# What is Upcasting and Downcasting in Java

Last Updated on 14 October 2017 | Print Email



In your daily coding, you will see (and use) ***upcasting*** and ***downcasting*** occasionally. You may hear the terms 'casting', 'upcasting', 'downcasting' from someone or somewhere, and you may be confused about them.

As you read on, you will realize that upcasting and downcasting are really simple.

Before we go into the details, suppose that we have the following class hierarchy:

**Mammal > Animal > Dog, Cat**

Mammal is the super interface:

```
1  public interface Mammal {
2      public void eat();
3
4      public void move();
5
6      public void sleep();
7  }
```

Animal is the abstract class:

```
1  public abstract class Animal implements Mammal {
2      public void eat() {
3          System.out.println("Eating...");
4      }
5
6      public void move() {
7          System.out.println("Moving...");
8      }
9
10     public void sleep() {
11         System.out.println("Sleeping...");
12     }
13
14 }
```



Dog and Cat are the two concrete sub classes:

```
public class Dog extends Animal {
    public void bark() {
        System.out.println("Gow gow!");
    }
    public void eat() {
        System.out.println("Dog is eating...");
    }
}

public class Cat extends Animal {
    public void meow() {
        System.out.println("Meow Meow!");
    }
}
```

# 1. What is Upcasting?

***Upcasting*** is casting a subtype to a supertype, upward to the inheritance tree. Let's see an example:

```
Dog dog = new Dog();
Animal anim = (Animal) dog;
anim.eat();
```

Here, we cast the Dog type to the Animal type. Because Animal is the supertype of Dog, this casting is called upcasting.

Note that the actual object type does not change because of casting. The Dog object is still a Dog object. Only the reference type gets changed. Hence the above code produces the following output:

```
Dog is eating…
```

Upcasting is always safe, as we treat a type to a more general one. In the above example, an Animal has all behaviors of a Dog.

This is also another example of upcasting:

```
Mammal mam = new Cat();
Animal anim = new Dog();
```

# 2. Why is Upcasting?

Generally, upcasting is not necessary. However, we need upcasting when we want to write general code that deals with only the supertype. Consider the following class:

```
public class AnimalTrainer {
    public void teach(Animal anim) {
        anim.move();
        anim.eat();
    }
}
```

Here, the teach() method can accept any object which is subtype of Animal. So objects of type Dog and Cat will be upcasted to Animal when they are passed into this method:

```
Dog dog = new Dog();
Cat cat = new Cat();

AnimalTrainer trainer = new AnimalTrainer();
trainer.teach(dog);
trainer.teach(cat);
```



# 3. What is Downcasting?

**Downcasting** is casting to a subtype, downward to the inheritance tree. Let's see an example:

```
Animal anim = new Cat();
Cat cat = (Cat) anim;
```

Here, we cast the Animal type to the Cat type. As Cat is subclass of Animal, this casting is called downcasting.

Unlike upcasting, downcasting can fail if the actual object type is not the target object type. For example:

```
Animal anim = new Cat();
Dog dog = (Dog) anim;
```

This will throw a ClassCastException because the actual object type is Cat. And a Cat is not a Dog so we cannot cast it to a Dog.

The Java language provides the **instanceof** keyword to check type of an object before casting. For example:

```
if (anim instanceof Cat) {
    Cat cat = (Cat) anim;
    cat.meow();
} else if (anim instanceof Dog) {
    Dog dog = (Dog) anim;
    dog.bark();
}
```

So if you are not sure about the original object type, use the instanceof operator to check the type before casting. This eliminates the risk of a ClassCastException thrown.

## 4. Why is Downcasting?

Downcasting is used more frequently than upcasting. Use downcasting when we want to access specific behaviors of a subtype.

Consider the following example:

```
public class AnimalTrainer {
    public void teach(Animal anim) {
        // do animal-things
        anim.move();
        anim.eat();

        // if there's a dog, tell it barks
        if (anim instanceof Dog) {
            Dog dog = (Dog) anim;
            dog.bark();
        }
    }
}
```

Here, in the teach() method, we check if there is an instance of a Dog object passed in, downcast it to the Dog type and invoke its specific method, bark().

Okay, so far you have got the nuts and bolts of upcasting and downcasting in Java. Remember:

- Casting does not change the actual object type. Only the reference type gets changed.
- Upcasting is always safe and never fails.
- Downcasting can risk throwing a ClassCastException, so the instanceof operator is used to check type before casting.

That's all for upcasting and downcasting today.

Recommended Course: Complete Java Master Class

**Share this article:**