpleUML Diagram-New Diagram…**
- У вікні "**Choose a Name and Location**" задайте ім'я та розташування діаграм.

## Варіанти завдань

| № | Завдання | № | Завдання | № | Завдання | № | Завдання |
|---|---|---|---|---|---|---|---|
| **1** | 1,5,16,40 | **7** | 8,13,18,22 | 13 | 2,14,28,38 | 19 | 8,15,22,34 |
| **2** | 2,13,17,39 | **8** | 9,14,23,35 | 14 | 3,5,29,39 | 20 | 9,13,21,35 |
| **3** | 3,15,18,38 | **9** | 5,10,24,34 | 15 | 4,15,26,30 | 21 | 10,14,20,36 |
| **4** | 4,14,19,37 | **10** | 11,15,25,35 | 16 | 5,13,25,31 | 22 | 5,11,19,37 |
| **5** | 5,6,20,36 | **11** | 12,14,26,36 | 17 | 5,7,24,32 | 23 | 12,15,18,38 |
| **6** | 7,15,17,21 | **12** | 1,13,27,37 | 18 | 14,6,23,33 | 24 | 1,13,17,33 |

1. Побудувати програму для роботи з класом для зберігання даних про криву *y(x)* другого порядку - гіперболу. Програма повинна забезпечувати: розрахунок *y* по *x* і навпаки, введення/виведення значень.
2. Побудувати програму для роботи з класом для зберігання даних про криву *y(x)* другого порядку - еліпс. Програма повинна забезпечувати простіші функції: розрахунок *y* по *x* і навпаки, введення/виведення значень.

3. Побудувати програму для роботи з класом для зберігання даних про криву *y(x)* другого порядку - параболу. Програма повинна забезпечувати простіші функції: розрахунок *y* по *x* і навпаки, введення/виведення значень.
4. Побудувати програму для роботи з класом для зберігання даних про погоду (напрямок, швидкість вітру, температура, хмарність, опади). Програма повинна забезпечувати функції введення/виведення значень.
5. Побудувати програму для роботи з класом для зберігання даних про повідомлення в форумі (автор, тема, текст, час, дата створення та редагування). Програма повинна забезпечувати функції введення, редагування, виведення значень.
6. Побудувати програму для роботи з класом *Дата*. Програма повинна забезпечувати функції збільшення/зменшення на 1 день, введення/виведення значень.
7. Побудувати програму для роботи з класом *Час*. Програма повинна забезпечувати функції збільшення/зменшення на 1 годину, хвилину, секунду, введення/виведення значень.
8. Побудувати програму для роботи з класом *Правильний дріб*, який повинен включати поля *чисельник*, *знаменник*. Програма повинна забезпечувати функції додавання, віднімання, множення, ділення, виведення дробу в зручній формі.
9. Побудувати програму для роботи з класом *Комплексне число*. Клас повинен включати поля дійсної та уявної частини числа. Програма повинна забезпечити функції додавання, віднімання, множення, виведення дробу в зручній формі.
10. Створити класс типа *Прямокутник*. Поля - висота і ширина. Функції-члени обчислюють площу, периметр, встановлюють і повертають значення полів. Функції-члени встановлення значень полів класу повинні перевіряти коректність заданих параметрів.

11. Створити клас *Гра в хрестики-нулики*. Поле класу - масив 3x3. Ставити можна лише на вільні місця.
12. Створити клас *Коло*. Поле - радіус. Функції-члени обчислюють площу обмеженого колом круга, довжину кола, встановлюють і повертають значення полів. Функції-члени встановлення значень полів повинні перевіряти коректність заданих значень.
13. Створити клас *Квадрат*. Поля - сторона. Функції-члени обчислюють площу, периметр квадрата, встановлюють і повертають значення полів. Функції-члени встановлення значень полів повинні перевіряти коректність заданих значень.
14. Створити клас *Трикутник*. Поля - сторони. Функції-члени обчислюють площу, периметр трикутника, встановлюють і повертають значення полів. Функції-члени встановлення значень полів повинні перевіряти коректність заданих значень.
15. Створити клас *Лінія*. Поля - координати початку и кінця. Функції-члени обчислюють довжину лінії, переміщують її, встановлюють і повертають значення полів. Функції-члени встановлення значень полів повинні перевіряти коректність заданих значень.
16. Написати клас для відділу кадрів Співробітник (поля: прізвище, ім'я, по батькові, дата народження, стать, освіта, номер документа про освіту, навчальний заклад, дата початку роботи, домашня адреса).
17. Створити клас записної книжки (поля: ім'я, нік, мобільний телефон, адреса електронної пошта, номер ICQ).
18. Створити клас для одиниць товару на складі (поля: товар, виробник, кількість, дата виготовлення, срок поставки, постачальник, телефон постачальника, телефон виробника, ціна за 1 одиницю).
19. Створити клас для обліку продаж (поля: товар, виробник, покупець, кількість, ціна за одиниць, загальна вартість).

20. Створити клас для каталогу музичних компакт-дисків (поля: виконавець, композитор, назва диска, улюблений трек, дані про покупки, кому видано диск, кількість треків, тривалість).
21. Створити клас для каталогу фільмів (поля: назва, режисер, виконавець головної ролі, рік виходу, кому видано на перегляд, мова звукової доріжки, ліцензійний чи ні).
22. Створити базовий клас - «*Дата*», похідний клас - «*Записна книжка*», що включає ПІБ, телефон, дату народження і функцію обчислення кількості днів до дня народження.
23. Створити базовий клас – «*Час»*, похідний клас - «*Розклад*», що включає дисципліну, аудиторію, час початку заняття і функцію обчислення часу до початку заняття;
24. Створити базовий клас - «*Вікно*» включає координати (left, top, right, bottom) і колір вікна, похідний клас - «*Вікно з текстом*», що включає текст, колір тексту у вікні.
25. Створити базовий клас - «*Поліном*» (масив коефіцієнтів), похідний клас «*Раціональний вираз*», що включає поліном в чисельнику, поліном в знаменнику.
26. Створити базовий клас «*Матриця*», що включає матрицю, її визначник і функцію обчислення визначника, Похідний - «СЛАР», що включає матрицю, стовпець вільних членів, метод вирішення системи лінійних алгебраїчних рівнянь.
27. Створити базовий клас - «*Дріб*», похідний клас - «*Дробове комплексне число*», що включає дробову дійсну частину, дробову уявну частину і арифметичні операції (+, -, *) над комплексним дробом;
28. Створити базовий клас «*Комплексне число*» і похідний, який включає комплексне число в стандартній і експоненційній формі і функцію обчислення експоненційної форми числа.

29. Створити клас *Студент* з полями ім'я, курс і ідентифікаційний номер. Визначити конструктори і функцію друку. Створити похідний клас - *Студент-дипломник*, що має тему диплома.
30. Створити клас *Тварина*, що має поля вид, число кінцівок, число нащадків. Створити похідний клас - домашня тварина, що має кличку.
31. Створити клас *Машина*, що має марку, число циліндрів, потужність. Створити похідний клас - *Вантажівка*, що має вантажопідйомність.
32. Створити клас *Фігура*, яка має координати розташування. Створіть похідний клас - *Еліпс*. Напишіть віртуальний метод переміщення.
33. Створити базовий клас *Комп'ютер* (назва, частота процесор, кількість ядер, об'єм ОЗУ, постійної пам'яті), похідний клас *Ноутбук* (діагональ екрану, вага, об'єм батареї).
34. Створити базовий клас *Мобільний телефон* (назва, вага, діагональ екрану, кількість вбудованої пам'яті, наявність камери). Похідний клас - *Смартфон* (частота процесора, кількість ядер, об'єм ОП).
35. Створити клас *Фігура*, яка має координати розташування. Створити похідний клас *Прямокутник* (висота, ширина). Напишіть віртуальний метод переміщення.
36. Створити базовий клас *Дата* (рік, місяць, день). Похідний клас - *Дата з часом* (години, хвилини).
37. Створити базовий клас *Стек* на базі масиву цілих чисел, з максимальним розміром і методами додавання і вилучення елемента. Створіть клас *Чергу* - на базі того ж масиву.
38. Створити базовий клас *Трикутник* (три сторони, метод розрахунку площі). Похідний клас - *Чотирикутник* (+1 сторона і діагональ).

39. Створити базовий клас *Книга* (назва, автор(и), рік видання, тираж, кількість сторінок). Похідний клас - *Книга в бібліотеці* (інвентарний номер, хто взяв).
40. Створити базовий клас *Дріб* (чисельник, знаменник, арифметичні операції, перетворення в дійсний тип). Похідний клас - *Число з дробовою частиною*.

## Хід виконання роботи

1. Завантажте IntelliJ IDEA.
2. Встановіть плагін simpleUMLCE і перевантажте IntelliJ IDEA.
3. Створіть проект Maven.
4. Розкрийте дерево проекту і перейдіть в папку **java**.
5. Додайте до неї новий файл класу, в якому визначте інтерфейс завдання вашого варіанту.
6. Додайте новий файл класу, в якому надайте реалізацію створеного інтерфейсу для виконання завдання вашого варіанту.
7. Для створеного класу забезпечте інкапсуляцію даниих.
8. В створеному класі перевизначте методи **equals**(), **hashCode**(), **toString**() і т.п.
9. Створіть unit-тест для тестування методів розробленого класу.
10. Запустіть тест на виконання.
11. Якщо тест не пройдений, внесіть необхідні виправлення в код програми і перейдіть до п.10.
12. Якщо тест успішно пройдений, перейдіть до п.13.
13. Доповніть клас коментарями Javadoc.
14. Згенеруйте довідку Javadoc (вкладається в звіт).
15. Сформуйте діаграму класа (теж вкладається в звіт).
16. Повторіть пп.5-16 для інших завдань вашого варіанту.
17. Оформіть звіт про виконання роботи. Довідка Javadoc і діаграми класів розміщуються у додатках до звіту.

## Контрольні запитання

1. Як оголошуються класи в Java?
2. Яке призначення конструкторів класу?
3. Яку роль відіграють ініціатори класу?
4. Для чого використовуються модифікатори доступу?
5. Як забезпечується інкапсуляція даних в класах?
6. Як реалізується наслідування класів?
7. Які класи називаються абстрактними?
8. Яке призначення інтерфейсів?
9. Як в Java організоване документування коду?
10. Які теги можуть використовуватись в коментарях Javadoc? Яке їх призначення?
11. Як здійснюється генерація довідки Javadoc в середовищі IntelliJ IDEA?
12. Як в IntelliJ IDEA будувати діаграми класів?

# ЛІТЕРАТУРА


1. С.А. Кравчук, Шонин В.А. Основы программирования на языке Java. – К.: Норіта-плюс, 2007. – 280 с.
2. Ноутон П., Шилдт Г., Java 2. - СПб.: БХВ – Санкт-Петербург. 2000. – 1072 с.
3. Хабибулин И.Ш. Самоучитель Java. – СПб.: БХВ – СанктПетербург. 2001. – 464 с.
4. Шилд Герберт, Холмс Джеймс. Искуство программрования на Java. Пер. с англ. – М.: Издательский дом "Вильямс", 2005. – 336 с., ил.
5. Блинов, И.Н. Java. Промышленное программирование : практ. пособие / И.Н. Блинов, В.С. Романчик. – Минск : УниверсалПресс, 2007. – 704 с.
6. Бишоп Д. Эффективная работа: Java 2. СПб.: Питер; К.: Издательская группа BHV. 2002. – 592с.
7. К. Арнольд, Дж. Гослинг, Д. Холмс. Язык программирования Java. 3-е изд. // М: Вильямс, 2001. – 624 с.
8. Б. Эккель. Философия Java. // СПб: Питер, 2001. – 880 с.
9. Д. Блох. Java. Эффективное программирование. // М.: Лори, 2002. – 224 с.
10. С. Макконнелл. Совершенный код.// СПб: Питер, 2005. – 868 с.