(/)

Explanation of ClassCastException in Java

Last updated: January 8, 2024

Written by: Cristian Stancalau (https://www.baeldung.com/author/cristianstancalau)

Reviewed by: Bruno Fontana (https://www.baeldung.com/editor/brunofontana)

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

Exception (https://www.baeldung.com/tag/exception)

1. Overview

In this short tutorial, we'll focus on *ClassCastException* (https://javadoc.scijava.org/Java8/java/lang/ClassCastException.html), a common Java exception (/java-common-exceptions).

***ClassCastException* is an unchecked exception (/java-checked-unchecked-exceptions) that signals the code has attempted to cast a reference to a type of which it's not a subtype**.

Let's look at some scenarios that lead to this exception being thrown and how we can avoid them.

2. Explicit Casting

For our next experiments, let's consider the following classes:

```
public interface Animal {
    String getName();
}
```

```
public class Mammal implements Animal {
    @Override
    public String getName() {
        return "Mammal";
    }
}
```

```
public class Amphibian implements Animal {
    @Override
    public String getName() {
        return "Amphibian";
    }
}
```

```
public class Frog extends Amphibian {
    @Override
    public String getName() {
        return super.getName() + ": Frog";
    }
}
```

2.1. Casting Classes

By far, the most common scenario for encountering a *ClassCastException* is explicitly casting to an incompatible type.

For example, let's try to cast a *Frog* to a *Mammal*:

```
Frog frog = new Frog();
Mammal mammal = (Mammal) frog;
```

We might expect a *ClassCastException* here, but in fact, we get a compilation error: "incompatible types: Frog cannot be converted to Mammal". However, the situation changes when we use the common super-type:

```
Animal animal = new Frog();
Mammal mammal = (Mammal) animal;
```

Now, we get a *ClassCastException* from the second line:

```
Exception in thread "main" java.lang.ClassCastException: class Frog cannot be cast to class
Mammal (Frog and Mammal are in unnamed module of loader 'app')
at Main.main(Main.java:9)
```

A checked downcast to *Mammal* is incompatible from a *Frog* reference because *Frog* is not a subtype of *Mammal*. In this case, the compiler cannot help us, as the *Animal* variable may hold a reference of a compatible type.

It's interesting to note that the compilation error only occurs when we attempt to cast to an unequivocally incompatible class. The same is not true for interfaces because Java supports multiple interface inheritance, but only single inheritance for classes. Thus, the compiler can't determine if the reference type implements a specific interface or not. Let's exemplify:

```
Animal animal = new Frog();
Serializable serial = (Serializable) animal;
```

We get a *ClassCastException* on the second line instead of a compilation error:

```
Exception in thread "main" java.lang.ClassCastException: class Frog cannot be cast to class
java.io.Serializable (Frog is in unnamed module of loader 'app'; java.io.Serializable is
in module java.base of loader 'bootstrap')
at Main.main(Main.java:11)
```

2.2. Casting Arrays

We've seen how classes handle casting, now let's look at arrays. Array casting works the same as class casting. However, we might get confused by autoboxing and type-promotion, or lack thereof.

Thus, let's see what happens for primitive arrays when we attempt the following cast:

```
Object primitives = new int[1];
Integer[] integers = (Integer[]) primitives;
```

The second line throws a *ClassCastException* as autoboxing (/java-wrapper-classes) doesn't work for arrays.

How about type promotion? Let's try the following:

```
Object primitives = new int[1];
long[] longs = (long[]) primitives;
```

We also get a *ClassCastException* because the type promotion (/java-primitive-conversions) doesn't work for entire arrays.

2.3. Safe Casting

In the case of explicit casting, it is highly **recommended to check the compatibility of the types before attempting to cast** using *instanceof* (/java-instanceof).

Let's look at a safe cast example:

```
Mammal mammal;
if (animal instanceof Mammal) {
    mammal = (Mammal) animal;
} else {
    // handle exceptional case
}
```

3. Heap Pollution

As per the Java Specification (https://docs.oracle.com/javase/specs/jls/se8/html/jls-4.html#jls-4.12.2): "Heap pollution (https://en.wikipedia.org/wiki/Heap_pollution) can only occur if the program performed some operation involving a raw type that would give rise to a compile-time unchecked warning".

For our experiment, let's consider the following generic class:

```
public static class Box<T> {
    private T content;

    public T getContent() {
        return content;
    }

    public void setContent(T content) {
        this.content = content;
    }
}
```

We will now attempt to pollute the heap as follows:

```
Box<Long> originalBox = new Box<>();
Box raw = originalBox;
raw.setContent(2.5);
Box<Long> bound = (Box<Long>) raw;
Long content = bound.getContent();
```

The last line will throw a *ClassCastException* as it cannot transform a D*ouble* reference to *Long*.

4. Generic Types

When using generics in Java, we must be wary of type erasure (/java-type-erasure), which can lead to *ClassCastException* as well in some conditions.

Let's consider the following generic method:

```
public static <T> T convertInstanceOfObject(Object o) {
    try {
        return (T) o;
    } catch (ClassCastException e) {
        return null;
    }
}
```

And now let's call it:

```
String shouldBeNull = convertInstanceOfObject(123);
```

At first look, we can reasonably expect a null reference returned from the catch block. However, at runtime, due to type erasure, the parameter is cast to *Object* instead of *String*. Thus the compiler is faced with the task of assigning an *Integer* to *String*, which throws *ClassCastException*.

5. Conclusion

In this article, we have looked at a series of common scenarios for inappropriate casting.

Whether implicit or explicit, **casting Java references to another type can lead to *ClassCastException* unless the target type is the same or a descendent of the actual type**.

The code backing this article is available on GitHub. Once you're **logged in as a Baeldung Pro Member (/members/)**, start learning and coding on the project.

(/)

COURSES

ALL COURSES (/COURSES/ALL-COURSES)
BAELDUNG ALL ACCESS (/COURSES/ALL-ACCESS)
BAELDUNG ALL TEAM ACCESS (/COURSES/ALL-ACCESS-TEAM)
LOGIN COURSE PLATFORM (HTTPS://WWW.BAELDUNG.COM/MEMBERS/ACCOUNT)

SERIES

JAVA "BACK TO BASICS" TUTORIAL (/JAVA-TUTORIAL)
LEARN SPRING BOOT SERIES (/SPRING-BOOT)
SPRING TUTORIAL (/SPRING-TUTORIAL)
GET STARTED WITH JAVA (/GET-STARTED-WITH-JAVA-SERIES)
ALL ABOUT STRING IN JAVA (/JAVA-STRING)
SECURITY WITH SPRING (/SECURITY-SPRING)
JAVA COLLECTIONS (/JAVA-COLLECTIONS)

ABOUT

ABOUT BAELDUNG (/ABOUT)
THE FULL ARCHIVE (/FULL_ARCHIVE)
EDITORS (/EDITORS)
OUR PARTNERS (/PARTNERS/)
PARTNER WITH BAELDUNG (/PARTNERS/WORK-WITH-US)
EBOOKS (/LIBRARY/)
FAQ (/LIBRARY/FAQ)
BAELDUNG PRO (/MEMBERS/)

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

PRIVACY MANAGER