Практична робота 7. Розробка мобільного додатку в Android Studio з використанням мови Java

Теоретичні відомості

Створення класів та об'єктів у Java.

Мова Java є об'єктно орієнтованою та в повному обсязі використовує ці принципи. Навіть невеликі навчальні проекти часто складаються з багатьох класів. При цьому кожному загальнодоступному (public) класу відповідає свій файл, що має те саме ім'я. Для зручності Java передбачено спеціальний засіб угруповання класів, зване пакетом (package). Пакети забезпечують незалежні простір імен (namespaces), а також обмеження доступу до класів. Класи завжди задаються в якомусь пакеті. Щоб помістити клас у пакет, потрібно продекларувати ім'я пакета на початку файлу, в якому оголошено клас, у вигляді:

package ім’я_пакету;

Крім того, необхідно помістити вихідний код класу в папку, яка відповідає пакету. Якщо декларація імені пакета відсутня, вважається, що клас належить пакету під назвою default. При створенні проекту в середовищі Android Studio розміщення класу пакета відбувається автоматично.

Клас - це опис того, як буде влаштований об'єкт, що є екземпляром даного класу, а також які методи об'єкт може викликати. Класи в Java задаються в такий спосіб. Спочатку пишеться зарезервоване слово class, потім ім'я класу, після чого у фігурних дужках пишеться реалізація класу задаються його поля (глобальні змінні) та методи.

Об'єктні змінні є такими змінними, які мають об'єктний тип. У Java об'єктні змінні це самі об'єкти, лише посилання ними. Тобто, всі об'єктні типи є посилальними.

Оголошення об'єктної змінної здійснюється як і, як та інших типів змінних. Спочатку пишеться тип, а потім через пробіл ім'я змінної, що оголошується.

Наприклад, якщо ми задаємо змінну obj1 типу Circle, "коло", її завдання здійснюється так:

Circle obj1;

Зв'язування об'єктної змінної з об'єктом здійснюється шляхом надання. У правій частині присвоєння можна вказати або функцію, яка повертає посилання на об'єкт (адресу об'єкта), або ім'я іншої об'єктної змінної. Якщо об'єктній змінній не надано посилання, у ній зберігається значення null.

Об'єктні змінні можна порівнювати на рівність, зокрема рівність null. При цьому порівнюються не самі об'єкти, а їх адреси, що зберігаються в об'єктних змінних.

Створюється об'єкт за допомогою виклику спеціальної підпрограми, яка задається в класі і називається конструктором. Конструктор повертає посилання на створений об'єкт. Ім'я конструктора Java завжди збігається з ім'ям класу, екземпляр якого створюється. Перед іменем конструктора під час дзвінка ставиться оператор new

"Новий", що означає, що створюється новий об'єкт. Наприклад, виклик obj1=new Circle(); означає, що створюється новий об'єкт типу Circle, "коло", і посилання на нього (адреса об'єкта) записується в змінну obj1. Змінна obj1 до цього вже має бути оголошена. Оператор new відповідає за динамічне виділення пам'яті під об'єкт, що створюється. Часто поєднують завдання об'єктної змінної та призначення об'єкта. У нашому випадку воно виглядатиме як

Circle obj1 = new Circle ();

Конструктор, як і будь-яка підпрограма, може мати список параметрів. Вони потрібні для того, щоб встановити початковий стан об'єкта при його створенні. Наприклад, ми хочемо, щоб у створюваного кола можна було при виклику конструктора задати координати x, y її центру та радіус r. Тоді при написанні класу Circle можна передбачити конструктор, в якому першим параметром визначається координата x, другим y, третім радіус кола r. Тоді завдання змінної obj1 може мати такий вигляд:

Circle obj1 = new Circle (130,120,50);

Воно означає, що створюється об'єкт коло, що має центр у точці з координатами x=130, y=120, і у якої радіус r=50. Якщо розробники класу не створили жодного конструктора, у реалізації класу автоматично створюється конструктор за замовчуванням, який має порожній список параметрів. І його можна викликати в програмі так, як ми спочатку робили для класу Circle.

Прийнято імена об'єктних типів писати з великої літери, а імена об'єктних змінних з невеликою.

У Java прийнято називати об'єктні змінні так само, як їх типи, але починати ім'я з малої літери.

Наступні рядки програмного коду створюють два незалежні об'єкти з однаковими початковими параметрами:

```
Circle circle1 = new Circle (130,120,50);
Circle circle2 = new Circle (130,120,50);
```

За допомогою об'єктних змінних здійснюється доступ до полів даних або методів об'єкта: спочатку вказується ім'я змінної, потім точка, після чого пишеться ім'я поля даних чи методу. Наприклад, якщо ім'я об'єктної змінної circle1, а ім'я цілого поля даних x, то присвоювання йому нового значення виглядатиме як

```
circle1.x=5;
```

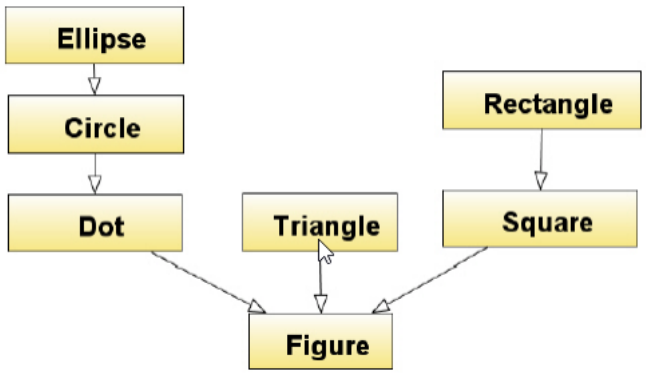
А якщо ім'я підпрограми show, у неї немає параметрів і вона не повертає жодного значення, її виклик буде виглядати як

```
circle1.show();
```

Наслідування класів

Спадкування дозволяє будувати на основі початкового класу нові, додаючи до класів нові поля даних та методи. У Java всі класи є нащадками класу Object. Тобто він є базовим всім класів.

Як приклад того, як будується ієрархія, розглянемо ієрархію фігур, які виводяться на екран.



У ній базовим класом є Figure, від якого успадковуються Dot "точка", Triangle "трикутник" та Square "квадрат". Від Dot успадковується клас Circle "коло", а від Circle успадкуємо Ellipse "еліпс". І, нарешті, від Square успадкуємо Rectangle "прямокутник". Зазначимо, що в ієрархії прийнято малювати стрілки у напрямку від спадкоємця до прабатька. Такий напрямок називається Generalization "узагальнення", "генералізація". Стрілки символізують напрямок у бік спрощення.

Базовий клас завжди називають ім'ям, яке характеризує всі об'єкти екземпляри класів спадкоємців, і яке виражає найбільш загальну абстракцію, яка застосовується до таких об'єктів. У нашому випадку це клас Figure. Будь-яка фігура матиме поля даних x та y координати фігур на екрані. Клас Dot ("точка") є спадкоємцем Figure, тому він матиме поля даних x і y, успадковані від Figure. Тобто в самому класі Dot задавати ці поля не треба. Від Dot ми успадковуємо клас Circle ("коло"), тому в ньому також є поля x і y, успадковані від Figure. Але з'являється додаткове поле даних. У Circle це поле, що відповідає радіусу. Ми назвемо його r. Крім того, для кола можлива операція зміни радіусу, тому в ній може з'явитися новий метод, що забезпечує цю дію, назвемо його setSize ("установити розмір"). Клас Ellipse має ті ж поля даних і забезпечує ту ж поведінку, що і Circle, але в цьому класі з'являється додаткове поле даних r2 довжина другої півосі еліпса, і можливість регулювати значення цього поля.

Порядок виконання роботи

1. Вивчити методичні вказівки до практичного заняття. Написати код програми згідно з варіантом завдання.

2. Запустити на виконання Android Studio.

3. Відкрити в середовищі Android Studio раніше створений проект і додати новий пакет. Або можна створити новий проект.

4. Розмістити на формі елементи керування.

5. У вікні java коду проекту додати рядки обробки натискання кнопки.

6. Запустити створену програму в емуляторі Android і спостерігати за появою цієї програми та результатів її роботи у вікні додатків емулятора.

7. Додати в проект інші елементи керування, налаштувати їх властивості та перевірити роботу програми в емуляторі Android.

Створення класів

1. Створіть проект Java. Назвіть пакет com.example, а головний клас EmployeeTest

2. Створіть пакет com.example.domain, а у ньому клас Employee.

```
public int empId;
public String name;
public String ssn;
public double salary;
```

3. Додайте конструктор класу:

```
public Employee() {}
```

4. Створіть методи читання та запису («гетери» та «сеттери») для кожного поля. Використовуйте для цього контекстне меню редактора Android Studio

5. Додайте до файлу класу EmployeeTest імпорт класу Employee import com.example.domain.Employee;

6. Додайте в процедуру main класу EmployeeTest команди створення об'єкта класу Employee та заповнення його полів

```
Employee emp = new Employee();
```

```
emp.setEmpId(101);
emp.setName("Jane Smith");
emp.setSalary(120_345.27);
emp
setSsn ("012 34 5678");
```

7. Додайте в процедуру main класу EmployeeTest команди відображення даних об'єкта класу Employee

```
System.out.println("Employee ID: "+emp.getEmpId());
System.out.println("Employee Name: "+emp.getName());
System.out.println("Employee Soc Sec #" + emp.getSsn());
System.out.println("Employee salary: " + emp.getSalary());
```

8. Запустіть програму.

Використання ієрархії класів

1. Скопіюйте папку з проектом із попередньої роботи та перейменуйте проект.

2. Застосуйте інкапсуляцію до класу Employee.

Для цього:

- замініть модифікатори доступу полів з public на private;
- замініть конструктор без параметрів на конструктор із параметрами

```
public Employee(int empId, String name, String ssn, double salary) {
    this.empId = empId;
    this.name = name; this.ssn = ssn; this.sa
    lary = salary;
}
```

- заберіть усі методи запису даних у поля («сеттери»), крім методу setName
- додайте метод збільшення зарплати raiseSalary:

```
public void raiseSalary(double increase){
    if (increase>0){
```

```
            salary += increase;
        }
    }
```

3. Створіть у тому пакеті підклас класу Employee і назвіть його Manager:

```
public class Manager extends Employee { }
```

4. Додайте поле deptName private String deptName;

5. Додайте конструктор класу з параметрами, що викликає конструктор батьківського класу та ініціює значення поля deptName :

```
public Manager(int empId, String name, String ssn, double salary, String deptName) {
    super(empId, name, ssn, salary);
    this.deptName = deptName;
}
```

6.Додайте до нього метод читання даних із поля

```
getDeptName : public String getDeptName() {
    return deptName;
}
```

7. Створіть у тому пакеті підклас класу Employee і назвіть його Admin. Запишіть у нього конструктор із параметрами:

```
public class Engineer extends Employee {
    public Engineer(int empId, String name, String ssn, double salary) {
        super(empId, name, ssn, salary);
    }
}
```

9. Створіть у тому пакеті підклас класу Manager і назвіть його Director:

```
public class Director extends Manager { }
```

10. Додайте до нього поле budget private double budget;

11. Додайте конструктор класу з параметрами, що викликає конструктор батьківського класу та ініціює значення поля budget:

public Director(int empId, String name, String ssn, double salary, String deptName, double budget) {

super(empId, name, ssn, salary, deptName);

this.budget = budget;

}

12. Додайте метод читання даних з поля budget:

```
public double getBudget () {
    return budget;
}
```

13. Збережіть усі класи

14. Додайте в процедуру main класу EmployeeTest команди імпорту створених класів

```
import com.example.domain.Admin;
import com.example.domain.Director;
import com.example.domain.Engineer;
import com.example.domain.Manager;
```

15. Запишіть у процедуру main класу EmployeeTest команди створення об'єктів створених класів та заповнення їх полів Engineer

eng = new Engineer(101, "Jane Smith", "012–34–5678", 120_345.27);

Manager mgr = new Manager(207, "Barbara Johnson", "054–12–2367", 109_501.36, "US Marketing");

Admin adm = new Admin(304, "Bill Munroe", "108–23–2367", 75_002.34);

Director dir = new Director(12, "Susan Wheeler", "099–45–2340", 120_567.36, "Global Marketing", 1_000_000.00);

16. Додайте до класу EmployeeTest статичний метод відображення даних об'єкта, представленого за посиланням класу Employee, батьківського для всіх новостворених класів

```
private static void printEmployee(Employee emp) {
    System.out.println("Employee ID: " + emp.getEmpId());
    System.out.println("Employee Name: " + emp.getName());
```

System.out.println("Employee Soc Sec # " + emp.getSsn());
System.out.println("Employee salary: " + emp.getSalary());
}

17. Додайте до класу EmployeeTest команди відображення даних створених об'єктів

```
printEmployee(eng);
printEmployee(mgr);
printEmployee(adm);
printEmployee(dir);
```

18. Збережіть усі класи та запустіть програму.

Вимоги до звіту

1. Назва та мета роботи.

2. Лістинг створеної програми в Android Studio за допомогою командних елементів керування, елементів вибору, елементів введення та елементів відображення.

3. Результати роботи програми в емуляторі телефону.

Завдання для виконання

№	Клас
1	Співробітник, 3 поля
2	Студент, 2 поля
3	Товар, 3 поля
4	Пес, 2 поля
5	Геометрична фігура, 2 поля
6	Програмне забезпечення, 3 поля
7	Апаратне забезпечення, 3 поля
8	Місто, 2 поля
9	Країна, 2 поля
10	Книга, 3 поля
11	Співробітник, 3 поля – 3 класи спадкоємця
12	Студент, 2 поля – 2 класи спадкоємця
13	Товар, 3 поля – 4 класи спадкоємця

14	Пес, 2 поля – 3 класи спадкоємця
15	Геометрична фігура, 2 поля – 3 класи спадкоємця
16	Програмне забезпечення, 3 поля – 2 класи спадкоємця
17	Апаратне забезпечення, 3 поля – 3 класи спадкоємця
18	Місто, 2 поля – 2 класи спадкоємця
19	Країна, 2 поля – 2 класи спадкоємця
20	Книга, 3 поля – 4 класи спадкоємця