Tomasz Fijałkowski

Java 8 in daily life

Today

We will find answers for the following questions:

make IT simple

Today

We will find answers for the following questions:

What is functional programing?

make IT simple

Today

We will find answers for the following questions:

What is functional programing?

When Java 8 helps as?

Today

We will find answers for the following questions:

What is functional programing?

When Java 8 helps as?

How to refactor to functional patterns?

Common example

```
Collections.sort(someList, new Comparator<String>() {
        @Override
        public int compare(String s1, String s2) {
            return reverse(s1).compareTo(reverse(s2));
        }
    });
```

make IT simple

Common example

```
Collections.sort(someList, new Comparator<String>() {
        @Override
        public int compare(String s1, String s2) {
            return reverse(s1).compareTo(reverse(s2));
        }
    });
```

Common example

```
Collections.sort(someList, new Comparator<String>() {
        @Override
        public int compare(String s1, String s2) {
            return reverse(s1).compareTo(reverse(s2));
        }
    });
```

```
Collections.sort(someList, (s1, s2) -> reverse(s1).compareTo(reverse(s2)));
```

Lambda syntax

Lambda syntax

```
Runnable action = () -> System.out.println("Hello, World!");
```

Lambda syntax

```
Runnable action = () -> System.out.println("Hello, World!");

Runnable otherAction = () -> {
        System.out.println("Ala ma kota,");
        System.out.println("a kot ma Alę");
    };
```

make IT simple

Lambda syntax

```
Runnable action = () -> System.out.println("Hello, World!");

Runnable otherAction = () -> {
        System.out.println("Ala ma kota,");
        System.out.println("a kot ma Alę");
    };

Predicate<BigDecimal> isPositive = num -> num.compareTo(BigDecimal.ZERO) > 0;
```

Lambda syntax

```
Runnable action = () -> System.out.println("Hello, World!");

Runnable otherAction = () -> {
        System.out.println("Ala ma kota,");
        System.out.println("a kot ma Alę");
    };

Predicate<BigDecimal> isPositive = num -> num.compareTo(BigDecimal.ZERO) > 0;

Comparator<String> reverseComperator =
            (s1, s2) -> reverse(s1).compareTo(reverse(s2));
```

make IT simple

Lambda syntax

```
Runnable action = () -> System.out.println("Hello, World!");

Runnable otherAction = () -> {
        System.out.println("Ala ma kota,");
        System.out.println("a kot ma Alę");
    };

Predicate<BigDecimal> isPositive = num -> num.compareTo(BigDecimal.ZERO) > 0;

Comparator<String> reverseComperator =
            (s1, s2) -> reverse(s1).compareTo(reverse(s2));

Comparator<String> reverseComperator =
            (String s1, String s2) -> reverse(s1).compareTo(reverse(s2));
```

make IT simple

@FunctionalInterface

Base functional Interfaces

	Arguments	Returns
Predicate<T>	T	boolean
Consumer<T>	T	void
Function<T, R>	T	R
Supplier<T>	None	T
UnaryOperator<T>	T	T
BinaryOperator<T>	(T, T)	T

make IT simple

Simplifications

make IT simple

```
private List<User> adults(List<User> users) {
        List<User> adults = new ArrayList<>();
        for (User user : users) {
            if (user.getAge() > 18) {
                adults.add(user);
            }
        }
        return adults;
    }
```

make IT simple

```java
private List<User> adults(List<User> users) {
    List<User> adults = new ArrayList<>();
    for (User user : users) {
        if (user.getAge() > 18) {
            adults.add(user);
        }
    }
    return adults;
}
```

stream refactoring

```java
private List<User> adults(List<User> users) {
    return users.stream()
            .filter(user -> user.getAge() > 18)
            .collect(Collectors.toList());
}
```

make IT simple

```
private List<User> adults(List<User> users) {
    return users.stream()
            .filter(user -> user.getAge() > 18)
            .collect(Collectors.toList());
}
```

make IT simple

```
private List<User> adults(List<User> users) {
    return users.stream()
            .filter(user -> user.getAge() > 18)
            .collect(Collectors.toList());
}
```

Method References refactoring

```
private List<User> adults(List<User> users) {
    return users.stream()
            .filter(this::isUserAdult)
            .collect(Collectors.toList());
}

private boolean isUserAdult(User user) {
        return user.getAge() > 18
}
```

```
private List<User> adults(List<User> users) {
    return users.stream()
            .filter(this::isUserAdult)
            .collect(Collectors.toList());
}
```

make IT simple

```
private List<User> adults(List<User> users) {
    return users.stream()
            .filter(this::isUserAdult)
            .collect(Collectors.toList());
}
```

parallel computations

```
private List<User> adults(List<User> users) {
    return users.stream()
            .parallel()
            .filter(this::isUserAdult)
            .collect(Collectors.toList());
}
```

Workshop time

make IT simple

Summary

make IT simple

Summary

Avoid state and mutable data

make IT simple

Summary

Avoid state and mutable data

Design with no side effects

Summary

Avoid state and mutable data

Design with no side effects

Use patterns

make IT simple

Summary

Avoid state and mutable data

Design with no side effects

Use patterns

Enjoy with lambda

Thank you

make IT simple