Prácticas POO
Curso 09/10

Alejandro Bellogín

Escuela Politécnica Superior
Universidad Autónoma de Madrid
Marzo 2010

http://www.eps.uam.es/~abellogin

Esquema

- IDE
- Cosas de Java (útiles para esta práctica)
- Organización P2:
  - Semana 1
  - Semana 2
  - Semana 3
  - Semana 4
- Más cosas de Java

IDE

- Deberíais saber
  - Crear proyectos (desde cero y con código)
  - Ejecutar
  - Depurar (breakpoint, watch, variables locales)
  - Generar Javadoc

Más Java

- Útil para esta práctica:
  - Constructores
  - Interfaces / Herencia
  - Control de acceso
  - ArrayList
  - HashMap
  - Enhanced for
  - Hilos
  - Excepciones
  - Entrada / salida

Constructores

- Métodos especiales que crean un objeto
  - Se invoca de manera automática
  - La JVM reserva memoria para ese objeto
  - Se devuelve una referencia a dicho objeto
  - Palabra reservada: *new*
- Inicializan las variables del objeto
- Puede haber más de un constructor
- Sin argumentos: constructor por defecto
  - Definido por defecto (salvo que se definan otros)
- Siempre hay que llamar a algún constructor

Interfaces y Herencia

- Interfaces
  - Imponen un protocolo de métodos a implementar
- Herencia
- Clases abstractas
  - Tipo especial de herencia, donde se definen métodos (que pueden ser llamados) pero no se implementan

```
public interface Dibujo {

    public void resize();
}
```

```
public abstract class Figura {

    public abstract double calculaArea();

    @Override
    public String toString() {
        return "Figura con área " + calculaArea();
    }
}
```

```
public class Circulo extends Figura {

    private double radio;

    public double calculaArea() {
        return Math.PI * radio * radio;
    }
}
```

Control de acceso

- Ocultación de
  - Variables
  - Métodos
  - Constructores

- Todas variables public ==> mal implementado (normalmente)

ArrayList

- Conjunto variable de cualquier tipo de objetos
- Similar a array, pero su capacidad aumenta o disminuye **dinámicamente**

- Desde 1.5: arrays tipados

```
protected   ArrayList<Evento>  eventos;
```

(en tiempo de compilación nos aseguramos el tipo del contenido)

HashMap

- Manera sencilla de tener una tabla (hash)

- Desde 1.5: tablas tipadas

```
private  HashMap<String, Cliente>  clientes;
//  …
clientes.put(cliente.getLogin(), cliente);
```

clientes ==

Login (String)	Cliente (Cliente)
“a”	Alguien{ Login=“a”; nombre=Yo}
…	…

Enhanced for

- Tipos de iteración:
  - Con iterador (clase Iterator)
  - Sin iterador (usando una variable como índice)
  - Enhanced for

```
for (Evento evento : servidor.getEventos(desde, hasta)){
        System.out.println(evento.getNombre());
}
```

Hilos

- El intérprete de Java hace un uso intensivo de hilos.
- Esto provoca situaciones raras:

```
Escribe su dirección:
Escribe su teléfono:
aaa
aaaa
```

- Veremos más cosas en la P3 (GUI)

Excepciones

- Cómo (y cuándo) lanzarlas

```
public void cerrarExpediente() throws IllegalArgumentException {
    if (!esPosibleCerrarExpediente()) {
        throw new IllegalArgumentException("No es posible cerrar el expediente");
    }
```

  - Throws: en la definición del método
  - Throw: dentro del método

- Cómo (y cuándo) tratarlas

```
try {
    cerrarExpediente();
} catch (IllegalArgumentException e) {
    // nada
}
```

  - Try-catch: se capturan las que se quieran, las demás se lanzan

Entrada / salida

- En esta práctica:

```
public static String leeLineaDeTeclado() {
    BufferedReader entrada = new BufferedReader(new InputStreamReader(System.in));
    String s = null;
    try {
        s = entrada.readLine();
    } catch (Exception e) {
    }
    return s;
}
```

```
private static void imprimirMenu() {
    System.out.println("Elige una de estas opciones:");
    System.out.println("0: Salir del sistema");
    System.out.println("1: Crear un alumno");
    System.out.println("2: Matricular a un alumno determinado de ur
    System.out.println("3: Añadir una nota a una asignatura");
    System.out.println("4: Cerrar actas");
    System.out.println("5: Cerrar expediente");
    System.out.println("6: Generar informes de asignaturas (alumnos
    System.out.println("7: Generar informes de asignaturas (resumer
    System.out.println("8: Generar informes de alumnos (expediente
    System.out.println("9: Generar informes de alumnos (expediente
    System.out.println("10: Salvar el estado actual");
    System.out.println("11: Leer un sistema salvado previamente");
}
```

Proyecto

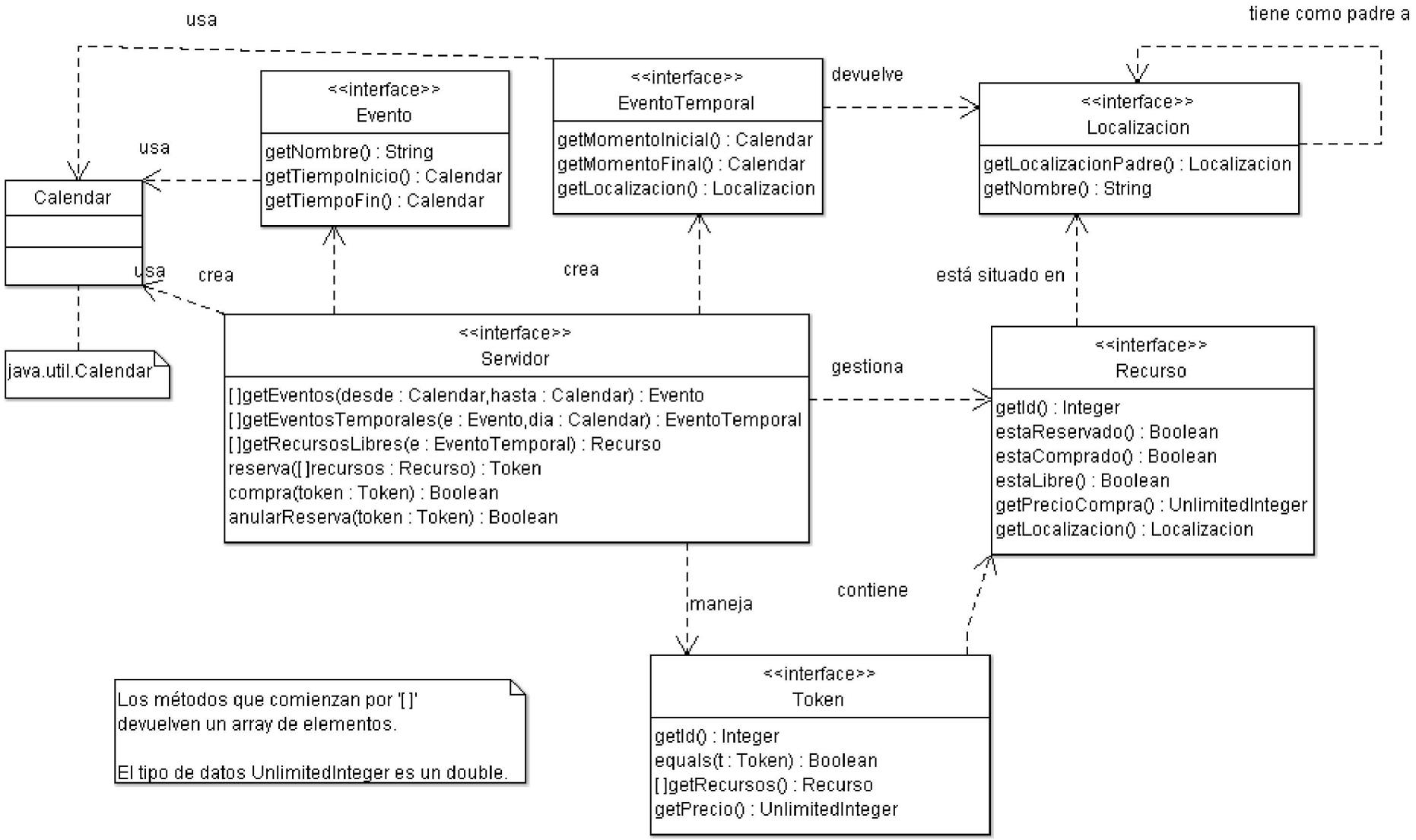
- Aplicación que:
  - Crea eventos y recursos
  - Permite comprar/reservar n recursos

  - Tanto local como remotamente

Práctica 2

- Implementar el diseño de la P1 (corregido / modificado)
- Integrar dicho diseño con las clases que se os ha entregado
- Hacer sistema distribuido

Práctica 2


usa
tiene como padre a
<<interface>>
Evento
getNombre() : String
getTiempoInicio() : Calendar
getTiempoFin() : Calendar
<<interface>>
EventoTemporal
getMomentoInicial() : Calendar
getMomentoFinal() : Calendar
getLocalizacion() : Localizacion
devuelve
<<interface>>
Localizacion
getLocalizacionPadre() : Localizacion
getNombre() : String
usa
Calendar
usa
crea
crea
está situado en
java.util.Calendar
<<interface>>
Servidor
[ ]getEventos(desde : Calendar,hasta : Calendar) : Evento
[ ]getEventosTemporales(e : Evento,dia : Calendar) : EventoTemporal
[ ]getRecursosLibres(e : EventoTemporal) : Recurso
reserva([ ]recursos : Recurso) : Token
compra(token : Token) : Boolean
anularReserva(token : Token) : Boolean
gestiona
<<interface>>
Recurso
getId() : Integer
estaReservado() : Boolean
estaComprado() : Boolean
estaLibre() : Boolean
getPrecioCompra() : UnlimitedInteger
getLocalizacion() : Localizacion
maneja
contiene
<<interface>>
Token
getId() : Integer
equals(t : Token) : Boolean
[ ]getRecursos() : Recurso
getPrecio() : UnlimitedInteger
Los métodos que comienzan por '[ ]'
devuelven un array de elementos.
El tipo de datos UnlimitedInteger es un double.

Semana 1

- Entender el diseño
- Ver qué parte del diseño (métodos, atributos) corresponde con cada funcionalidad
- Implementar funcionalidades, sin pensar en cliente/servidor

Semana 2

- Terminar de implementar
- Comprobar si el resultado permite aislar el cliente del servidor fácilmente (constructores en servidor)
- Hacer pruebas, depurar

Semana 3

- Una vez está implementada la práctica, hacer interfaz de cliente (por consola)

- Extensión del sistema: RMI

Semana 4

- Pruebas en local de RMI, ¿en distintos ordenadores?
- Opcionales
- Mirar la última transparencia (Aún más Java)

(Aún) Más Java

- Patrones de diseño:
  - Singleton
  - Factory
  - ...
- Entrada / Salida alternativa: java.io.Console
- Funciones matemáticas: java.lang.Math