ZetCode

Filtering a list in Java

last modified July 8, 2023

In this article we show how to filter a list in Java.

This tutorial shows six different ways to filter a list. We use four different libraries: Apache Commons, Google Guava, Eclipse Collections, and Spring core.

In all six examples, we are going to filter a list of persons. A `Person` is a Java class with three attributes: `age`, `name`, and `sex`.

Advertisements

Filtering a list with Java for loop

In the first example, we use iteration to filter a list in Java.

com/zetcode/Person.java

```
package com.zetcode;

enum Gender {
    MALE, FEMALE
}

public class Person {

    private int age;
    private String name;
    private Gender sex;

    public Person(int age, String name, Gender sex) {

        this.age = age;
        this.name = name;
        this.sex = sex;
    }

    public int getAge() {

        return age;
    }

    public void setAge(int age) {

        this.age = age;
    }

    public String getName() {
```

```
        this.name = name;
    }

    public Gender getSex() {

        return sex;
    }

    public void setSex(Gender sex) {

        this.sex = sex;
    }

    @Override
    public String toString() {

        final StringBuilder sb = new StringBuilder("Person{");

        sb.append("age=").append(age);
        sb.append(", name='").append(name).append('\'');
        sb.append(", sex=").append(sex);
        sb.append('}');

        return sb.toString();
    }
}
```

We have this `Person` bean. We are going to filter a list having these beans. The `toString` method gives a string representation of the bean. This is going to be helpful when we print the filtered list of elements.

com/zetcode/FilterListEx.java

```
package com.zetcode;

import java.util.ArrayList;
import java.util.List;

public class FilterListEx {

    public static void main(String[] args) {

        var p1 = new Person(34, "Michael", Gender.MALE);
        var p2 = new Person(17, "Jane", Gender.FEMALE);
        var p3 = new Person(28, "John", Gender.MALE);
        var p4 = new Person(47, "Peter", Gender.MALE);
        var p5 = new Person(27, "Lucy", Gender.FEMALE);

        var persons = List.of(p1, p2, p3, p4, p5);

        var result = new ArrayList<Person>();

        for (Person person: persons) {

            if (person.getAge() > 30) {

                result.add(person);
            }
        }

        System.out.println(result);
    }
}
```

```
    if (person.getAge() > 30) {

        result.add(person);
    }
}
```

A for loop is used to go through the list of persons and create a new one having persons above thirty.

```
[Person{age=34, name=Michael, sex=MALE}, Person{age=47, name=Peter, sex=MALE}]
```

Filtering a list with Java 8 streams

In the next example, we use a Java 8 stream API to filter a list.

com/zetcode/FilterListEx2.java

```
package com.zetcode;

import java.util.List;
import java.util.function.Predicate;
import java.util.stream.Collectors;

public class FilterListEx2 {

    public static void main(String[] args) {

        var p1 = new Person(34, "Michael", Gender.MALE);
        var p2 = new Person(17, "Jane", Gender.FEMALE);
        var p3 = new Person(28, "John", Gender.MALE);
        var p4 = new Person(47, "Peter", Gender.MALE);
        var p5 = new Person(27, "Lucy", Gender.FEMALE);

        var persons = List.of(p1, p2, p3, p4, p5);

        Predicate<Person> byAge = person -> person.getAge() > 30;

        var result = persons.stream().filter(byAge)
                .collect(Collectors.toList());

        System.out.println(result);
    }
}
```

The Java stream API is used to filter data to contain only persons older than thirty.

```
Predicate<Person> byAge = person -> person.getAge() > 30;
```

This predicate returns elements with age greater than thirty.

```
var result = persons.stream().filter(byAge)
        .collect(Collectors.toList());
```

The `persons` list is filtered with the predicate and a new result list is produced.

Advertisements

In the next example, we filter data with the Apache `CollectionUtils`. It provides utility methods and decorators for `Collection` instances.

```
<dependency>
    <groupId>commons-collections</groupId>
    <artifactId>commons-collections</artifactId>
    <version>3.2.2</version>
</dependency>

<dependency>
    <groupId>commons-lang</groupId>
    <artifactId>commons-lang</artifactId>
    <version>2.6</version>
</dependency>
```

We use these Maven dependencies. The `commons-lang` is used for the `ToStringBuilder`, which is used in the `toString` method.

com/zetcode/Person.java

```
package com.zetcode;

import org.apache.commons.lang.builder.ToStringBuilder;

enum Gender {
    MALE, FEMALE
}

public class Person {

    private int age;
    private String name;
    private Gender sex;

    public Person(int age, String name, Gender sex) {

        this.age = age;
        this.name = name;
        this.sex = sex;
    }

    public int getAge() {

        return age;
    }

    public void setAge(int age) {

        this.age = age;
    }

    public String getName() {

        return name;
    }

    public void setName(String name) {

        this.name = name;
    }

    public Gender getSex() {
```

```
        this.sex = sex;
    }

    @Override
    public String toString() {

        return new ToStringBuilder(Person.class).
                append("Age", age).
                append("Name", name).
                append("Sex", sex).
                toString();
    }
}
```

The `Person` bean is improved with the `ToStringBuilder` inside the `toString` method.

com/zetcode/FilterListEx3.java

```
package com.zetcode;

import org.apache.commons.collections.CollectionUtils;

import java.util.ArrayList;
import java.util.List;

public class FilterListEx3 {

    public static void main(String[] args) {

        var p1 = new Person(34, "Michael", Gender.MALE);
        var p2 = new Person(17, "Jane", Gender.FEMALE);
        var p3 = new Person(28, "John", Gender.MALE);
        var p4 = new Person(47, "Peter", Gender.MALE);
        var p5 = new Person(27, "Lucy", Gender.FEMALE);

        var persons = List.of(p1, p2, p3, p4, p5);

        var result = new ArrayList<>(persons);

        CollectionUtils.filter(result, o -> ((Person) o).getAge() < 30);

        System.out.println(result);
    }
}
```

The example filters a list of person beans using the Apache `CollectionUtils` from the Apache Commons library.

```
var result = new ArrayList<>(persons);
```

A new copy of the list is created.

```
CollectionUtils.filter(result, o -> ((Person) o).getAge() < 30);
```

The `CollectionUtils.filter` filters the collection by applying a predicate to each element. If the predicate returns `false`, the element is removed.

Filtering a list with Google Guava

In the following example, we filter a list using Google Guava. *Google Guava* is an open-source set of common libraries for

```
    <version>19.0</version>
</dependency>
```

For the Guava library, we use this dependency.

com/zetcode/Person.java

```
package com.zetcode;

import com.google.common.base.MoreObjects;

enum Gender {
    MALE, FEMALE
}

public class Person {

    private int age;
    private String name;
    private Gender sex;

    public Person(int age, String name, Gender sex) {

        this.age = age;
        this.name = name;
        this.sex = sex;
    }

    public int getAge() {

        return age;
    }

    public void setAge(int age) {

        this.age = age;
    }

    public String getName() {

        return name;
    }

    public void setName(String name) {

        this.name = name;
    }

    public Gender getSex() {

        return sex;
    }

    public void setSex(Gender sex) {

        this.sex = sex;
    }

    @Override
    public String toString() {

        return MoreObjects.toStringHelper(Person.class)
                .add("Age", age)
                .add("Name", name)
                .add("Sex", sex)
```

The `MoreObjects.toStringHelper` is used to improve the `toString` method.

com/zetcode/FilterListEx4.java

```
package com.zetcode;

import com.google.common.base.Predicate;
import com.google.common.collect.FluentIterable;
import com.google.common.collect.Lists;

public class FilterListEx4 {

    public static void main(String[] args) {

        var persons = Lists.newArrayList(

                new Person(34, "Michael", Gender.MALE),
                new Person(17, "Jane", Gender.FEMALE),
                new Person(28, "John", Gender.MALE),
                new Person(47, "Peter", Gender.MALE),
                new Person(27, "Lucy", Gender.FEMALE)
        );

        Predicate<Person> byGender = person -> person.getSex() == Gender.MALE;

        var results = FluentIterable.from(persons)
                .filter(byGender)
                .toList();

        System.out.println(results);
    }
}
```

The code example filters a list to contain only males.

```
var persons = Lists.newArrayList(

        new Person(34, "Michael", Gender.MALE),
        new Person(17, "Jane", Gender.FEMALE),
        new Person(28, "John", Gender.MALE),
        new Person(47, "Peter", Gender.MALE),
        new Person(27, "Lucy", Gender.FEMALE)
);
```

We use Guava's `newArrayList` method to create a mutable list in one shot.

```
Predicate<Person> byGender = person -> person.getSex() == Gender.MALE;
```

This predicate returns `true` for males.

```
var results = FluentIterable.from(persons)
        .filter(byGender)
        .toList();
```

Using a `FluentIterable`, we filter the original list using the predicate and place it into a new list.

Advertisements

Filtering a list with Eclipse Collections

In the following example, we are going to filter a list with Eclipse Collections.

Eclipse Collections is a collections framework for Java. It has JDK-compatible `List`, `Set` and `Map` implementations with a rich API, additional types not found in the JDK like `Bags`, `Multimaps` and set of utility classes that work with any JDK compatible `Collections`, `Arrays`, `Maps` or `Strings`.

```
<dependency>
    <groupId>org.eclipse.collections</groupId>
    <artifactId>eclipse-collections-api</artifactId>
    <version>7.1.0</version>
</dependency>

<dependency>
    <groupId>org.eclipse.collections</groupId>
    <artifactId>eclipse-collections</artifactId>
    <version>7.1.0</version>
</dependency>
```

For the program, we use these two Mave dependencies.

com/zetcode/FilterListEx5.java

```
package com.zetcode;

import org.eclipse.collections.api.block.predicate.Predicate;
import org.eclipse.collections.impl.factory.Lists;
import org.eclipse.collections.impl.utility.Iterate;

import java.util.List;

public class FilterListEx5 {

    public static void main(String[] args) {

        var persons = Lists.immutable.of(

                new Person(34, "Michael", Gender.MALE),
                new Person(17, "Jane", Gender.FEMALE),
                new Person(28, "John", Gender.MALE),
                new Person(47, "Peter", Gender.MALE),
                new Person(27, "Lucy", Gender.FEMALE)
        );

        Predicate<Person> lessThan30 = (Predicate<Person>) person -> person.getAge() < 30;

        var result = (List<Person>) Iterate.select(persons, lessThan30);
        System.out.println(result);
    }
}
```

The code example creates a filtered list containing persons younger than thirty.

```
Predicate<Person> lessThan30 = (Predicate<Person>) person -> person.getAge() < 30;
```

A predicate is created to accept elements whose age is lower than thirty.

```
var result = (List<Person>) Iterate.select(persons, lessThan30);
```

In the next example, we are going to filter a list with Spring's `CollectionUtils`. It contains miscellaneous collection utility methods.

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>5.1.7.RELEASE</version>
</dependency>
```

The project contains a Maven dependency for the Spring Core JAR.

com/zetcode/FilterListEx6.java

```
package com.zetcode;

import org.springframework.cglib.core.CollectionUtils;

import java.util.ArrayList;
import java.util.Arrays;

public class FilterListEx6 {

    public static void main(String[] args) {

        var p1 = new Person(34, "Michael", Gender.MALE);
        var p2 = new Person(17, "Jane", Gender.FEMALE);
        var p3 = new Person(28, "John", Gender.MALE);
        var p4 = new Person(47, "Peter", Gender.MALE);
        var p5 = new Person(27, "Lucy", Gender.FEMALE);

        var persons = Arrays.asList(p1, p2, p3, p4, p5);

        var result = new ArrayList<>(persons);

        CollectionUtils.filter(result, p -> ((Person) p).getAge() > 30);

        System.out.println(result);
    }
}
```

The code example uses Spring's `CollectionUtils` to create a filtered list which contains persons older than thirty.

```
var result = new ArrayList<>(persons);
```

Similar to the Apache `CollectionUtils`, a copy of the original list is created. The example will modify the `result` list in place.

```
CollectionUtils.filter(result, p -> ((Person) p).getAge() > 30);
```

The `CollectionUtils.filter` method filters the `result` list with the given predicate.

```
[Person{age=34, name=Michael, sex=MALE}, Person{age=47, name=Peter, sex=MALE}]
```

In this article we have used six different ways to filter a list in Java.

Advertisements

Author

My name is Jan Bodnar and I am a passionate programmer with many years of programming experience. I have been writing programming articles since 2007. So far, I have written over 1400 articles and 8 e-books. I have over eight years of experience in teaching programming.

List all Java tutorials.

© 2007 - 2023 Jan Bodnar admin(at)zetcode.com