


# Przykłady wyrażeń Lambda

`() -> 223.76`



```
double metoda() {  
    return 223.76;  
}
```

---

```
() -> Math.random() * 100
```

---

```
(n) -> (n % 2) == 0
```

# Interfejs funkcyjny

```
interface MyInterface{  
    String myMethod();  
}
```

Wyrażenie lambda



Interfejs funkcyjny

# Przykład



MyInterface.java

```
interface MyInterface{  
    String myMethod();  
}
```



Main.java

```
public class Main {  
  
    public static void main(String[] args) {  
        MyInterface myIn;  
        myIn = () -> "Hello World";  
        System.out.println(myIn.myMethod());  
    }  
}
```



StringProcessing.java

# Przykład

```
public interface StringProcessing {  
    String process(String s);  
}
```



Main.java

```
public class Main {  
    public static void main(String[] args) {  
        String s="Java is Easy";  
        StringProcessing sp=(str)->{  
            return str.toUpperCase();  
        };  
        System.out.println(stringOperation(sp, s));  
  
        System.out.println(stringOperation((str)->str.toLowerCase(), s));  
    }  
  
    static String stringOperation(StringProcessing sp, String s){  
        return sp.process(s);  
    }  
} //Main
```

# Referencje do metod



Main.java

...

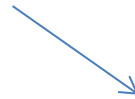
System.out.println(stringOperation(**Main::reverse**, s));

...

```
static String reverse(String s){  
    String result = "";  
    int i;  
    for(i = s.length()-1; i >= 0; i--)  
        result += s.charAt(i);  
    return result;  
}
```

Klasa

Metody w klasie





Main.java

# Referencje do metod

...

```
Processor proc=new Processor();  
proc.str= "!!!";  
System.out.println(stringOperation(proc::removeSpaces, s));
```



Processor.java

...

```
public class Processor {  
    String str = "";  
  
    String removeSpaces(String s){  
        String result = "";  
        int i;  
        for(i = 0; i < s.length(); i++)  
            if(s.charAt(i) != ' ')  
                result += s.charAt(i);  
        return result +str;  
    }  
}
```

Referencja do obiektu

Metoda zaimplementowana  
w obiekcie



Main.java

# Interfejs generyczny/Dostęp do pól

```
public class Main {  
    public static void main(String[] args) {  
  
        String str="Hello";  
        Double cena=2.95;  
  
        MyFunc <String> string=(s)->{return str+s;};  
        System.out.println(string.function("!"));  
  
        MyFunc <Double> double=(c)->{return cena+c;};  
        System.out.println(double.function(3.45)+"zł");  
  
    }  
} //Main
```



MyFunc.java

```
public interface MyFunc <T> {  
    T function(T s);  
}
```



# Interfejs generyczny/metoda generyczna 1/2

```
public interface MyAveFunc <T> {  
    String average(T[] s);  
}
```

```
public class MyClass {  
    static <T> String calculate (T [] s){  
        double sum= 0;  
        for(T sx:s) sum=sum+ Double.parseDouble(String.valueOf(sx));  
        return String.valueOf(sum/s.length);  
    }  
}
```

# Interfejs generyczny/metoda generyczna 2/2

```
public class Main {  
    public static void main(String[] args) {  
  
        String stringTab []={"4","5","3"};  
        Double doubleTab []={new Double(2), new Double(5), new Double(4) };  
        Integer integerTab []={new Integer(5), new Integer(10), new Integer(4) };  
  
        MyAveFunc <String> ave=MyClass::<String>calculate;  
        System.out.println(ave.average(stringTab));  
  
        MyAveFunc <Double> aveDoub=MyClass::<Double>calculate;  
        System.out.println(aveDoub.average(doubleTab));  
  
        MyAveFunc <Integer> aveInt=MyClass::<Integer>calculate;  
        System.out.println(aveInt.average(integerTab));  
  
    }  
  
} //Main
```

# Odwołania do Konstruktorów

```
public interface MyFunction <R,T>{  
    R function(T t);  
}
```

---

```
public class MyData{  
    private String data;  
  
    MyData(String data) {  
        this.data = data;  
    }  
    public String getData(){  
        return this.data;  
    }  
}
```

---

```
public class Main {  
  
    public static void main(String[] args) {  
        MyFunction<MyData,String> myDataConst=MyData::new;  
        MyData md=myDataConst.function("Important message!");  
        System.out.println(md.getData());  
    }  
}
```

# Odwołania do Konstruktorów

Klasa parametryczna

```
public interface MyFunction <R,T>{  
    R function(T t);  
}
```

---

```
public class MyDataGen <T>{  
    private T data;
```

```
    MyDataGen(T data) {  
        this.data = data;  
    }
```

```
    public T getData() {  
        return this.data;  
    }
```

```
}
```

---

```
public class Main {
```

```
    public static void main(String[] args) {  
        MyFunction<MyDataGen<Double>, Double> myDataConst2=  
            MyDataGen<Double>::new;  
        MyDataGen <Double> mdg=myDataConst2.function(100.21);  
        System.out.println(mdg.getData());  
    }
```

```
}
```

# Gotowe interfejsy funkcjonalne **JAVA.UTIL.FUNCTION**

<b>Interfejs funkcjonalny</b>	<b>Typy parametrów</b>	<b>Typ zwracany</b>	<b>Nazwa metody abstrakcyjnej</b>	<b>Opis</b>
Runnable	brak	void	Run	Uruchamia działanie bez parametrów i wartości zwracanej
Supplier<T>	brak	T	Get	Dostarcza wartość typu T
Consumer<T>	T	void	Accept	Pobiera wartość typu T
BiConsume<T, U>	T, U	void	Accept	Pobiera wartości typu T i U
Function<T, R>	T	R	Apply	Funkcja z parametrem typu T
BiFunction<T, U, R>	T, U	R	Apply	Funkcja z parametrami typu T i U
UnaryOperator<T>	T	T	Apply	Operator jednoargumentowy dla typu T
BinaryOperator<T>	T, T	T	Apply	Operator dwuargumentowy dla typu T
Predicate<t>	T	boolean	Test	Funkcja zwracająca wartość logiczną
BiPredicate<T, U>	T, U	boolean	Test	Dwuargumentowa funkcja zwracająca wartość logiczną