

# Theory: Type Bounds

⌚ 18 minutes   0 / 5 problems solved

Skip this topic

Start practicing

1048 users solved this topic. Latest completion was about 18 hours ago.

## §1. Introduction

In previous articles, we have mentioned that simple generics can easily accept any type parameter and make it possible to reuse some code once written. Imagine that we have a generic `Storage<T>` class that can contain objects of any class we want it to. But there are some situations when we want to restrict these objects. We can say, for example, that the storage has to be able to contain only books, and there are different types of books. In this case, we should use Type Bounds.

## §2. Usage

Let us take a closer look at what we've got in the introduction. Consider this code:

```
1 class Storage<T> {
2     private T nameOfElements;
3     //other methods
4 }
```

Above, we can put any type of an object inside `Storage<T>`. We can put books there by casting `T` to `books`. This way, we will also be able to put any other object inside it. As we said, we would like to limit these objects and put only different types of books inside. Let's assume we have a `Books` class to represent all books. Then we can implement our limitation by adding `<T extends Books>`:

```
1 class Storage<T extends Books> {
2     private T nameOfElements;
3     //other methods
4 }
```

Let us create three classes:

```
1 public class Books { }
2 public class Brochures extends Books { }
3 public class Computers { }
```

Now creating three `Storage` objects will lead to different results:

```
1 Storage<Books> storage1 = new Storage<>(); //no problem
2 Storage<Brochures> storage2 = new Storage<>(); //no problem
3 Storage<Computers> storage3 = new Storage<>(); //a compile-time error
```

Two first lines will compile without problems unlike the third one: there we will get the `Type parameter 'Computers' is not within its bound; should extend 'Books'` error. Since this error is compile-time, we can catch this problem before any problems in real usage have appeared. That is why the usage of type bounds is safe and can be critically important in some cases.

**Note** that "extends" can mean not only an extension of a certain class but also an implementation of an interface. Generally speaking, this word is used as a replacement for an extension of usual classes because generic classes do not extend this way: you cannot write `Storage<Brochures> extends Storage<Books>`, because it will lead to an error.

## §3. Principles

Actually, type bounding implies that two keywords are used: "extends" and "super", and there are special rules regulating their utilization. However, within the current topic, we deal with the usual type bounds, where it has no

Current topic:

Type Bounds ...

Topic depends on:

✗ Generics and Object ...

✗ Generic methods ...

Topic is required for:

Type Erasure ...

Table of contents:

1 Type Bounds

§1. Introduction

§2. Usage

§3. Principles

§4. Multiple Bounds

§5. Conclusion

Feedback & Comments

sense to set a lower bound. Normally, in usual type bounds, only “extends” keyword is used. We will learn more about the principles underlying the usage of these keywords later when we’ll deal with “Wildcards” topic.

**Note**, that under the hood every type variable declared as a type parameter has bound. If no bound is declared for a type variable, `Object` is assumed. For this reason

```
1 | class SomeClass<T> {...}
```

is equivalent to

```
1 | class SomeClass<T extends Object> {...}
```

## §4. Multiple Bounds

Type bound may have the form of a single type variable:

```
1 | <T extends A>
```

or have multiple bounds:

```
1 | <T extends A & B & C & ...>
```

Here "A" is a class or an interface; "B", "C", ... are necessarily interfaces.


**Important** that if `T` has a bound with a class, this class must be specified first! Otherwise, it results in a compile-time error:

```
1 | <T extends B & C & A & ...> //an error if A is a class
```

## §5. Conclusion

Type bounds are widely used in specific coding when some type parameters need to be restricted. Normally, usual type bounds are specified with the "extends" keyword. In real situations, wildcards have a wider application: this topic is closely related to type bounds. You will learn it in the next article.

 Report a typo

82 users liked this theory.  didn't like it. What about you?



Start practicing

[Comments \(1\)](#)[Hints \(0\)](#)[Useful links \(0\)](#)[Show discussion](#)