Object Oriented Programming with Java 8

Akshita Chanchlani

Upcasting

When the reference variable of super class refers to the object of subclass, it is known as widening or **upcasting in java**.

when subclass object type is converted into superclass type, it is called widening or upcasting.

Superclass s = new SubClass();

Up casting : *Assigning child class object to parent class reference .*
Syntax for up casting : **Parent p = new Child();**
Here **p** is a parent class reference but point to the child object. *This reference p can access all the methods and variables of parent class but only overridden methods in child class.*

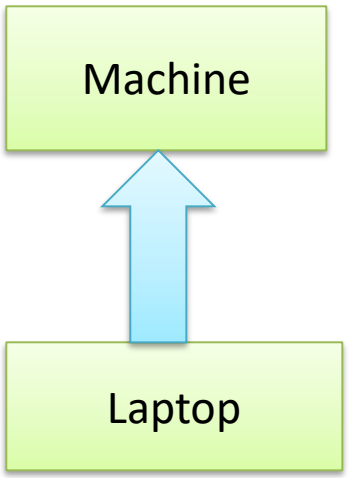
Upcasting gives us the flexibility to access the parent class members, but it is not possible to access all the child class members using this feature. Instead of all the members, we can access some specified members of the child class. For instance, we can only access the overridden methods in the child class.

Downcasting

Down casting : *Assigning parent class reference (which is pointing to child class object) to child class reference .*

Syntax for down casting : **Child c = (Child)p;**

Here **p** is pointing to the object of child class as we saw earlier and now we cast this parent reference **p** to child class reference **c**. *Now this child class reference **c** can access all the methods and variables of child class as well as parent class.*



*For example, if we have two classes, **Machine** and **Laptop** which extends **Machine** class. Now for upcasting, every laptop will be a machine but for downcasting, every machine may not be a laptop because there may be some machines which can be **Printer**, **Mobile,** etc.*
Downcasting is not always safe, and we explicitly write the class names before doing downcasting. So that it won't give an error at compile time but it may throw **ClassCastExcpetion** at run time, if the parent class reference is not pointing to the appropriate child class.

Explanation

```
Machine machine = new Machine ();

Laptop laptop = (Laptop)machine;//this won't give an error while compiling
```

//laptop is a reference of type Laptop and machine is a reference of type Machine and points to Machine class Object .So logically assigning machine to laptop is invalid because these two classes have different object strucure.And hence throws ClassCastException at run time .

To remove ClassCastException we can use instanceof operator to check right type of class reference in case of down casting .

```
if(machine instanceof Laptop)
{
Laptop laptop = machine;      //here machine must be pointing to Laptop class object .
}
```

Polymorphisim

- The ability to have many different forms
- An object always has only one form
- A reference variable can refer to objects of different forms

Method Binding

Static Binding

- Static binding
- Compile time
- early binding
- Resolved by java compiler
- Achieved via method overloading

Example :

In class Test :

```
void test(int i,int j){...}
void test(int i) {..}
void test(double i){..}
void test(int i,double j,boolean flag){..}
int test(int a,int b){...}   //javac error
```

Dynamic Binding

- Dynamic binding
- Run time / Late binding
- Resolved by java runtime environment
- Achieved by method overriding (Dynamic method dispatch)
- Method Overriding is a Means of achieving run-time polymorphism

All java methods can be overridden : if they are not marked as private or static or final

Super-class form of method is called as overridden method

sub-class form of method is called as overriding form of the method

Example :

```
class A {
  A getInstance()
        {
          return new A();
        }
  }
```

```
class B extends A
{
   B getInstance()
         {
           return new B();
         }
}
```

Run time polymorphism or Dynamic method dispatch

- Super -class ref. can directly refer to sub-class object(direct=w/o type casting) as its the example of up-casting(similar to widening auto. conversion) .

- When such a super class ref is used to invoke the overriding method then the method to send for execution that decision is taken by JRE & not by compiler.
- In such case overriding form of the method(sub-class version) will be dispatched for exec.
- Javac resolves the method binding by the type of the reference & JVM resolves the method binding by type of the object it's referring to.
- Super class reference can directly refer to sub-class instance BUT it can only access the members declared in super-class directly.
- eg : A ref=new B();
  ref.show(); // this will invoke the sub-class: overriding form of the show () method

Java compiler resolves method binding by type of the reference & JVM resolves it by the type of the object, reference is referring to.

Nested Class

- In Java, we can define class inside scope of another class. It is called nested class.
- Nested class represents encapsulation.

```
//Top-Level class
class Outer{    //Outer.class
    //Nested class
    class Inner{    //Outer$Inner.class
       //TODO
    }
}
```

- Access modifier of top level class can be either package level private or public only.
- We can use any access modifier on nested class.
- Types of nested class:
    1. Non static nested class / Inner class
    2. Static nested class

Non Static Nested Class

- Non static nested class is also called as inner class.
- If implementation of nested class depends on implementation of top level class then nested class should be non static.
- **Implementation Hint :** For the simplicity, consider non static nested class as non static method of class.

```
class Outer{
    public class Inner{
        //TODO
    }
}
```

```
Instantiation of top level class:
Outer out = new Outer( );
```

```
* Instantiation of top level class:
- method 1
Outer out = new Outer( );
Outer.Inner in = out.new Inner( );
```

```
- method 2
Outer.Inner in = new Outer( ).new Inner( );
```

Non Static Nested Class

- Top level class can contain static as well as non static members.
- Inside non static nested class we can not declare static members.
- If we want to declare any field static then it must be final.
- Using instance, we can access members of non static nested class inside method of top level class.
- Without instance, we can use all the members of top level class inside method of non static nested class.
- But if we want refer member of top level class, inside method of non static nested class then we should use "TopLevelClassName.this" syntax.

Non Static Nested Class

```
class Outer{
    private int num1 = 10;

    public class Inner{
        private int num1 = 20;

        public void print( ) {
            int num1 = 30;
            System.out.println("Num1     :    "+Outer.this.num1); //10
            System.out.println("Num1     :    "+this.num1);    //20
            System.out.println("Num1     :    "+num1);     //30
        }
    }
}
public class Program {
    public static void main(String[] args) {
        Outer.Inner in = new Outer().new Inner();
        in.print();
    }
}
```

Static Nested Class

- If we declare nested class static then it is simply called as static nested class.
- We can declare nested class static but we can not declare top level class static.
- If implementation of nested class do not depends on implementation of top level class then we should declare nested class static.
- **Implementation hint:** For simplicity consider static nested class as a static method of a class.

```
class Outer{
    public static class Inner{
        //TODO
    }
}
```

```
* Instantiation of top level class:
Outer out = new Outer( );

* Instantiation of static nested class:
Outer.Inner in = new Outer.Inner();
```

Static Nested Class

- Static nested class can contain static members.
- Using instance, we can access all the members of static nested class inside method of top level class.
- If we want to use non static members of top level class inside method of static nested class then it is mandatory to create instance of top level class.

Nested Class

```
class LinkedList implements Iterable<Integer>{
    static class Node{
        //TODO
    }
    //TODO
    class LinkedListIterator implements Iterator<Integer>{
        //TODO
    }
}
```

Local Class

- In Java, we can define class inside scope of another method. It is called local class / method local class.

- Types of local class:
  1. Method local inner class
  2. Method local anonymous inner class.

Method Local Inner Class

- In Java, we can not declare local variable /class static hence local class is also called as local inner class.
- We can not use reference/instance of method local class outside method.

```
public class Program {  //Program.class
    public static void main(String[] args) {
        class Complex{  //Program$1Complex.class
            private int real = 10;
            private int imag = 20;
            public void print( ) {
                System.out.println("Real Number :    "+this.real);
                System.out.println("Imag Number :    "+this.imag);
            }
        }
        Complex c1 = new Complex();
        c1.print();
    }
}
```

Method local anonymous inner class.

- In java, we can create instance without reference. It is called anonymous instance.
- Example:
- ```
  new Object( );
  ```
- We can define a class without name. It is called anonymous class.
- If we want to define anonymous class then we should use new operator.
- We can create anonymous class inside method only hence it is also called as method local anonymous class.
- We can not declare local class static hence it is also called as method local anonymous inner class.
- To define anonymous class, we need to take help of existing interface / abstract class / concrete class.
  ```

Method local anonymous inner class.

- Consider anonymous inner class using concrete class.

```
public static void main(String[] args) {
    //Object obj;   //obj => reference
    //new Object( );    // new Object( ) => Anonymous instance
    //Object obj = new Object( );//Instance with reference
    Object obj = new Object( ) {    //Program$1.class
        private String message = "Hello";
        @Override
        public String toString() {
            return this.message;
        }
    };
    String str = obj.toString();
    System.out.println(str);
}
```

Method local anonymous inner class.

- Consider anonymous inner class using abstract class:

```
abstract class Shape{
    protected double area;
    public abstract void calculateArea( );
    public double getArea() {
        return area;
    }
}
```

```
public class Program {
    public static void main(String[] args) {
        Shape sh = new Shape() {
            private double radius = 10;
            @Override
            public void calculateArea() {
                this.area = Math.PI * Math.pow(this.radius, 2);
            }
        };

        sh.calculateArea();
        System.out.println("Area    :    "+sh.getArea());
    }
}
```

Method local anonymous inner class.

- Consider anonymous inner class using interface.

```
interface Printable{
    void print( );
}
```

```
public class Program {
    public static void main(String[] args) {
        Printable p = new Printable() {
            @Override
            public void print() {
                System.out.println("Hello");
            }
        };
        p.print();
    }
}
```

Thank You.

akshita.chanchlani@sunbeaminfo.com