Лабораторна робота № 4

Тема: Динамічна диспетчеризація методів

Мета: Вивчення принципів поліморфізму та динамічної диспетчеризації методів

Перевизначення методів є основою для одного із найбільш ефективних принципів Java — динамічної диспетчеризації методів. Динамічна диспетчеризація методів — це механізм, за допомогою якого виклик перевизначеного методу дозволяється під час виконання, а не компіляції. Динамічна диспетчеризація методів важлива тому, що завдяки їй поліморфізм Java реалізується під час виконання.

Змінна посилання із суперкласу може посилатися на об'єкт підкласу. Цей принцип використовується Java для дозволу викликів перевизначених методів під час виконання. Відбувається це таким чином: коли перевизначений метод викликається за посиланням на суперклас, необхідний його варіант обирається в Java залежно від типу об'єкту, на який робиться посилання в останній момент виклику. За посиланням різні типи об'єктів викликатимуть різні варіанти перевизначеного методу. Варіант перевизначеного методу залежить від типу об'єкту, на який робиться посилання, а не від типу змінної посилання. Якщо суперклас містить метод, що перевизначається у підкласі, то за посиланням на різні типи об'єктів через змінну посилань із суперкласу будуть виконуватися різні варіанти цього методу.

Приклад 1

```
// Динамічна диспетчеризація методів
class A {
        void callme ( ) {
        System.out.println( " Це метод callme( ) із класу A " ) ;
        }
}

class B extends A {
// перевизначення методу callme( )
        void callme( ) {
        System.out.println( " Це метод callme ( ) із класу B " ) ;
        }
}

class C extends A {
// перевизначення метода callme( )
        void callme( ) {
```

```
        System.out.println( " Це метод callme ( ) із класу C " ) ;
        }
}

class Dispatch {
  public static void main(String args[]) {
    A a = new A(); // об’єкт класу A
    B b = new B(); // об’єкт класу B
    C c = new C(); // об’єкт класу C
    A r; // змінна посилання типу A

    r = a; // r посилається на об’єкт класу A
    r.callme(); // викликається метод callme із класу A

    r = b; // r посилається на об’єкт класу B
    r.callme(); // викликається метод callme із класу B

    r = c; // r посилається на об’єкт класу C
    r.callme(); // викликається метод callme із класу C
  }
}
```

У наведеній програмі створюється один суперклас A та два його підкласи B та C. У підкласах B та C перевизначається метод callme(), який оголошено у класі A. У методі main() утворюються об’єкти класів A, B та C, а також змінна r, що посилається на об’єкт типу A. Далі змінній r привласнюється за чергою посилання на об’єкт кожного із класів A, B та C, і вже за посиланням на об’єкт викликається метод callme(). Виконуваний варіант методу callme() визначається типом об’єкту, на який робиться посилання під час виклику.

Особливе значення поліморфізму для ООП пояснюється тим, що він дозволяє визначити в загальному класі методи, які стануть загальними для усіх похідних від нього класів, а у підкласах можуть змінюватись реалізації деяких або всіх цих методів.

Перевизначені методи надають ще один спосіб реалізувати Java принцип поліморфізму — один інтерфейс, безліч методів. Однією з основних умов успішного застосування поліморфізму є розуміння, що суперкласи та підкласи утворюють ієрархію за ступенем збільшення спеціалізації. Якщо суперклас застосовується правильно, він надає всі елементи, які можуть використовуватися безпосередньо у підкласі. У ньому також визначаються ті методи, які мають бути реалізовані у похідному класі. Це дає можливість визначати у підкласі його методи, зберігаючи однаковість інтерфейсу. Таким чином, поєднуючи успадкування з перевизначеними методами, у суперкласі можна визначити загальну форму для методів, які використовуватимуться у всіх його підкласах.

Динамічний, тобто реалізований під час виконання поліморфізм є одним із найефективніших механізмів об'єктно-орієнтованої архітектури, що забезпечують повторне використання та надійність коду. Можливість викликати із бібліотек вже існуючий код методу для екземплярів нових класів, не вдаючись до повторної компіляції і водночас зберігаючи ясність абстрактного інтерфейсу, є дуже дієвим засобом.

У прикладі, що наводиться далі, створюється суперклас Figure для зберігання розмірів двовимірного об'єкту, а також визначається метод area() для розрахунку площі цього об'єкта. Крім того, створюються два класи, Rectangle та Triangle, похідні від класу Figure. Метод area() перевизначається у кожному з цих підкласів, щоб повертати площу чотирикутника та трикутника відповідно.

Приклад 2

```
// Використовуємо динамічний поліморфізм
class Figure {
  double dim1;
  double dim2;

  Figure(double a, double b) {
    dim1 = a;
    dim2 = b;
  }

  double area() {
    System.out.println("Площа фігури не визначено");
    return 0;
  }
}

class Rectangle extends Figure {
  Rectangle(double a, double b) {
    super(a, b);
  }

  // перевизначаємо метод area для прямокутника
  double area() {
    System.out.println("Площа прямокутника");
    return dim1 * dim2;
  }
}

class Triangle extends Figure {
  Triangle(double a, double b) {
```

```
    super(a, b);
  }

  // перевизначаємо метод area для трикутника
  double area() {
    System.out.println("Площа трикутника");
    return dim1 * dim2 / 2;
  }
}

class FindAreas {
  public static void main(String args[]) {
    Figure f = new Figure(10, 10);
    Rectangle r = new Rectangle(9, 5);
    Triangle t = new Triangle(10, 8);

    Figure figref;

    figref = r;
    System.out.println("Площа фігури = " + figref.area());

    figref = t;
    System.out.println("Площа фігури = " + figref.area());

    figref = f;
    System.out.println("Площа фігури = " + figref.area());
  }
}
```

Подвійний механізм ООП - наслідування та поліморфізм, дозволяє визначити єдиний інтерфейс, що під час виконання використовується різними типами об’єктів, які пов'язані через їх суперклас. Якщо об'єкт відноситься до класу, похідного від класу Figure, його площу можна розрахувати, викликавши метод area(). Інтерфейс виконання цієї операції залишається незмінним незалежно від виду фігури, а результат буде залежати від неї.

Інколи суперклас потрібно визначити таким чином, щоб оголосити в ньому структуру заданої абстракції, не надаючи повної реалізації кожного методу. Це означає створити суперклас, що визначає лише узагальнену форму для спільного використання усіма його підкласами, у кожному з яких можуть бути додані необхідні деталі. У такому класі визначаються методи, які мають бути реалізовані у підкласах. Подібна ситуація може виникнути, коли у суперкласі не вдається повністю визначитись з реалізацією методу. Так у попередньому прикладі в класі Figure визначення методу area() грає роль шаблону, не дозволяючи розрахувати та вивести площу об'єкта якогось

типу. У випадку, коли потрібно переконатися, що у підкласі перевизначаються необхідні методи, Java для реалізації цієї мети використовує абстрактний тип методу, що позначається модифікатором типу abstract. Іноді такі методи називаються методами під відповідальністю підкласу, оскільки у суперкласі їм ніякої реалізації не передбачено. Отже, ці методи мають бути перевизначені у підкласі, де не можна просто скористатися їх варіантом, визначеним у суперкласі. Для оголошення абстрактного методу використовується наступна загальна форма:

abstract тип ім’я (список параметрів) ;

Будь-який клас, що містить один або більше абстрактних методів, повинен бути оголошений як абстрактний. Для цього достатньо вказати ключове слово abstract перед ключовим словом класу на початку оголошення класу. Абстрактний клас не може мати жодних об'єктів. Екземпляр абстрактного класу може бути отриманий безпосередньо за допомогою оператора new. Не можна оголошувати абстрактні конструктори чи абстрактні статичні методи. Будь-який підклас, що є похідним від абстрактного класу, повинен реалізувати всі абстрактні методи зі свого суперкласу або сам бути оголошений як абстрактний.

Нижче наведено простий приклад класу, що вміщує абстрактний метод:

Приклад 3

```
// Приклад демонстрації використання абстрактного класу
abstract class A {
  abstract void callme();

// конкретний метод може залишатись доступним в абстрактному класі
  void callmetoo() {
    System.out.println("Це не абстрактний метод");
  }
}

class B extends A {
  void callme() {
    System.out.println("Реалізація абстрактного методу callme в класі B");
  }
}

class AbstractDemo {
  public static void main(String args[]) {
    B b = new B();

    b.callme();
```

```
    b.callmetoo();
  }
}
```

У наведеній програмі об’єкти класу А не оголошуються, тому що це абстрактний клас. У абстрактному класі можуть реалізовуватись конкретні методи, наприклад, у класі А реалізується метод callmetoo( ). До абстрактного класу може бути додано реалізації будь-якої кількості конкретних методів. Незважаючи на те, що абстрактні класи не дозволяють утворювати об’єкти їх типу, але такі класи можна використовувати для створення посилань на об’єкти. У наведеному нижче прикладі показано, як створювати посилання на абстрактний клас Figure, після чого робиться та використовуються посилання на конкретний об’єкт підкласу.

Приклад 4

```
// Абстрактний клас з абстрактним методом розрахунку площі
abstract class Figure {
  double dim1;
  double dim2;

  Figure(double a, double b) {
    dim1 = a;
    dim2 = b;
  }

  // Це абстрактний метод
  abstract double area();
}

class Rectangle extends Figure {
  Rectangle(double a, double b) {
    super(a, b);
  }

  // Реалізація абстрактного методу для прямокутника
  double area() {
    System.out.println("Прямокутник");
    return dim1 * dim2;
  }
}

class Triangle extends Figure {
  Triangle(double a, double b) {
    super(a, b);
```

```
  }

  // Реалізація абстрактного методу для прямокутного трикутника
  double area() {
    System.out.println("Трикутник");
    return dim1 * dim2 / 2;
  }
}

class DemoAreas {
  public static void main(String args[]) {
  // Figure f = new Figure(2, 2); // створити об’єкт типу Figure неможливо
    Rectangle r = new Rectangle(9, 5);
    Triangle t = new Triangle(10, 8);

    Figure figref; // допустиме оголошення, об’єкт не створюється

    figref = r;
    System.out.println("Площа фігури = " + figref.area());

    figref = t;
    System.out.println("Площа фігури = " + figref.area());
  }
}
```

Завдання до роботи

1. Вивчить та відтворіть приклади 1-4 теоретичної частини лабораторної роботи. Визначить результати роботи прикладів, зробіть їх скриншот та надайте пояснення.
2. В одній із задач прикладної механіки необхідно використовувати значення визначених інтегралів від *a* до *b* для різних функцій *f(x)*. Припускається, що множина функцій, що використовується, буде розширюватись за необхідністю. Розробіть абстрактний клас FunctionIntegral з трьома абстрактними методами чисельного інтегрування (наприклад, методом прямокутників, трапецій та методом Симпсона), для яких *a* та *b* використовуються як аргументи. Для функцій свого варіанту розробіть 3 підкласи класу FunctionIntegral, які реалізують його абстрактні методи.
4. Створіть клас, у якому для розроблених класів демонструється принцип динамічної диспетчеризації методів за допомогою змінної посилання на суперклас FunctionIntegral. Отримайте результати роботи прикладів, зробіть їх скриншоти та надайте пояснення.
5. Підготуйте звіт.

Контрольні запитання

1. Дайте визначення поліморфізму.
2. Чим відрізняються перевизначення та перезавантаження методів?
3. У чому полягає механізм динамічної диспетчеризації методів?
4. Чи можна використати динамічну диспетчеризацію для абстрактних методів? Що для чого потрібно?
5. Чому механізм динамічної диспетчеризації методів є можливим?
6. Наведіть приклад використання оператору switch-case для реалізації розгалуження під час використання динамічної диспетчеризації методів.