Java

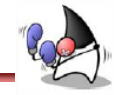
Tema 6: Uso avanzado de clases

Ampliación de clases

Fase II

Contenidos

1. Fechas
2. Sobrecarga (overloading)
3. Paquetes
4. Enumerados



Java

Gestión de fechas en java 8

3

LocalDateTime , LocalDate y LocalTime

- La clase **LocalDate** (fecha), **LocalTime** (hora) y **LocalDateTime** (fecha y hora) son las encargadas de almacenar una fecha y hora.
- Están en el **paquete java.time**;
- *Ejemplo: fecha de hoy (formato de notación internacional estandar ISO 8601 YYYY-MM-DD)*

```
LocalDate hoy= LocalDate.now();
System.out.println( hoy );
```

→ `2016-01-12`

- *Ejemplo: la hora actual (formato ISO 8601 hh:mm:ssZ)*

```
LocalTime ahora= LocalTime.now();
```

→ `18:10:04.004`

- *Ejemplo: la fecha y hora actuales*

```
LocalDateTime instante= LocalDateTime.now();
```

→ `2016-01-12T18:21:09.084`

- *Ejemplo: crear una fecha 02-01-2016*

```
LocalDate fechaTope = LocalDate.of(2016, 01, 02);
LocalDateTime fecha = LocalDateTime.of(fechaTope, ahora);
```

4

Enumerados DayOfWeek y Month

- Enumerado **DayOfWeek,** dia de la semana.
  DayOfWeek lunes = DayOfWeek.**MONDAY**;
- Sumar o restar dias.
  System.out.println("8 días despues será: " + lunes.**plus**(8));
  System.out.println("2 días antes fue: " + lunes.**minus**(2));
- Enumerado **Month,** nombre del mes
  Month enero = Month.**JANUARY**;
  LocalDate fecha = LocalDate.**of**(2016, Month.JULY, 02);
- Sumar o restar meses.
  System.out.println("2 meses despues será: " + enero.**plus**(2));
  System.out.println("1 mes antes fue: " + enero.**minus**(1));

5

Métodos de LocalDate

- **boolean isAfter(LocalDate fecha)** : saber si es posterior a la fecha pasada por parámetro
  if (hoy.**isAfter**(fechaTope)) System.out.println("Has llegado tarde");
- **LocalDate plusMonths(int x)**: devuelve la fecha con X meses más
  LocalDate devolucion = hoy.**plusMonths**(2)
- **LocalDate plusDays(int x)** : devuelve la fecha con X dias más
  LocalDate devolucion = hoy.**plusDays**(10);
  LocalDate devolucion = hoy.**plusMonths**(1).**plusDays**(12);
- **DayOfWeek getDayOfWeek():** dia de la semana de una fecha
  System.out.println("Hoy es: " + hoy.**getDayOfWeek**());

6

Métodos de LocalTime

- ***LocalTime minusMinutes(int x)*** *: devuelve la hora con X minutos menos*
- ***LocalTime plusHours(int x)*** *: devuelve la hora con X horas más*

```
LocalTime finPelicula = ahora.plusHours(2);
```

- ***int getHour()*** *: devuelve la hora*

```
int hora = ahora.getHour();
```


clase DateTimeFormatter

- Clase que sirve para formatear una fecha y/u hora según un patrón.
  - **Definir nuestro patron** con el método **ofPattern()**

```
DateTimeFormatter patron =
                        DateTimeFormatter.ofPattern("dd/MM/yyyy");
```

  - **format() → convertir una fecha LocalDate a String** según un patrón

```
String fecha = hoy.format(patron);
```

  - Contienen **patrones predefinidos** para formatear fechas

```
String fechaFormateada = hoy.format(DateTimeFormatter.ISO_DATE);
```

  - **parse() → Convertir un texto a clase LocalDate** según un patron

```
String fecha = "24/12/2015";
LocalDate fecha2 = LocalDate.parse(fecha , patron);
```

clase DateTimeFormatter

- Carácteres que se pueden utilizar en **ofPattern()**

Símbolo	Significado	Presentacion	Ejemplo
G	Era	Texto	AD
y	Año	Número	2009
M	Mes del año	Texto & Número	July & 07
d	Día del mes	Número	10
h	Hora con formato am/pm (1-12)	Número	12
H	Hora del día (0-23)	Número	0
m	Minuto en hora	Número	30
s	Segundo en hora	Número	55
S	Milisegundo	Número	978
E	Día de la semana	Texto	Tuesday
D	Día del año	Número	189
F	Día de la semana según el mes	Número	2 (2nd Wed in July)
w	Semana del año	Número	27
W	Semana del mes	Número	2
a	Indicador am/pm	Texto	PM
k	Hora del día (1-24)	Número	24
K	Hora en formato am/pm (0-11)	Número	0
z	Zona horaria	Texto	Pacific Standard Time
'	Símbolo de escape	Delimitador	
''	Comilla simple	Literal	'

9

clase Duration

- Clase que representa un **intervalo de tiempo en segundos.**
- Tiene el método static **between**(*Instant* inicio , *Instant* fin) que calcula el tiempo existente entre inicio y fin

```
Instant ahora = Instant.now();
Instant finCurso = Instant.parse("2019-06-21T00:00:00.00Z");

Duration faltan = Duration.between(ahora, finCurso);

System.out.println(faltan);
System.out.println("Faltan " + faltan.toDays() + " dias para fin de curso");
System.out.println("Faltan " + faltan.toHours() + " horas para fin de curso");
```

```
run-single:
PT3726H36M11.5436057S
Faltan 155 dias para fin de curso
Faltan 3726 horas para fin de curso
BUILD SUCCESSFUL (total time: 1 second)
```

10

clase Period

- Clase que representa un **intervalo de tiempo entre dos fechas.** Se almacena como YMD (años, meses, dias)

- Tiene el método static **between**(LocalDate inicio, LocalDate fin) que calcula el tiempo existente entre inicio y fin

```
LocalDate   hoy= LocalDate.now();
LocalDate   finCurso= LocalDate.of(2019,Month.JUNE, 21);

Period  quedan = Period.between(hoy , finCurso);
System.out.println("Quedan: " + quedan);
System.out.print("Faltan para fin de curso ");
…println(quedan.getMonths() + " meses y "+quedan.getDays() + "dias);
```

```
Quedan: P5M5D
Faltan para fin de curso 5 meses y 5 dias
BUILD SUCCESSFUL (total time: 1 second)
```

11

Java

Sobrecarga

(Overloading)

12

Sobrecarga de métodos (overloading)

- En Java es posible definir dos o más métodos que tengan el mismo nombre dentro de una clase, pero con distintos parámetros. En este caso, se dice que los métodos están **sobrecargados**

- Cuando se llama a un método sobrecargado, Java utiliza el tipo y/o número de argumentos para saber qué versión del método se debe llamar.

13

Sobrecarga de métodos

```
public class Matematicas {

    // Sobrecarga del metodo mult() con dos parámetros enteros
     public  int mult ( int a, int b ) {
        return a * b;
    }
    // Sobrecarga del método mult() con dos parámetros double
    public  double mult ( double a,double b ) {
         return a * b;
    }
}
public class DemoSobrecarga {
  public static void main ( String args[ ] ) {
    Matematicas m = new Matematicas();
    System.out.println("Resultado:"+ m.mult(10,20));
    System.out.println("Resultado:"+ m.mult(5.3,6.7));
  }
}
```

14

Sobrecarga de constructores

```
public class Rectangulo extends Figura{
    private double base, altura;

    public Rectangulo (double base, double altura, double x,
  double y){
        super(x,y);
        this.base = base;
        this.altura = altura;
    }

    public Rectangulo (double base, double altura) {
        this.base = base;
        this.altura = altura;
    }

    public Rectangulo( ) { // Rectángulo base y altura 0
     this.altura = 0; this.base = 0;
    }
}
```


Java

Paquetes

Paquetes

- Los paquetes son agrupaciones de clases, enumerados, interfaces y otros paquetes (subpaquetes) relacionados entre sí.
- Evitan solapamiento de nombres de clases.
- Los paquetes Java equivalen a directorios. Es decir, cada miembro del paquete (separado por puntos) se traduce a un directorio en el Sistema de Archivos. Por ejemplo, en windows, un paquete declarado como

```
package net.fpmislata.graficos2d;
```

debe almacenarse en

```
net\fpmislata\graficos2d
```

17

Usando Paquetes

- Para indicar que una clase pertenece a un determinado paquete empleamos la palabra reservada **package**.
- Por ejemplo, para indicar que la clase Circulo pertenece al paquete graficos2d, tenemos que poner en la primera línea no comentada del fichero que contiene la clase Circulo:

```
package net.fpmislata.graficos2d;

class Circulo {
   private double radio;
   public double area(){
          return Math.PI * radio * radio;
   }
}
```

18

Importando Paquetes

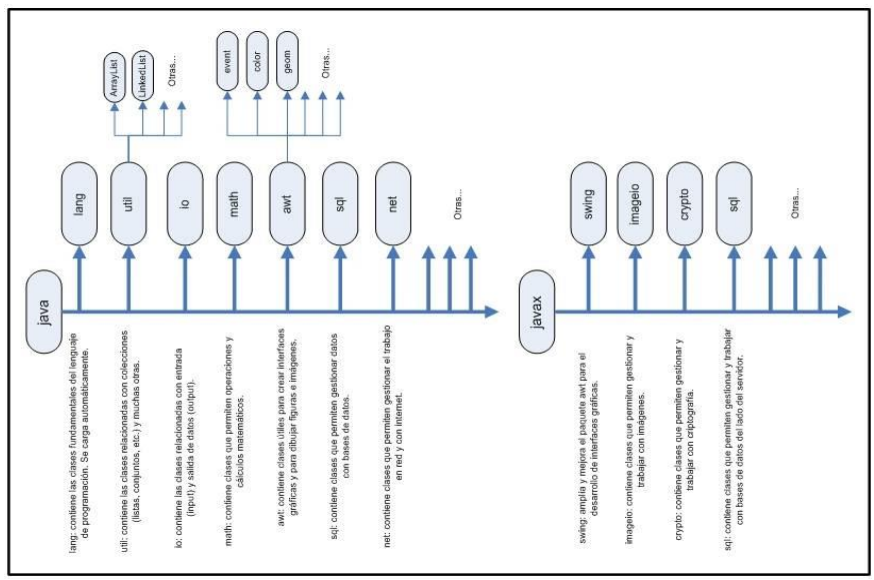
- Para usar la clase Circulo del paquete graficos2d usaremos:

```
import net.fpmislata.graficos2d.Circulo;

Circulo circulo = new Circulo(5.5);
```

19

Paquetes del API de Java



20

Paquetes del API de Java

- **java.applet:** Este paquete contiene clases diseñadas para usarse con applets. Hay la clase Applet y tres interfaces: AppletContext, AppletStub y AudioClip.
- **java.awt** El paquete Abstract Windowing Toolkit (awt) contiene clases para generar widgets y componentes para la Interfaz Gráfico de Usuario (GUI). Incluye las clases Button, Checkbox, Choice, Component, Graphics, Menu, Panel, TextArea, TextField...
- **java.beans**: El paquete que contiene el modelo de componentes JavaBeans para el desarrollo de elementos de software reusables.
- **java.io:** El paquete de entrada/salida contiene las clases de acceso a ficheros: FileInputStream y FileOutputStream.
- **java.lang:** Este paquete incluye las clases del lenguaje Java propiamente dicho: Object, Thread, Exception, System, Integer, Float, Math, String, Package, Process, Runtime, etc.

21

Paquetes de Java

- **java.math:** Este paquete contiene clases para realizar operaciones con números enteros de precisión arbtraria y aritmética de punto flotante.
- **java.net:** Este paquete contiene clases para accesos a traves de internet da soporte a las conexiones del protocolo TCP/IP y, además, incluye las clases Socket, URL y URLConnection.
- **java.security:** Este paquete contiene clases e interfaces que implementan sistemas de autenticación y control de acceso. Soporta el envío y recepción de mensajes encriptados y firmas digitales.
- **java.util:** Este paquete es una miscelánea de clases de utilidades varias. Se incluyen, entre otras, Date (fecha), Dictionary (diccionario), List (lista), Map (mapa), Random (números aleatorios) y Stack (pila FIFO).
- **java.time:** Contiene clases relativas a gestiones de fechas, horas, instantes, periodos, zonas, formatos. Incluye clases como: LocalDate, LocalDateTime, Instant, Duration, Period, DateTImeFormatter

22

Paquetes de Java

- **java.text:** Este paquete da soporte a la internacionalización de textos en aplicaciones Java, proporcionando clases e interfaces que trabajan con cadenas de texto.

- **javax.net:** Este paquete define clases e interfaces para dar soporte a sockets cliente y servidor, que sean de diferente tipo a los que se establecen por defecto. Estas clases permiten comunicación encriptada a través de la red utilizando el protocolo Secure Sockets Layers (SSL).

- **javax.xml:** Este paquete proporciona un API de alto nivel para el análisis de documentos XML utilizando analizadores DOM y SAX. También proporciona un API para la transformación de documentos XML usando el motor XSLT.

Java

Enumerados

Enumerados

- Un *enumerado* es un tipo de datos definido por el programador, que define una lista de constantes con nombre.
- Un *enumerado* se crea usando la palabra clave enum.

```
enum Identificador { valores_separados_por_coma }

enum ColorSemaforo{ROJO, VERDE, AMBAR}
```

  - Los identificadores ROJO, VERDE y AMBAR son *constantes de enumeración*. Cada una que se declara es un elemento publico y estático de ColorSemaforo.
  - El tipo de las constantes de enumeración es del tipo de enumeración en el que están declaradas, en este caso del tipo ColorSemaforo.
  - Se recomienda escribirlas en mayúsculas como valores constantes que son.

25

Enumerados: manejo

- Una vez definido un enumerado se pueden declarar variables objeto de ese tipo, SIN necesidad de utilizar **new**

```
ColorSemaforo  color;
```

- Asignar un valor de la enumeración a esta variable:

```
color =  ColorSemaforo.ROJO;
// devuelve el enumerado que coincide exactamente con el String
color =  ColorSemaforo.valueOf("ROJO");
```

- Comparar dos constantes de enumeración con el operador ==

```
if (color == ColorSemaforo.VERDE){ . . . }
```

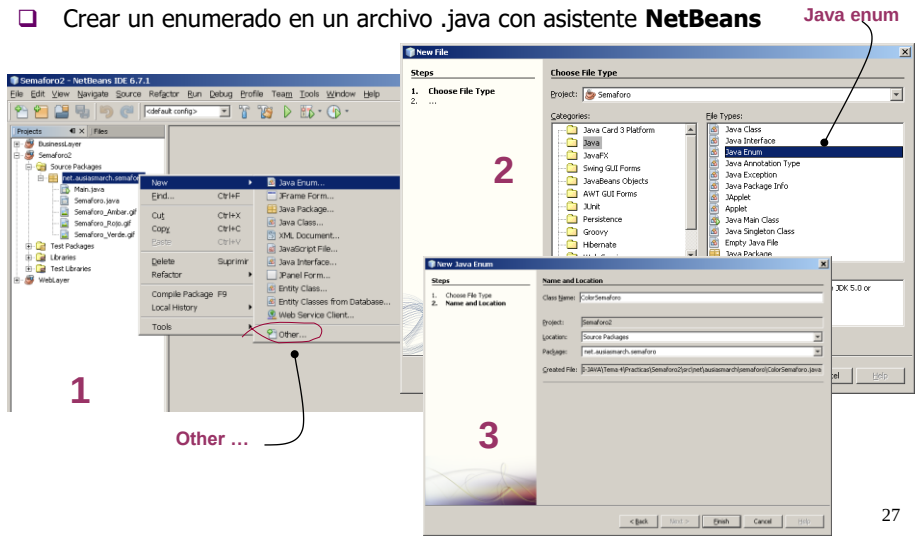
- Recuperar todos los valores del enum:

```
ColorSemaforo[]  colores =  ColorSemaforo.values();
```

26

Enumerados públicos

- Crear un enumerado en un archivo .java con asistente **NetBeans**



27

Enumerados dentro de una clase

- **Otra forma de creación**. Se puede definir el enum dentro del archivo de una clase pero fuera de su definición.

```
enum EstadoTanque1 {
     //El tanque está lleno de Agua
      LLENO,
     //El tanque está vacío de Agua
      VACIO,
    //El tanque está a la mitad de su capacidad
      MEDIO
  }

public class Clase1{
    ……...
}
```

28

Enumerados dentro de una clase

- **Otra forma de creación**. También se puede definir el enum dentro de la definición de una clase. Su acceso entonces será siempre a través de esa clase.

```
public class Clase1{
  …….
  enum EstadoTanque2 {
     //El tanque está lleno de Agua
      LLENO,
     //El tanque está vacío de Agua
      VACIO,
    //El tanque está a la mitad de su capacidad
      MEDIO
  }
  ……….
}
```


Enumerados

- Asignando valores a un enumerado definido dentro de una clase, y comparándolos

```
EstadoTanque1          estadoTanque1;
Clase1.EstadoTanque2   estadoTanque2;


estadoTanque1 = EstadoTanque.LLENO;
estadoTanque2 = Clase1.EstadoTanque.VACIO;

if (estadoTanque1 == estadoTanque2) {
   System.out.println("mismo estado");
}
```