

Sentencias de Control

if y switch

Fundamentos de Programación
Departamento de Lenguajes y Sistemas Informáticos



2024/25

Sentencias de Control

if-else

Ejecuta las sentencias del primer bloque de sentencias si la *expresión* evalúa a **true** y el segundo bloque si evalúa a **false**

El segundo bloque se puede omitir, en cuyo caso, si la *expresión* evalúa a **false** no se ejecuta sentencia alguna.

```
if (expresión) {  
    sentencia11;  
    sentencia12;  
    ...  
}  
else {  
    sentencia21;  
    sentencia22;  
    ...  
}
```

Ejemplo:

```
int a=1,b=0,m=9;  
...  
if (a>b) {  
    a++;  
    b=7;  
}  
else {  
    texto="mi texto";  
    m=25;  
}
```





2024/25

Sentencias de Control

switch-case

Ejecuta las sentencias inmediatamente siguientes a los dos puntos (:) cuyo “*valor-n*” coincida con el valor de la *expresión*, hasta encontrar **break**.

Si no empareja se ejecutan la sentencia a continuación de **default**. Si se omite esta última etiqueta y ningún valor-n coincide con el de la *expresión*, no se ejecuta ninguna sentencia.

La *expresión* debe ser int, String o enumerado



```
switch (expresión) {  
  case valor1:sentencia11;  
    sentencia12;  
    ...  
  break;  
  case valor2:sentencia21;  
    sentencia22;  
    ...  
  break;  
  ...  
  default:  sentenciad1;  
    sentenciad2;  
    ...  
}
```

Ejemplo1:

```
int a=1,b=0;  
switch (a+1) {  
  case 1: System.out.println("vale 1");  
    b=7;  
    break;  
  case 2: System.out.println("vale 2");  
    break;  
  default: System.out.println("Otro valor");  
}
```



2024/25

Sentencias de Control

switch-case

Ejecuta las sentencias inmediatamente siguientes a los dos puntos (:) cuyo “*valor-n*” coincida con el valor de la *expresión*, hasta encontrar **break**.

Si no empareja se ejecutan la sentencia a continuación de **default**. Si se omite esta última etiqueta y ningún valor-n coincide con el de la *expresión*, no se ejecuta ninguna sentencia.

La *expresión* debe ser int, String o enumerado

```
switch (expresión) {  
    case valor1:sentencia11;  
        sentencia12;  
        ...  
    break;  
    case valor2:sentencia21;  
        sentencia22;  
        ...  
    break;  
    ...  
    default:  sentenciad1;  
        sentenciad2;  
        ...  
}
```

Ejemplo1:

```
int a=1,b=0;  
switch (a+1) {  
    case 1: System.out.println("vale 1");  
        b=7;  
        break;  
    case 2: System.out.println("vale 2");  
        break;  
    default: System.out.println("Otro valor");  
}
```



2024/25

Sentencias de Control

switch-case

Ejecuta las sentencias inmediatamente siguientes a los dos puntos (:) cuyo “*valor-n*” coincida con el valor de la *expresión*, hasta encontrar **break**.

Si no empareja se ejecutan la sentencia a continuación de **default**. Si se omite esta última etiqueta y ningún valor-n coincide con el de la *expresión*, no se ejecuta ninguna sentencia.

La *expresión* debe ser int, String o enumerado

```
switch (expresión) {  
  case valor1:sentencia11;  
                sentencia12;  
                ...  
  break;  
  case valor2:sentencia21;  
                sentencia22;  
                ...  
  break;  
  ...  
  default:  sentenciad1;  
            sentenciad2;  
            ...  
}
```


Ejemplo2: (*Color es un tipo enumerado*)

```
Color color=...;  
...  
switch (color) {  
  case ROJO: System.out.println("Parar");  
                break;  
  case AMARILLO: System.out.println("Reducir");  
                break;  
  case VERDE: System.out.println("Pasar");  
}
```



Tipo String

métodos más habituales



Fundamentos de Programación
Departamento de Lenguajes y Sistemas Informáticos



2024/25

Trabajando con String

Métodos habituales para el manejo de String

```
String s1="Fundamentos";  
String s2=" de ";  
String s3="Programación"
```

*char cad.***charAt(p):**

Devuelve el carácter que ocupa la posición “*p*” de *cad*. (recordad: las posiciones empiezan en cero)

Ejemplo: `s1.charAt(3)` → `'d'`

*String cad.***concat(cad2):**

Concatena a la cadena *cad* la cadena *cad2*.

Ejemplo: `s1.concat(s2).concat(s3)` → `"Fundamentos de Programación"`

*boolean cad.***contains(cad2):**

Devuelve `true` o `false`, según *cad* contenga o no a *cad2* (mayúsculas != minúsculas)

Ejemplo: `s1.contains("men")` → `true`
`s1.contains("Men")` → `false`



2024/25

Trabajando con String

Métodos habituales para el manejo de String

```
String s1="Fundamentos";  
String s2=" de ";  
String s3="Programación"
```

*boolean cad.***endsWith(cad2):**

Devuelve `true` o `false`, según `cad` termine, o no, con `cad2`

Ejemplo: `s1.endsWith("tos")` → `true`
`s1.endsWith("na")` → `false`

*int cad.***indexOf(cad2):**

Devuelve la posición de `cad` en la que encuentra por primera vez la `cad2`. Si no la encuentra devuelve -1

Ejemplo: `s1.indexOf("dam")` → `3`
`s1.indexOf("ana")` → `-1`

*int cad.***lastIndexOf(cad2):**

Devuelve la posición de `cad` en la que encuentra por primera vez la `cad2` buscando desde el final. Si no la encuentra devuelve -1

Ejemplo: `s1.lastIndexOf("ent")` → `6`
`s1.lastIndexOf("ana")` → `-1`



2024/25

Trabajando con String

Métodos habituales para el manejo de String

```
String s1="Fundamentos";  
String s2=" de ";  
String s3="Programación"
```

*String cad.***length():**

Devuelve la longitud (el número de caracteres) de *cad*.

Ejemplo: `s1.length()` → `11`

*String cad.***replace(*cad1*, *cad2*):**

Reemplaza en *cad* todas las ocurrencias de *cad1* por *cad2* y devuelve *cad* modificada.

Ejemplo: `s1.replace("n", "123")` → `"Fu123dame123tos"`
`s1.replace("damentos", "ción")` → `"Función"`

*String cad.***trim():** 

Devuelve *cad* quitándole los espacios en blanco del principio y del final.

Ejemplo: `s2.trim()` → `"de"`



2024/25

Trabajando con String

Métodos habituales para el manejo de String

```
String s1="Fundamentos";  
String s2=" de ";  
String s3="Programación"
```

*String cad.***substring(p1,p2):** 

Devuelve la subcadena de *cad* desde la posición "*p1*" hasta la posición "*p2*" (excluida esta última). Si se omite "*p2*", llega hasta el final de la cadena.

Ejemplo: *s1.substring(4,8)* → "amen"
s1.substring(4) → "amento"

*String cad.***toUpperCase():**

Devuelve la cadena *cad* con todos sus caracteres en mayúsculas

Ejemplo: *s1.toUpperCase()* → "FUNDAMENTOS"

*String cad.***toLowerCase():**

Devuelve la cadena *cad* con todos sus caracteres en mayúsculas

Ejemplo: *s3.toLowerCase()* → "programación"

FECHAS, HORAS Y DURACIONES

LocalDate, Localtime, LocalDateTime, DateTimeFormatter,
ChronoUnit, Duration



Fundamentos de Programación
Departamento de Lenguajes y Sistemas Informáticos



2024/25

Trabajando con Fechas y Horas

Tipos relacionados con fechas, horas y duraciones

- `java.time.LocalDate` (para manejar *fechas*)
- `java.time.LocalTime` (para manejar *horas*)
- `java.time.LocalDateTime` (para manejar *fechas* y *horas*)
- `java.time.format.DateTimeFormatter` (para “*formatear*” fechas y/u horas)
- `java.time.temporal.ChronoUnit` (para indicar *unidades* temporales)
- `java.time.Duration` (para manejar *duraciones* -espacios de tiempo-)



LocalDate

2024/25

LocalDate

Sirve para trabajar con objetos tipo *fechas*

Se construyen con métodos estáticos e inician de la siguiente manera:

```
LocalDate f = LocalDate.now (); //la fecha actual (hoy)
```

```
LocalDate f = LocalDate.of (2021,2,26); // año, mes y día
```

```
LocalDate f = LocalDate.parse ("2014-09-03"); // año, mes y día
```

parse con un solo parámetro espera una fecha con el formato "yyyy-MM-dd"

Se puede modificar con un patrón (dos ejemplos):

```
LocalDate f = LocalDate.parse ("20/2/2024", DateTimeFormatter.ofPattern("d/M/y"));
```

```
LocalDate f = LocalDate.parse ("23/9-58", DateTimeFormatter.ofPattern("d/M-yy"));
```





LocalTime

Sirve para representar **horas**.

Se construyen e inician de la siguiente manera:

LocalTime h = *LocalTime.now*(); // la actual

LocalTime h = *LocalTime.of*(16,55); // hora y minutos

LocalTime h = *LocalTime.of*(14,59,45); // hora, minutos y segundos

LocalTime h = *LocalTime.parse*("14:09:05"); // hora, minutos y segundos

parse con un solo parámetro espera una hora con el formato "HH:mm:ss"

Se puede modificar con un patrón (*dos ejemplos*):

LocalTime h = *LocalTime.parse*("17:9:5", DateTimeFormatter.*ofPattern*("H:m:s"));

LocalTime h = *LocalTime.parse*("14/59-5", DateTimeFormatter.*ofPattern*("H/m-s"));





2024/25

LocalDateTime

LocalDateTime

Sirve para representar fechas y, horas conjuntamente.

Se construyen e inicializan de la siguiente manera:

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

```
LocalDateTime fH=LocalDateTime.of (2014,9,23,11,00)
```

```
LocalDateTime fH=LocalDateTime.of (2014,9,23,10,4,11);
```

```
LocalDateTime fH=LocalDateTime.parse ("2014-09-03T02:04:01");
```



parse con un solo parámetro espera fecha y hora con el formato “yyyy-MM-ddT HH:mm:ss”

Se puede modificar con un patrón (dos ejemplos):

```
LocalDateTime fH=LocalDateTime.parse("3/9/2014 2:4:1",  
DateTimeFormatter.ofPattern("d/M/y H:m:s"))
```

```
LocalDateTime fH=LocalDateTime.parse("3/9-2014#2/4&1",  
DateTimeFormatter.ofPattern("d/M-y#H/m&s"))
```



2024/25

LocalDate, LocalTime y LocalDateTime

Algunos métodos de estos tipos:

- isBefore/isAfter/isEquals
- getDayOfWeek/getMonth/getMonthValue/getYear
- getHour/getMinute/getSecond
- minusHour/minusSeconds (para restar)
- minusYears/minusMonths/minusDays (para restar)
- plusHour/plusSeconds (para sumar)
- plusYears/plusMonths/plusDays (para sumar)
- minus o plus(cantidad, **ChronoUnit.DAYS**) (para restar o sumar)
- until(fecha, **ChronoUnit.DAYS**) (devuelve un periodo de tiempo)

Tipo enumerado



2024/25

Duration

Duration

Sirve para representar duraciones (espacios de tiempo)

Se construyen e inicializan de la siguiente manera:

Duration d1 = *Duration.ofMillis* (60000);

Duration d2 = *Duration.ofSeconds* (60);

Duration d3 = *Duration.ofMinutes* (1);

Duration d4 = *Duration.ofHours* (20);

Duration d5 = *Duration.ofDays* (5);

Duration d6 = *Duration.of* (5, *ChronoUnit.DAYS*); ← **Tipo enumerado**

Duration d7 = *Duration.parse* ("cadena");

donde **cadena** puede ser → "PnDTnHnMn.nS"

Ejemplos: "PT60S" "PT1M" "PT20H"

"P5D" "PT2H4M" "P2DT4M"

"P5DT20H4M60S", ...





Duration


2024/25

Una vez construida una Duración

```
Duration d = Duration.parse("P5DT27H62M58S");
```

Observar que hay:

5 días = 120 horas

- 27 horas = 1 día + 4 horas y a su vez, 62 minutos = 1 hora + 2 minutos
- System.out.println(d); → PT148H2M58S
- System.out.println(d.toDays()); → 6
- System.out.println(d.toHours()); → 148
- System.out.println(d.toHoursPart()); → 4 
- System.out.println(d.toMinutes()); → 8882
- System.out.println(d.toMinutesPart()); → 2
- System.out.println(d.toSeconds()); → 532978
- System.out.println(d.toSecondsPart()); → 58



Duration

2024/25

Una vez construida una Duración

```
Duration d = Duration.parse("P5DT27H62M58S");
```

Observar que hay:

5 días = 120 horas

- 27 horas = 1 día + 4 horas y a su vez, 62 minutos = 1 hora + 2 minutos
- `System.out.println(d);` → **PT148H2M58S**
- `System.out.println(d.toDays());` → **6**
- `System.out.println(d.toHours());` → **148**
- `System.out.println(d.toHoursPart());` → **4**
- `System.out.println(d.toMinutes());` → **8882**
- `System.out.println(d.toMinutesPart());` → **2**
- `System.out.println(d.toSeconds());` → **532978**
- `System.out.println(d.toSecondsPart());` → **58**



2024/25

Ejercicio. Aeropuerto

Ejercicio:

Realizar el enunciado Aeropuerto 01

Muy importante: Este proyecto va a ir creciendo a lo largo de cuatrimestre teniendo más de 10 entregas y servirá de base para ir practicando los conceptos de teoría.

Hay que llevarlo “al día”