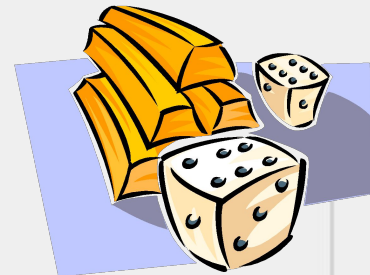


Ejemplo:

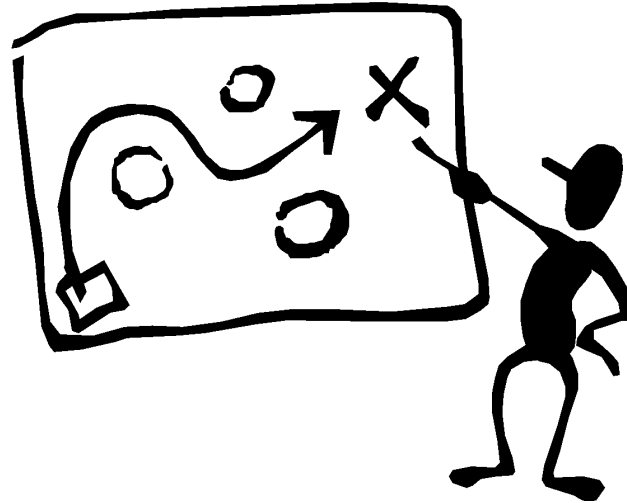
Juego de Dados

Dados

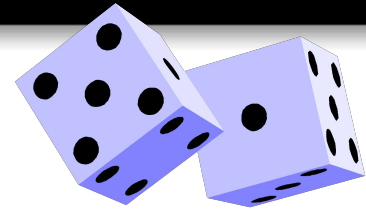


Objetivos

- Identificar los objetos que intervienen en varios sistemas simples.
- Identificar sus responsabilidades y la forma en que colaboran entre sí.



Ejemplo: Un Juego de Dados

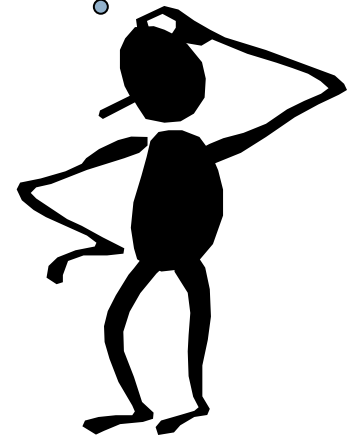


■ Reglas del Juego:

- Los jugadores tiran dos dados diez veces.
- En cada tirada, el jugador que obtiene la mayor puntuación es el ganador (siempre que sume más que 7).

**Analizando
alternativas**

- ¿Cuales son las responsabilidades de los objetos de este juego?
- ¿Que objetos deben colaborar entre sí ?
- Importante: este es un paso creativo con una variedad de soluciones
 - Un aspecto importante de este curso es determinar “que es un buen diseño y porque”



Alternativa 1

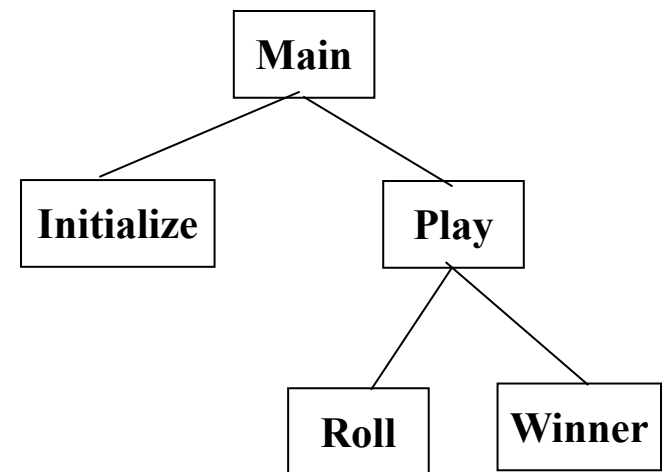
```
public class Juego {  
    int puntos1;  
    int puntos2;  
  
    public Juego() {  
        puntos1=0;  
        puntos2=0;}  
  
    public void jugar() {  
        for (int i = 0; i < 10; i++)    {  
            int r1, r2;  
            r1= tirar() + tirar();  
            r2= tirar() + tirar();  
            if (r1 >= 7 && r1 > r2) puntos1++;  
            else if (r2 >= 7 && r2 > r1) puntos2++; }  
  
            if (ganador() != null) System.out.println("Ganador: "+ ganador());  
            else System.out.println("Empate");  
        }  
    }
```

Alternativa 1

```
public String ganador() {  
    if (puntos1 > puntos2) return "Jugador 1";  
    else if (puntos2 > puntos1) return "Jugador 2";  
    else return null; }  
  
public int tirar() { return (int) (Math.random() * 6) + 1;  
}  
  
public static void main( String args[] ) {  
    Juego juego=new Juego();  
    juego.jugar(); } }
```



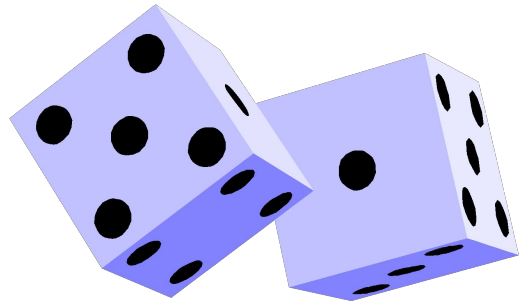
- Todo centralizado
 - Prácticamente una solución procedural
- No se identificaron correctamente los objetos. El modelo dista más de la realidad.
- No se distribuyeron correctamente las responsabilidades.
- Difícil de mantener, modificar, extender.



Que Objetos debería tener el juego?



Alternativa 2: Modelando un Dado como un Objeto



Dado
ultimoValor
tirar() ultimoValor() : int

■ Un dado:

- Es responsable de rodar por sí mismo—principio “Lo hago Yo mismo” (tirar)
- Es responsable de responder cuál es el valor de cara obtenido (ultimoValor)
- Recuerda el valor obtenido (ultimoValor).

Clase Dado

```
public class Dado{  
    private int ultimoValor;
```

acceso

tipo

Tipo de Retorno

```
public int ultimoValor()  
{  
    return ultimoValor;  
}
```

Nombre del Método

Sentencia de Retorno

```
public void tirar()  
{  
    ultimoValor= (int) (Math.random() * 6 ) + 1;  
}  
}
```

Genera un número al azar

El método `main()` – Comienzo de la Aplicación

- El primer método que es ejecutado cuando la aplicación comienza
- Generalmente utilizado para crear y enviar mensajes a otros objetos

```
public static void main ( String[] args )  
{  
    Dado d = new Dado( );  
    d.tirar( );  
}
```

- Cualquier clase puede contener un método `main()`.

Utilizando dados

```
public class TestDado  
{
```

punto de comienzo de
ejecución de un Programa
Java

```
public static void main( String args[] )
```

```
{
```

Se crea una instancia de dado

```
Dado d1 = new Dado();
```

```
for ( int i = 0; i < 10; i++ )
```

Se itera 10
veces

```
{
```

```
d1.tirar();
```

Se le envía el
mensaje "roll" al dado

```
int valor = d1.ultimoValor();
```

```
System.out.println( valor);
```

```
}
```

```
}
```

```
}
```

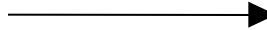
Imprime el
resultado en
Pantalla

Clase Juego

```
public class Juego {  
    int puntos1, puntos2;  
    private Dado dado1 = new Dado();  
    private Dado dado2 = new Dado();  
  
    public void jugar() {  
        for (int i = 0; i < 10; i++) {  
            int r1, r2;  
            d1.tirar(); d2.tirar();  
            r1= d1.ultimoValor() + d2.ultimoValor();  
            d1.tirar(); d2.tirar();  
            r2= d1.ultimoValor() + d2.ultimoValor();  
            if (r1 >= 7 && r1 > r2) puntos1++;  
            else if (r2 >= 7 && r2 > r1) puntos2++; }  
  
            if (ganador() != null) System.out.println("Ganador: "+ ganador());  
            else System.out.println("Empate");  
        }  
    }  
}
```

Alternativa 3: Modelando un Jugador como un Objeto

Player



Jugador

nombre: String
puntos: int

Jugador(String n)

incrementarPuntos()

■ Un jugador

- Sabe cuantos puntos tiene y su nombre

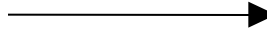
```
public void Jugar() {  
    for (int i = 0; i < 10; i++)    {  
        int r1, r2;  
        d1.tirar(); d2.tirar();    r1= d1.ultimoValor() + d2.ultimoValor();  
        d1.tirar(); d2.tirar();    r2= d1.ultimoValor() + d2.ultimoValor();  
        if (r1 >= 7 && r1 > r2) jug1.incrementarPuntos();  
        else if (r2 >= 7 && r2 > r1) jug2.incrementarPuntos(); }  
  
        if (ganador() != null) System.out.println("Ganador: "+ ganador());  
        else System.out.println("Empate"); }  
}
```

Qué más hace un jugador?



Alternativa 3: Modelando un Jugador como un Objeto II

Player



Player

nombre: String
puntos: int

Jugador(String n)
tirar(Dado d1,d2)

incrementarPuntos()

■ Un jugador

- Además sabe jugar

```
public void play() {  
    for (int i = 0; i < 10; i++)    {  
        int r1, r2;  
        r1= jug1.tirar(d1,d2);  
        r2= jug2. tirar(d1,d2);  
        if (r1 >= 7 && r1 > r2) jug1.incrementarPuntos();  
        else if (r2 >= 7 && r2 > r1) jug2.incrementarPuntos(); }  
  
    if (ganador() != null) System.out.println("Ganador: "+ ganador());  
    else System.out.println("Empate"); }
```

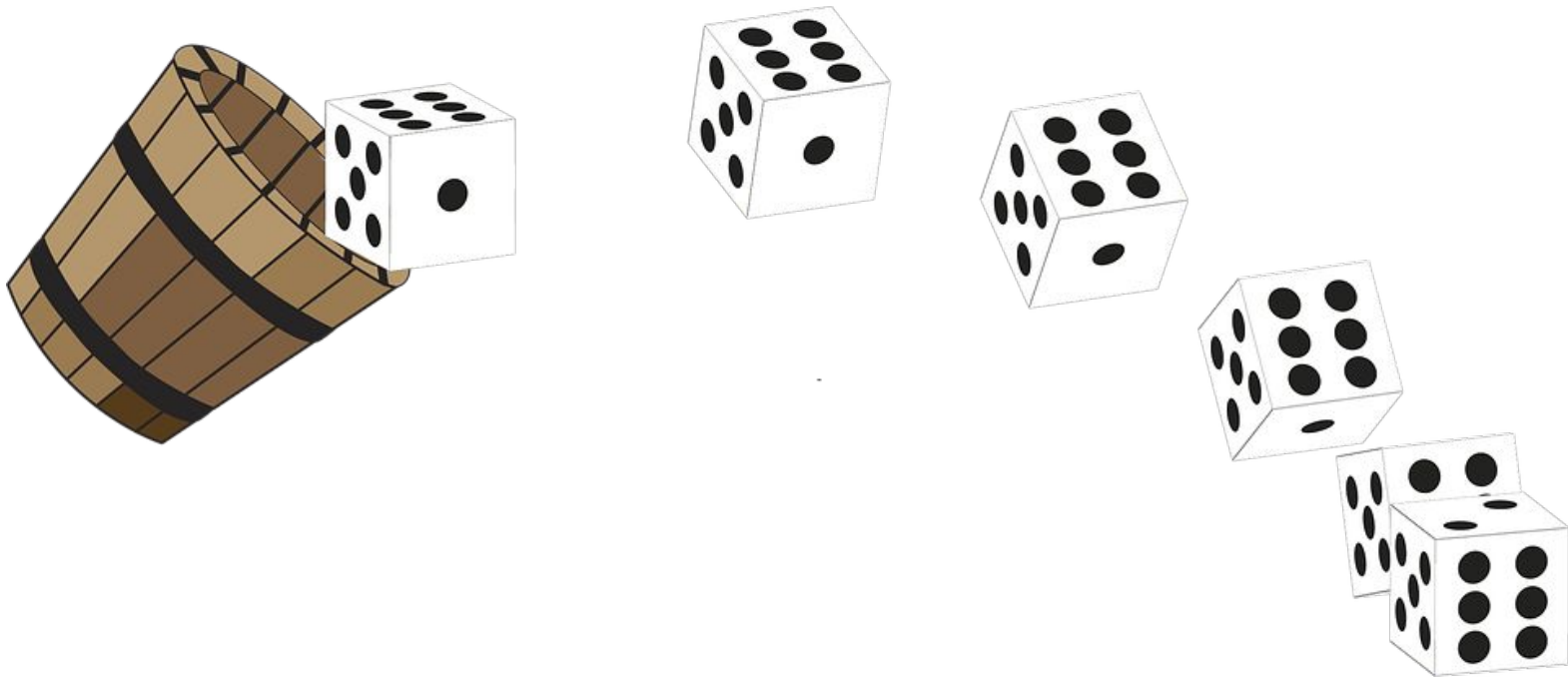
Modificando el Juego



Composición de Objetos

Un Cubilete tiene un conjunto de dados, es decir que se **compone** de dados

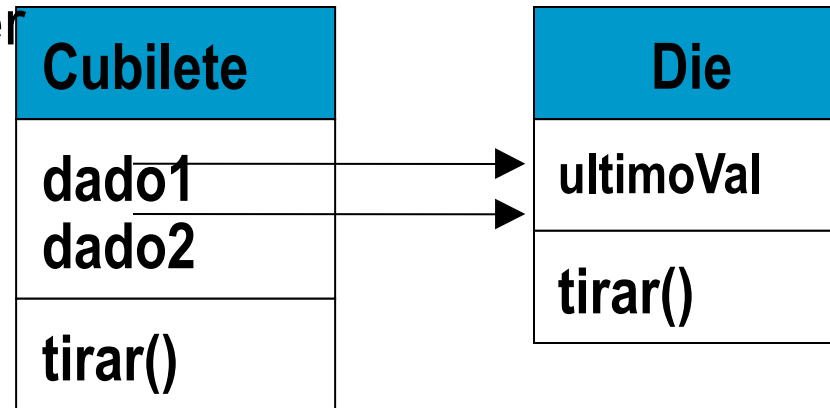
Cuando tiro el cubilete, en realidad tiro todos los dados



Composición de Objetos

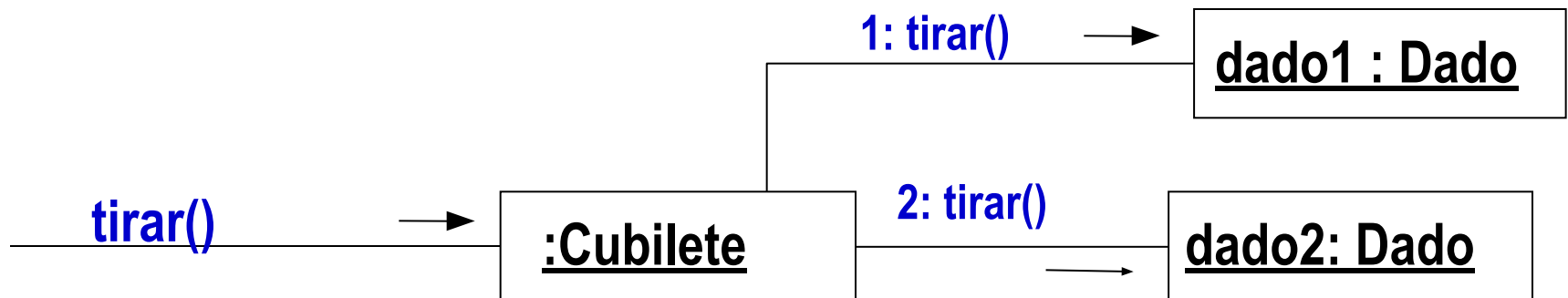
- Los objetos pueden contener o componerse de otros objetos

- La complejidad se reduce
- Los objetos están conectados



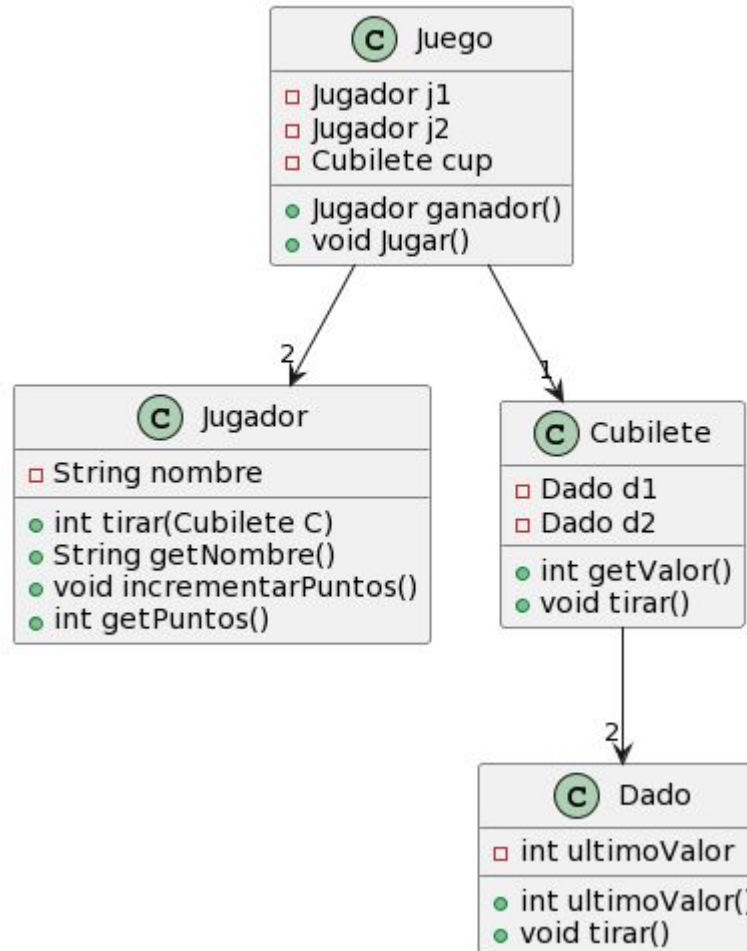
- Interacción / Colaboración

- Los objetos colaboran para resolver tareas
- Un objeto es el emisor del mensaje y el otro el receptor



Solución Final

Classes - Juego Dados



Clases Dado y Cubilete

```
public class Dado{  
    private int ultimoValor;  
  
    public int ultimoValor() {  
        return ultimoValor; }  
  
    public void tirar() {  
        ultimoValor= (int) (Math.random() * 6) + 1; } }
```

```
public class Cubilete{  
    private Dado d1 = new Dado(); private Dado d2 = new Dado();  
  
    public int getValor() {  
        return d1.ultimoValor() + d2.ultimoValor(); }  
  
    public void roll() {  
        d1.tirar(); d2.tirar(); } }
```

Clase Jugador

```
public class Jugador {  
    private String nombre; private int puntos;  
  
    public Jugador(String nom) {  
        this.nombre=nom; puntos=0; }  
  
    public String getNombre() { return nombre; }  
  
    public int getPuntos() { return puntos; }  
  
    public int tirar(Cubilete cup) {  
        cup.tirar();  
        return cup.getValor(); }  
  
    public void incrementarPuntos() { puntos++; } }
```

Clase Juego

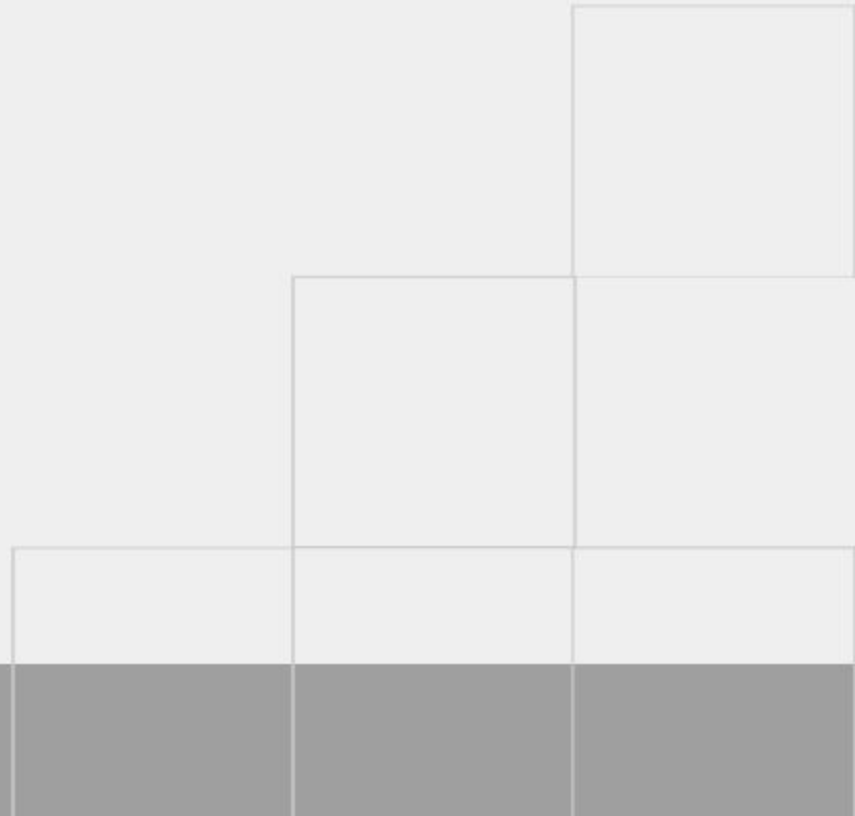
```
public class Juego{  
    private Jugador p1= new Jugador("Jugador 1");  
    private Jugador p2= new Jugador("Jugador 2");  
    private Cubilete cup = new Cubilete();  
  
    public Jugador Ganador() {  
        if (p1.getPuntos() > p2.getPuntos())  
            return p1;  
        else if (p2.getPuntos() > p1.getPuntos()) return p2;  
        else return null;}  
}
```

Clase Juego

```
public void Jugar() {
    for (int i = 0; i < 10; i++) {
        int r1, r2;
        r1= p1.tirar(cup);
        r2= p2.tirar(cup);

        if (r1 >= 7 && r1 > r2) p1.incrementarPuntos();
        else if (r2 >= 7 && r2 > r1) p2.incrementarPuntos();
    }
    if (ganador() != null)
        System.out.println("El Ganador es: " + ganador().getNombre());
    else System.out.println("Empate");
}
public static void main( String args[] ) {
    Juego game=new Juego();
    game.jugar(); }}
```

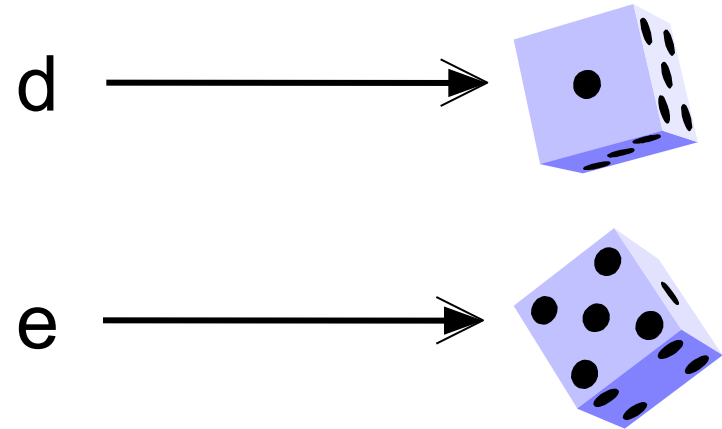
Java



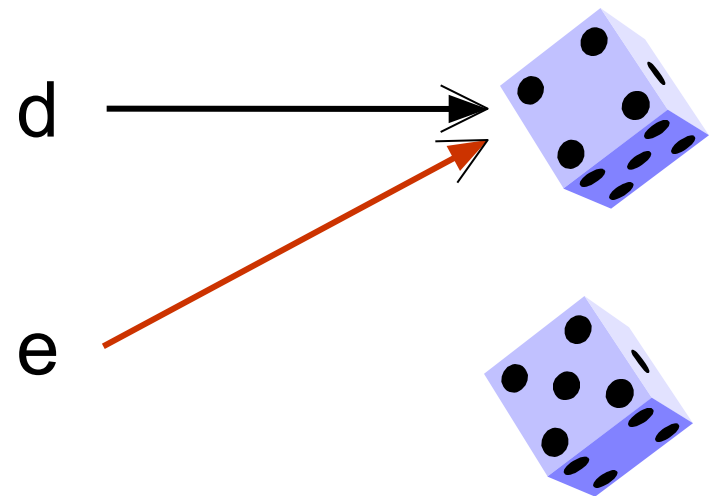
Referencias a Objetos y Asignaciones

- La asignación de objetos copia la referencia (dirección de memoria).

```
Dado d = new Dado();  
Dado e = new Dado();  
d.setValor( 1 );  
e.setValor( 5 );
```



```
e = d;  
e.setValor ( 4 );  
d.ultimoValor();  
    //returns 4
```



El operador ==

“ == ” compara direcciones de memoria, no valores

```
Dado d = new Dado( 5 );  
Dado e = new Dado( 5 );
```

Los dados *d* y *e* tienen un valor 5 de cara

```
if ( e == d )  
    x++;
```

Falso.

== no se fija en los valores de las variables de instancia

```
e = d;
```

```
if ( e == d )  
    y++;
```

Verdadero

d y *e* referencian al mismo objeto

Utilizando “this”

- **this** se refiere al objeto sobre el cual el método fue invocado

```
Dado d1 = new Dado( );  
d1.tirar( );
```

```
public void tirar( )  
{  
    // ...  
    this.valor( val );    // legal, pero no se  
                           utiliza  
    valor( val );         // equivalente; idiomatic  
}
```

- El compilador implícitamente agrega **this** a la invocación de un método cuando no se especifica el receptor del mensaje