Metode avansate de programare

Informatică Româna, 2017-2018

Curs 5 - Java 8 features – Partea I

Interfețe cu o singură metodă abstractă

- **SAM Interfaces**
  - `java.lang.Runnable, void run();`
  - `java.awt.event.ActionListener, void actionPerformed(ActionEvent e);`
  - `java.util.Comparator, int compare(T o1, T o2);`
  - `java.util.concurrent.Callable, V call() throws Exception`
  - ….
- **Ce au în comun toate aceste interfețe?**
  - Declară o singură metodă abstractă (de obicei, cu numele unor verbe precum: *run*, *execute*, *perform*, *apply*, *compare*, …..)
  - *Interfețele au și metode care nu sunt abstracte?*

Metode default sau statice în interfețe

```
interface Formula{
    double pi=3.14;
    double  calculate(double a, double b);

    default double sqrt(double a) {return Math.sqrt(a);}

    default double power(double a, double  b) {return Math.pow(a, b);}

    default double numarLaPatrat(double nr){return power(nr,2);}

    default double numarLaCub(double nr){return power(nr,3);}

    default double patratBinom(double x, double y){ return Math.pow(x+y,2); }

    static double cubBinom(double x, double y){ return Math.pow(x+y,3); }

    static double suma(double x, double y) {return x+y;}
}
```

Java 8 permite interfețelor adaugarea metodelor care nu sunt abstracte (default sau statice) precum și a constantelor!!!

Interfețe în contextul claselor interne anonime

```
Formula patratBinom=new Formula() {
    @Override
    public double calculate(double a, double b) {
        return patratBinom(a,b);
    }
};

double a=2.1, b=2.2;
double res=patratBinom.calculate(a,b);
System.out.format("(%.2f+%.2f)^2=%.2f",a,b,res);
```

Codul este lipsit de concizie, prea complicat!!!!

Interfețe funcționale

- O interfaţă funcţională (functional interface) este orice interfaţă ce conţine doar o metodă abstractă.
- Astfel putem omite numele metodei atunci când implementăm interfaţa şi putem elimina folosirea claselor anonime. În locul lor vom avea **lambda expresii** sau **referințe la metode**
- O interfaţă funcţională este adnotată cu **`@FunctionalInterface`**

```
@FunctionalInterface
interface Formula {
        double calculate(double a, double b);
        // others default or static methods
}
```

Utilizarea interfețelor funcționale

```
Formula f=referintaLaOMetoda sau oExpresie
```

Este posibil datorită faptului că avem o singură metodă abstractă in interfața Formula.

Referință la o metodă de clasă

```
class FormulaHelper{

    public static double patratBinom(double x, double y){
        return Math.pow(x+y,2);
    }
}
```

```
Formula bin2=FormulaHelper::patratBinom;  //method reference - static method patratBinom
double a=2.1, b=2.2;
System.out.format("(%.2f+%.2f)^2=%.2f",a,b,bin2.calculate(a,b));
```

Referință la o metodă de instanță

```
class FormulaHelper{
    private double a;
    private double b;

    public FormulaHelper(double a, double b) { this.a = a;this.b = b; }

    public static double patratBinom(double x, double y){
        return Math.pow(x+y,2);
    }

    public double distanta(double x, double y){
        return FormulaHelper.sqrt(Math.pow(x-a,2) +Math.pow(y-b,2));
    }

    public static double sqrt(double a) {
        return Math.sqrt(a);
    }
}
```

```
FormulaHelper helper=new FormulaHelper(a,b);
Formula dist=helper::distanta;
System.out.format("d(A(%.2f,%.2f),B(%.2f,%,2f))=%.2f",a,b,a,b,dist.calculate(a,b));
```

Referință la o metodă de instanță .....

- "Reference to an instance method of an arbitrary object of a particular type"

```
class Boeing implements Comparable<Boeing>{
    int height;

    public Boeing(int height) {
        this.height = height;
    }

    public int getHeight() {
        return height;
    }

    @Override
    public int compareTo(Boeing o) {
        return this.height-o.height;
    }
}
```

```
@FunctionalInterface
interface Flyable<T> {
    int canFly(T t); // the hight reached by T
}
```

```
Flyable<Boeing> f=Boeing::getHeight;
int n=f.canFly(new Boeing(23));
```

Referință la constructor

```
interface StudentFactory<S extends Student> {
    S create(int id, String nume, float media);
}
```

```
//referinte la constructori
StudentFactory<Student> studentFactory=Student::new;
studentFactory.create(1, "POp", 8.9f);
```

Funcții lambda

- O functie lambda (funcţie anonimă) este o funcţie definită şi apelată fără a fi legată de un identificator.

- Funcţiile lambda sunt o formă de funcţii ,,încuibate” (nested functions) în sensul că permit accesul la variabilele din domeniul funcţiei în care sunt conţinute.

Funcții Lambda. Exemplu

```
@FunctionalInterface
interface Formula {
    double calculate(int a, int b);
}
…

double a=2.1, b=2.2;
Formula f1=(x,y)->{ return FormulaHelper.patratBinom(a,b);};
System.out.format("(%.2f+%.2f)^2=%.2f",a,b,f1.calculate(a,b));

FormulaHelper helper=new FormulaHelper(a,b);
Formula f2=(x,y)->helper.distanta(a,b);
System.out.format("d(A(%.2f,%.2f),B(%.2f,%,2f))=%.2f",a,b,a,b,f1.calculate(a,b));
```

Lambda. Domenii de accesibilitate

- Expresiile lambda pot avea acces la:
  - Variabilele statice
  - Variabile de instanță
  - Parametrii metodelor
  - Variabilele locale

- Amintiti-vă cum era în cazul caselor anonime!!!!!!!!!!!!!!!

Accesarea variabilelor locale

```
public static void locVariable()
{
      int patrat=2;
      Formula patratulBinomuluiLambda1=(double a, double b)->{
          // patrat=5; eroare
          return Math.pow(a+b,patrat);
      };
      double res1=patratulBinomuluiLambda1.calculate(3.1,5);
      System.out.printf("(%d + %d)^2=%.0f %n",3,5,res1);
}
```

Putem referi variabile locale în funcția lambda, dar acestea sunt implicit **final** (nu le putem modifica).

Accesarea membrilor de clasa și de instanță

```
class FormuleMatematice{
    private static int outerStaticPutere=1;
    private int outerPutere=1;
    public double PatratBinom(double x, double y){
        Formula f=(a,b)->{ outerPutere=2; return Math.pow(a+b,outerPutere);};
        return f.calculate(x,y);
    }
    public double CubBinom(double x, double y){
        return f.calculate(x,y);
    }
    Formula f=(a,b)->{ outerStaticPutere=3; return Math.pow(a+b,outerStaticPutere);};
}
```

În contrast cu variabilele locale, variabilele de clasă și cele de instanță pot fi accesate și modificate în funcții lambda.

Accesarea metodelor default în funcții lambda

```
interface Formula {
    double pi=3.14;
    double calculate(double a, double b);

    default double numarLaPatrat(double nr)
    {
        return power(nr,2);
    }
}
```

```
Formula patratBinomLambda2=(x,y)->numarLaPatrat(x+y); // eroare
```

În funcții lambda nu putem accesa metode default din interfață!

Built-in Functional Interfaces

- Predicates
- Functions
- Suppliers
- Consumers
- Comparators

- …

Consumer

https://docs.oracle.com/javase/8/docs/api/java/util/function/Consumer.html

- Operatii effectuate pe un singur argument.
- Verbul sugestiv pentru metoda abstractă accept

Method Summary

All Methods	Instance Methods	Abstract Methods	Default Methods

Modifier and Type	Method and Description
void	**accept**(T t) Performs this operation on the given argument.
default Consumer<T>	**andThen**(Consumer<? super T> after) Returns a composed Consumer that performs, in sequence, this operation followed by the after operation.

```
Consumer<Student> consumer=System.out::println; //method reference
consumer.accept(new Student(123,"Dan",4.5f));


Consumer<Student> consumer2=x-> System.out.println(x); //lambda
consumer.accept(new Student(123,"Dan",4.5f));

Consumer<Student> consumer3=Student::toString; //method reference
consumer.accept(new Student(123,"Dan",4.5f));
```

Metode adiționale colecțiilor - forEach, removeIf

```
List<Student> list= new ArrayList(Arrays.asList(
        new Student(22,"Aprogramatoarei",5.6f),
        new Student(23,"Popescu",9.6f),
        new Student(24,"Birlanescu",8.6f)));

list.forEach(x-> System.out.println(x));
list.forEach(System.out::println);
```

```
Predicate<Student> estePromovat=x->x.getMedia()>=5;  //lambda function

list.forEach(x-> {if (estePromovat.test(x)) System.out.println(x);} );

list.removeIf(estePromovat.negate());

list.forEach(System.out::println);
```

Predicate

https://docs.oracle.com/javase/8/docs/api/java/util/function/Predicate.html

- Predicatele sunt funcții de un singur argument care întorc o valoare logică
- Verbul sugestiv pentru metoda abstractă: test

Modifier and Type	Method and Description
default Predicate<T>	and(Predicate<? super T> other) Returns a composed predicate that represents a short-circuiting logical AND of this predicate and another.
static <T> Predicate<T>	isEqual(Object targetRef) Returns a predicate that tests if two arguments are equal according to Objects.equals(Object, Object).
default Predicate<T>	negate() Returns a predicate that represents the logical negation of this predicate.
default Predicate<T>	or(Predicate<? super T> other) Returns a composed predicate that represents a short-circuiting logical OR of this predicate and another.
boolean	test(T t) Evaluates this predicate on the given argument.

```
Predicate<Student> estePromovat=x->x.getMedia()>=5;  //lambda function

Predicate<Student> estePromovat2=StudentHelper::promovat; //method reference
System.out.println(estePromovat.test(new Student(24,"Birlanescu",4.6f)));
```

Predicate – default, static methods

https://docs.oracle.com/javase/8/docs/api/java/util/function/Predicate.html

Modifier and Type	Method and Description
default Predicate<T>	and(Predicate<? super T> other) Returns a composed predicate that represents a short-circuiting logical AND of this predicate and another.
static <T> Predicate<T>	isEqual(Object targetRef) Returns a predicate that tests if two arguments are equal according to Objects.equals(Object, Object).
default Predicate<T>	negate() Returns a predicate that represents the logical negation of this predicate.
default Predicate<T>	or(Predicate<? super T> other) Returns a composed predicate that represents a short-circuiting logical OR of this predicate and another.
boolean	test(T t) Evaluates this predicate on the given argument.

```
Predicate<Student> promovatSiIncepeCuA=estePromovat.and(x->x.getNume().startsWith("A"));
System.out.println(estePromovat.test(new Student(24,"Birlanescu",4.6f))); //false
```

Predicate. Alte Exemple

```
Student s=new Student(3,"Ana",5.6f);
Predicate<Student> nonNull = Objects::nonNull;
System.out.println(nonNull.test(s));  //true


Predicate<Student> isNull = Objects::isNull;
System.out.println(isNull.test(s));  //false
```

Exerciții - final Curs 5

- Să se șteargă dintr-o listă de șiruri de caractere, toate elementele care încep cu “a”.
- Să se șteargă dintr-o listă de șiruri de caractere, toate elementele care sunt prefixele unui cuvânt. De exemplu “Anamaria”. Folositi funție lambda si referință la metodă.
- Să se șteargă dintr-o listă de șiruri de caractere, toate elementele care conțin sirul vid. Folositi funție lambda si referință la metodă.
- Să se șteargă toți studenții a căror nume începe cu “B”, dintr-o listă de studenți.