**Typecasting in Java**

Typecasting in Java is the process of converting one data type to another data type using the casting operator. When you assign a value from one primitive data type to another type, this is known as type casting. To enable the use of a variable in a specific manner, this method requires explicitly instructing the Java compiler to treat a variable of one data type as a variable of another data type.

**Syntax:**

<datatype> variableName = (<datatype>) value;

**Types of Type Casting**

There are two types of Type Casting in java:

* Widening Type Casting
* Narrow Type Casting

**Widening Type Casting**

A lower data type is transformed into a higher one by a process known as widening type casting. Implicit type casting and casting down are some names for it. It occurs naturally. Since there is no chance of data loss, it is secure. Widening Type casting occurs when:

* The target type must be larger than the source type.
* Both data types must be compatible with each other.

**Syntax:**

larger\_data\_type variable\_name = smaller\_data\_type\_variable;

Java

|  |
| --- |
| // Java program to demonstrate Widening TypeCasting  **import** java.io.\*;    **class** GFG {  **public** **static** **void** main(String[] args)      {  **int** i = 10;            // Wideing TypeCasting (Automatic Casting)          // from int to long  **long** l = i;            // Wideing TypeCasting (Automatic Casting)          // from int to double  **double** d = i;            System.out.println("Integer: " + i);          System.out.println("Long: " + l);          System.out.println("Double: " + d);      }  } |

**Output**

Integer: 10

Long: 10

Double: 10.0

**Narrow Type Casting**

The process of downsizing a bigger data type into a smaller one is known as narrowing type casting. Casting up or explicit type casting are other names for it. It doesn’t just happen by itself. If we don’t explicitly do that, a compile-time error will occur. Narrowing type casting is unsafe because data loss might happen due to the lower data type’s smaller range of permitted values. A cast operator assists in the process of explicit casting.

**Syntax:**

smaller\_data\_type variable\_name = (smaller\_data\_type) larger\_data\_type\_variable;

**Example:**

Java

|  |
| --- |
| // Java Program to demonstrate Narrow type casting  **import** java.io.\*;    **class** GFG {  **public** **static** **void** main(String[] args)      {  **double** i = 100.245;            // Narrowing Type Casting  **short** j = (**short**)i;  **int** k = (**int**)i;            System.out.println("Original Value before Casting"                             + i);          System.out.println("After Type Casting to short "                             + j);          System.out.println("After Type Casting to int "                             + k);      }  } |

**Output**

Original Value before Casting100.245

After Type Casting to short 100

After Type Casting to int 100

**Types of Explicit Casting**

Mainly there are two types of Explicit Casting:

* Explicit Upcasting
* Explicit Downcasting

**Explicit Upcasting**

Upcasting is the process of casting a subtype to a supertype in the inheritance tree’s upward direction. When a sub-class object is referenced by a superclass reference variable, an automatic process is triggered without any further effort.

**Example:**

Java

|  |
| --- |
| // Java Program to demonstrate Explicit Upcasting  **import** java.io.\*;    **class** Animal {  **public** **void** makeSound()      {          System.out.println("The animal makes a sound");      }  }    **class** Dog **extends** Animal {  **public** **void** makeSound()      {          System.out.println("The dog barks");      }    **public** **void** fetch()      {          System.out.println("The dog fetches a ball");      }  }  **class** GFG {  **public** **static** **void** main(String[] args)      { // Upcasting          Animal animal = **new** Dog();          // Calls the overridden method in Dog class          animal.makeSound();          // This would give a compile error as fetch() is not          // a method in Animal class          // animal.fetch();      }  } |

**Output**

The dog barks

**Explicit Downcasting**

When a subclass type refers to an object of the parent class, the process is referred to as downcasting. If it is done manually, the compiler issues a runtime ClassCastException error. It can only be done by using the instanceof operator. Only the downcast of an object that has already been upcast is possible.

**Example:**

Java

|  |
| --- |
| // Java Program to demonstrate Explicit downcasting  **import** java.io.\*;  **class** Animal {  **public** **void** eat()      {          System.out.println("The animal is eating.");      }  }    **class** Cat **extends** Animal {  **public** **void** meow()      {          System.out.println("The cat is meowing.");      }  }    **class** GFG {  **public** **static** **void** main(String[] args)      {          Animal animal = **new** Cat();          animal.eat();            // Explicit downcasting          Cat cat = (Cat)animal;          cat.meow();      }  } |

**Output**

The animal is eating.

The cat is meowing.