**Stream в Java**

Потокът от данни предоставя интерфейс към поредица от елементи от определен тип, но за разлика от нея не се фокусира върху запазването и достъпа им от паметта, а върху изчисленията, които трябва се извършат с тях. Най-често данните биват "подавани" от колекция (масив) или друго място. Поддържат се различни операции като filter, reduce, sort и други.

Пример: private static Employee[] arrayOfEmps = {

new Employee(1, "Jeff Bezos", 100000.0),

new Employee(2, "Bill Gates", 200000.0),

new Employee(3, "Mark Zuckerberg", 300000.0)

};

Stream.of(arrayOfEmps);

**Функционален интерфейс в Java**

Функционален интерфейс е този, в който има точно един абстрактен метод. За да може да се прилага ламбда синтаксисът, типът на аргумента, на който се подава ламбда функцията, трябва да бъде "функционален интерфейс".

В java.util.List има метод sort с единствен аргумент от тип Comparator, който е функционален интерфейс:

public interface Comparator <T > {

int compare ( T o1 , T o2 );

}

List <Person> people = Arrays.asList (

new Person ("Ivan", Gender.MALE, 22),

new Person ("Ivanka", Gender.FEMALE, 34),

new Person ("Peter", Gender.FEMALE, 11));

people.sort((p1, p2) -> p1.getAge()-p2.getAge());

**Наследяване (inheritance) в Java**

1. Интерфейс може да наследява само интерфейс!
2. Абстрактен клас може да наследява абстрактен клас и интерфейс!
3. Обикновен клас може да наследява друг обикновен клас, абстрактен клас и интерфейс!

Най-просто казано наследяването е механизъм, чрез който на един клас е позволено да наследява характеристиките (полета и методи) на друг клас.

**Наследяването в Java има своите особености, които са:**

1. Eдин клас може да наследи не повече от един клас (за разлика от C++, който поддържа множествено наследяване).
2. Наследяващият клас разполага с всички променливи и методи на класа родител (обратния вариант не е валиден).
3. Родителския клас не трябва да е final и не трябва да е с модификатор за достъп private.
4. Наследяващият клас задължително трябва да съдържа нови полета или нови методи.

Наследяването става с ключовата дума extends, след което се посочва кой е наследяваният клас.

**File Writing/Reading в Java**

**Характеристики на FileWriter:**

1. Този клас се наследява от класа OutputStream.
2. Конструкторите на този клас приемат, че кодирането на символи по подразбиране и размера на байт-буфера по подразбиране са приемливи.
3. FileWriter е предназначен за писане на потоци от знаци.
4. FileWriter създава изходния файл, ако той вече не присъства.

**Характеристики на FileReader:**

1. Този клас е наследен от класа InputStreamReader.
2. Конструкторите на този клас приемат, че кодирането на символи по подразбиране и размера на байт-буфера по подразбиране са подходящи.
3. FileReader е предназначен за четене на потоци от знаци.

**Generic в Java**

Генеричен клас просто означава, че елементите или функциите в този клас могат да бъдат обобщени с параметъра (пример T), за да посочим, че можем да добавим произволен тип като параметър на мястото на T като Integer, Character, String, Double или всеки друг потребител - дефиниран тип.

**Интерфейси в Java**

1. Методите в интерфейса са винаги само абстрактни.
2. Интерфейса поддържа множествено наследяване.
3. Интрефейса може да дефинира данни, но те са задължитено константи - "static" и "final".
4. Интерфейса може да наследява (extends) един или повече интерфейси.
5. Методите в интерфейса са задължително public (независимо дали са декларирани или не като такива).

**Абстрактни класове в Java**

1. Абстрактните класове може да съдържат абстрактни и не-абстрактни методи.
2. Абстрактните класове не поддържат множествено наследяване.
3. Абстрактните класове може да съдържа final, non-final, static, non-static променливи.
4. Абстрактните класове може да имплементират множество интерфейси
5. Абстрактните класове може да имат методи с модификатор за достъп public, private, protected.

**Видове цикли в Java**

1. **While** цикъл - при while цикъла първо се изпълнява булевия израз (условието), ако резултатът от него е true се изпълнява и последователността от операции. Това се повтаря докато условния израз не върне false. Има следната структура:

**while (условие) {**

**тяло;**

**}**

1. **Do-while** цикълът е аналогичен на while цикъла, само че при него проверката на булевия израз се прави след изпълнението на операциите в цикъла. Изглежда по следния начин:

**do {**

**тяло;**

**} while (условие);**

1. **For цикъл** - използва се тогава, когато предварително е ясен броят на итерациите, които трябва да се извършват, чрез предварително задаване на условие (брояч), което ще се изпълни след определения брой превъртания на цикъла. Може да го представим чрез следния пример:

**for(int i = 0; i < 5; i++) {**

**тяло;**

**}**

**Полиморфизъм в Java**

Полиморфизмът е една от характеристиките на ООП, която ни позволява да извършваме едно действие по различни начини. Например, да кажем, че имаме родителки клас Test, който има метод sound().

public class Test{

public void sound(){

System.out.println("....");

}

}

Нека създадем нов клас Test2, който ще наследява класа Test:

public class Test2 extends Test{

@Override

public void sound(){

System.out.println("////");

}

}

В случая, ако наследяващия клас има същия метод, който e деклариран в родителския клас, той е известен като отмяна на метода (Override). По този начин можем да използваме метода, колкото пъти поискаме, като съответно спазим йерархията на наследяването.