Полиморфизм

Полиморфизм

Полиморфизм - возможность представления единого интерфейса для объектов разных типов или возможность использование единого идентификатора для представления разных типов.

Возможность представления единого интерфейса для объектов разных типов чаще всего означает возможность описания функции которая может работать с объектами разных типов или возможность вызова одной и той же функции у объектов разных типов.

Существует несколько типов полиморфизма:

- Ad-hoc полиморфизм
- Параметрический полиморфизм
- Полиморфизм подтипов

Ad-hoc полиморфизм

Если параметру функции сопоставлен ровно один тип, то такая функция называется мономорфной. Некоторые языки программирования (в том числе и Java) разрешают назначение нескольким мономорфным функциям единого идентификатора (имя функции). Например в Java это механизм перегрузки методов. В этом случае, в исходном коде становится возможным осуществлять вызов функции с фактическими параметрами разных типов, но в скомпилированном коде происходит вызов различных функций. Такая возможность называется ad-hoc полиморфизмом.

Ad-hoc полиморфизм (перегрузка методов)

```
public class Main {

	public static void main(String[] args) {

		SampleClassA sca = new SampleClassA();

		int result1 = sca.add(1, 2);

		String result2 = sca.add("Hello ", "world");

		System.out.println(result1);
		System.out.println(result2);

	}

}

class SampleClassA {

	public int add(int a, int b) {
		return a + b;
	}

	public String add(String a, String b) {
		return a + b;
	}
}
```

Вызов перегруженных методов

Перегруженные методы

Ad-hoc полиморфизм (перегрузка методов)

На самом деле при перегрузке методов для вызова используется обычная мономорфная функция характерная для данного типа параметра. В предыдущем примере для вызова с параметрами int вызывается один метод, а при вызове с параметрами String другой. Эти методы обладают одинаковым идентификатором (имя метода) но вариант какой именно метод будет вызван определяется на этапе компиляции.

Ad-hoc полиморфизм (приведение типов)

При описании функции можно указать в качестве типа формального параметра, такой тип к которому возможно автоматическое приведение других типов. В таком случае также возможен вызов этой функции с параметрами разных типов, при вызове сначала произойдет автоматическое приведение типов и только потом вызов функции. Однако такая функция все равно будет мономорфной, так как она работает с параметром одного типа.

Ad-hoc полиморфизм (приведение типов)

```
public class Main {

	public static void main(String[] args) {

		SampleClassB scb = new SampleClassB();

		int a1 = 3;
		int b1 = 4;
		double result1 = scb.add(a1, b1);

		double a2 = 3.5;
		double b2 = 4.5;
		double result2 = scb.add(a2, b2);

		System.out.println(result1);
		System.out.println(result2);

	}
}

class SampleClassB {

	public double add(double a, double b) {
		return a + b;
	}
}
```

Вызов метода с значениями типа int

Вызов метода с значениями типа double

Метод принимающий значения типа double

Параметрический полиморфизм

Функция параметр которой может быть представлен разными типами называется полиморфной функцией. Полиморфными также могут быть некоторые типы данных (например списки, работа списка целых чисел, по сути не отличается от работы списка строк).

Параметрический полиморфизм позволяет описать функцию (или тип данных) в обобщенном виде, что бы они могли обрабатывать данные разных типов одинаково, независимо от их типа. К реализации параметрического полиморфизма в Java можно отнести применение обобщений (generic)

Параметрический полиморфизм

```
public class Main {

     public static void main(String[] args) {

          Integer[] array1 = new Integer[] { 5, 3, 7, 2 };
          Integer result1 = getMaximum(array1);

          String[] array2 = new String[] { "Java", "Python", "Fortran", "C" };
          String result2 = getMaximum(array2);

          System.out.println(result1);
          System.out.println(result2);

     }

     public static <T extends Comparable<T>> T getMaximum(T[] array) {
          T max = array[0];
          for (int i = 0; i < array.length; i++) {
               if (array[i].compareTo(max) > 0) {
                    max = array[i];
               }
          }
          return max;
     }
}
```

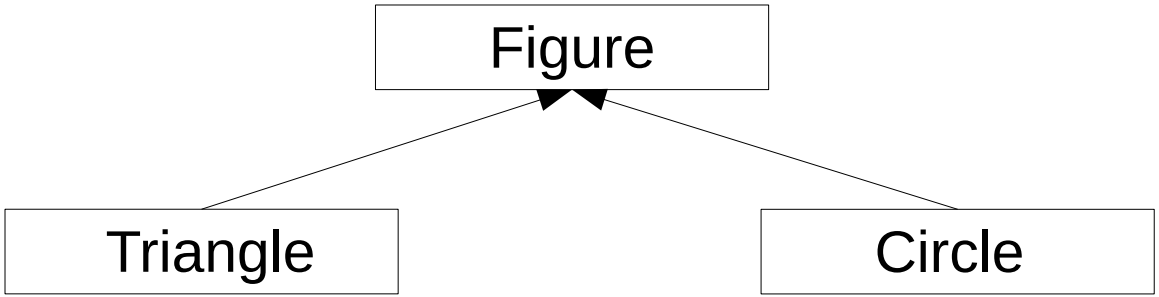
Пример метода использующего обобщения

Полиморфизм подтипов

Полиморфизм подтипов - механизм связывающий один тип (подтип) с другим типом (супертип) понятием заменяемости. Это означает что любые программные компоненты где может быть использован супертип, могут использовать и подтип. В Java полиморфизм подтипов опирается на механизм наследования.

В ООП под термином полиморфизм понимают именно полиморфизм подтипов.

Пример иерархии классов



Класс Figure будет абстрактным, а классы Triangle, Circle будут подклассами Figure.

Супертип

```
public abstract class Figure {

    public abstract double getArea();

    public abstract double getPerimeter();

    @Override
    public String toString() {
        return "Figure []";
    }

}
```

Класс Triangle как подкласс Figure

```
public class Triangle extends Figure {
	private double sideA;
	private double sideB;
	private double sideC;

	public Triangle(double sideA, double sideB, double sideC) {
		this.sideA = sideA;
		this.sideB = sideB;
		this.sideC = sideC;
	}

	public Triangle() {
		super();
	}

	public double getSideA() {
		return sideA;
	}
	public void setSideA(double sideA) {
		this.sideA = sideA;
	}
	public double getSideB() {
		return sideB;
	}
	public void setSideB(double sideB) {
		this.sideB = sideB;
	}
	public double getSideC() {
		return sideC;
	}
	public void setSideC(double sideC) {
		this.sideC = sideC;
	}

	@Override
	public double getArea() {
		double halfPerimeter = getPerimeter() / 2.0;
		return Math.sqrt(halfPerimeter * (halfPerimeter - sideA) * (halfPerimeter - sideB) * (halfPerimeter - sideC));
	}

	@Override
	public double getPerimeter() {
		return sideA + sideB + sideC;
	}

	@Override
	public String toString() {
		return "Triangle [sideA=" + sideA + ", sideB=" + sideB + ", sideC=" + sideC + "]";
	}
}
```

Полиморфизм подтипов

```
public class GameBoard {
	private Figure[] figures = new Figure[4];

	public void addFigure(Figure fig) {
		for (int i = 0; i < figures.length; i++) {
			if (figures[i] == null) {
				figures[i] = fig;
				break;
			}
		}
	}

	public double getTotalArea() {
		double area = 0;
		for (Figure figure : figures) {
			if (figure != null) {
				area += figure.getArea();
			}
		}
		return area;
	}

	@Override
	public String toString() {
		String result = "GameBoard [figures=";
		for (Figure figure : figures) {
			if (figure != null) {
				result += ", " + figure.toString();
			}
		}
		result = result.substring(0, result.length() - 2);
		return result + "]";
	}

}
```

Метод принимающий параметр супертипа

Вызов метода характерного для супертипа

Полиморфизм подтипов

```
public class Main {

     public static void main(String[] args) {

          Triangle triangle = new Triangle(3, 4, 5);

          GameBoard gb = new GameBoard();

          gb.addFigure(triangle);

          System.out.println(gb);

     }
}
```

Использование подтипа в методе

И хотя метод добавления фигуры на доску использует ссылку типа Figure при вызове можно использовать ссылку тип которой определяется любым подклассом класса Figure.

Статический и динамический полиморфизм

Полиморфизм можно также разделить на виды в зависимости от времени выбора реализации функции.

Статический полиморфизм - выбор реализации производиться на этапе компиляции (раннее связывание) ad-hoc полиморфизм и параметрический полиморфизм относятся именно к статическому полиморфизму.

Динамический полиморфизм - выбор реализации производится на этапе выполнения (позднее связывание). Полиморфизм подтипов относится к динамическому полиморфизму.

Список литературы

1) Герберт Шилдт Java 8. Полное руководство 9-е издание ISBN 978-5-8459-1918-2
2) https://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html
3) https://docs.oracle.com/javase/tutorial/java/IandI/override.html