Universidad Autónoma de Madrid

Escuela Politécnica Superior

Análisis y Diseño de Software (ADSOF)

Curso 2017-2018

Práctica 5

Genericidad, colecciones, lambdas y patrones de diseño

Inicio: semana del 16 de abril

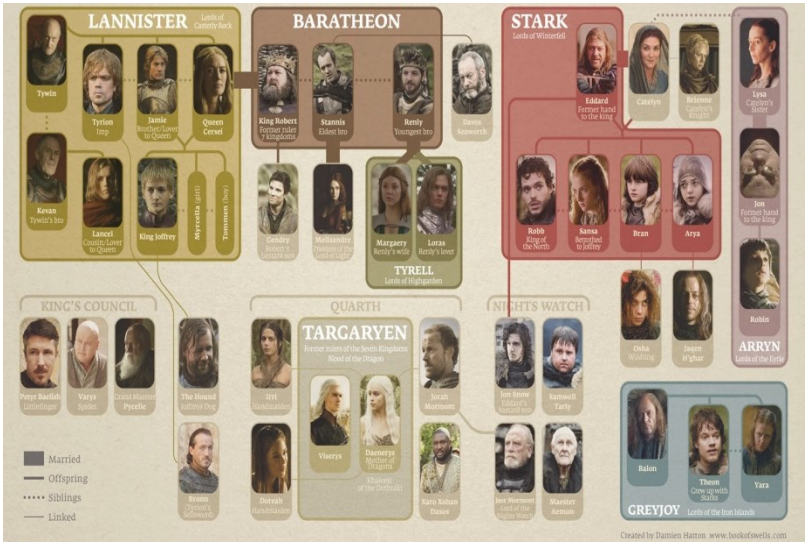
Duración: 3 semanas

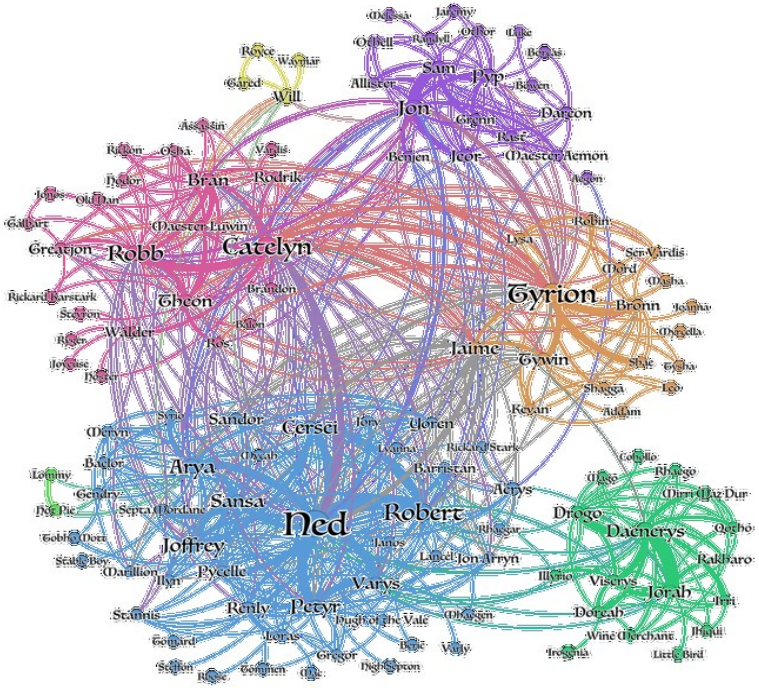
Entrega: lunes 7 de mayo – 8:45h (todos los grupos)

Peso en la calificación de prácticas: 30%

Apartado 2: Expresiones Lambda

- **Grafo de Juego de Tronos *(Games of Thrones, GOT)***
  - Package `adsof1718.grafos.got`
  - Grafo ponderado no dirigido, `GrafoGOT extends GrafoNoDirigido`
  - Vértices:
    - Datos de tipo `PersonajeGOT`, con el nombre y la "casa" de un personaje
  - Constructor:
    - `GrafoGOT(String nombeFicheroVertices, String nombreFicheroArcos)`

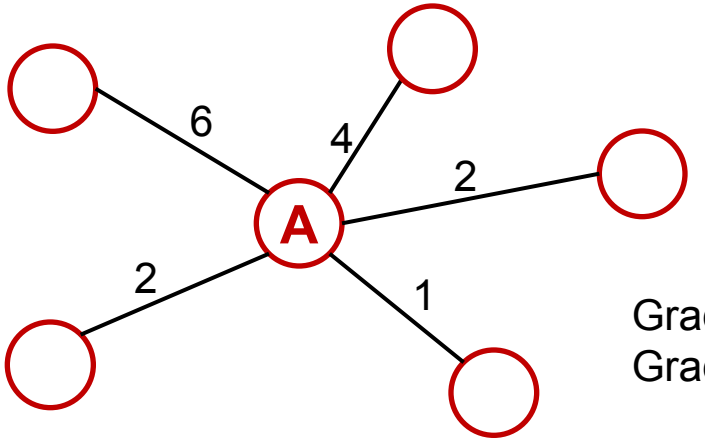




Apartado 2: Expresiones Lambda

- **Métodos a implementar en GrafoGOT**
  - `Vertice<PersonajeGOT> getVertice(String nombre)`
  - `List<String> casas()`
  - `List<String> miembrosCasa(String casa)`
  - `Map<String, Integer> gradoPersonajes()`
  - `Map<String, Integer> gradoPonderadoPersonajes()`
  - `Map<String, Integer> personajesRelevantes()`

```
// aquellos con grado mayor que el grado medio del grafo
```



Grado(A) = 5
GradoPonderado(A) = 6 + 4 + 2 +1 + 2 = 15

- **Restricción: usar expresiones lambda y operaciones sobre los "streams" de vértices**

Apartado 2: Expresiones Lambda

- **Expresión Lambda (Java 8)**
  - Bloque de código sin nombre, formado por:
    - Lista de parámetros
    - Separador "`->`"
    - Cuerpo
  - Ejemplo

```
(int x) -> x + 1
```

  - Parece un método, pero no lo es
  - Es (una notación compacta para) una instancia de una "clase anónima" tipada por una interfaz funcional

Apartado 2: Expresiones Lambda

- **Ejemplo: imprimir por pantalla los elementos de una lista**

```
List<Producto> productos = Arrays.asList(  new Producto(20, "Sal"),
                                           new Producto(40, "Azucar"),
                                           new Producto (5, "Vino"));

Consumer<Producto> consumer = p -> {
    p.incrementaPrecio(10);
    System.out.println(p.getNombre()+": "+p.getPrecio());
};

productos.forEach(consumer);
```

Apartado 2: Expresiones Lambda

- **Ejemplo: imprimir por pantalla los elementos de una lista**

```
List<Producto> productos = Arrays.asList(  new Producto(20, "Sal"),
                                           new Producto(40, "Azucar"),
                                           new Producto (5, "Vino"));

Consumer<Producto> consumer = p -> {
    p.incrementaPrecio(10);
    System.out.println(p.getNombre()+": "+p.getPrecio());
};

productos.forEach(consumer);
```

- **Simplificación del ejemplo anterior**

```
List<Producto> productos = Arrays.asList(  new Producto(20, "Sal"),
                                           new Producto(40, "Azucar"),
                                           new Producto (5, "Vino"));

productos.forEach(p -> {
    p.incrementaPrecio(10);
    System.out.println(p.getNombre()+": "+p.getPrecio());
});
```

Apartado 2: Expresiones Lambda

- **Ejemplo: filtrado de los elementos de una lista**
  - Lista de “productos”

```
class Producto {
  private int precio;
  public Producto(int p) { this.precio = p; }
  public int getPrecio() { return this.precio; }
}
```

```
List<Producto> productos = Arrays.asList(
                    new Producto(20),
                    new Producto(40),
                    new Producto (5));
```

  - Sin expresiones lambda

```
List<Producto> descuentos = new ArrayList<Producto>();
for (Producto p : productos)
    if (p.getPrecio()>10.0)
      descuentos.add(p);
```

Apartado 2: Expresiones Lambda

- **Ejemplo: filtrado de los elementos de una lista**
  - Lista de “productos”

```
class Producto {
  private int precio;
  public Producto(int p) { this.precio = p; }
  public int getPrecio() { return this.precio; }
}
```

```
List<Producto> productos = Arrays.asList(
                    new Producto(20),
                    new Producto(40),
                    new Producto (5));
```

  - Sin expresiones lambda

```
List<Producto> descuentos = new ArrayList<Producto>();
for (Producto p : productos)
    if (p.getPrecio()>10.0)
      descuentos.add(p);
```

  - Con expresiones lambda

```
List<Producto> descuentos = productos.stream().
          filter(p -> p.getPrecio()>10.0).     // filtramos los > 10
          collect(Collectors.toList());        // los ponemos en una lista
```

Apartado 2: Expresiones Lambda

- **Stream**
  - Secuencia de datos que soporta operaciones secuenciales o paralelas de agregación
  - Ejemplo:

```
List<Integer> numbers =
        Arrays.asList(1, 2, 3, 4, 5);

int sum = 0;
for (int n : numbers) {
  if (n % 2 == 1) {
    int square = n * n;
    sum = sum + square;
  }
}

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

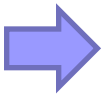
Apartado 2: Expresiones Lambda

- **Stream**
  - Secuencia de datos que soporta operaciones secuenciales o paralelas de agregación
  - Ejemplo:

```
List<Integer> numbers =
        Arrays.asList(1, 2, 3, 4, 5);

int sum = 0;
for (int n : numbers) {
  if (n % 2 == 1) {
    int square = n * n;
    sum = sum + square;
  }
}

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



```
List<Integer> numbers =
     Arrays.asList(1, 2, 3, 4, 5);

int sum = numbers.stream()
          .filter(n -> n % 2 == 1)
          .map(n -> n * n)
          .reduce(0, Integer::sum);

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

Apartado 2: Expresiones Lambda

- **Operaciones sobre streams: ejemplo**

```
int suma = Stream.of(1, 2, 3, 4, 5)
           .peek(e -> System.out.println("Taking integer: " + e))
           .filter(n -> n % 2 == 1)
           .peek(e -> System.out.println("Filtered integer: " + e))
           .map(n -> n * n)
           .peek(e -> System.out.println("Mapped integer: " + e))
           .reduce(0, Integer::sum);

System.out.println("Sum = " + suma);
```

```
Taking integer: 1
Filtered integer: 1
Mapped integer: 1
Taking integer: 2
Taking integer: 3
Filtered integer: 3
Mapped integer: 9
Taking integer: 4
Taking integer: 5
Filtered integer: 5
Mapped integer: 25
Sum = 35
```

Apartado 2: Expresiones Lambda

- **Operaciones sobre streams (I)**

Operación	Tipo	Descripción
distinct	intermedia	Devuelve un stream con los elementos distintos
filter	Intermedia	Devuelve un stream con los elementos que cumplen el predicado
flatMap	Intermedia	Devuelve un stream con el resultado de aplicar una función sobre los elementos del stream. La función produce un stream por cada elemento, que se aplana.
limit	Intermedia	Devuelve un stream de longitud menor o igual que el límite que se le pasa
map	Intermedia	Devuelve un stream con el resultado de aplicar una función sobre los elementos del stream
peek	Intermedia	Devuelve este stream, pero aplica una acción al consumir elementos (útil para debug)
skip	Intermedia	Descarta los n primeros elementos y devuelve el stream con los siguientes.
sorted	intermedia	Devuelve un stream ordenado de acuerdo al orden natural o a un Comparator

Apartado 2: Expresiones Lambda

- **Operaciones sobre streams (II)**

Operación	Tipo	Descripción
allMatch	terminal	Devuelve true si todos los elementos del stream cumplen el predicado
anyMatch	terminal	Devuelve true si algún elemento del stream cumplen el predicado
findAny	terminal	Devuelve elemento del stream. Se devuelve un Optional vacio si el stream está vacio.
findFirst	terminal	Devuelve el primerl elemento del stream (si está desordenado es uno arbitrario)
noneMatch	terminal	Devuelve true si ningún elemento del stream cumple el predicado
forEach	terminal	Aplica una acción a cada elemento del stream
reduce	terminal	Aplica una operación de reducción que calcula un valor único para el stream

Apartado 3: Patrones de diseño

- **Patrón de diseño Observer (I)**
  - Dependencia 1-a-N entre objetos
  - Cuando un objeto cambia de estado, se notifica a los objetos dependientes para que se actualicen automáticamente

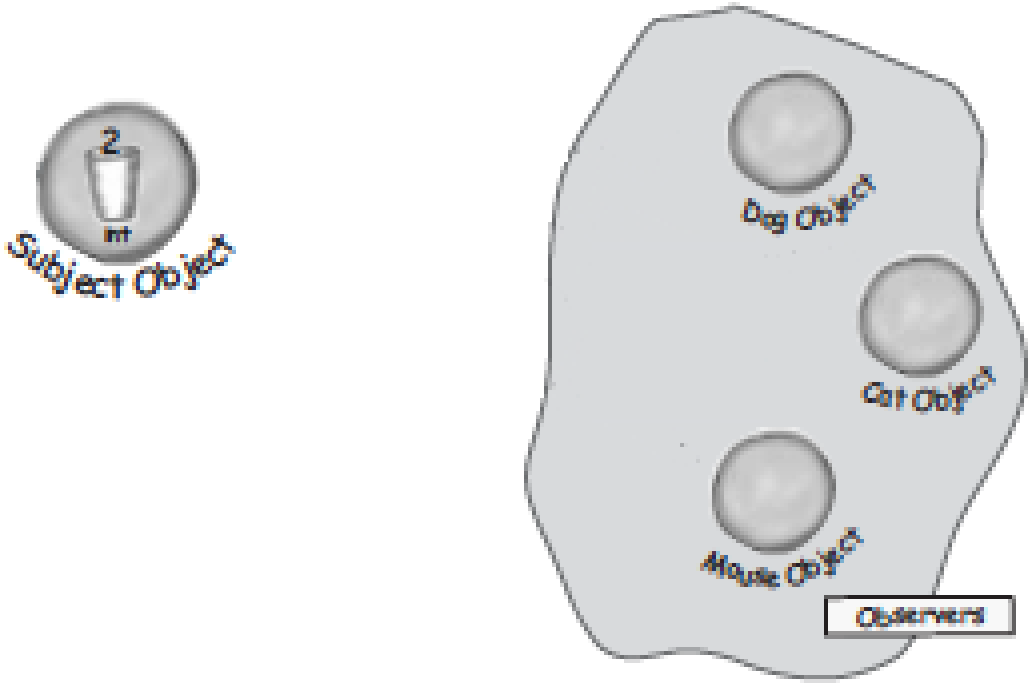


Figura adaptada de Freeman, E., Robson, E., Bates, B., & Sierra, K. (2004). *Head First Design Patterns: A Brain-Friendly Guide*. " O'Reilly Media, Inc.".

Apartado 3: Patrones de diseño

- **Patrón de diseño Observer (I)**
  - Dependencia 1-a-N entre objetos
  - Cuando un objeto cambia de estado, se notifica a los objetos dependientes para que se actualicen automáticamente

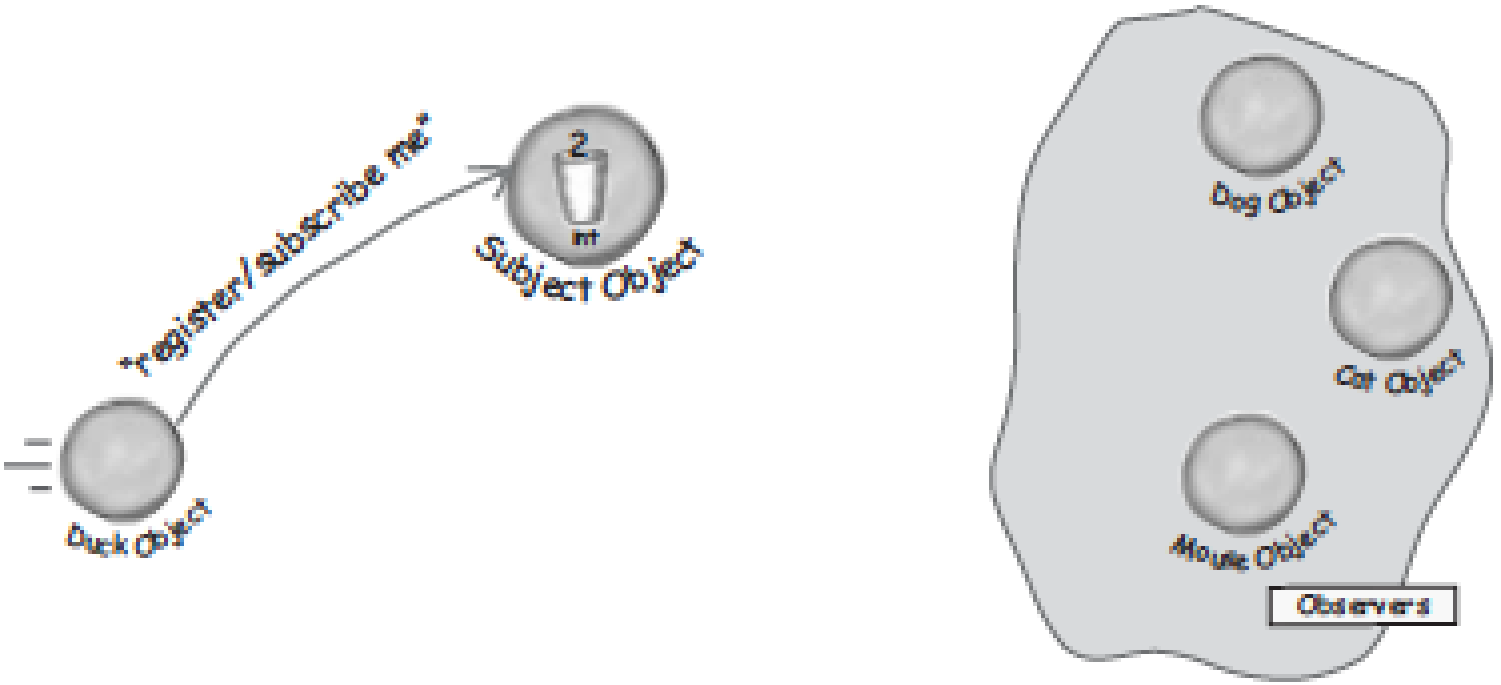


Figura adaptada de Freeman, E., Robson, E., Bates, B., & Sierra, K. (2004). *Head First Design Patterns: A Brain-Friendly Guide*. " O'Reilly Media, Inc.".

Apartado 3: Patrones de diseño

- **Patrón de diseño Observer (I)**
  - Dependencia 1-a-N entre objetos
  - Cuando un objeto cambia de estado, se notifica a los objetos dependientes para que se actualicen automáticamente

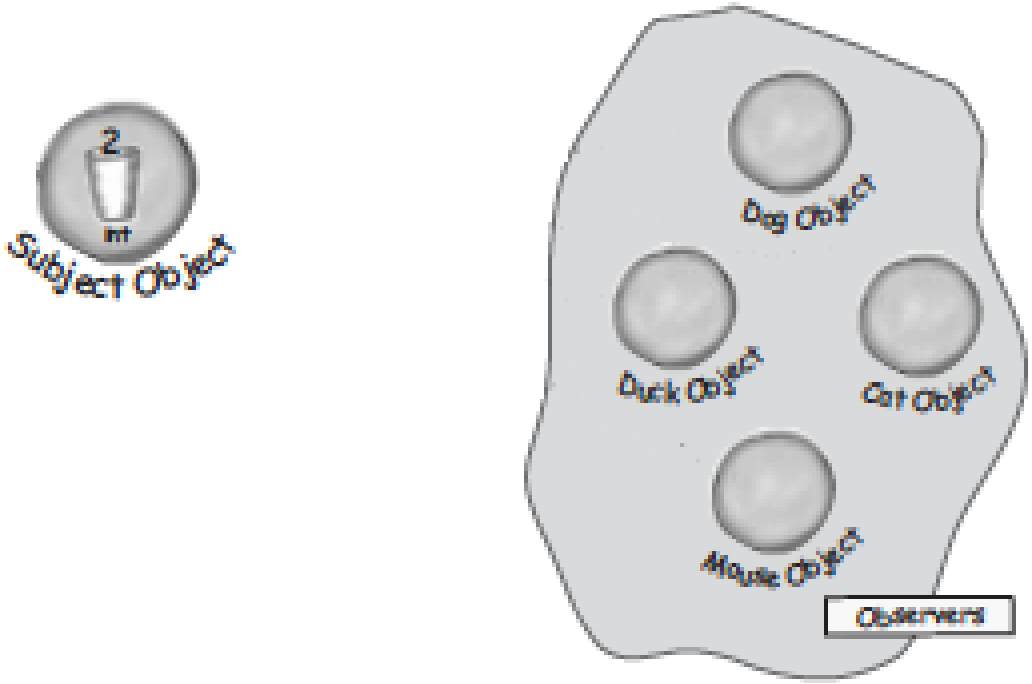


Figura adaptada de Freeman, E., Robson, E., Bates, B., & Sierra, K. (2004). *Head First Design Patterns: A Brain-Friendly Guide*. " O'Reilly Media, Inc.".

Apartado 3: Patrones de diseño

- **Patrón de diseño Observer (I)**
  - Dependencia 1-a-N entre objetos
  - Cuando un objeto cambia de estado, se notifica a los objetos dependientes para que se actualicen automáticamente

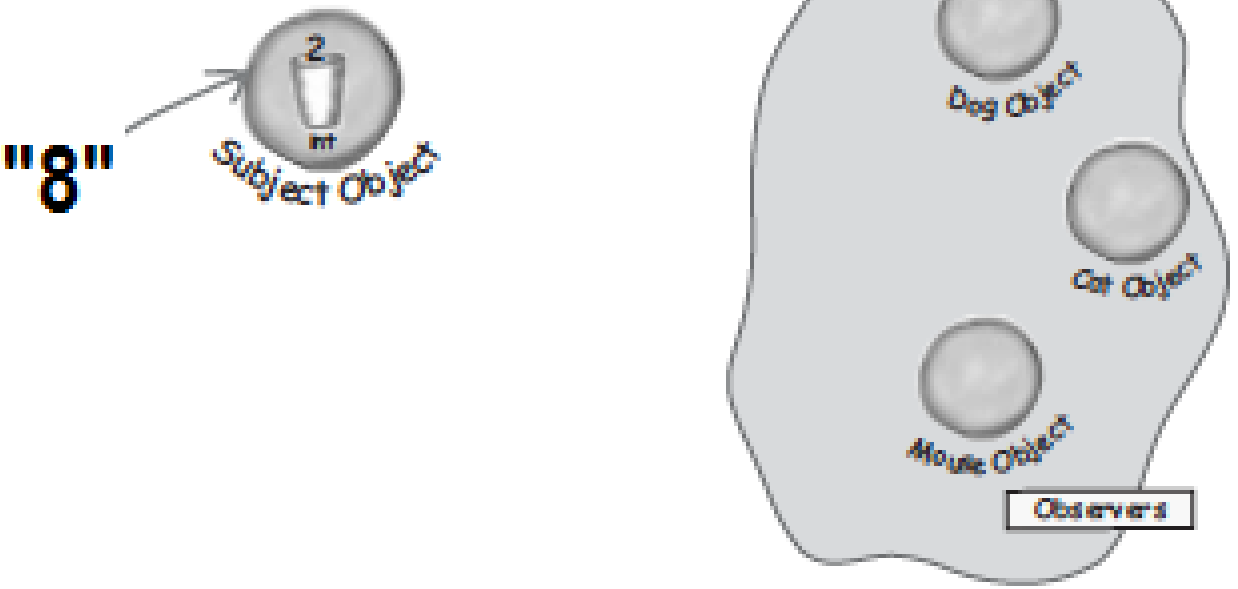


Figura adaptada de Freeman, E., Robson, E., Bates, B., & Sierra, K. (2004). *Head First Design Patterns: A Brain-Friendly Guide*. " O'Reilly Media, Inc.".

Apartado 3: Patrones de diseño

- **Patrón de diseño Observer (I)**
  - Dependencia 1-a-N entre objetos
  - Cuando un objeto cambia de estado, se notifica a los objetos dependientes para que se actualicen automáticamente

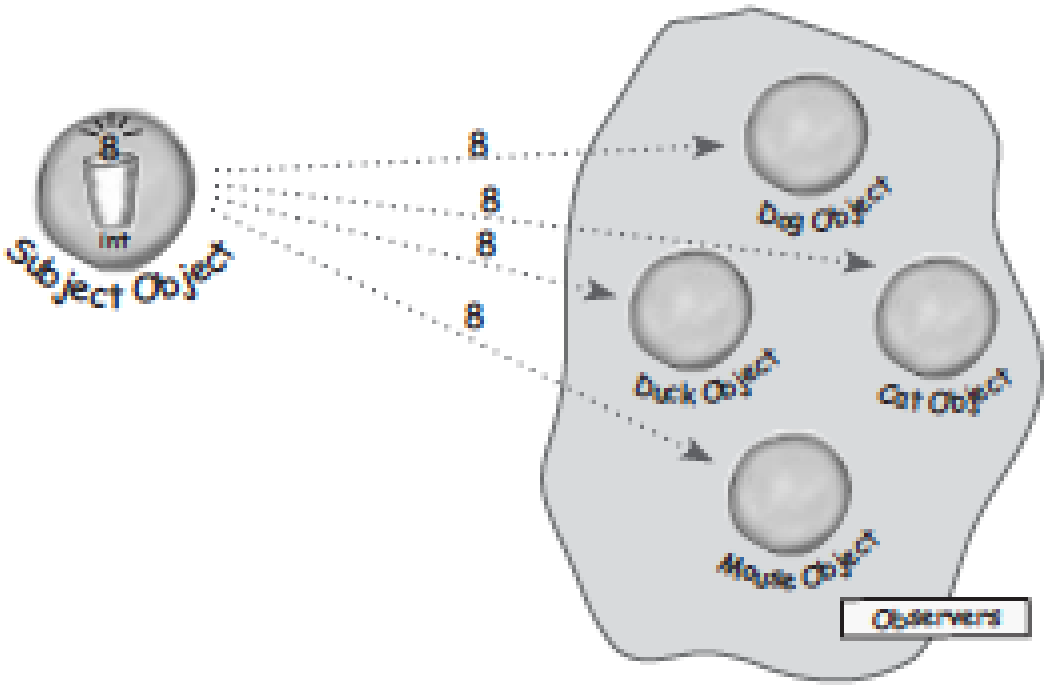


Figura adaptada de Freeman, E., Robson, E., Bates, B., & Sierra, K. (2004). *Head First Design Patterns: A Brain-Friendly Guide*. " O'Reilly Media, Inc.".

Apartado 3: Patrones de diseño

- **Patrón de diseño Observer (I)**
  - Dependencia 1-a-N entre objetos
  - Cuando un objeto cambia de estado, se notifica a los objetos dependientes para que se actualicen automáticamente

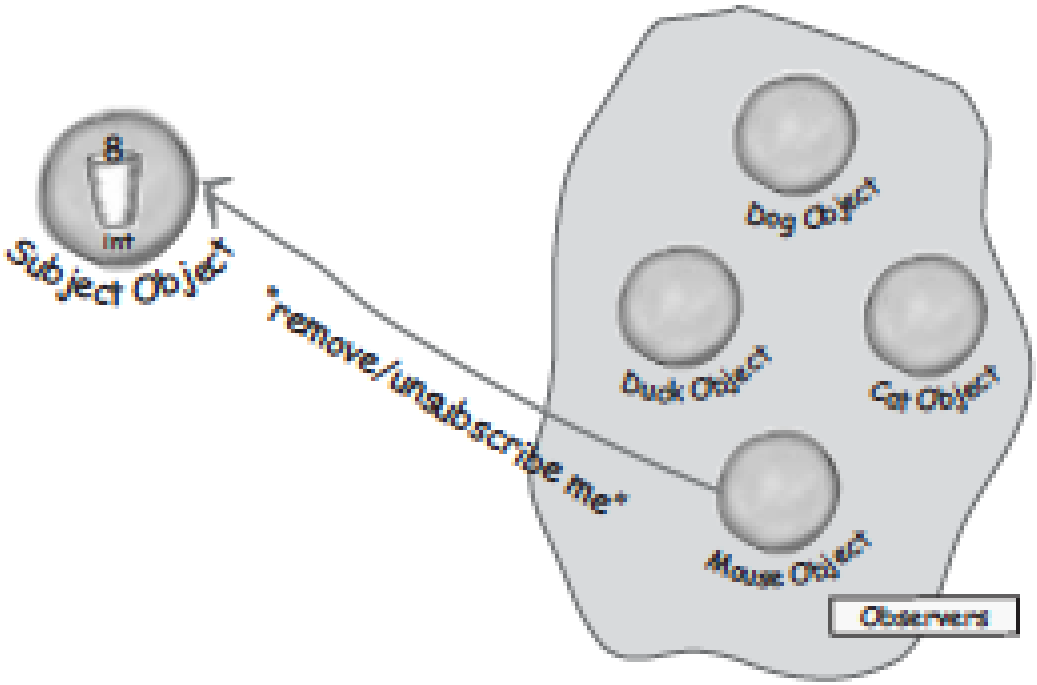


Figura adaptada de Freeman, E., Robson, E., Bates, B., & Sierra, K. (2004). *Head First Design Patterns: A Brain-Friendly Guide*. " O'Reilly Media, Inc.".

Apartado 3: Patrones de diseño

- **Patrón de diseño Observer (I)**
  - Dependencia 1-a-N entre objetos
  - Cuando un objeto cambia de estado, se notifica a los objetos dependientes para que se actualicen automáticamente

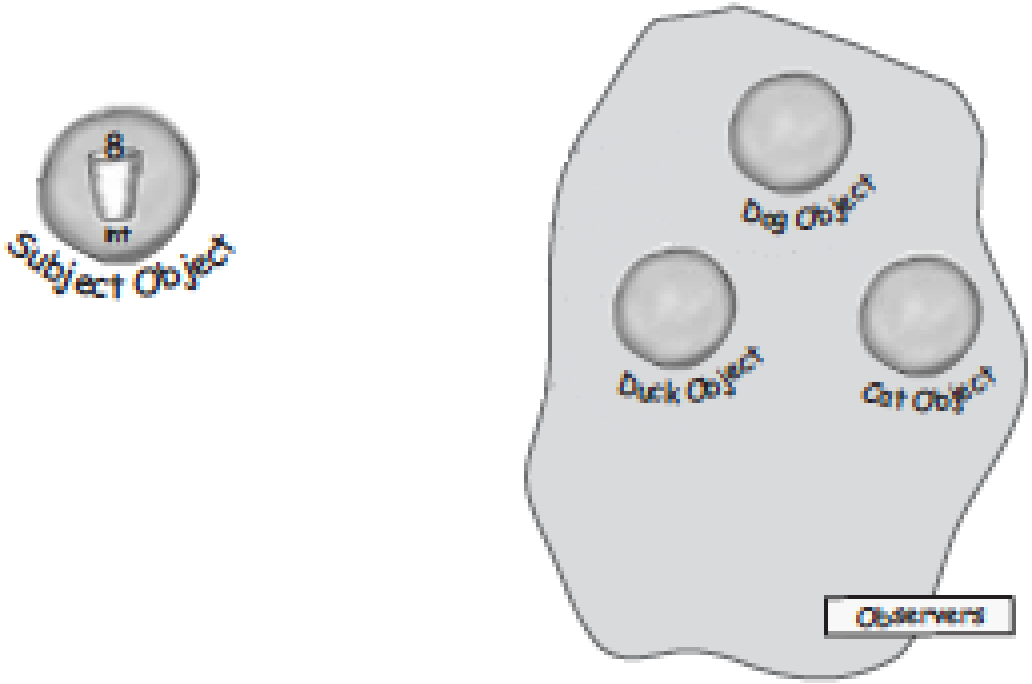
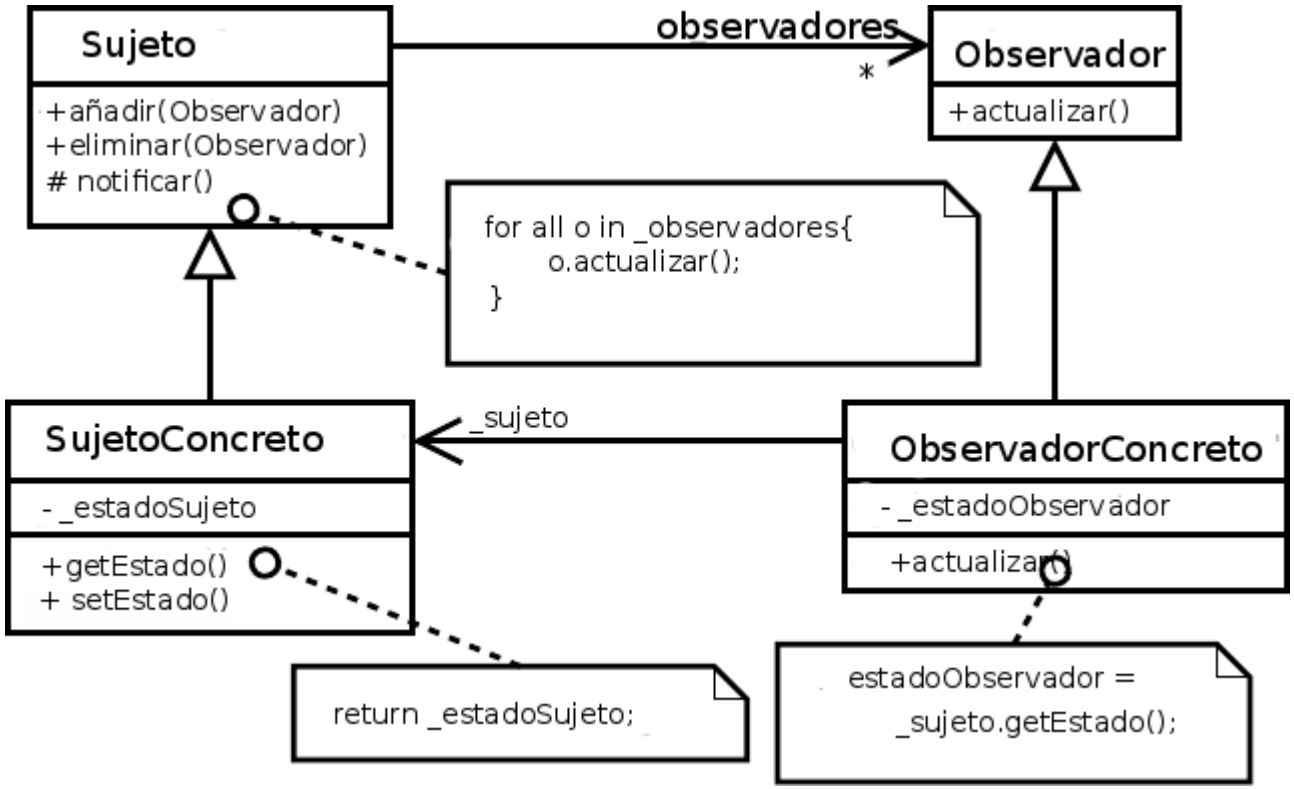


Figura adaptada de Freeman, E., Robson, E., Bates, B., & Sierra, K. (2004). *Head First Design Patterns: A Brain-Friendly Guide*. " O'Reilly Media, Inc.".

Apartado 3: Patrones de diseño

- **Patrón de diseño Observer (II)**
  - Dependencia 1-a-N entre objetos
  - Cuando un objeto cambia de estado, se notifica a los objetos dependientes para que se actualicen automáticamente



Apartado 3: Patrones de diseño

- **En la práctica 5:**
  - Clases (package `adsof1718.grafos.got.simulador`)
    - `SujetoConcreto` → `SimuladorGOT`
    - `ObservadorConcreto` → `ObservadorGOT` (asociado a un personaje)
  - *N (p.e. 10000)* acciones que se envían a `SimuladorGOT` (sujeto), invocando su método "`interaccion`":

    1. Seleccionar un personaje (vértice) u al azar de la red social (grafo).
    2. Para cada vecino v de u:
       - Calcular probabilidad de interacción entre u y v como:
         $p(u,v) = peso(u,v) / \sum_w peso(u,w)$,
         donde w son los "vecinos" de u, y $peso(u,v)$ es el peso del arco que une los vértices u y v.
       - Generar un número aleatorio $r \in [0,1]$.
       - Si $r < p(u,v)$, registrar "interacción" entre u y v.

  - Datos que se registran en un `ObservadorGOT`, invocando a su método "`actualizar`" desde el simulador
    - Para su personaje: número de interacciones total, número de interacciones con miembros de su casa, número de interacciones con cada una las otras casas

Apartado 3: Patrones de diseño

- **En la práctica 5:**
  - Parte de un ejemplo de salida por pantalla:

```
Jon Snow
  Interacciones: 704
    Con miembros de su casa: 78
    Con miembros de casa ajena
      : 410
      Baratheon: 3
      Cassel: 3
      Greyjoy: 6
      Lannister: 5
      Marsh: 28
      Mormont: 24
      Noye: 49
      Rykker: 21
      Targaryen: 20
      Tarly: 37
      Thorne: 20
```