### 3.2. Ознайомлення з класами

Класи є ядром мови *Java*. Що б ми не захотіли реалізувати в *Java*-програмі, її неохідно оформити у вигляді класу. Найважливіше, що необхідно знати про клас – це те, що він визначає новий тип даних. Один раз описаний клас можна надалі використовувати з метою створення об’єктів цього класу. Отже, клас є шаблоном (*template*) об’єкта, а об’єкт – екземпляром (*instance*) класу.

#### 3.2.1. Визначення класу

Визначення класу полягає в описі його вигляду і природи. Це відбувається за допомогою визначення даних, які в ньому містяться, і програмного коду, який керує цими даними. Загальний вигляд оголошення класу записують так:

*class ім’я\_класу{ тип об’єктна\_змінна1; тип об’єктна\_змінна 2;*

*//… тип об’єктна\_змінна N;*

*тип ім’я\_методу1(список параметрів){*

*// тіло методу*

*}*

*тип ім’я\_методу2(список параметрів){*

*// тіло методу*

*}*

*//…*

*тип ім’я\_методуN(список параметрів){*

*// тіло методу*

*} }*

Методи і змінні, визначені всередині класу, називають членами (*members*) цього класу.

Дані або змінні, визначені у блоці *class,* можуть належати конкретному екземплярові класу (об’єктові), внаслідок чого їх називають змінними екземпляра, або бути глобальними змінними (*instance variables*), спільними для всіх екземплярів конкретного класу і носити назву змінних класу. З метою визначення змінної класу перед її оголошенням необхідно додати модифікатор *static.* Статичні змінні зберігаються в одному місці оперативної пам’яті і доступні завжди протягом виконання програми з усіх екземплярів класу. Якщо ще в оголошенні змінної додати модифікатор *final,* то значення змінної не можна змінювати в підкласах. Фактично − це оголошення константи. За загальноприйнятими правилами імена таких змінних пишуть великими літерами.

Програмний код міститься в методах. Головні частини методу − це ім’я, параметри, тип значення, яке повертається, і тіло. Методи в мові *Java* створюються тільки як частини класу. Це її основна відмінність від інших мов програмування, в яких можна окремо написати і виконати функцію або процедуру. Список параметрів задає типи та імена для інформації, яку необхідно передати в метод. Ім’я методу і список параметрів винятково ідентифікують метод. Для виходу з методу використовують оператор *return,* за яким іде вираз, значення якого необхідно повернути. Метод може повертати значення довільного типу або не повертати нічого. У таких випадках на місці типу в оголошенні методу необхідно написати слово *void*.

Як приклад простого класу наведемо клас *Box* (коробка) *class Box { double width; double heigth;*

*double depth; }*

Цей клас визначає тип даних *Box*. Його можна використовувати для визначення об’єктів. Важливо пам’ятати, що описуючи клас, ми задаємо лише шаблон. Реальний об’єкт типу *Box* при цьому не створюється. Для того, щоб створити об’єкт типу

*Box*, необхідно записати оператор виду

*Box mybox = new Box(); // створює об’єкт з іменем mybox*

Для доступу до змінних класу *Box* необхідно використати оператор "крапка" (.). Розглянемо приклад програми, яка використовує клас Box:

*// оголошення класу Box class Box { double width; double heigth;*

*double depth;*

*}*

*// у цьому класі створюється об’єкт типу Box class BoxDemo{ public static void main(String args[ ]){ Box mybox=new Box();*

*double vol;*

*// присвоєння значень змінним класу mybox.width=10; mybox.heigth=20; mybox.depth=30;*

*// обчислення об’єму коробки vol=mybox.width \* mybox.heigth \* mybox.depth;*

*System.out.println ("об’єм коробки ="+vol);*

*} }*

Для створення об’єкта типу *Box*, використовують оператор:

*Box mybox = new Box();*

У цьому операторі об’єднано два кроки: оголошення змінної *mybox* і створення реального об’єкта в пам’яті на стадії виконання.

Його можна розбити на два:

*Box mybox; // Оголошує посилання на об’єкт типу Box,*

*// яка має значення null;*

*mybox=new Box();//Розміщує об’єкт типу Box в пам’яті і*

*// присвоює mybox адресу, // куди розмістився об’єкт Box.* Посилання на об’єкти – це покажчики на комірки в пам’яті, яку ще називають "купою". У мові *Java* не можна присвоїти цьому покажчикові довільне значення або оперувати з ним як з цілим (на відміну від *С/С++*).

Файл з програмою необхідно назвати *BoxDemo.java*, тому що метод *main()* міститься в цьому класі. Класи *Box* і *BoxDemo* можна розмістити також в окремих файлах *Box.java, BoxDemo.java*.

***Детальніший погляд на оператор*** *new****.*** Оператор *new* динамічно (під час виконання) розміщує об’єкт в оперативній пам’яті загального призначення (купі).

Його загальний вигляд *змінна\_класу= new ім’я\_класу();*

Ім’я класу, після якого стоять круглі дужки – це виклик конструктора класу (*constructor*). Конструктор описує дії, які виконуються при створенні об’єкта цього класу. Конструктор задається всередині визначення класу. Якщо конструктор явно не заданий (як і було в класі *Box*), *Java* автоматично викликає заданий за замовчуванням конструктор.

Оператор *new* виокремлює пам’ять для об’єкта під час виконання програми. Якщо пам’яті для наступного об’єкта не вистачає, виникає виняткова ситуація, яку необхідно опрацювати. Цим мова *Java* відрізняється від мови *С++*, де у випадку невдачі оператор *new* повертає *null*.

***Присвоєння значень змінним-посиланням на об’єкти.*** Після виконання фрагмента програми

*Box b1=new Box();*

*Box b2=b1; b1* і *b2* посилаються на один і той же об’єкт в пам’яті. Внаслідок присвоєння *b2=b1* не буде зроблено копії об’єкта *b1*. Тобто внаслідок присвоєння значення однієї змінної-посилання на об’єкт іншій змінній- посиланню створюється копія посилання, а не копія цього об’єкта. Для копіювання об’єктів існують інші засоби.

#### 3.2.2. Методи класу

Класи сформовано переважно з двох частин: змінних і методів. Кожний з методів за своїми функціями можна зачислити до однієї з семи категорій:

1. Конструктори. Викликаються для створення екземплярів об’єктів деякого класу.
2. Деструктори. Викликаються, коли робота з екземпляром об’єкта закінчена.
3. Копіювальники. Використовуються для копіювання екземпляра об’єкта в інший.
4. Set-методи. Викликаються з метою присвоєння значення змінній класу.
5. Get-методи. Зчитування значення змінної класу.
6. Методи введення-виведення. Викликаються для виконання взаємодії з зовнішніми пристроями.
7. Методи, специфічні для області визначення застосування.

Чимало експертів з програмування вважає, що тіло методу повинно містити до 20-ти стрічок або менше, тобто приблизно одну сторінку екранного тексту. В цьому випадку метод можна прочитати і зрозуміти з першого разу.

***Методи в класі Box.*** Розширимо визначення вже згадуваного класу *Box*, додавши два методи:

*class Box { double width; double height; double depth;*

*// метод обчислення об’єму double volume( ){ return width\*height\*depth;*

*} // завершення методу volume*

*// метод задання розмірів коробки void setDim(double w, double h, double d){ width=w; height=h; depth=d;*

*} // завершення методу setDim*

*} // завершення визначення класу* Декілька зауважень до тексту програми:

1. у методах класу посилання на змінні класу відбувається безпосередньо без посилання на об’єкт;
2. у методі *SetDim* використані параметри (*параметр* – це визначена в методі змінна, якій внаслідок виклику методу присвоюється значення, яке називають *аргументом).*
3. коректно розроблена програма – це програма, в якій доступ до змінних класу відбувається через методи класу (у нашому випадку реалізовано лише присвоєння цим змінним значень у методі *SetDim()*).

***Передавання параметрів у мові Java.*** Передавання аргументів у методи здійснюється двома способами. Прості типи в мові *Java* (як і в більшості мов) передаються за значеннями (*call-byvalue*), тобто у формальний параметр копіюється значення аргументу. Все, що відбувається з параметром, не впливає на аргумент.

Об’єкти передаються за посиланням (*call-by-reference*). Тобто параметрові присвоюється значення посилання на об’єкт. Обидва посилання − і аргумент, і параметр − вказують на один і той же об’єкт у пам’яті. Тому всі зміни, які відбулися в методі, впливають на об’єкт, переданий як аргумент.

***Використання об’єктів у ролі параметрів.*** В усіх попередніх прикладах у ролі параметрів методів передавалися прості типи. Проте в метод класу в ролі параметра можна передати об’єкт будь-якого типу. Розглянемо програму:

*clas Test {*

*int a, b;*

*Test (int і, int j){ // конструктор класу a=i; b=j;*

*}*

*// метод, який повертає true, якщо об’єкт 0 рівний тому,*

*// для якого викликається цей метод boolean eguals (Test 0){ if (0.a==a && 0.b==b) return true; else return false;*

*}*

*} // завершення класу Test class Passob {*

*public static void main (String args [ ]){*

*Test ob1=new Test(100,22);*

*Test ob2=new Test(100,22);*

*Test ob3=new Test(-1,-1);*

*System.out.println ("ob1==ob2: "+ob1.equals(ob2));*

*System.out.println ("ob1==ob3: "+ob1.equals(ob3));*

*}*

*}// завершення класу Passob*

Результатом виконання цієї програми буде:

*ob1==ob2: true ob1==ob3: false.*

Цей приклад засвідчує, що параметром може бути не тільки будь-який вбудований тип, але й клас, для якого цей метод визначається.

Особливо зручно робити це в конструкторах для створення точної копії вже існуючого об’єкта. Наприклад, у класі *Box* можна додати ще один варіант конструктора:

*// створення дубліката об’єкта*

*Box (Box ob) { widht=ob.width; height =ob.height;*

*depth=ob.depth;*

*}*

***Повернення об’єктів методами.*** Метод може повертати не тільки прості типи даних, але і значення, описане як клас. Розглянемо програму

*// Приклад повернення об’єкта з методу*

*class Test { int a;*

*Test(int i ){ a=i; }*

*// Цей метод повертає об’єкт типу Тest,*

*// в якому a збільшене на 10*

*Test incrByten( ) {*

*Test temp = new Test(a+10);*

*return temp; }*

*}// завершення класу Test class Retob { public static void main (String args[ ]){ Test ob1=new Test(2), ob2; ob2=0b1.incrByten( ); ob2=0b2.incrByten( );*

*System.out.println ("ob1.a= "+ob1.a);*

*System.out.println ("ob2.a= "+ob2.a);*

*System.out.println ("ob2.a= "+ob2.a);*

*}*

*}*

Результатом виконання цієї програми будуть значення:

*ob1.a=2 ob2.a=12 ob2.a=22.*

Отже об’єкт *ob2* створюється не викликом безпосередньо конструктора класу, а методу *incrByten*, який повертає об’єкт. Хоча оператор *new* виокремив пам’ять у методі, який закінчив свою роботу, об’єкт буде існувати до того часу, доки в програмі буде хоча б одне посилання на нього.

***Рекурсія.*** У мові *Java* підтримується механізм рекурсії (*recursion*), за допомогою якого метод може звертатися до самого себе. Такий метод називають рекурсивним.

Класичним прикладом рекурсії є обчислення факторіала:

*// Приклад рекурсивного методу*

*class Factorial { int fact(int n){ int result; if (n==1) return 1;*

*// метод викликає сам себе result = fact(n-1)\*n; return result;*

*} // завершення методу fact*

*} // завершення класу Factorial*

*class Recursion { public static void main(String args[ ]) {*

*Factorial f = new Factorial( );*

*System.out.println ("Факторіал 4=" +f.fact(4));*

*}*

*}*

При рекурсивному викликові методу в стек поміщаються нові локальні змінні і параметри і код методу починає виконуватися з початку із новими змінними. При рекурсивному виклику не створюється нова копія методу, новими стають тільки аргументи. При обчисленні факторіалу звертання відбувається доти, поки не буде повернена одиниця, яка в оберненому порядку помножиться на значення *n* (для кожного виклику своє).

При написанні рекурсивних методів треба використовувати хоча б один оператор *if*, щоб метод завершив роботу.

Рекурсивні версії багатьох програм можуть працювати довше, ніж їх ітераційні еквіваленти, через затрати на додаткові виклики функції. Надзвичайно велика кількість викликів методу може стати причиною переповнення стеку, в якому зберігаються параметри і локальні змінні методу. Але є думка, що мислити в термінах рекурсії простіше, ніж в термінах ітерацій.

#### 3.2.3. Конструктори

При створенні чергового екземпляра класу проводити ініціалізацію усіх його змінних є клопітно, можна щось випадково упустити. Для того в класах існують спеціальні методи, які називають конструкторами. Їхнє призначення полягає в ініціалізації внутрішнього стану об’єкта так, щоб внаслідок виконання оператора *new()* ми отримували цілковито завершений щодо користування об’єкт.

Конструктор (*constructor*) здійснює ініціалізацію об’єкта при його створенні. Ім’я конструктора завжди збігається з іменем класу, а за синтаксисом він нагадує метод. При написанні конструктора явно не вказується тип значення, яке він повертає, зокрема і *void*. Насправді, він повертає об’єкт типу, який визначено класом, конструктором якого він є.

Якщо конструктор класу не описано явно, *Java* створює для цього класу конструктор за замовчуванням. Конструктор за замовчуванням присвоює змінним класу нульові значення. Для простих класів цього буває достатньо. Якщо при визначенні класу є явно описаний конструктор, то при створенні екземпляра класу конструктор за замовчуванням не викликається: не можна викликати конструктор класу без параметрів, попередньо не написавши його.

У конструктор можна передавати параметри. У визначення класу *Box* (п. 3.2.1) можна додати конструктор:

*// конструктор класу Box*

*Box(double w, double h, double d){ width = w; height=h; depth=d; }*

*// демонстрація виклику конструктора з параметрами class BoxDemo{ public static void main(string args[ ]){ Box mybox = new Box(10, 20, 15); double vol = mybox.volume( ); } }*

#### 3.2.4. Перевантаження методів

Одним із способів реалізації поліморфізму є перевантаження методів (*metod overloading*). У мові *Java* (як і в інших мовах) можна в одному класі визначати декілька методів з одним іменем, однак різною кількістю параметрів. Ці методи називають перевантаженими (*overloaded*).

При перевантаженні методу, щоб визначити, яку версію методу треба виконати, використовується кількість і (або) тип аргументів. Типи значень, які повертає метод, можуть бути різними і вони не використовуються як ознака, за якою розрізняють дві версії методу. Коли компілятор *Java* зустрічає виклик перевантаженого методу, виконується та версія, параметри якого відповідають заданим під час виклику аргументам.

*// Демонстрація перевантаження методів test() class OverloadDemo { // метод без параметрів*

*void test() {*

*System.out.println("Без параметрів");*

*}*

*// метод з одним цілочисельним параметром void test(int a) {*

*System.out.println ("Ціле число a="+a);*

*}*

*// метод з двома цілочисельними параметрами void test(int a, int b) {*

*System.out.println ("Цілі числа a і b"+a+","+b);*

*}*

*// метод з параметрам типу double*

*double test(double a) {*

*System.out.println ("double a="+a);*

*return a\*a;*

*}*

*} // завершення визначення класу OverloadDemo class Overload { public static void main(String args[ ] ){*

*OverloadDemo ob=new OverloadDemo( ); double result;*

*// виклик усіх версій методу test*

*ob.test(); ob.test(10); ob.test(10, 20); result=ob.test(123.2);*

*System.out.println ("Результат="+result);*

*} }*

Якщо типи параметрів не збігаються з параметрами хоча б одного з методів, то відбувається автоматичне перетворення типів. Заберемо в попередньому прикладі у визначенні класу *OverloadDemo* версію методу *test(int a)*. Тоді при звертанні *ob.test(10)* відбудеться перетворення 10 в *double* і виклик методу *test(double a)*. При наявності необхідного методу перетворення типів не відбувається і звертання здійснюється до відповідного методу без перетворення типів.

Без реалізації перевантажених методів кожний метод повинен був би мати унікальне ім’я. У мовах програмування, які не допускали перевантаження методів (наприклад, *Fortran, С*), для обчислення усіх математичних функцій визначено свої назви для аргументів цілого типу, дійсного, з подвійною точністю. Функція *аbs()* - повертала модуль цілого числа, *labs()* – цілого числа подвійної довжини, *fabs()* – дійсного числа і т. п.

Цінність перевантаження методів полягає у можливості присвоїти одне ім’я спорідненим методом. Ім’я у цьому випадку вказує на спільну дію, яка не залежить від типів аргументів. Вибір конкретної версії методу здійснює компілятор.

Унаслідок перевантаження методів кожна його версія може виконувати будь-які дії. Немає правила, яке б вимагало, щоб перевантажені методи мали виконувати подібні функції. Проте з погляду стилю ООП, під час перевантаження методів вже зрозуміло, що це реалізація поліморфізму і виконуються споріднені операції.

#### 3.2.5. Перевантаження конструкторів

У мові *Java* перевантаження конструкторів – це норма, що й розглянемо на прикладі класу *Box* (п. 3.2.1*)*. Для цього класу було визначено конструктор з трьома параметрами (розмірами коробки). Це означає, що кожне створення нового об’єкта типу *Box* вимагає задати певні розміри. Як бути у випадку, коли вони невідомі, або несуттєві при створенні нового об’єкта. Розв’язати ці очевидні ситуації при створенні об’єкта типу *Box* можна за допомогою перевантаження конструктора. У цьому випадку реалізація класу *Box* буде такою:

*class Box { double width; double height; double depth;*

*// конструктор, коли всі розміри задано Box (double w, double h, double d){*

*width =w; height=h;*

*depth=d;*

*}*

*// конструктор, коли всі розміри не задано*

*Box( ) {*

*width =-1; height=-1;*

*depth=-1;*

*}*

*// конструктор, який створює куб*

*Box(double len) {*

*width = height = depth = len;*

*}*

*// метод обчислення об’єму коробки double volume( ) { return width\*height\* depth;*

*}*

*} // завершення визначення класу Box class OverloadCons { public static void main(String args [ ]){*

*// створення об’єктів Box*

*// за допомогою різних конструкторів Box mybox1=new Box(10,20,15);*

*Box mybox2=new Box( );*

*Box mycube=new Box(7); double vol;*

*vol = mybox1.volume( );*

*}*

*}*

#### 3.2.6. Основи керування доступом

Інкапсуляція – це не тільки зв’язування даних з кодом, який ними маніпулює, але й керування доступом (*accesscontrol*). Завдяки інкапсуляції можна дозволяти або забороняти доступ до елементів класу.

Спосіб доступу до елементів класу можна задати за допомогою специфікатора доступу (*access control*). У мові *Java* підтримується декілька спеціфікаторів доступу. Деякі з них призначені переважно для забезпечення спадковості або роботи з пакетами (набір групи класів). Ці механізми керування доступом розглянемо далі.

У мові *Java* є такі специфікатори доступу: *public, private і protected.* Існує поняття доступу, задане за замовчуванням. Член класу, визначений як *public,* є доступним з довільного місця програми, а член класу, визначений як *private,* –тільки для членів даного класу. Якщо специфікатор не задано, елемент класу вважається доступним в рамках його пакета, але не ззовні пакета.

Наведемо приклад класу, доступ до змінних якого здійснюється тільки через методи. Для оголошення змінної загальнодоступною мають бути дуже вагомі причини. Дана програма реалізує цілочисельний стек:

*class Stack { private int stck [ ]=new int [10]; private int tos;*

*// Ініціалізація вершини стека*

*Stack( ){ tos=-1;*

*}*

*// метод, який додає елемент у стек*

*void push(int item) { if (tos==9) System.out.println ("Стек повний"); else stck[++tos]=item;*

*}*

*// метод, який добуває елемент зі стека int pop( ) { if (tos<0) {*

*System.out.println ("Вихід за нижню границю"); return 0; }*

*else*

*return stck[tos--];*

*}*

*}*

#### 3.2.7. Поняття статичних даних

Бувають випадки, коли необхідно визначити член класу (змінну або метод), призначені для використання без створених конкретних об’єктів класу. Для створення такого члена класу необхідно перед його оголошенням поставити ключове слово *static*. До таких членів класу можна звертатися ще до створення хоча б одного екземпляра (об’єкта) класу. Статичними можуть бути як змінні, так і методи.

Екземпляри змінних, оголошених як *static*, є, за змістом, глобальними. При створенні об’єктів даного класу змінні не копіюються. Усі екземпляри класу користуються однією і тією ж статичною змінною.

На методи, оголошені як *static*, накладено такі обмеження:

− вони викликають тільки статичні методи;

− вони використовують тільки статичні змінні; − вони не можуть посилатися на *this* і *super*.

Виклик статичного методу класу здійснюють так:

*ім’я\_класу.ім’я\_методу( );*

Якщо для ініціалізації статичних змінних необхідно виконати певні обчислення, можна оголосити блок як *static,* який виконається тільки один раз − під час першого завантаження класу.

*// Демонстрація статичних змінних, методів і блоків class UseStatic {*

*static int a=3; static int b;*

*static void meth(int x){*

*System.out.println ("x"+x);*

*System.out.println ("a"="+a);*

*System.out.println ("b"="+b);*

*}*

*static {*

*System.out.println ("Статичний блок ініціалізації"); b=a\*4; }*

*public static void main (String args[ ]){ meth(42); }*

*}*

За допомогою статичних членів класу в мові *Java* реалізовано аналоги глобальних функцій і глобальних змінних.

#### 3.2.8. Збирання сміття. Метод finalize( )

Зважаючи на те, що об’єкти розміщуються в пам’яті динамічно за допомогою оператора *new*, виникає питання: як вони знищуються і як звільняється у цьому випадку пам’ять. У мові *С++* це здійснюють оператором *delete*. У мові *Java* звільнення пам’яті відбувається автоматично. Цю технологію називають збиранням сміття (*garbage collection*). Періодично виконавча система *Java* переглядає, чи є посилання на об’єкт. Якщо немає, то об’єкт знищують і пам’ять вивільняється. Однак збирання сміття може і не здійснюватися, якщо програма навіть не наближається до критичного використання ресурсів. У таких випадках пам’ять повертається в операційну систему з завершенням роботи програми. При різних запусках однієї і тієї ж програми мовою *Java* збирання сміття відбувається неідентично.

Трапляються випадки, коли перед знищенням об’єкта необхідно виконати певні дії: закрити файл і т. п. З цією метою в *Java* розроблений спеціальний механізм, який називають фіналізацією (*finalization*) і реалізують у методі *finalizе( )*:

*protected void finalize( ){*

*//дії, які необхідно виконати перед знищенням об’єкта*

*}*

Збирання сміття в *Java* відбувається періодично або не відбувається зовсім, тому не можна передбачити, коли спрацює метод *finalize()*. Для нормального режиму роботи програми покладатися на метод *finalize()* не можна. Насправді його використовують з метою стеження за процесом збирання сміття (за умови, що воно відбувається).

Для запуску позачергового процесу збирання сміття використовують виклик спеціального методу класу *System:*

*System.gc();*

Процес збирання сміття в мові *Java* впливає також на швидкість виділення пам’яті для нових об’єктів. Віртуальна машина *Java* (*JVM*) за часом розміщення об’єкта в пам’яті "купі" майже досягає рівня виділення пам’яті стека в інших мовах програмування.