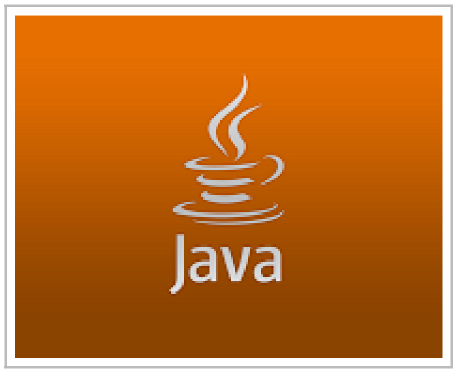
What is Type Casting in Java - Casting one Class to other class or interface Example

Type casting in Java is to cast one type, a class or interface, into another type i.e. another class or interface. Since Java is an Object oriented programming language and supports both Inheritance andPolymorphism, It's easy that Super class reference variable is pointing to Sub Class object but catch here is that there is no way for Java compiler to know that a Super class variable is pointing to Sub Class object. Which means you can not call method which isdeclared on sub class. In order to do that, you first need to cast theObject back into its original type. This is called **type-casting in Java**. This will be more clear when we see an example of type casting in next section. Type casting comes with risk of`ClassCastException` in Java, which is quite common with method which accept Object type and later type cast into morespecific type. we will see when `ClassCastException` comes during type casting and How to avoid it in coming section of this article. Another worth noting point here is that from Java 5 onwards you can use Generics to write type-safe code toreduce amount of type casting in Java which also reduces risk of `java.lang.ClassCastException` at runtime.

What is type casting in Java



From first paragraph, we pretty much know *What is type casting in Java*. Anyway, In simple words type casting is process of converting one type, which could be aclass or interface to another, But as per rules of Java programming language only classes or interfaces (collectively known as Type) from same Type hierarchy can be cast or converted into each other. If you try to cast two object which doesn't share same type hierarchy, i.e. there is no parent child relationship between them, you will get compile time error. On the other hand if you type cast objects from same type hierarchy but the object which you are casting are not of the same type on which you are casting then it will throw`ClassCastException` in Java. Some people may ask *why do you need type casting*? well you need type casting to get access of fields and methods declared on target type or class. You can not access them with any other type. Let's see a simple example of type casting in Java with two classes `Base` and `Derived` which shares same type hierarchy.

Type casting example in Java

In this Example of type casting in Java we have two classes, `Base` and `Derived`. `Derived` class extends `Base` i.e. Base is a Super class and Derived is a Sub class. So there type hierarchy looks like following tree :

```
Base
 |
Derived
```

Now look at following code :

```
Base b = new Derived(); //reference variable of Base class points object of Derived
class
Derived d = b; //compile time error, requires casting
Derived d = (Derived) b; // type casting Base to Derived
```

Above code type casting object of `Derived` class into `Base` class and it will throw `ClassCastExcepiton` if `b` is not an object of `Derived` class. If `Base` and `Derived` class are not related to each other and doesn't part of same type hierarchy, cast will throw compile time error. for example you can not cast String and StringBuffer, as they are not from same type hierarchy.

Type-casting and ClassCastExcepiton in Java

As explained in last section type casting can result in `ClassCastException` in Java. If the object you are casting is of different type. `ClassCastException` is quite common while using Java collection framework classes e.g. ArrayList,LinkedList or HashSet etc because they accept object of type `java.lang.Object`, which allows insertion of any object into collection. let's a real life example of `ClassCastException` in Java during type casting :

```
ArrayList names = new ArrayList();
names.add("abcd"); //adding String
names.add(1);   //adding Integer

String name = (String) names.get(0); //OK
name = (String) names.get(1); // throw ClassCastException because you can not convert
Integer to String
```

In above example we have an `ArrayList` of `String` which stores names. But we also added an incorrect name which is`Integer` and when we retrieve Object from collection they are of type `java.lang.Object` which needs to be cast onrespective for performing operation. This leads into `java.lang.ClassCastException` when we try to type cast second object, which is an Integer, into String. This problem can be avoided by using Generics in Java which we will see in next section.

Generics and Type Casting in Java

Generics was introduced in Java 5 along with another type-safe feature Enum, which ensures type safety of code during compile time, So rather you getting `ClassCastException` during runtime by type casting you get compile timer error why your code violate type safety. Use of Generics also removes casting from many places e.g. now while retrieving object from Collection you don't need to type cast into respective type. Here is the modified version of same code which is free of`ClassCastException` because use of Generics :

```
ArrayList<String> names = new ArrayList<String>(); //ArrayList of String only accept
String
names.add("abcd");
names.add(1);    //compile time error you can not add Integer into ArrayList of String

String name =  names.get(0); // no type casting requires
```

If you are new to Generics, those angle bracket denotes type. to learn more about Generics, See How Generics works in Java.

In this Java tutorial we learn **What is type casting in Java**, how to cast one class to another class or interface in Java with a simple type casting Example. We have also seen risk of `ClassCastException` related to type casting and How can we reduce that risk by using Generics in Java. In Summary minimize type casting in Java program and use type-safe Enum and Generics to reduce or completely eliminate type casting from Java code.