Реферат
з теми “Узагальнення в Java”

Кузьмук Артур, 3 курс группа комп. мат. 2

Основні поняття.

Починаючи з самого появи мова Java зазнав масу змін, які, безсумнівно, привнесли позитивні моменти в його функціональність. Одним з таких важливих змін є введення Java Generic або узагальнення. Дана функціональність зробила мова не тільки гнучкіше і універсальніше, але і набагато безпечніше в плані приведення типів даних.

По суті, узагальнення - це параметризовані типи. Такі типи важливі, оскільки вони дозволяють оголошувати класи, інтерфейси і методи, де тип даних, якими вони оперують, зазначений у вигляді параметра.
Використовуючи узагальнення. можна, наприклад, створити єдиний клас, який буде автоматично звертатися з різнотипними даними.

Класи, інтерфейси або методи, які оперують параметризовані типами, називаються узагальненими.

Отже, в старому коді посилання типу Object використовувалися в узагальнених класах, інтерфейси або методах з метою оперувати різнотипними об'єктами. Але справа в тому, що вони не могли забезпечити типову безпеку.

Саме узагальнення внесли в мову типову безпеку типів. якої так бракувало раніше. Вони також спростили процес виконання, оскільки тепер немає потреби в явному приведення типів для перетворення об'єктів типу Object в конкретні типи оброблюваних даних.

Завдяки узагальнень всі операції приведення типів виконуються автоматично і неявно. Таким чином, узагальнення розширили можливості повторного використання коду, дозволивши робити це легко і безпечно.

Приклади.

Розглянемо проблему, в якій можуть знадобитися узагальнення.
Припустимо, ми визначаємо клас для представлення банківського рахунку. Наприклад, він міг би мати такий вигляд:

```
class Account{

    private int id;
    private int sum;

    Account(int id, int sum){
        this.id = id;
        this.sum = sum;
    }

    public int getId() { return id; }
    public int getSum() { return sum; }
    public void setSum(int sum) { this.sum = sum; }
}
```

Клас Account має два поля: id - унікальний ідентифікатор рахунку і sum - сума на рахунку.

У цьому разі ідентифікатор задано як цілочисельне значення, наприклад, 1, 2, 3, 4 і так далі. Однак також нерідко для ідентифікатора використовують і рядкові значення. І числові, і рядкові значення мають свої плюси і мінуси. І на момент написання класу ми можемо точно не знати, що краще вибрати для зберігання ідентифікатора - рядки або числа. Або, можливо, цей клас використовуватимуть інші розробники, які можуть мати свою думку щодо цієї проблеми. Наприклад, як тип id вони захочуть використовувати якийсь свій клас.

І на перший погляд ми можемо розв'язати цю проблему таким чином: задати id як поле типу Object, який є універсальним і базовим суперкласом для всіх інших типів:

```
public class Program{

    public static void main(String[] args) {

        Account acc1 = new Account(2334, 5000); // id - число
        int acc1Id = (int)acc1.getId();
        System.out.println(acc1Id);

        Account acc2 = new Account("sid5523", 5000);    // id - строка
        System.out.println(acc2.getId());
    }
}
class Account{
```

```
    private Object id;
    private int sum;

    Account(Object id, int sum){
        this.id = id;
        this.sum = sum;
    }

    public Object getId() { return id; }
    public int getSum() { return sum; }
    public void setSum(int sum) { this.sum = sum; }
}
```

У цьому випадку все чудово працює. Однак тоді ми стикаємося з проблемою безпеки типів. Наприклад, у такому випадку ми отримаємо помилку:

```
Account acc1 = new Account("2345", 5000);
int acc1Id = (int)acc1.getId(); // java.lang.ClassCastException
System.out.println(acc1Id);
```

Проблема може здатися штучною, оскільки в даному випадку ми бачимо, що в конструктор передається рядок, тому ми навряд чи будемо намагатися перетворювати його до типу int. Однак у процесі розробки ми можемо не знати, який саме тип представляє значення в id, і під час спроби отримати число в цьому разі ми зіткнемося з виключенням java.lang.ClassCastException.

Писати для кожного окремого типу свою версію класу Account теж не є гарним рішенням, оскільки в цьому випадку ми змушені повторюватися.

Ці проблеми були покликані усунути узагальнення або generics. Узагальнення дозволяють не вказувати конкретний тип, який буде використовуватися. Тому визначимо клас Account як узагальнений:

```
class Account<T>{

    private T id;
    private int sum;

    Account(T id, int sum){
        this.id = id;
        this.sum = sum;
    }

    public T getId() { return id; }
```

```
    public int getSum() { return sum; }
    public void setSum(int sum) { this.sum = sum; }
}
```

За допомогою літери T у визначенні класу class Account<T> ми вказуємо, що даний тип T буде використовуватися цим класом. Параметр T у кутових дужках називається універсальним параметром, оскільки замість нього можна підставити будь-який тип. При цьому поки ми не знаємо, який саме це буде тип: String, int або якийсь інший. Причому буква T обрана умовно, це може і будь-яка інша буква або набір символів.

Після оголошення класу ми можемо застосувати універсальний параметр T: так далі в класі оголошується змінна цього типу, якій потім присвоюється значення в конструкторі.

Метод getId() повертає значення змінної id, але оскільки ця змінна представляє тип T, то цей метод також повертає об'єкт типу T: public T getId().

Використовуємо цей клас:

```
public class Program{

    public static void main(String[] args) {

        Account<String> acc1 = new Account<String>("2345", 5000);
        String acc1Id = acc1.getId();
        System.out.println(acc1Id);

        Account<Integer> acc2 = new Account<Integer>(2345, 5000);
        Integer acc2Id = acc2.getId();
        System.out.println(acc2Id);
    }
}
class Account<T>{

    private T id;
    private int sum;

    Account(T id, int sum){
        this.id = id;
        this.sum = sum;
    }

    public T getId() { return id; }
    public int getSum() { return sum; }
    public void setSum(int sum) { this.sum = sum; }
}
```

При визначенні змінної даного класу і створенні об'єкта після імені класу в кутових дужках потрібно вказати, який саме тип буде використовуватися замість універсального параметра. При цьому треба враховувати, що вони працюють тільки з об'єктами, але не працюють із примітивними типами. Тобто ми можемо написати Account<Integer>, але не можемо використовувати тип int або double, наприклад, Account<int>. Замість примітивних типів треба використовувати класи-обгортки: Integer замість int, Double замість double тощо.

Наприклад, перший об'єкт використовуватиме тип String, тобто замість T підставлятиметься String:

```
Account<String> acc1 = new Account<String>("2345", 5000);
```

У цьому випадку як перший параметр у конструктор передається рядок.

А другий об'єкт використовує тип int (Integer):

```
Account<Integer> acc2 = new Account<Integer>(2345, 5000);
```

Деякі особливості узагальнень.

Компілятор Java не створює жодних різних версій класу Pair. Насправді в процесі компіляції вся інформація про узагальнений тип видаляється. Замість цього виконується приведення відповідних типів, створюючи спеціальну версію класу Pair. Однак у самій програмі, як і раніше, існує єдина узагальнена версія даного класу. Цей процес називається в Java Generic очищення типу.
Відзначимо важливий момент. Посилання на різні версії одного і того ж java generic класу не можуть вказувати на один і той же об'єкт. Тобто, припустимо, у нас є два посилання: Pair obj1 і Pair obj2. Отже, у рядку obj1 = obj2 виникне помилка. Хоча обидві змінні відносяться до типу Pair об'єкти , на які вони посилаються, різні. Це яскравий приклад забезпечення безпеки типів Java Generic.

Обмеження, що накладаються на узагальнені класи.

Важливо знати, що узагальнення можуть застосовуватися тільки до посилальних типів, тобто передається параметру generic class java аргумент обов'язково повинен бути типом класу. Такі прості типи, як, наприклад, або long double, передавати не можна. Іншими словами, наступна рядок оголошення класу Pair неприпустима: Pair obj. Тим не менше дане обмеження не становить серйозної проблеми, так як в Java для кожного примітивного типу є відповідний клас-оболонка. Строго кажучи, якщо в класі Pair ви хочете инкапсулировать ціле і логічне значення, автоупаковка зробить все за вас: Pair obj = new Pair <> (25 true).
Ще одним серйозним обмеженням є неможливість створення екземпляра параметра типу. Так, наступна рядок викличе помилку компіляції: T first = new T(). Це очевидно, оскільки ви наперед не знаєте, чи як аргумент передаватися повноцінний клас або абстрактний, або зовсім інтерфейс. Те ж саме стосується створення масивів.

Обмежені типи.

Досить часто виникають ситуації, коли необхідно обмежити перелік типів, які можна передавати в якості аргументу java generic класу. Припустимо, що в нашому класі Pair ми хочемо инкапсулировать виключно числові значення для подальших математичних операцій над ними. Для цього нам необхідно вказати верхню межу параметра типу. Реалізується це за допомогою оголошення суперкласу, що успадковується всіма аргументами, що передаються в кутових дужках. Це буде виглядати наступним чином: class Pair . Таким способом компілятор дізнається, що замість параметра T можна підставляти або клас Number або один з його підкласів. Це поширений прийом. Такі обмеження часто використовуються для забезпечення сумісності параметрів типу в одному і тому ж класі. Розглянемо приклад на нашому класі Pair: class Pair . Тут ми повідомляємо компілятору, що тип T може бути довільним, а тип V обов'язково повинен бути типом T, або одним з його підкласів. Обмеження «знизу» відбувається точно таким же чином, але замість слова extends пишеться слово super. Тобто оголошення class Pair говорить про те, що замість T може бути підставлений або ArrayList, або будь-який клас або користувача, які він наслідує.