КЛАСИ та НАСЛІДУВАННЯ В JAVA

Лекція 4

доц. кафедри Інформатики
Сінельнікова Т.Ф.

ХНУРЕ, Кафедра Інформатики

informatika@kture.kharkov.ua

- Основи наслідування
- Приклад наслідування
- Доступ до елементів і наслідування
- Створення багаторівневої ієрархії
- Модифікатори обмеження доступу до елементів при спадкуванні
- Поняття та використання абстрактних класів
- Остаточні члени і класи
- Клас Object
- Методи object
- Використання ключового слова super (перший вид)
- Використання другої форми super
- Перевизначення методів
- Динамічна диспетчеризація методів
- Використання ключового слова final з наслідуванням
- Використання final для відмови від перевизначення
- Використання final для скасування наслідування

Основи наслідування

-Успадкування дозволяє створювати ієрархічні класифікації.

- Використовуючи спадкування, можна створити головний клас, який визначає властивості, загальні для набору пов'язаних елементів.
- Клас, який успадкований, називається суперкласом (superclass).
- Клас, який виконує спадкування, називається підкласом (subclass) - це спеціалізована версія суперкласу. Він успадковує всі змінні екземпляра і методи, визначені суперкласом, і додає свої власні унікальні елементи .

Основи наслідування

- Щоб успадковувати клас, потрібно просто включити визначення одного класу до іншого, використовуючи ключове слово extends.

Приклад наслідування

```
// Створити суперклас,
class A { int i, j;
void showi j () {.
System.out.println("i и j: " + i + " " + j); } }
// Створити підклас розширенням класу A.
class B extends A { int k;
void showk() {
System.out.printIn("к: " + к); } void sum() {
System.out.println("i+j+k: " + (i+j+k)); } }
class Simplelnheritance {
public static void main(String args[])
 { A superOb = new A(); B subOb = new B();
// Суперклас може бути використаний сам по
    собі.
superOb.i = 10;
superOb.j = 20;
System.out.println("Содержимое superOb: ");
superOb.showij();
System.out.println();
/* Підклас має доступ до всіх public-членам його
    суперкласу. */
subOb.i = 7;
subOb.j =8;          ,
subOb.k = 9;
System.out.println("Содержимое of subOb: ");
    subOb.showij(); subOb.showk();
    System.out.println();
System.out.println("Сумма i, j и к в subOb:");
    subOb.sum(); }
}
```

Доступ до елементів і наслідування

- Хоча підклас включає всі елементи (члени) свого суперкласу, він не може звертатися до тих елементів суперкласу, які були оголошені як private.

Створення багаторівневої ієрархії

- Ми можемо зробити опис своїх домашніх тварин (pets): кішок (cats), собак (dogs), корів (cows) та інших наступним чином:

```
class Pet {/ / Тут описуємо загальні властивості всіх домашніх
улюбленців
Master person; / / Господар тварини
int weight, age, eatTime; / / вага, вік, час годування
int eat (int food, int drink, int time) {/ / Процес годування
                                        / / Початкові дії ...
if (time == eatTime) person.getFood (food, drink);
            / / Метод споживання їжі
}
void voice (); / / Звуки, що видаються тваринам
              / / Інше ...
}
```

Створення багаторівневої ієрархії

Потім створюємо класи, описують більш конкретні об'єкти, пов'язуючи їх із загальним класом:

```
class Cat extends Pet {/ / Описуються властивості,
притаманні лише кішкам:
int mouseCatched; / / число спійманих мишей
void toMouse (); / / процес лову мишей
                    / / Інші властивості
}

class Dog extends Pet {/ / Властивості собак:
void preserve (); / / охороняти
}
```

Створення багаторівневої ієрархії

Зауважте, що ми не повторюємо загальні властивості, описані в класі Pet. Вони успадковуються автоматично. Ми можемо визначити об'єкт класу Dog і використовувати в ньому всі властивості класу Pet так, як ніби вони описані в класі Dog:

Dog tuzik = new Dog (), sharik = new Dog ();

Після цього визначення можна буде написати

tuzik.age = 3;
int p = sharik.eat (30, 10, 12);

А класифікацію продовжити так:

class Pointer extends Dog {... } / / Властивості породи Пойнтер
class Setter extends Dog {... } / / Властивості сеттерів

Створення багаторівневої ієрархії

Зауважте, що на кожному наступному рівні ієрархії в клас додаються нові властивості, але ні одна властивість не пропадає. Тому і вживається слово extends - "розширює" і говорять, що клас Dog - розширення (extension) класу Pet. З іншого боку, кількість об'єктів при цьому зменшується: собак менше, ніж усіх домашніх тварин. Тому часто кажуть, що клас Dog - підклас (subclass) класу Pet, а клас Pet - суперклас (superclass) або надклас класу Dog.

Часто використовують генеалогічну термінологію: батьківський клас, дочірній клас, клас-нащадок, клас-предок. Клас Dog успадковує клас Pet.

Опишемо в класі Master власника домашнього зоопарку.

```
class Master {/ / Господар тварини
String name; / / Прізвище, ім'я
        / / Інші відомості
void getFood (int food, int drink); / / Годування
        / / Інше
}
```

Модифікатори обмеження доступу до елементів при спадкуванні

```
class Bisection2 {
    private static double final EPS = 1e-8; / / Константа
    private double a = 0.0, b = 1.5, root; / / Закриті поля
    public double getRoot (} {return root;} / / Метод
    доступу
    private double f (double x)
    {
    return x * x * x - 3 * x * x + 3; / / Або щось інше
    }
    private void bisect () {/ / Параметрів ні -
                    / / Метод працює з полями примірника
    double y = 0.0; / / Локальна змінна - не поле
    do {
    root = 0.5 * (a + b); y = f (root);
    if (Math.abs (y) <EPS) break;
    / / Корінь знайдений. Виходимо з циклу
    / / Якщо на кінцях відрізка [a; root]
    / / Функція має різні знаки:
if (f (a) * y <0.0} b = root;
        / / Означає, корінь тут
        / / Переносимо точку b в точку root
        / / В іншому випадку:
    else a = root;
        / / Переносимо точку a в точку root
        / / Продовжуємо, поки [a; Ь] не стане малий
    } While (Math.abs (b-a)> = EPS);
    }

    public static void main (String [] args) {
    Bisection2 b2 = new Bisection2 ();
    b2.bisect ();
    System.out.println ("x =" +
    b2.getRoot () + / / Звертаємося до кореня
    / / Через метод доступу
    ", F () =" + b2.f (b2.getRoot ()));
    }
    }
```

Поняття та використання абстрактних класів

- При описі класу Pet ми не можемо поставити в методі voice () ніякої корисний алгоритм, оскільки у всіх тварин абсолютно різні голоси. В таких випадках ми записуємо тільки заголовок методу і ставимо після закриває список параметрів дужки крапку з комою. Цей метод буде абстрактним (abstract).
  Використовувати абстрактні класи можна тільки породжуючи від них підкласи, в яких перевизначені абстрактні методи.

Поняття та використання абстрактних класів

- Хоча елементи масиву singer [] посилаються на підкласи Dog, Cat, Cow, але все-таки це змінні типу Pet і посилатися вони можуть тільки на поля і методи, описані в суперклас Pet. Додаткові поля підкласу для них недоступні. Якщо звернутися, наприклад, до поля k класу Dog, написавши singer [0]. k, то отримаємо відгук про неможливість реалізувати таке посилання. Тому метод, який реалізується в кількох підкласах, доводиться виносити в суперклас, а якщо там його не можна реалізувати, то оголосити абстрактним. Абстрактні класи групуються на вершині ієрархії класів.
- Можна задати порожню реалізацію методу, просто поставивши пару фігурних дужок, нічого не написавши між ними, наприклад: void voice () {}
  Вийде повноцінний метод. Але це штучне рішення, заплутують структуру класу.

Остаточні члени і класи

- Позначивши метод модифікатором final, можна заборонити його перевизначення в підкласах. Це зручно в цілях безпеки. Ви можете бути впевнені, що метод виконує ті дії, які ви задали. Саме так визначено математичні функції sin (), cos () та інші в класі Math. Ми впевнені, що метод Math.cos (x) обчислює саме косинус числа x.
- Зрозуміло, такий метод не може бути абстрактним.
  Для повної безпеки, поля, оброблювані остаточними методами, слід зробити закритими (private).

Остаточні члени і класи

- Якщо ж позначити модифікатором final весь клас, то його взагалі не можна буде розширити. Так визначений, наприклад, клас Math:
  public final class Math {. . . }
- Для змінних модифікатор final має зовсім інший сенс. Якщо позначити модифікатором final опис змінної, то її значення (а воно має бути обов'язково задано або тут же, або в блоці ініціалізації або в конструкторі) не можна змінити ні в підкласах, ні в самому класі. Мінлива перетворюється на константу. Саме так у мові Java визначаються константи:
  public final int MIN_VALUE = -1, MAX_VALUE = 9999;
- За угодою "Code Conventions" константи записуються прописними буквами, слова в них поділяються знаком підкреслення.
  На самій вершині ієрархії класів Java стоїть клас Object.

Клас Object

- В Java визначений один спеціальний клас - object. Всі інші класи є його підкласами, object - це суперклас всіх інших класів. Це означає, що посилальна змінна типу object може звертатися до об'єкта будь-якого іншого класу. Крім того, т. к. масиви реалізуються як класи, змінна типу object може також звертатися до будь-якого масиву.

Клас Object

- Якщо при описі класу ми не не пишемо слово extends та ім'я класу за ним, то Java вважає цей клас розширенням класу object, і компілятор дописує це за нас:
  class Pet extends Object {. . . }
- Можна записати це розширення і явно.
- Сам же клас object не є нічиїм спадкоємцем, від нього починається ієрархія будь-яких класів Java. Зокрема, всі масиви - прямі спадкоємці класу object.
- Оскільки такий клас може містити лише загальні властивості всіх класів, в нього включено лише кілька самих загальних методів, наприклад, метод equals (), що порівнює цей об'єкт на рівність з об'єктом, заданим в аргументі, і повертає логічне значення. Його можна використовувати так:

```
Object obj1 = new Dog(), obj2 = new Cat();
if (obj1.equals(obj2)) ...
```

Клас Object

- Об'єкт obj1 активний, він сам порівнює себе з іншим об'єктом. Можна, звичайно, записати і obj2.equals (obj1), зробивши активним об'єкт obj2, з тим же результатом.
  Посилання можна порівнювати на рівність і нерівність:
  obj1 == obj2; obj1! = obj 2;
  В цьому випадку зіставляються адреси об'єктів, ми можемо дізнатися, не вказують чи обидві посилання на один і той же об'єкт.

Клас Object

- Метод equals () ж порівнює вміст об'єктів в їх поточному стані, фактично він реалізований в класі object як тотожність: об'єкт дорівнює тільки самому собі.
- Тому його часто перевизначають в підкласах, більш того, правильно спроектовані класи повинні перевизначити методи класу object, якщо їх не влаштовує стандартна реалізація.
- Другий метод класу object, який слід перевизначати в підкласах, - метод tostring (). Це метод без параметрів, який намагається вміст об'єкта перетворити в рядок символів і повертає об'єкт класу string.
- До цього методу виконуюча система Java звертається щоразу, коли потрібно представити об'єкт у вигляді рядка, наприклад, у методі printing.

Методи object

Метод	Мета
Object clone ()	Створює новий об'єкт, який є таким же, як імітованим об'єкт
boolean equals (Object object)	Визначає, чи є один об'єкт рівним іншому
void finalize ()	Викликається перш, ніж невикористану об'єкт буде перероблений (складальником сміття)
Class getciass ()	Отримує клас об'єкта під час виконання
int hashCode()	Повертає хеш-код, пов'язаний з викликом об'єкта
void notify()	Відновлює виконання потоку, що очікує на об'єкті виклику
void notifyAU()	Відновлює виконання всіх потоків, які очікують на об'єкті виклику
string toStringO	Повертає рядок, який описує об'єкт
void wait () void wait(long milllsrconds) void wait(long mllllsrconds, int nanoseconds)	Чекає виконання на іншому потоці

Методи object

- Методи getClass (), notify (), notifyAU () і wait () оголошені як final. Інші можна перевизначати.
- Тут відзначимо два методи: equals () і toString ().
- Метод equals () про порівнює вміст двох об'єктів. Він повертає true, якщо об'єкти еквівалентні, і false-в противному випадку.
- Метод ToString () повертає рядок, що містить опис об'єкта, на якому він викликається. Крім того, цей метод викликається автоматично, коли об'єкт виводиться методом println ().

Використання ключового слова super (перший вид)

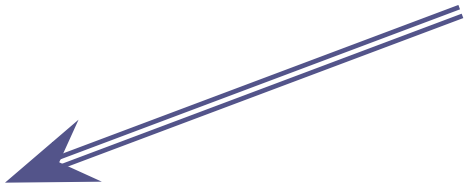
- Підклас може викликати метод конструктора, визначений його суперкласом, за допомогою наступної форми super:
  super (parameter-list);
- Тут parameter-list - список параметрів, який визначає будь-які параметри, необхідні конструктору в суперкласу. Схожий за формою на конструктор super () повинен завжди бути першим оператором, що виконуються всередині конструктора підкласу.

```
/ / BoxWeight тепер використовує
    / / Super для ініціалізації
    / / Box-атрибутів.
    class BoxWeight extends Box {
    double weight; / / вага блоку
    / / Ініціалізувати width,
    / / Height і depth, використовуючи
    / / Super ()
    BoxWeight (double w, double h, double d,
    double m) {
    super (w, h, d); / / викликати
    / / Конструктор суперкласу
    weight = m;}}
```

Використання другої форми super

- Загальний формат такого використання super має вигляд:
  super. member
  де member може бути або методом, або змінною екземпляра.
  Друга форма super найбільше застосовна до ситуацій, коли імена елементів (членів) підкласу приховують елементи з тим же ім'ям в суперкласу.

  i з суперкласу: 1
  i з підкласу: 2

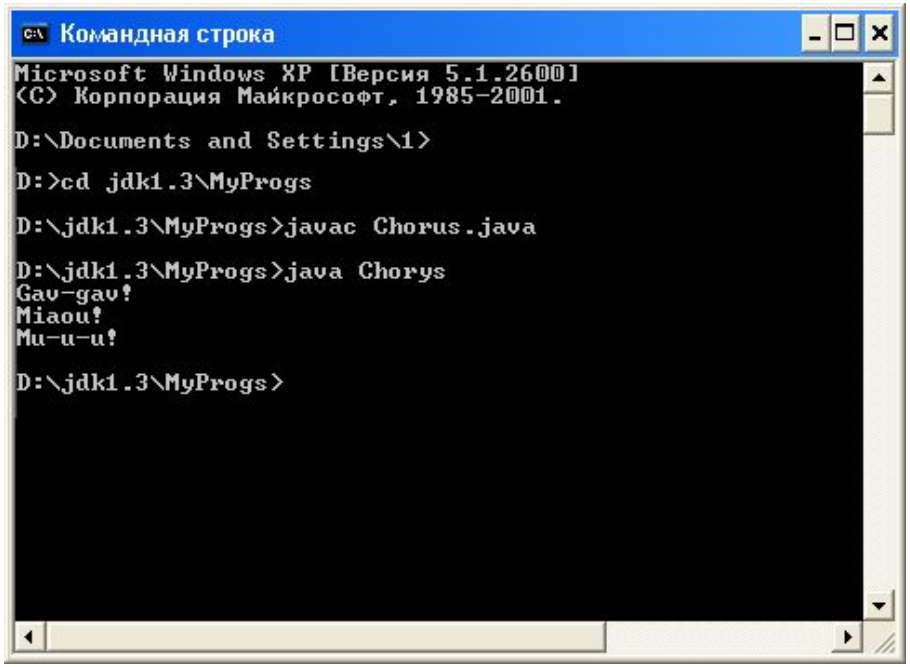


```
/ / Використання super для
/ / Подолання приховування імен,
class A {int i;}
/ / Створення підкласу B розширенням
/ / Класу A.
class B extends A {
int i; / / цей i приховує i в A
B (int a, int b) {
super.i = a; / / i з A
i = b; / / i з B
}
void show () {
System.out.println ("i з суперкласу:" + super.i);
System.out.println ("i з підкласу:" + i);}}
class UseSuper {

public static void main (String args []) {
B subOb = new B (1, 2);
subOb.show ();}
}
```

Перевизначення методів

```
abstract class Pet{
   abstract void voice();  }
class Dog extends Pet{
void voice(){
      System.out.printin("Gav-gav!");    } }
class Cat extends Pet{    void voice () {
     System.out.printin("Miaou!");     } }
class Cow extends Pet{     void voice(){
      System.out.printin("Mu-u-u!");    } }

public class Chorus(
   public static void main(String[] args){
      Pet[] singer = new Pet[3];
      singer[0] = new Dog();
      singer[1] = new Cat();
      singer[2] = new Cow();
      for (int i = 0; i < singer.length; i++)
         singer[i].voice();
   }
}
```

Вся справа тут у визначенні поля singer []. Хоча масив посилань singer [] має тип Pet, кожен його елемент посилається на об'єкт свого типу Dog, Cat, cow. При виконанні програми викликається метод конкретного об'єкта, а не метод класу, яким визначалося ім'я посилання.



Перевизначення методів

```
class A { int i,   j;
A(int a,   int b)   {
i  = a;
j  = b; }
// показати i та j на екрані
void show() (
System.out.println("i и j: " + i + " " + j); } }
class B extends A { int k;
B(int a, int b, int c) {
super(a, b);
k = c; }
// Показати на екрані к (цейshow(i перевизначає show() з A)
void show() {
System.out.println("k: " + k); } }
class Override {
public static void main(String args[]) { B subOb = new B(1, 2, 3);
subOb.showO;  // тут викликається show() з B
} }
```

Динамічна диспетчеризація методів

- Динамічна диспетчеризація методів це механізм, за допомогою якого рішення на виклик перевизначення функцій приймається під час виконання, а не під час компіляції.
- Принцип: посилальна мінлива суперкласу може звертатися до об'єкта підкласу. Java використовує цей факт, щоб приймати рішення про виклик перевизначених методів під час виконання.

Динамічна диспетчеризація методів

- Коли перевизначення методів викликається через посилання суперкласу, Java визначає, яку версію цього методу слід виконувати, грунтуючись на типі об'єкта, на який вказує посилання в момент виклику. Це визначення робиться під час виконання. Коли посилання вказує на різні типи об'єктів, будуть викликатися різні версії перевизначеного методу.
- Іншими словами, саме тип об'єкта, на який зроблено посилання (а не тип посилальної змінної) визначає, яка версія перевизначеного методу буде виконана.

Динамічна диспетчеризація методів

```
class A {
void callme () {
System.out.println ("Усередині A метод
callme");}}
class B extends A (
/ / Перевизначити callme ()
void callme {) {
System.out.println ("Усередині B метод
callme");
}}
class З extends A {
/ / Перевизначити callme ()
void callme () {
System.out.println ("Усередині З метод
callme");
}}
```

```
class Dispatch {
    public static void main (String args []) {?
    A a = new A (); / / об'єкт типу A
    B b = new B (); / / об'єкт типу B
    C з = new C (); / / об'єкт типу C
    A r; / / визначити посилання типу A
    r = a; / / r на A-об'єкт
    r.callme () ;/ / викликає A-версію callme
    r = b; / / r вказує на B-об'єкт
    r.callme () ;/ / викликає B-версію callme
    r = c; / / r вказує, на C-об'єкт,
    r.callme () ;/ / викликає C-версію callme
    }
    }
```

Використання ключового слова final з наслідуванням

- Ключове слово final має три застосування. Перше - його можна використовувати для створення еквіваленту іменованої константи. Два інших застосування final пов'язані зі спадкуванням.

Використання final для відмови від перевизначення

- Щоб скасувати перевизначення методу, вкажіть модифікатор final на початку його оголошення. Методи, оголошені як final, не можуть перевизначатися.

```
class A {
final void meth () {
System.out.println ("Це метод final.");
}}
class B extends A {
void meth () {/ / ПОМИЛКА! Не можна перевизначати.
System.out.println ("Помилка!");
}}
```

- Оскільки meth про оголошений як final, він не може бути перевизначений в класі B. Якщо ви спробуєте зробити це, то отримаєте помилку під час компіляції.

Використання final для скасування наслідування

- Іноді потрібно розірвати спадкову зв'язок класів (скасувати успадкування одного класу іншим). Щоб зробити це, передуватимете оголошення класу ключовим словом final, що дозволить неявно оголосити і всі його методи. Неприпустимо оголошувати клас одночасно як abstract і final.
  final class A {/ / ...
  }
  / / Наступний клас незаконний.
  class B extends A {/ / ПОМИЛКА! B не може бути підкласом A
  / / .. . }
- Коментар тут означає, що в не може успадковувати A, тому що A оголошений як final.