**CASTING IN JAVA(TYPE CASTING)**

**TYPECASTING**:

Converting one type to another type is called type casting

There are two types of casting in java

(i)Primitive casting

(ii)Derived casting

**Q) What is primitive type casting?**

Converting one primitive data type to another primitive data type is called primitive typecasting

Primitive casting can be classified into two types

1. Auto widening

2. Explicit Narrowing

**Auto widening:**

Converting a smaller primitive type to the any of the bigger primitive type is known as widening, since widening is done automatically by the compiler, it’s called as auto widening.

byte short int long float double

**Explicit narrowing:**

Converting the bigger primitive type to the smaller primitive type is known as explicit narrowing.

It should be explicitly specified in the program using ( ) cast operator by the programmer

byte short int long float double

int i=10;;

Double d=10.20;

String s=”Rama”;

If the right and left sides are of same type the its called **homogeneous statements**.

.

int i=10.20;;

Double d=10;

If the right and left sides are of same type the its called **heterogeneous statements.**

**Auto widening:**

Auto widening is done by compiler

Example to understand above point:

int i=10;

long l=i;

Here, long is greater than int. So, when we assign like this compiler will automatically do widening

**Explicit narrowing:**

Explicit narrowing--programmer should explicitly tell to the compiler

Example: to understand above point:

double d1=10.20;

Int k=d1;

We cannot do like this, because int is shorter than byte, compiler will not automatically widen it, we have to mention explicitly,

double d1=10.20;

Int k=(int)d1;

Program:

package com.qsp.pack1;

class Demo64

{

public static void main(String args[])

{

int i=100;

long l=i;// Auto widening

System.out.println(i);

byte b=(byte)l;// Explicit widening

System.out.println(b);

l=260;

b=(byte)l;// Explicit widening

System.out.println(b);

double d=12.44;

int k=(int)d;// Explicit widening

System.out.println(k);

char ch=25;

System.out.println(ch);

int i1=65;

char ch1=(char)i1;// Explicit widening

System.out.println(ch1);

double d1=65.45;

char ch2=(char)d1;// Explicit widening

System.out.println(ch2);

long l1=100;

//char ch1=l1;//cannot be done, as char expects only char or number, we cannot assign variable to char

double d2=100.0;

System.out.println(d2);

int i2=(int)100.0;// Explicit widening

System.out.println(i2);

}

}

O/P:

C:\practice\_java\bin>java com.qsp.pack1.Demo64

100

100

4

12

A

A

100.0

100

long l1=(long)(float)(long)(int)(short)(byte)f; correct

int i1=(int)(double)(byte)(short)(int)(long)d; correct

int d1=(double)(byte)(short)(int)f; wrong

**Note:**

In the above, we can convert or cast from one type to any other possible type any number of times in a statement but when the value is being assigned it should cast or narrowed to the type expected on the left side.

**Q) What is derived casting?**

A) Converting an object to behave like another possible type is called derived casting. In order to convert an object to another type the class should have IS A relationship.

**Derived casting:**

There are two types of derived casting

Up casting

Down casting

**Up casting:**

Converting an object of sub class to behave like any of the super type. Its automatically done by the compiler hence is called auto up casting.

Program:

package com.qsp.pack1;

class Animal

{

void eat()

{

System.out.println("eating");

}

void sleep()

{

System.out.println("sleeping");

}

}

class Dog extends Animal

{

void bark()

{

System.out.println("barking");

}

void wagtail()

{

System.out.println("wagging tail");

}

}

class Demo65

{

public static void main(String args[])

{

Dog d1=new Dog();

d1.sleep();

d1.eat();

d1.bark();

d1.wagtail();

Animal a1=new Dog();// up-casting

a1.sleep();

a1.eat();

//a1.bark();// after up-casting, it cannot access its own methods.

//a1.wagtail();

}

}

O/P:

C:\practice\_java\bin>java com.qsp.pack1.Demo65

sleeping

eating

barking

wagging tail

sleeping

eating

**Down-casting:**

Converting an up-cast object to behave like sub class type is called as down-casting

It should be explicitly specified in the program by the user. Down-casting cannot be directly done, only an up-casted object can be down-casted.

A explicit downcast using ( ) of a super class object to any of the sub class type compile but throws runtime exception- class cast exception.

Consider the above program, when we observe

class Demo65

{

public static void main(String args[])

{

Dog d1=new Dog(); // normal creation of object

d1.sleep();

d1.eat();

d1.bark();

d1.wagtail();

Animal a1=new Dog();// up-casting

a1.sleep();

a1.eat();

Now, a1 is pointing to Dog object which can access only animal object methods.

We are making sub class to behave like super class

Here, we are creating a sub class and assigning it super class Animal a1 reference variable,

Now we create animal object (which is super class) and assign it to Dog d reference variable(which is sub class)

Dog d=new Animal();

This is called direct down-casting.

This will throw compile time error -stating that not compatible type error

So, super class will never behave like sub class, Ex: every dog will behave like an animal, but every animal cannot behave like dog.

We will see whether above stating is possible or not using casting

Dog d=(Dog)new Animal();

This is called explicit down-casting.

This is also not possible.

This is will compile, but during execution JVM will throw “class cast Exception”.

How to down cast then?

Observe....

Here,

Animal a1=new Dog();// up-casting

a1.sleep();

a1.eat();

Dog is up-casted, Now we are down-casting the up-casted object

Dog d=(Dog) a1;

we are assigning the a1 to the Dog reference variable d using cast operator.

Observe the program:

Program:

package com.qsp.pack1;

class Animal

{

void eat()

{

System.out.println("eating");

}

void sleep()

{

System.out.println("sleeping");

}

}

class Mammal extends Animal

{

void walking()

{

System.out.println("walking");

}

void run()

{

System.out.println("Running");

}

}

class Monkey extends Mammal

{

void jump()

{

System.out.println("jumping");

}

void trek()

{

System.out.println("trekking");

}

}

class Demo70

{

public static void main(String args[])

{

//Monkey m1=new Monkey();, normally creating the object

Mammal m2 =new Monkey();// upcasting

m2.walking();

m2.run();

Animal a1=new Mammal();

a1.eat();

a1.sleep();

//now Monkey() and Mammal() being up-casted, now down-casting the up-casted object

Monkey m3=(Monkey)m2;//down-casted

m3.jump();

m3.trek();

m3.walking();

m3.run();

m3.eat();

m3.sleep();

Mammal m4= (Mammal)a1;//down-casted

m4.walking();

m4.run();

m4.eat();

m4.sleep();

}

}

O/P:

G:\practice\_java\bin>java com.qsp.pack1.Demo70

walking

Running

eating

sleeping

jumping

trekking

walking

Running

eating

sleeping

walking

Running

eating

sleeping