# CAST ENTRE  DE REFERENCIAS

Para entender el concepto de  polimorfismo previamente necesitamos saber:

- Posibilidad  de asignación a una referencia de tipo superclase un objeto subclase y posibilidades  de cast entre diferentes tipos de referencia
- Concepto de ligadura dinámica

***A una referencia de un tipo, no se le puede asignar una referencia u objeto de otro tipo, salvo que haya relación jerárquica.***

```
class A{
   int x;
}
class B{
   int y;
}

 class App{
   public static void main(String[] args) {
      A a;
      B b;
      a= new A();
      //b=a; ERROR
      b=new B();
      //a=b; ERROR
   }
}
```

Se escogieron para el ejemplo dos clases con la misma estructura, ambas tienen un int, pero a pesar de esto observamos errores en tiempo de compilación. Todos los errores son de la misma naturaleza. Al hacer

```
a=b;
```

El compilador informa de una asignación prohibida entre referencias de distinto tipo. El compilador mira el tipo de a y b para determinar compatibilidad

Al hacer

b= new A();

Un tanto de lo mismo,  b es de tipo B y le intentamos asignar un objeto de tipo A. Respecto a la indicación del enunciado "salvo relación jerárquica", lo vemos en el siguiente apartado.

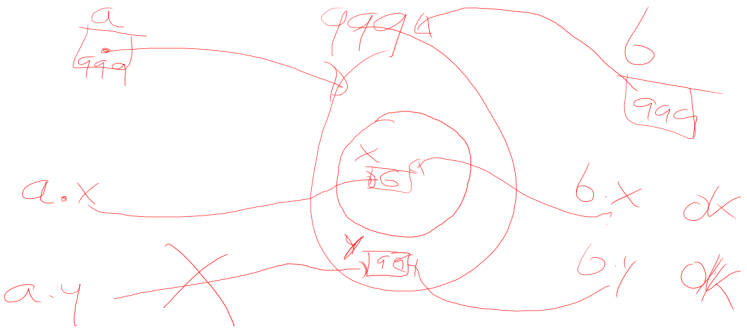
***Asignar a una referencia de superclase, un objeto de subclase***

Sin embargo,  es posible hacer en cierta medida asignaciones entre referencias de distinto tipo si entre ellas hay relación  jerárquica.

**Ejemplo:**

```
class A{
   int x;
}
class B extends A{
   int y;
}
```

```
class App{
   public static void main(String[] args) {
      A a;
      a= new B(); //con extends no hay error. Quita y comprueba.
      a.x=6;
      a.y=99;//error a sólo puede acceder a parte superclase
   }
}
```

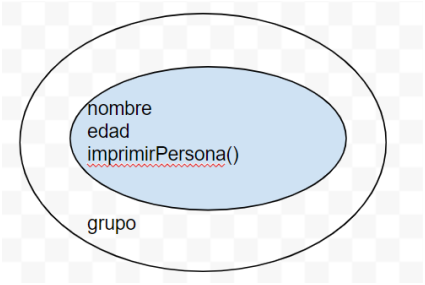


POR TANTO: si *B* extiende a *A*, *a* puede referenciar a un objeto de tipo B, pero sólo puede acceder a la parte que se corresponde con la superclase, en este caso, al atributo x;

**EJEMPLO:** Recuerda el siguiente dibujo que mostraba como un objeto Alumno contiene en su interior el objeto padre Persona

```
class Persona {
    String nombre;
    int edad;

    public void imprimePersona() {System.out.println("Datos personales: " + nombre + ", "+ edad );
    }
}
class Alumno extends Persona{
    char grupo;
}
```




Comprobamos con un main como puedo referenciar a un objeto alumno con una referencia Persona

```
class Persona {
    String nombre;
    int edad;

    public void imprimePersona() {System.out.println("Datos personales: " + nombre + ", "+ edad );
    }
}
class Alumno extends Persona{
    char grupo;
}
class App {
    public static void main(String[] args) {
        Persona p1 = new Persona();
        p1.nombre="Elías";
```

```
      p1.edad=5;
      p1.imprimePersona();
      //comprobamos como alumno hereda de persona
      Alumno a1= new Alumno();
      a1.nombre="Román";
      a1.edad=3;
      a1.grupo='a';
      a1.imprimePersona();

      //ESTO ES LO NUEVO: UNA REFERENCIA PERSONA PUEDE SEÑALAR A UN OBJETO ALUMNO
      p1=a1;
      //pero ojo, sólo puede acceder a la parte del subobjeto Persona
      System.out.println(p1.nombre);
      System.out.println(p1.edad);
      //ERROR ¡no puedo referenciar lo que está fuera de la zona padre!
      System.out.println(p1.grupo);

   }
}
```

## Conversión (cast)de referencias

¿Recuerdas la conversión (cast) entre tipos primitivos? ¿Es posible el cast entre referencias y objetos? Sí, dependiendo de la relación entre los tipos de objetos y referencias. Veamos una serie de casos.

Si *B extends A*, Podemos usar el operador cast para que una referencia de tipo A se convierta en una Referencia de tipo B. La conversión la hace el programador por su cuenta y riesgo, si hay algún tipo de incompatibilidad "salta" una excepción en tiempo de ejecución.

## Ejemplo: cast de referencias correcto

```
class A {
    int x=2;
}
class  B extends A {
    int y=3;
}
public class App{
   public static void main(String[] args) {
      int temp;
      A a;  // Referencia a objetos de la clase A
      a= new B (); // "a" es una referencia a objetos clase A, pero referencia realmente a un objeto clase B
      //temp=a.y; //error, "a" apunta a un objeto de clase B pero solo tiene acceso a subobjeto  superclase
      B b= (B) a;//la referencia a la convierto a otra de tipo B
      temp=b.y; //algo de tipo B si puede acceder a y
  }
}
```

## Ejemplo: cast de referencias correcto con Persona/Alumno

```
class Persona {

   String nombre;
   int edad;

   public void imprimePersona() {
      System.out.println("Datos personales: " + nombre + ", " + edad);
   }
}

class Alumno extends Persona {

   char grupo;
}
```

```
class App {

   public static void main(String[] args) {

      Alumno a1 = new Alumno();
      a1.nombre = "Román";
      a1.edad = 3;
      a1.grupo = 'a';

      Persona p1 = a1;
      Alumno a2 = (Alumno) p1;
      System.out.println(a2.grupo);//OK

   }
}
```

**Ejemplo: cast de referencias que provoca excepción en tiempo de ejecución.**

En el ejemplo anterior el programador debe de estar seguro que cuando hace

```
B b= (B) a;//la referencia  la convierto a otra de tipo B
```

realmente *a* está referenciado a un objeto *B*, en caso contrario habrá problemas en tiempo de ejecución

```
class A {
    int x=2;
}
class  B extends A {
    int y=3;
}
public class App{
   public static void main(String[] args) {
      int temp;
      A a;  // Referencia a objetos de la clase A
      a= new A (); //diferente que en ejemplo anterior
      B b= (B) a;//la refencia a la convierto a otra de tipo B
      temp=b.y; //algo de tipo B si puede acceder a y
  }
}
```

Observa que es exactamente el mismo código que en el ejemplo anterior pero sustituyendo

```
a= new B ();
```

por

```
a= new A ();
```

Comprueba que no hay errores en tiempo de compilación pero al ejecutar el programa salta la excepción.No se detecta el error en tiempo de compilación ya que a diferencia de los tipos primitivos, los objetos se crean en memoria en tiempo de ejecución, por tanto, en muchos casos no es detectable el error hasta ese momento.

En el caso anterior, la responsabilidad es del programador, ya que por mucho cast que hagamos la máquina java no puede crear una parte de un objeto sin saber a qué atenerse, y por tanto genera una excepción.

Recuerda que para que una referencia pueda "navegar" por un objeto al que referencia, debe ser del tipo de ese objeto lo que le garantiza que conoce su estructura y poder por tanto desplazarse por el.

**Ejemplo: cast de referencias que provoca excepción en tiempo de ejecución con Persona/alumno**

Fíjate bien en el único new que hay en el main y observa que es lógico que se produzca el error en tiempo de ejecución. Pero observa que no hay error en tiempo de compilación.

```
class Persona {

   String nombre;
   int edad;

   public void imprimePersona() {
      System.out.println("Datos personales: " + nombre + ", " + edad);
   }
}

class Alumno extends Persona {

   char grupo;
}

class App {

   public static void main(String[] args) {

      Persona p = new Persona();
      p.nombre = "Román";
      p.edad = 3;


      Alumno a = (Alumno) p;
      System.out.println(a.grupo);

   }
}
```

**Ejemplo: cast de referencias que provoca error en  tiempo de compilación.**

En este ejemplo B no extiende a A y no se permite convertir algo de tipo A en B. En este contexto, el cast erróneo se detecta en tiempo de compilación.

```
class A {
    int x=2;
}

class  B { //B no extiende a A
    int y=3;
}
public class App{
   public static void main(String[] args) {
     int temp;
     A a;
     a= new A ();
     B b= (B) a;//este cast no es posible

  }
}
```

CONCLUSIÓN: El cast de referencias tiene sentido cuando una referencia padre referencia a un objeto hijo, y luego, por la razón que sea queremos convertir convertir la referencia padre en tipo hijo.

**EJERCICIO:** Escribir la clase MiRacional

Tenemos la siguiente clase Racional incompleta

```
package racional;
public class Racional{
   protected  int numerador;
```

```
    protected   int denominador;

    public Racional(int numerador,int denominador) {
        this.numerador=numerador;
        this.denominador=denominador;
    }
    public Racional  multiplicar(Racional otro){
        return new Racional(this.numerador*otro.numerador,this.denominador*otro.denominador);
    }
    //falta sobrescribir toString() y equals()

}
```

Y el siguiente main()

```
package miscalculos;

import racional.Racional;
class App{
 public static void main(String[]  args){
        Racional r1= new Racional(2,3);
        Racional r2= new Racional(5,3);
        Racional r3;
        r3=r1.multiplicar(r2);
        //probar toString()
        System.out.println("multiplicación: "+ r3);

        //probamos equals
        r2= new Racional(2,3); //ahora r1 y r2 tienen igual numerador y denominador
        System.out.println(r1.equals(r2));
        System.out.println(r1.equals(r3));

    }
}
```

Que debe producir la siguiente salida

10/9

true

false

SE PIDE:

Completar la clase Racional sobrescribiendo toString() y equals() de object.

Consideramos que equals() debe ser true si los numeradores de los dos números son iguales entre sí y los denominadores son iguales entre sí.

Observa que para sobrescribir equals() su parámetro tiene que ser tipo Object, pero en el cuerpo del método te verás forzado a hacer un cast.

Cuando veamos colecciones hablaremos de forma más formal de la sobrescritura de equals(), ahora simplemente, consigue sobreescribirlo para obtener una salida como la anterior