Generic Constructors in Java

Last modified: April 21, 2019

by baeldung (https://www.baeldung.com/author/baeldung/)

Java (https://www.baeldung.com/category/java/) +

Core Java (https://www.baeldung.com/tag/core-java/)

1. Overview

We previously discussed the basics of Java Generics (https://www.baeldung.com/java-generics). In this tutorial, we'll have a look at Generic Constructors in Java.

A generic constructor is a constructor that has at least one parameter of a generic type.

We'll see that generic constructors don't have to be in a generic class, and not all constructors in a generic class have to be generic.

2. Non-Generic Class

First, we have a simple class *Entry*, which is not a generic class:

```
public class Entry {
    private String data;
    private int rank;
}
```

In this class, we'll add two constructors: a basic constructor with two parameters, and a generic constructor.

2.1. Basic Constructor

The first *Entry* constructor is a simple constructor with two parameters:

```
public Entry(String data, int rank) {
    this.data = data;
    this.rank = rank;
}
```

Now, let's use this basic constructor to create an *Entry* object:

```
@Test
public void givenNonGenericConstructor_whenCreateNonGenericEntry_thenOk
    Entry entry = new Entry("sample", 1);

    assertEquals("sample", entry.getData());
    assertEquals(1, entry.getRank());
}
```

2.2. Generic Constructor

Next, our second constructor is a generic constructor:

```
public <E extends Rankable & Serializable> Entry(E element) {
    this.data = element.toString();
    this.rank = element.getRank();
}
```

Although the *Entry* class isn't generic, it has a generic constructor, as it has a parameter *element* of type *E*.

The generic type *E* is bounded and should implement both *Rankable* and *Serializable* interfaces.

Now, let's have a look at the *Rankable* interface, which has one method:

```
public interface Rankable {
    public int getRank();
}
```

And, suppose we have a class *Product* that implements the *Rankable* interface:

```
public class Product implements Rankable, Serializable {
    private String name;
    private double price;
    private int sales;

    public Product(String name, double price) {
        this.name = name;
        this.price = price;
    }

    @Override
    public int getRank() {
        return sales;
    }
}
```

We can then use the generic constructor to create *Entry* objects using a *Product*:

```
@Test
public void givenGenericConstructor_whenCreateNonGenericEntry_thenOK()
    Product product = new Product("milk", 2.5);
    product.setSales(30);

    Entry entry = new Entry(product);

    assertEquals(product.toString(), entry.getData());
    assertEquals(30, entry.getRank());
}
```

3. Generic Class

Next, we'll have a look at a generic class called *GenericEntry*:

```
public class GenericEntry<T> {
    private T data;
    private int rank;
}
```

We'll add the same two types of constructors as the previous section in this class as well.

3.1. Basic Constructor

First, let's write a simple, non-generic constructor for our *GenericEntry* class:

```
public GenericEntry(int rank) {
    this.rank = rank;
}
```

Even though *GenericEntry* is a generic class, this is a simple constructor that doesn't have a parameter of a generic type.

Now, we can use this constructor to create a *GenericEntry<String>*:

```
@Test
public void givenNonGenericConstructor_whenCreateGenericEntry_thenOK()
    GenericEntry<String> entry = new GenericEntry<String>(1);

    assertNull(entry.getData());
    assertEquals(1, entry.getRank());
}
```

3.2. Generic Constructor

Next, let's add the second constructor to our class:

```
public GenericEntry(T data, int rank) {
    this.data = data;
    this.rank = rank;
}
```

This is a generic constructor, as it has a *data* parameter of the generic type *T*. Note that we don't need to add *<T>* in the constructor declaration, as it's implicitly there.

Now, let's test our generic constructor:

```
@Test
public void givenGenericConstructor_whenCreateGenericEntry_thenOK() {
    GenericEntry<String> entry = new GenericEntry<String>("sample", 1);

    assertEquals("sample", entry.getData());
    assertEquals(1, entry.getRank());
}
```

4. Generic Constructor with Different Type

In our generic class, we can also have a constructor with a generic type that's different from the class' generic type:

```
public <E extends Rankable & Serializable> GenericEntry(E element) {
    this.data = (T) element;
    this.rank = element.getRank();
}
```

This *GenericEntry* constructor has a parameter *element* with type *E*, which is different from the *T* type. Let's see it in action:

```
@Test
public void givenGenericConstructorWithDifferentType_whenCreateGeneric
    Product product = new Product("milk", 2.5);
    product.setSales(30);

    GenericEntry<Serializable> entry = new GenericEntry<Serializable>(

    assertEquals(product, entry.getData());
    assertEquals(30, entry.getRank());
}
```

Note that:

- In our example, we used *Product* (*E*) to create a *GenericEntry* of type *Serializable* (*T*)
- We can only use this constructor when the parameter of type *E* can be cast to *T*

5. Multiple Generic Types

Next, we have the generic class *MapEntry* with two generic types:

```
public class MapEntry<K, V> {
    private K key;
    private V value;

    public MapEntry(K key, V value) {
        this.key = key;
        this.value = value;
    }
}
```

***MapEntry* has one generic constructor with two parameters, each of a different type.** Let's use it in a simple unit test:

```
@Test
public void givenGenericConstructor_whenCreateGenericEntryWithTwoTypes_
    MapEntry<String,Integer> entry = new MapEntry<String,Integer>("samp

    assertEquals("sample", entry.getKey());
    assertEquals(1, entry.getValue().intValue());
}
```

6. Wildcards

Finally, we can use wildcards in a generic constructor:

```
public GenericEntry(Optional<? extends Rankable> optional) {
    if (optional.isPresent()) {
        this.data = (T) optional.get();
        this.rank = optional.get().getRank();
    }
}
```

Here, we used wildcards in this *GenericEntry* constructor to bound the *Optional* type:

```
@Test
public void givenGenericConstructorWithWildCard_whenCreateGenericEntry
    Product product = new Product("milk", 2.5);
    product.setSales(30);
    Optional<Product> optional = Optional.of(product);

    GenericEntry<Serializable> entry = new GenericEntry<Serializable>(

    assertEquals(product, entry.getData());
    assertEquals(30, entry.getRank());
}
```

Note that we should be able to cast the optional parameter type (in our case, *Product*) to the *GenericEntry* type (in our case, *Serializable*).

7. Conclusion

In this article, we learned how to define and use generic constructors in both generic and non-generic classes.

The full source code can be found over on GitHub (https://github.com/eugenp/tutorials/tree/master/core-java-lang-oop-2).

Leave a Reply

Start the discussion...

Subscribe

CATEGORIES

SPRING (HTTPS://WWW.BAELDUNG.COM/CATEGORY/SPRING/)
REST (HTTPS://WWW.BAELDUNG.COM/CATEGORY/REST/)
JAVA (HTTPS://WWW.BAELDUNG.COM/CATEGORY/JAVA/)
SECURITY (HTTPS://WWW.BAELDUNG.COM/CATEGORY/SECURITY-2/)
PERSISTENCE (HTTPS://WWW.BAELDUNG.COM/CATEGORY/PERSISTENCE/)
JACKSON (HTTPS://WWW.BAELDUNG.COM/CATEGORY/JSON/JACKSON/)
HTTP CLIENT (HTTPS://WWW.BAELDUNG.COM/CATEGORY/HTTP/)
KOTLIN (HTTPS://WWW.BAELDUNG.COM/CATEGORY/KOTLIN/)

SERIES

JAVA "BACK TO BASICS" TUTORIAL (/JAVA-TUTORIAL)
JACKSON JSON TUTORIAL (/JACKSON)
HTTPCLIENT 4 TUTORIAL (/HTTPCLIENT-GUIDE)
REST WITH SPRING TUTORIAL (/REST-WITH-SPRING-SERIES)
SPRING PERSISTENCE TUTORIAL (/PERSISTENCE-WITH-SPRING-SERIES)
SECURITY WITH SPRING (/SECURITY-SPRING)

ABOUT

ABOUT BAELDUNG (/ABOUT)
THE COURSES (HTTPS://COURSES.BAELDUNG.COM)
CONSULTING WORK (/CONSULTING)
META BAELDUNG (HTTP://META.BAELDUNG.COM/)
THE FULL ARCHIVE (/FULL_ARCHIVE)
WRITE FOR BAELDUNG (/CONTRIBUTION-GUIDELINES)
EDITORS (/EDITORS)
OUR PARTNERS (/PARTNERS)
ADVERTISE ON BAELDUNG (/ADVERTISE)

TERMS OF SERVICE (/TERMS-OF-SERVICE)
PRIVACY POLICY (/PRIVACY-POLICY)
COMPANY INFO (/BAELDUNG-COMPANY-INFO)
CONTACT (/CONTACT)