Спадкування і поліморфізм

Спадкування дозволяє будувати на основі початкового класу інші, додаючи в класи нові поля даних і методи.

Початковий клас називається ***прабатьком*** (ancestor), нові класи - його ***нащадками*** (descendants).

Набір класів, пов'язаних ставленням спадкування, називається ***ієрархією класів***.

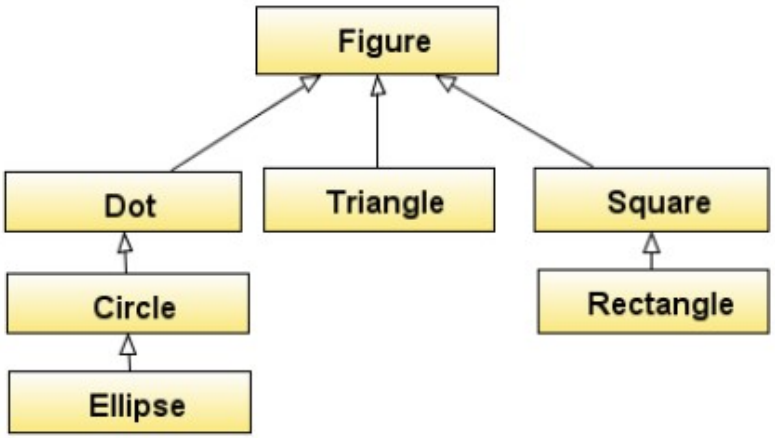
Клас, що стоїть на чолі ієрархії, від якого успадковані всі інші (прямо або опосередковано), називається ***базовим класом ієрархії***.

Поліморфізм (з грецької «має багато форм») - наявність коду, написаного з використанням посилань, що мають тип базового класу ієрархії.

При цьому такий код повинен правильно працювати для будь-якого об'єкта, що є екземпляром класу з даної ієрархії, незалежно від того, де цей клас розташований в ієрархії. Такий код і називається ***поліморфним***. При написанні поліморфного коду заздалегідь невідомо, для об'єктів якого типу він буде працювати - один і той же метод буде виконуватися по-різному в залежності від типу об'єкта.

Перевага об'єктного програмування полягає в можливості написання поліморфного коду. Саме для цього створюється ієрархія класів. Поліморфізм дозволяє різко збільшити ***коефіцієнт повторного використання програмного коду*** і ***модифікованість*** цього коду в порівнянні з процедурним програмуванням.

Приклад ієрархії класів для зображення фігур на екрані



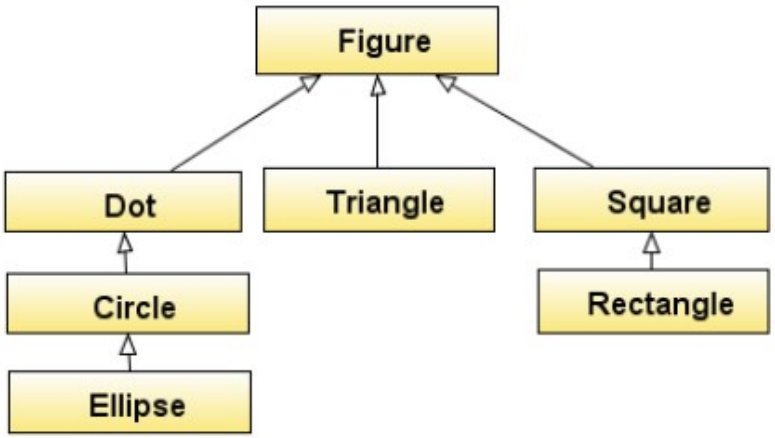
Клас **Figure**: поля даних ***x*** і ***y*** - координати фігури на екрані.

Клас **Dot**: поля даних ***x*** і ***y***, успадковані від **Figure**. У самому класі **Dot** задавати ці поля не треба

Клас **Circle**: поля ***x*** і ***y***, успадковані від **Figure**,
нове поле ***r***, відповідне радіусу, новий метод ***setSize***, що забезпечує зміну радіуса

Клас **Ellipse**: поля ***x*** і ***y***, успадковані від **Figure**, поле ***r*** і метод ***setSize***, успадковані від **Circle**, нове поле даних ***r2***, що задає довжину другої півосі еліпса

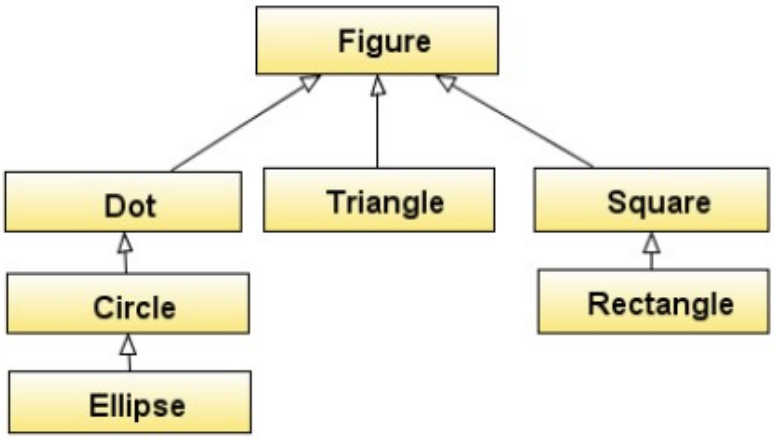
Приклад ієрархії класів для зображення фігур на екрані



Клас **Square**: поля даних ***x*** і ***y***, успадковані від **Figure**, нове поле ***a***, відповідне стороні квадрата.

Клас **Rectangle**: поля даних ***x*** і ***y***, успадковані від **Figure**, поле ***a***, успадковане від Square, нове поле ***b***, відповідне стороні прямокутника.

Приклад ієрархії класів для зображення фігур на екрані



Клас **Triangle**: поля даних ***x*** і ***y***, успадковані від **Figure**, як нові, не успадковані поля даних, можуть виступати або координати вершин трикутника, або координати однієї з вершин та довжини прилеглих до неї сторін і кут між ними і т.д.

Нащадки повинні володіти більш складною структурою і поведінкою в порівнянні з прабатьком.

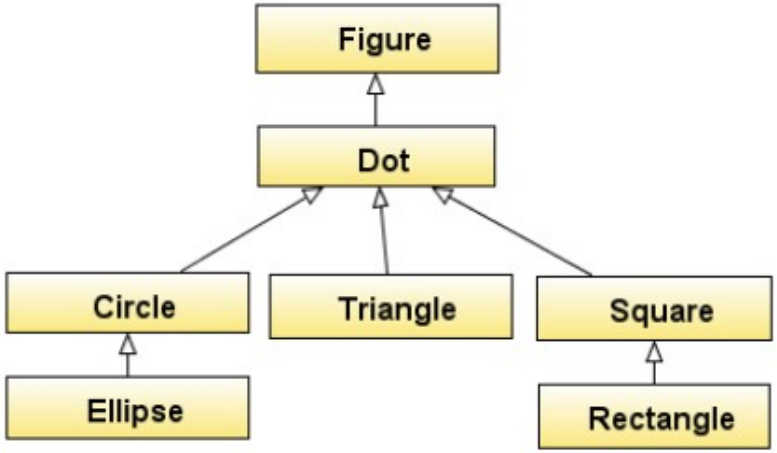
Кожен об'єкт класу-нащадка ***при будь-яких значеннях полів*** потрібно розглядати як екземпляр класу-прабатька.

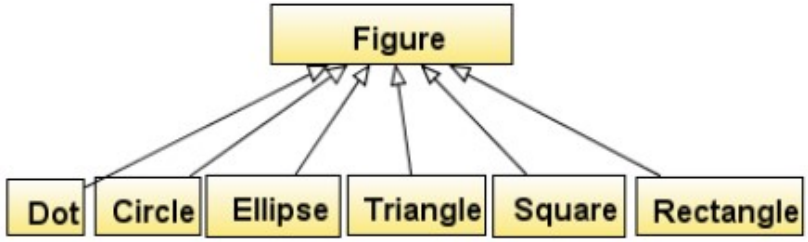
По своїй поведінці об'єкт-еліпс може розглядатися як будь-який екземпляр типу **Circle** і навіть вести себе в точності як окружність, але не навпаки: об'єкт типу **Circle** не володіє поведінкою **Ellipse**.

КРИТЕРІЙ ЗАСТОСОВНОСТІ УСПАДКУВАННЯ

Якщо є класи **A1** і **A2** і можна вважати, що клас **A2** є модифікованим (ускладненим або зміненим) варіантом класу **A1** зі збереженням всіх особливостей поведінки **A1**, то **A2** можна описувати як нащадок **A1**. На рівні абстракції, яка описує поведінку, об'єкт типу **A2** повинен вести себе, як об'єкт типу **A1** при будь-яких значеннях полів даних.

Альтернативні варіанти ієрархії класів для зображення фігур на екрані



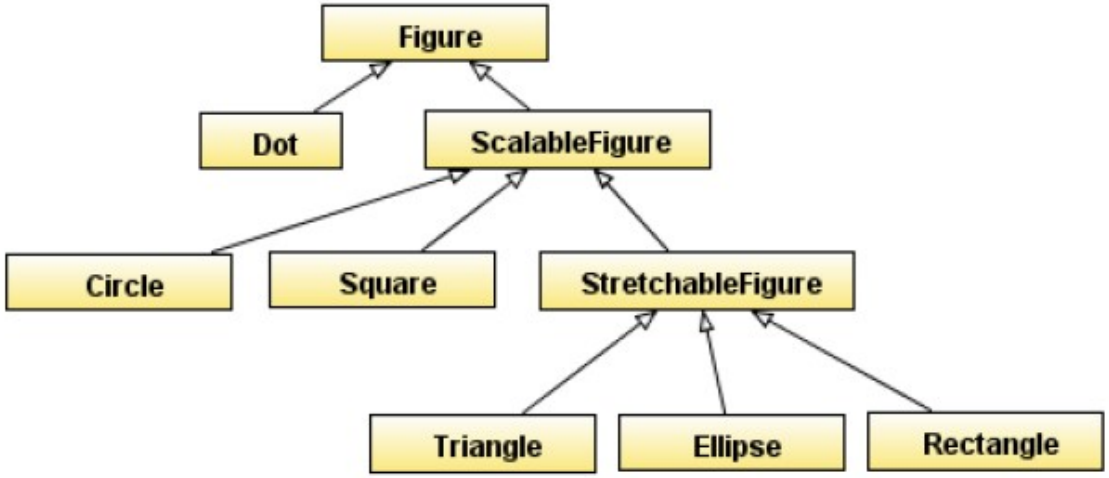


Один з важливих принципів при побудові ієрархій - ***відповідність представлень з предметної області ієрархії, що будується***.

Спадкування від класів, у яких бувають екземпляри (***не абстрактних***), небажано.

Класи, у яких немає примірників, називаються ***абстрактними***.

Ієрархія з використанням абстрактних класів:



Основна перевага такої ієрархії в порівнянні з попередніми - можливість писати поліморфний код для найбільш загальних різновидів фігур.

Введення проміжних рівнів спадкування, що відповідають відповідним абстракцій, - характерна риса об'єктного програмування.
При цьому класи **Figure**, **ScalableFigure** і **StretchableFigure** будуть абстрактними - екземпляри такого типу створювати не передбачається, оскільки не буває фігури (масштабованої або розтягуваної) в загальному вигляді, без зазначення її конкретної форми.

Продумування того, як влаштовані класи, тобто які в них повинні бути поля і методи (без уточнення конкретної реалізації цих методів), а також опис того, яка повинна бути ієрархія наслідування, називається ***проектуванням***. Це складний процес, який набагато важливіше написання конкретних операторів в реалізації (***кодування***).

Критерії правильності побудови ієрархії :

1. У процесі успадкування має йти розширення (ускладнення, спеціалізація, конкретизація) класів, а не навпаки.

2. Спадкування має йти тільки від абстрактних класів (або інтерфейсів як варіанти повністю абстрактних класів).

3. Назви всіх методів повинні давати точне уявлення про те, що робить метод. Причому ці імена повинні сприйматися як команди (встановити щось, прочитати щось, показати щось і т.д.).

4. Неприпустимі назви методів, що містять в'язку "і" ("and"). Наприклад, ***readAndShowSpeed***, ***calculateIntegralAndWriteToFile*** і т.п. Такого роду гібриди слід розділяти на два і більше незалежних методи - ***readSpeed*** і ***ShowSpeed***; ***calculateIntegral*** і ***writeToFile***, і т. д.

5. Назви всіх класів повинні давати чітке уявлення про відповідні абстракціях поведінки, в тому числі для неабстрактне класів.

Спадкування. Перевизначення методів

При завданні класу-нащадка спочатку йдуть модифікатори, потім після ключового слова **class** йде ім'я декларованого класу, потім йде зарезервоване слово **extends**, після чого потрібно вказати ім'я класу-батька. Якщо не вказується, від якого класу йде спадкування, то батьком вважається клас **Object**.

Далі в фігурних дужках йде реалізація класу - опис його полів і методів. При цьому поля даних і методи, наявні в прабатьків, в нащадку описувати не потрібно - вони успадковуються. **У класі-нащадку прабатьківський метод можна реалізувати по-іншому**. Тоді метод необхідно задекларувати і реалізувати в класі-нащадку. Крім того, в нащадку можна задавати нові поля даних і методи, відсутні в прабатьків.

Модифікатори при завданні класу-нащадка:

• **public** - модифікатор, що задає публічний (загальнодоступний) рівень видимості. Якщо він відсутній, то діє пакетний рівень доступу - клас доступний тільки елементам того ж пакета;

• **abstract** - модифікатор, який вказує, що клас абстрактний, т. е. у нього не буває примірників (об'єктів). Обов'язково оголошувати клас абстрактним в разі, якщо який-небудь метод оголошений як абстрактний;

• **final** - модифікатор, який вказує, що клас остаточний (final), т. е. що у нього не може бути нащадків.

Таким чином, завдання класу-спадкоємця має наступний формат:

Модифікатори **class** *ідентифікатор* **extends** *суперклас*
{
тіло-класу
}

Приклад: Завдання абстрактного класу Figure

```
public abstract class Figure {  // абстрактний клас
int x=0;
int y=0;
java.awt.Color color;
java.awt.Graphics graphics;
java.awt.Color bgColor;
public abstract void show();    // абстрактний метод
public abstract void hide();    // абстрактний метод

public void moveTo(int x, int y){
hide();
this.x= x;
this.y= y;
show();
}
}
```

Приклад: Завдання класу Dot – нащадка Figure

```
package java_gui_example;
import java.awt.*;
public class Dot extends Figure {        // Створює новий
                                    екземпляр типу Dot
public Dot(Graphics graphics,Color bgColor) {
this.graphics=graphics;
this.bgColor=bgColor; }
public void show() {
Color oldC=graphics.getColor();
graphics.setColor(Color.BLACK);
graphics.drawLine(x,y,x,y);
graphics.setColor(oldC); }
public void hide() {
Color oldC=graphics.getColor();
graphics.setColor(bgColor);
graphics.drawLine(x,y,x,y);
graphics.setColor(oldC); }
}
```

Спадкування і правила видимості

Поля і методи, помічені як **private**, успадковуються, але в класах-спадкоємцях недоступні. Це зроблено з метою забезпечення безпеки.

Модифікатор **protected** призначений для використання відповідних полів і методів розробниками класів-спадкоємців. Він дає трохи більшу відкритість, ніж пакетний вид доступу, оскільки на додаток до видимості з поточного пакета дозволяє забезпечити доступ до **protected** з класів-спадкоємців, які перебувають в інших пакетах.

Модифікатором **protected** корисно позначати різного роду службові методи, непотрібні користувачам класу, але необхідні для функціональності цього класу.

Зарезервоване слово super

Іноді виникає необхідність викликати поле або метод з прабатьківського класу. Зазвичай це буває в тих випадках, коли в класі-нащадку задано поле з таким же ім'ям або перевизначений метод. В результаті видимість прабатьківського поля даних або методу в класі-нащадку загублена. Кажуть, що поле або метод ***затіняється*** в нащадку.
У цих випадках використовують виклик:

super.ім'яПоля або **super. ім'яМетода(список параметрів)**

Слово **super** тут означає скорочення від **superclass**. Якщо метод або поле задані не в безпосередньому прабатьку, а успадковані від більш далекого прабатька, то відповідні виклики все одно будуть працювати, але комбінації виду **super.super.імя** не дозволені. Виклики за допомогою слова **super** дозволені тільки для методів і полів даних об'єктів. Для методів і змінних класу виклики за допомогою посилання **super** заборонені.

Клас Object

Класи Java є вузлами єдиного ієрархічного дерева. Коренем цього дерева є клас **Object** і, якщо визначення суперкласу в оголошенні будь-якого класу відсутнє, такий клас вважається подклассом класу **Object** і на початку виконання конструктора такого класу відбувається звернення до конструктору класу **Object**.

У класі **Object** визначений конструктор без параметрів, який не виконує ніяких дій.

Методи класу **Object**:

protected Object clone() – створює і повертає копію об'єкта;

protected void finalize() – викликає «збирач сміття», що видаляє об'єкт, якщо на нього більше немає посилань;

public boolean equals(Object obj) – визначає дорівнює чи один об'єкт іншому;

public final Class getClass() – отримує об'єкт типу **Class** для викликаючого об'єкта;

public int hashCode() – дозволяє отримати хеш-код об'єкта - ціле число;

public String toString() повертає рядкове представлення об'єкту - ім'я класу для даного об'єкта і хеш-код об'єкта;

три перевантажених методи **wait(), notify()** і **notifyAll()** – використовуються для управління обчислювальними потоками.

Перетворення змінних типу класів і масивів

В Java можливо перетворення змінних типу класів і масивів одного об'єктного типу в інший об'єктний тип. Як і для примітивних змінних, для посилальних змінних визначені розширююче і звужуюче перетворення.

Розширюючими перетвореннями для змінних типу класів і масивів є **перетворення типу змінної підкласу в тип змінної суперкласу**, а також **перетворення null в об'єкт будь-якого класу**. Такі перетворення не вимагають ніяких дій під час виконання і не призводять до помилок.

Звужуючими перетвореннями для змінних типу класів і масивів є **перетворення типу змінної суперкласу в тип змінної підкласу**.

Звужуючі перетворення виконуються, як і для примітивних типів, за допомогою оператора приведення типу:

(ім'я-класу) ім'я-змінної

Приклади перетворення типів:

```
class MyClass {…}
…
class MyFirstClass extends MyClass {…}
MyClass var1, var2;
MyFirstClass fvar1, fvar2, fvar3;
Object objvar1, objvar2;
…
objvar1 = fvar1;              // Розширююче перетворення
var1 = fvar1;                 // Розширююче перетворення
fvar2 = (MyFirstClass)var2;       // Звужуюче перетворення
fvar3 = (MyFirstClass)objvar2;  // Звужуюче перетворення
```

Лектор:

Старший викладач кафедри Електроніки и комп'ютерної техніки Сумського державного університету

Горячев О. Є.

В лекції використано матеріали авторів:

Шонін В.А.
Монахов В.В.