Java Reference Type Casting

Source	http://javasoft.phpnet.us/res/tut_typecast.html

In java one object reference can be cast into another object reference. The cast can be to its own class type or to one of its subclass or superclass types or interfaces. There are compile-time rules and runtime rules for casting in java. The casting of object references depends on the relationship of the classes involved in the same hierarchy. Any object reference can be assigned to a reference variable of the type Object, because the Object class is a superclass of every Java class.

There can be 2 types of casting

- Upcasting
- Downcasting

When we cast a reference along the class hierarchy in a direction from the root class towards the children or subclasses, it is a downcast.

When we cast a reference along the class hierarchy in a direction from the sub classes towards the root, it is an upcast. We need not use a cast operator in this case.

The compile-time rules are there to catch attempted casts in cases that are simply not possible. This happens when we try to attempt casts on objects that are totally unrelated (that is not subclass super class relationship or a class-interface relationship)

At runtime a ClassCastException is thrown if the object being cast is not compatible with the new type it is being cast to.

Below is an example showing when a ClassCastException can occur during object casting

```
                //X is a supper class of Y and Z which are sibblings.

public class RunTimeCastDemo{
        public static void main(String args[]){
                X x = new X();
                Y y = new Y();
                Z z = new Z();

                X xy = new Y();  // compiles ok (up the hierarchy)
                X xz = new Z();  // compiles ok (up the hierarchy)
//              Y yz = new Z();   incompatible type (siblings)

//              Y y1 = new X();   X is not a Y
//              Z z1 = new X();   X is not a Z
```

```
            X x1 =  y;      // compiles ok (y is subclass of X)
            X x2 =  z;      // compiles ok (z is subclass of X)

            Y y1 = (Y) x;   // compiles ok but produces runtime error
            Z z1 = (Z) x;   // compiles ok but produces runtime error
            Y y2 = (Y) x1;  // compiles and runs ok (x1 is type Y)
            Z z2 = (Z) x2;  // compiles and runs ok (x2 is type Z)
//          Y y3 = (Y) z;    inconvertible types (siblings)
//          Z z3 = (Z) y;    inconvertible types (siblings)

            Object o  = z;
            Object o1 = (Y)o; // compiles ok but produces runtime error

      }
}
```

Casting Object References: Implicit Casting using a Java Compiler

In general an implicit cast is done when an Object reference is assigned (cast) to:

A reference variable whose type is the same as the class from which the object was instantiated.

An Object as Object is a super class of every Java Class.

A reference variable whose type is a super class of the class from which the object was instantiated.

A reference variable whose type is an interface that is implemented by the class from which the object was instantiated.

A reference variable whose type is an interface that is implemented by a super class of the class from which the object was instantiated.

Consider an interface Vehicle, a super class Car and its subclass Ford. The following example shows the automatic conversion of object references handled by the java compiler

```
interface Vehicle {
}
class Car implements Vehicle {
  }

class Ford extends Car {
  }
```

Let c be a variable of type Car class and f be of class Ford and v be an vehicle interface reference. We can assign the Ford reference to the Car variable:

I.e. we can do the following

Example 1

c = f; //Ok Compiles fine

Where c = new Car();

And, f = new Ford();

The compiler automatically handles the conversion (assignment) since the types are compatible (sub class - super class relationship), i.e., the type Car can hold the type Ford since a Ford is a Car.

Example 2

v = c; //Ok Compiles fine

c = v; // illegal conversion from interface type to class type results in compilation error

Where c = new Car();

And v is a Vehicle interface reference (Vehicle v)

The compiler automatically handles the conversion (assignment) since the types are compatible (class – interface relationship), i.e., the type Car can be cast to Vehicle interface type since Car implements Vehicle Interface. (Car is a Vehicle).

Casting Object References: Explicit Casting

Sometimes we do an explicit cast in java when implicit casts don't work or are not helpful for a particular scenario. The explicit cast is nothing but the name of the new "type" inside a pair of matched parentheses. As before, we consider the same Car and Ford Class

```
class Car {
      void carMethod(){
    }
}
```

```
class Ford extends Car {
      void fordMethod () {
   }
}
```

We also have a breakingSystem() function which takes Car reference (Superclass reference) as an input parameter.

The method will invoke carMethod() regardless of the type of object (Car or Ford Reference) and if it is a Ford object, it will also invoke fordMethod(). We use the instanceof operator to determine the type of object at run time.

```
public void breakingSystem (Car obj) {
   obj.carMethod();
```

```
    if (obj instanceof Ford)

        ((Ford)obj).fordMethod ();
}
```

To invoke the fordMethod(), the operation (Ford)obj tells the compiler to treat the Car object referenced by obj as if it is a Ford object. Without the cast, the compiler will give an error message indicating that fordMethod() cannot be found in the Car definition.

The following java shown illustrates the use of the cast operator with references.

Note: Classes Honda and Ford are Siblings in the class Hierarchy. Both these classes are subclasses of Class Car. Both Car and HeavyVehicle Class extend Object Class. Any class that does not explicitly extend some other class will automatically extends the Object by default. This code instantiates an object of the class Ford and assigns the object's reference to a reference variable of type Car. This assignment is allowed as Car is a superclass of Ford.

In order to use a reference of a class type to invoke a method, the method must be defined at or above that class in the class hierarchy. Hence an object of Class Car cannot invoke a method present in Class Ford, since the method fordMethod is not present in Class Car or any of its superclasses. Hence this problem can be colved by a simple downcast by casting the Car object reference to the Ford Class Object reference as done in the program.

Also an attempt to cast an object reference to its Sibling Object reference produces a ClassCastException at runtime, although compilation happens without any error.

```
 class Car extends Object{
        void carMethod() {
    }
}

class HeavyVehicle extends Object{

}

class Ford extends Car {
        void fordMethod () {
                System.out.println("I am fordMethod defined in Class Ford");
    }
}

class Honda extends Car {
```

```
        void fordMethod () {
                System.out.println("I am fordMethod defined in Class Ford");
        }
}

public class ObjectCastingEx{
  public static void main(
                    String[] args){
        Car obj = new Ford();
//    Following will result in compilation error
//    obj.fordMethod();        //As the method fordMethod is undefined for the Car Type
//  Following will result in compilation error
//    ((HeavyVehicle)obj).fordMethod();    //fordMethod is undefined in the HeavyVehicle
Type
//  Following will result in compilation error

        ((Ford)obj).fordMethod();

    //Following will compile and run
//        Honda hondaObj = (Ford)obj;        Cannot convert as they are sibblings

  }
}
```

One common casting that is performed when dealing with collections is, you can cast an object reference into a String.

```
import java.util.Vector;

public class StringCastDemo{
        public static void main(String args[]){
                String username = "asdf";
                String password = "qwer";
                Vector v = new Vector();
                v.add(username);
                v.add(password);


//              String u = v.elementAt(0); Cannot convert from object to String
                Object u = v.elementAt(0);  //Cast not done
                System.out.println("Username : " +u);
```

```
            String uname = (String) v.elementAt(0); // cast allowed
            String pass = (String) v.elementAt(1); // cast allowed

            System.out.println();
            System.out.println("Username : " +uname);
            System.out.println("Password : " +pass);
        }
}
```

Output

```
Username : asdf
Username : asdf
Password : qwer
```

~~~ End of Article ~~~