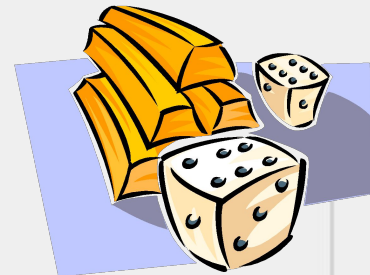


Ejemplo:

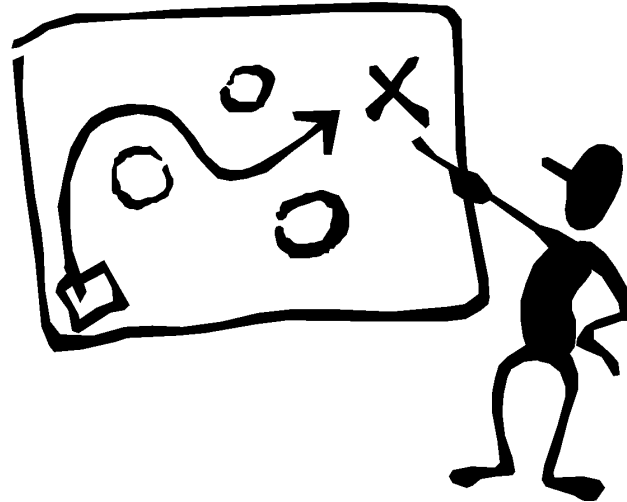
Juego de Dados

DiceGame

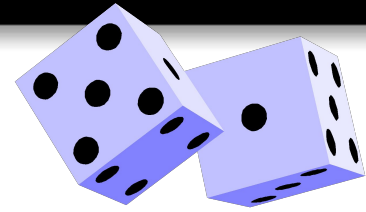


Objetivos

- 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

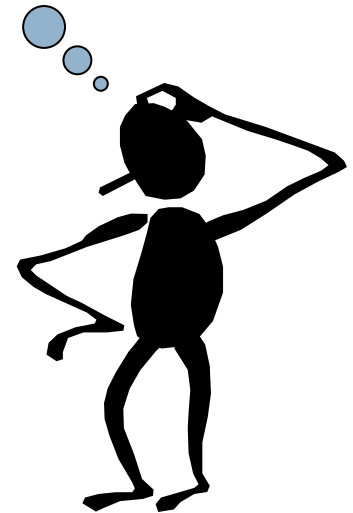


■ Reglas del Juego:

- Cada jugador tira dos dados diez veces.
- El jugador que obtiene la mayor cantidad de tiradas sumando siete o más es el ganador.

**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

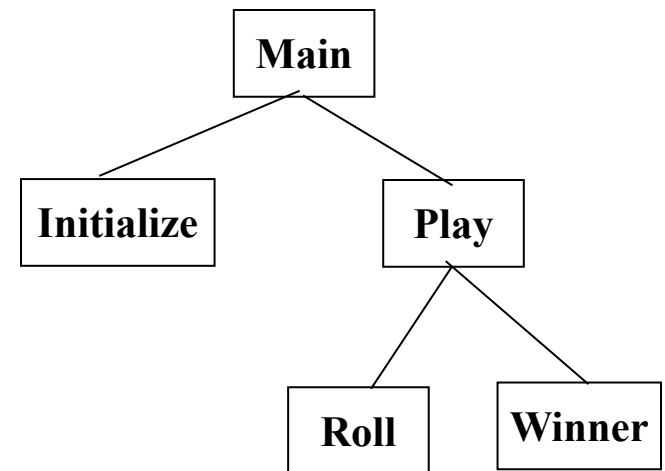
```
public class Juego {  
    int puntos1;  
    int puntos2;  
  
    public Juego() {  
        puntos1=0;  
        puntos2=0;}  
  
    public void play() {  
        for (int i = 0; i < 10; i++)    {  
            int r1, r2;  
            r1= roll() + roll();  
            r2= roll() + roll();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

Alternativa 1

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



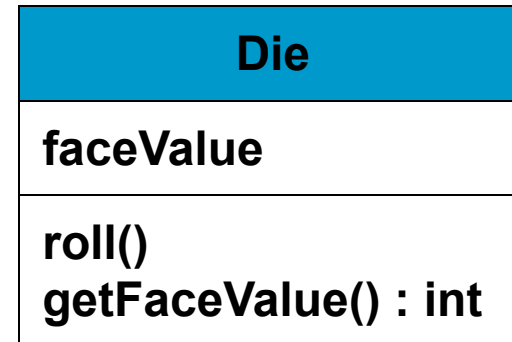
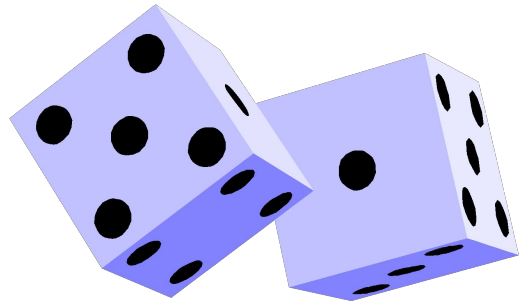
- Todo centralizado
 - Prácticamente una solución procedural
- No se identificaron correctamente los objetos. El modelo dista más de la realidad.
- No se distribuyeron correctamente las responsabilidades.
- Difícil de mantener, modificar, extender.



Que Objetos debería tener el juego?



Alternativa 2: Modelando un Dado como un Objeto



■ Un dado:

- Es responsable de rodar por sí mismo—principio “Lo hago Yo mismo” (roll)
- Es responsable de responder cual es el valor de cara obtenido (getFaceValue)
- Recuerda el valor obtenido (face value).

Clase Dado

```
public class Die {  
    private int faceValue;
```

tipo

acceso

Tipo de Retorno

```
    public int getFaceValue()  
    {  
        return faceValue;  
    }
```

Nombre del Método

Sentencia de Retorno

```
    public void roll()  
    {
```

```
        faceValue = (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 )  
{  
    Die d = new Die( );  
    d.roll( );  
}
```

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

Utilizando dados

```
public class TestDie  
{
```

punto de comienzo de
ejecución de un Programa
Java

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

Se crea una instancia de dado

```
    Die d1 = new Die();
```

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

Se itera 10
veces

```
    {
```

```
        d1.roll();
```

Se le envía el
mensaje "roll" al dado

```
        int value = d1.getFaceValue();
```

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

```
    }
```

```
}
```

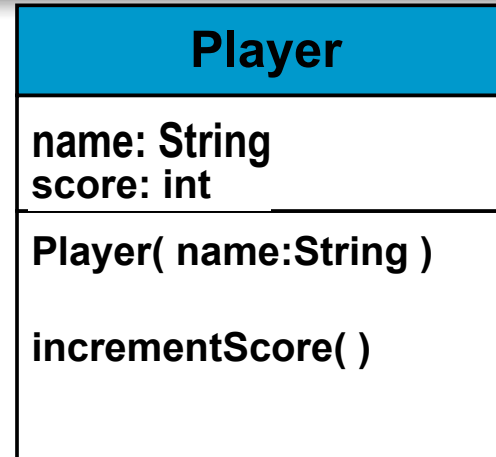
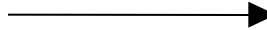
```
}
```

Imprime el
resultado en
Pantalla

Clase Juego

```
public class Juego {  
    int puntos1, puntos2;  
    private Die die1 = new Die();  
    private Die die2 = new Die();  
  
    public void play() {  
        for (int i = 0; i < 10; i++) {  
            int r1, r2;  
            d1.roll(); d2.roll();  
            r1= d1.faceValue() + d2.faceValue();  
            d1.roll(); d2.roll();  
            r2= d1.faceValue() + d2.faceValue();  
            if (r1 >= 7 && r1 > r2) puntos1++;  
            else if (r2 >= 7 && r2 > r1) puntos2++; }  
  
            if (winner() != null) System.out.println("Ganador: "+ winner());  
            else System.out.println("Empate");  
        }  
    }  
}
```

Alternativa 3: Modelando un Jugador como un Objeto



■ Un jugador

- Sabe cuantos puntos tiene y su nombre

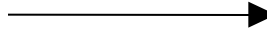
```
public void play() {  
    for (int i = 0; i < 10; i++)    {  
        int r1, r2;  
        d1.roll(); d2.roll();      r1= d1.faceValue() + d2.faceValue();  
        d1.roll(); d2.roll();      r2= d1.faceValue() + d2.faceValue();  
        if (r1 >= 7 && r1 > r2) jug1.incrementScore();  
      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

Que más hace un jugador?



Alternativa 3: Modelando un Jugador como un Objeto II

Player



Player

name: String
score: int

Player(name:String)
takeTurn():int
incrementScore()

■ Un jugador

- Además sabe jugar

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

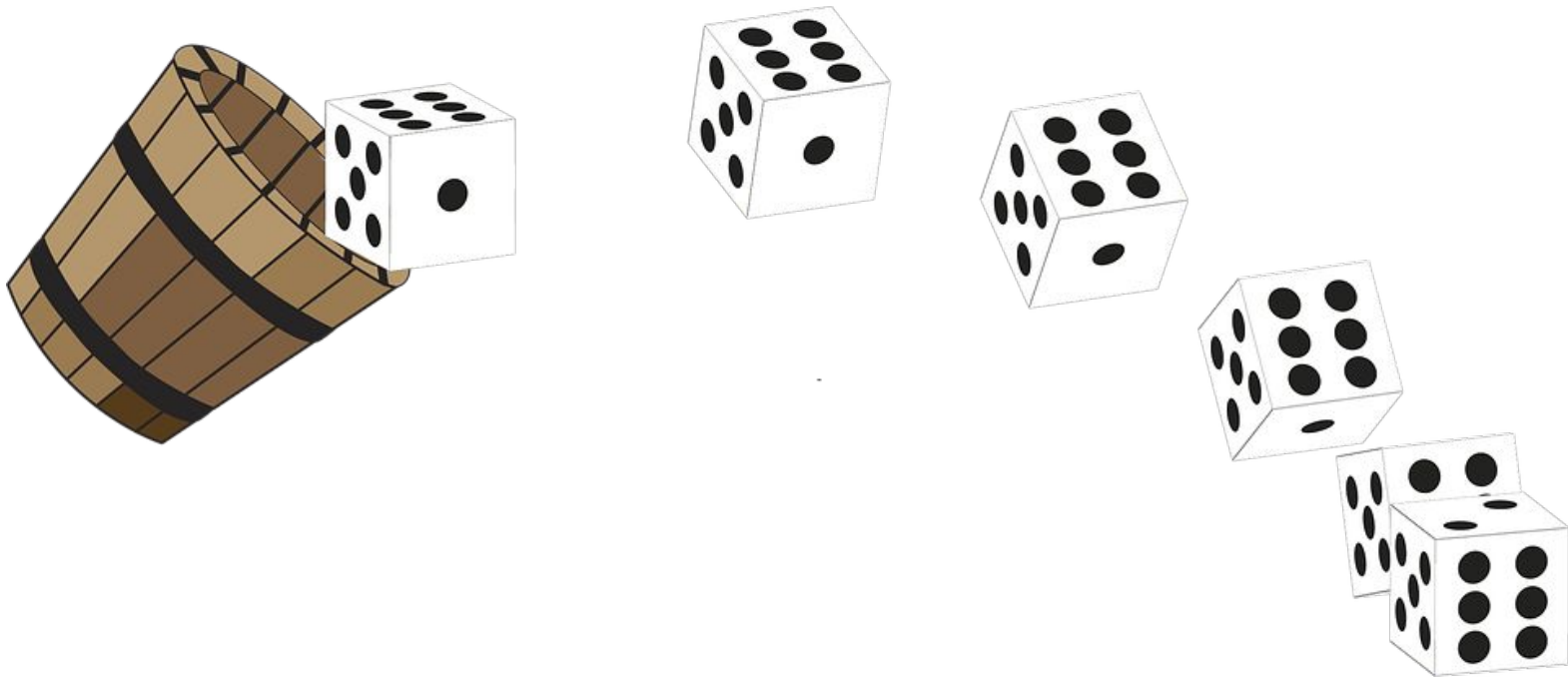
Modificando el Juego



Composición de Objetos

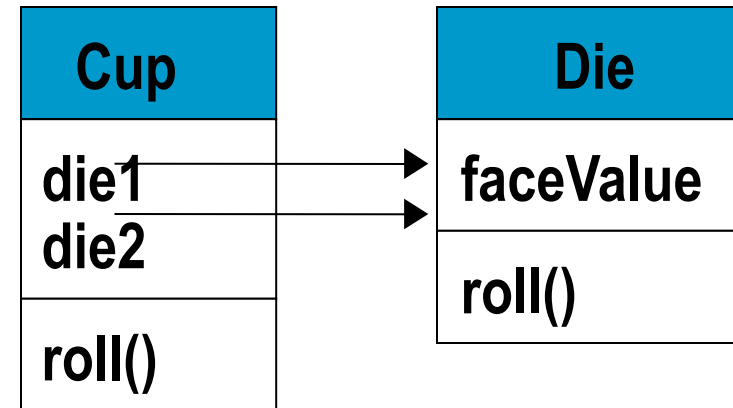
Un Cubilete tiene un conjunto de dados, es decir que se **compone** de dados

Cuando tiro el cubilete, en realidad tiro todos los dados

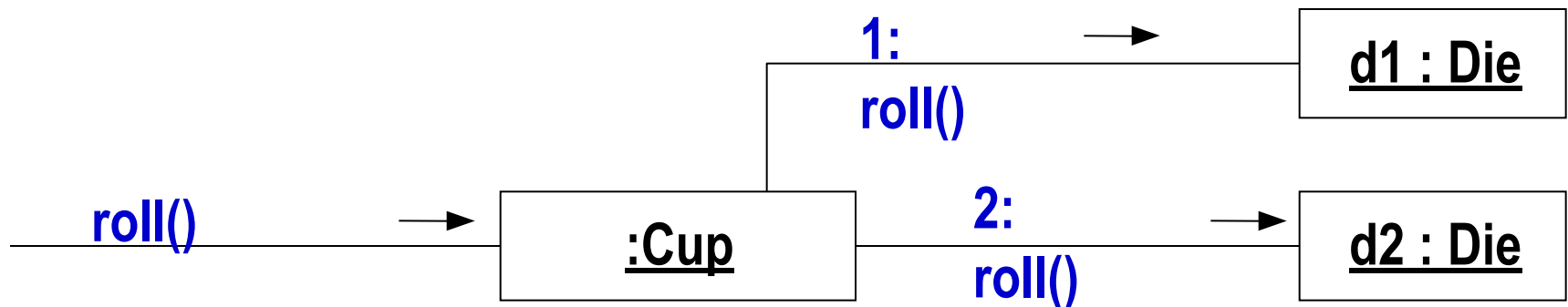


Composición de Objetos

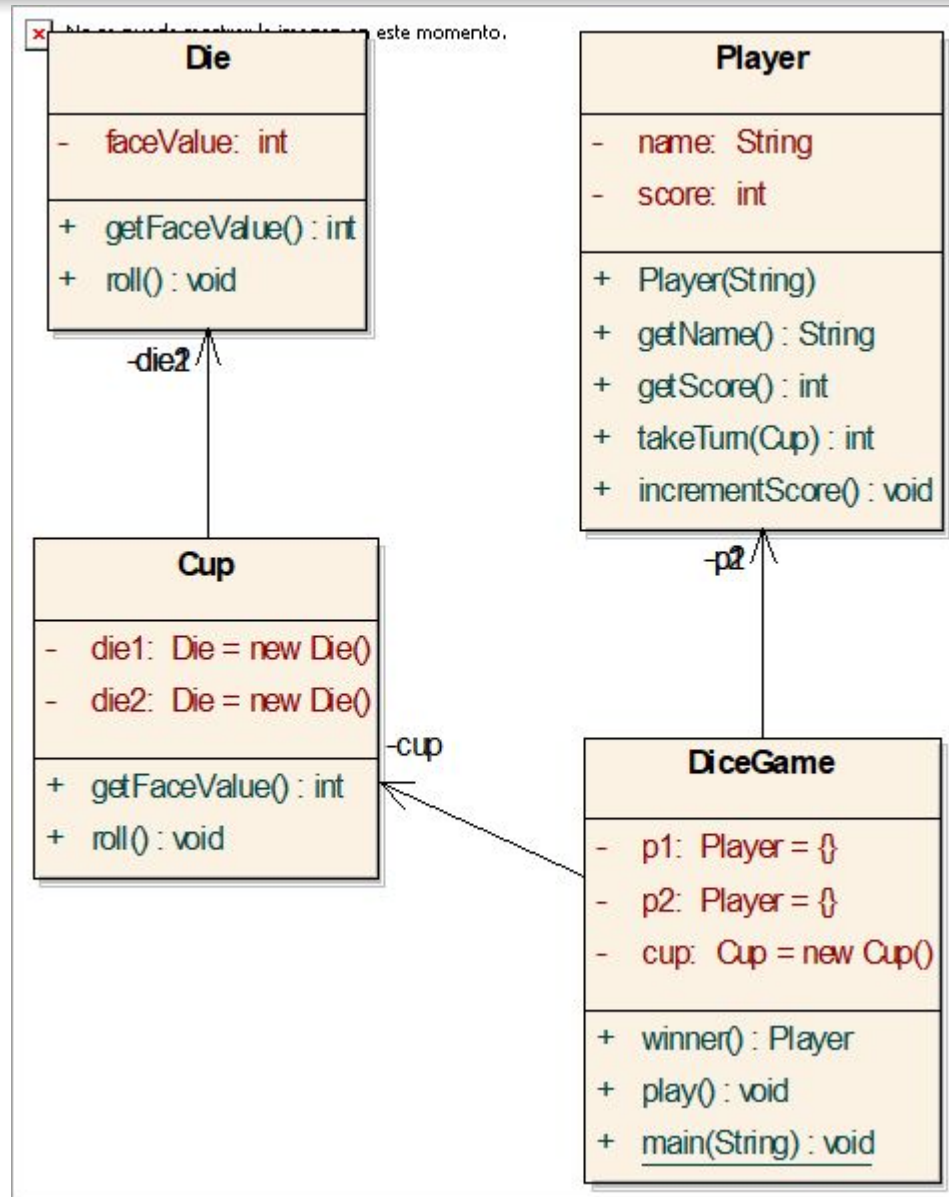
- Los objetos pueden contener o componerse de otros objetos
 - La complejidad se reduce
 - Los objetos están conectados



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



Solucion Final



Clases Dado y Cubilete

```
public class Die {  
    private int faceValue;  
  
    public int getFaceValue() {  
        return faceValue; }  
  
    public void roll() {  
        faceValue = (int) (Math.random() * 6) + 1;} }
```

```
public class Cup {  
    private Die die1 = new Die(); private Die die2 = new Die();  
  
    public int getFaceValue() {  
        return die1.getFaceValue() + die2.getFaceValue(); }  
  
    public void roll() {  
        die1.roll(); die2.roll(); } }
```

Clase Player

```
public class Player {  
    private String name; private int score;  
  
    public Player(String name) {  
        this.name=name;   score=0; }  
  
    public String getName() { return name; }  
  
    public int getScore() { return score; }  
  
    public int takeTurn(Cup cup) {  
        cup.roll();  
        return cup.getFaceValue(); }  
  
    public void incrementScore() {   score++; } }
```

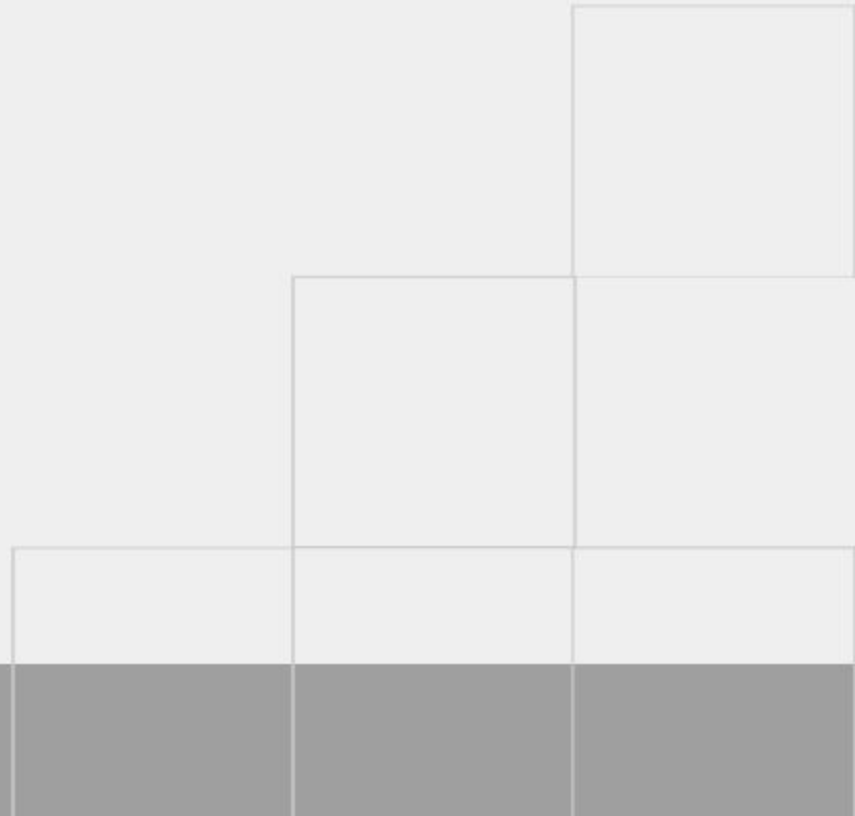
Class DiceGame

```
public class DiceGame {  
    private Player p1= new Player("Jugador 1");  
    private Player p2= new Player("Jugador 2");  
    private Cup cup = new Cup();  
  
    public Player winner() {  
        if (p1.getScore() > p2.getScore())  
            return p1;  
        else if (p2.getScore() > p1.getScore()) return p2;  
        else return null;}  
}
```

Clase DiceGame

```
public void play() {  
    for (int i = 0; i < 10; i++)    {  
        int r1, r2;  
        r1= p1.takeTurn(cup);  
        r2= p2.takeTurn(cup);  
  
        if (r1 >= 7 && r1 > r2) p1.incrementScore();  
        else if (r2 >= 7 && r2 > r1) p2.incrementScore();  
    }  
    if (winner() != null)  
        System.out.println("El Ganador es: " + winner().getName());  
    else System.out.println("Empate");  
}  
public static void main( String args[] ) {  
    DiceGame game=new DiceGame();  
    game.play(); }}
```

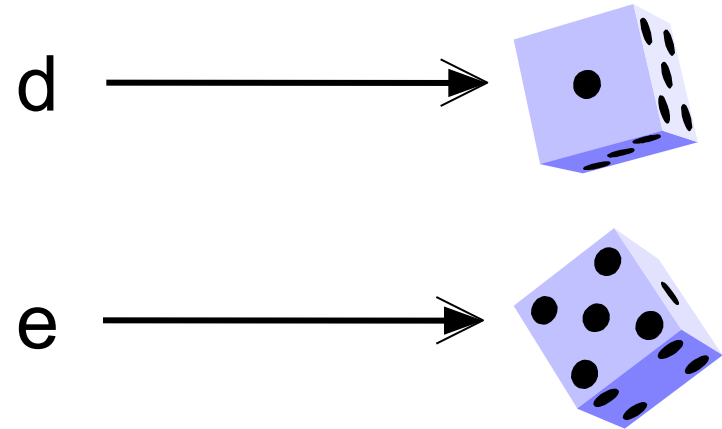
Java



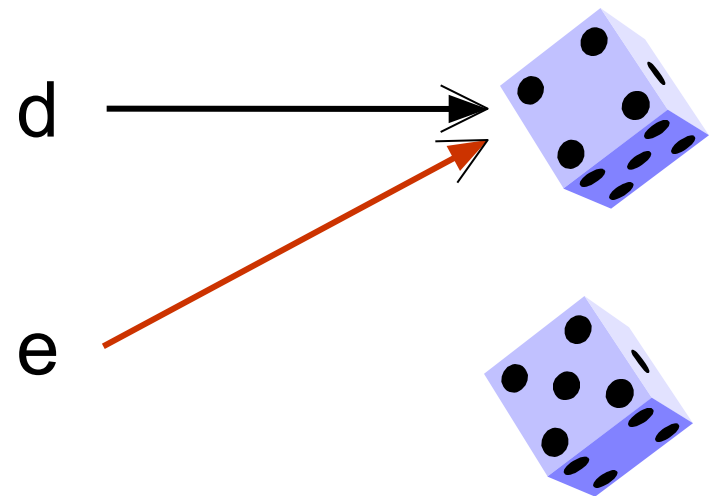
Referencias a Objetos y Asignaciones

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

```
Die d = new Die();  
Die e = new Die();  
d.setFaceValue( 1 );  
e.setFaceValue( 5 );
```



```
e = d;  
e.setFaceValue ( 4 );  
d.getFaceValue();  
    //returns 4
```



El operador ==

“ == ” compara direcciones de memoria, no valores

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

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

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

Falso.

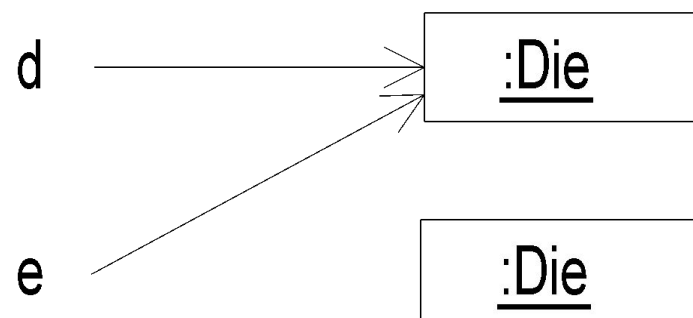
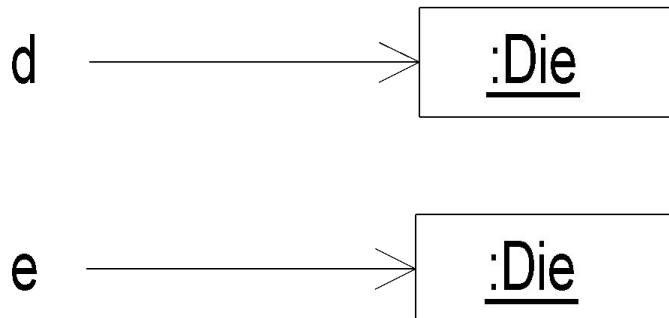
== no se fija en los valores de las variables de instancia

```
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

```
Die d1 = new Die( );  
d1.roll( );
```

```
public void roll( )  
{
```

```
    // ...
```

```
    this.setFaceValue( val );    // legal, pero no se  
                                utiliza
```

```
    setFaceValue( val );        // equivalente; idiomatic  
}
```

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