Programación Orientada a Objetos

Tema 2:
Sintaxis del lenguaje POO Java

Tema 3-2: Clases básicas de Java

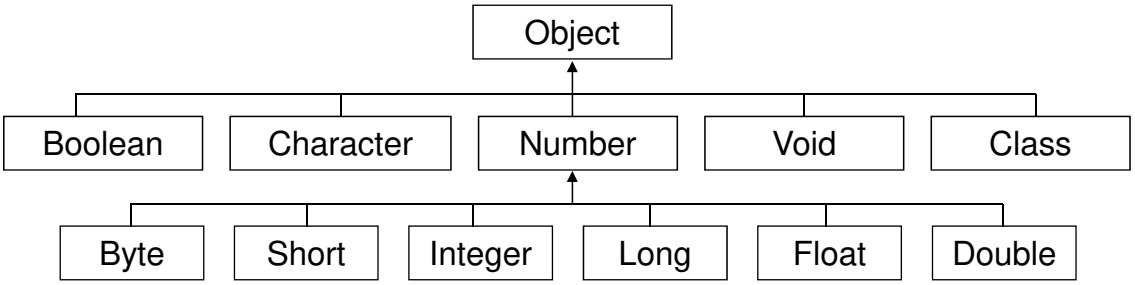
Universidad de Alcalá
Departamento Ciencias de la Computación

Contenidos

- Tema 3-2: Clases básicas de Java

CLASES DE ENVOLTURA

- Cada tipo primitivo tiene asociada una clase de envoltura que contiene un valor del tipo correspondiente y un conjunto de métodos relacionados. Los métodos se usan principalmente para realizar transformaciones entre tipos de datos y representar valores desde o hacia el tipo String.
- Las clases de envoltura tienen dos funciones principales:
  1. Proporcionar una estructura orientada a objetos de los tipos primitivos con sus principales atributos y métodos.
  2. Poder crear objetos que almacenan los valores de un tipo primitivo para poder ser manipulados por clases que solo pueden manejar referencias Object.
- La jerarquía de tipos para las clases de envoltura es:



Universidad de Alcalá
Departamento Ciencias de la Computación

CLASES DE ENVOLTURA

- Todas las clases de envoltura tienen definidos los siguientes métodos:
  - ***public static Tipo valueOf(String cad)***: Devuelve un nuevo objeto del Tipo especificado con el valor de cad.
    Ejemplo: Integer.valueOf("16")
  - ***public String toString()***: Devuelve un objeto String representando el valor del objeto al que se le aplica el método.
  - ***public tipo tipoValue()***: Devuelve el valor del tipo primitivo correspondiente al actual objeto de envoltura.
  - ***public boolean equals(Object obj)***: Hace la comparación entre dos objetos y devuelve true si son iguales o false en caso contrario.
    No confundir con == que sirve para variables de tipo primitivo.
  - ***public int compareTo(Tipo/Object obj)***: Hace la comparación entre dos objetos y devuelve un valor menor, mayor o igual a 0 resultado de esa comparación.

- Las clases numéricas tienen los atributos MIN_VALUE, MAX_VALUE y TYPE para indicar cual es el menor y el mayor valor que pueden tomar y la clase a la que pertenece las variables de ese tipo.

LA CLASE INTEGER

Declaraciones

int i; //tipo primitivo
Integer I; //Clase - objeto

Conversiones de Clase/Cadena

String toString()
static Integer valueOf(String s)
static int parseInt(String s);

Conversiones Objeto/variable

int intValue()
long longValue()
float floatValue()
double doubleValue()

Comparaciones

boolean equals(Object obj)
int compareTo(Integer anotherInteger)

Universidad de Alcalá

LA CLASE INTEGER

Departamento Ciencias de la Computación

```
//Ejemplo Clase Integer
import java.io.*;
class ClaseInteger {
        public static void main( String args[] ) throws IOException {
                BufferedReader entrada =
                        new BufferedReader(new InputStreamReader(System.in));
                int var = 7;
                Integer obj1 = new Integer(5);
                Integer obj2 = Integer.valueOf("5");
                String cad = Integer.toString(var);;

                var = obj1.intValue();
                var += obj2;      //Solo jdk1.5
                System.out.println("Valor de la variable var: "+var);

                if (obj1.equals(obj2)) System.out.println("Los dos objetos
                                                        contienen lo mismo.");
                //Pedimos un número por pantalla al usuario
                System.out.println("\nIntroduce un número: ");
                cad = entrada.readLine();
                var = Integer.parseInt(cad);
                System.out.println("\nEl cuadrado del número es: "+ var*var);
        }
}
```

LA CLASE DOUBLE

Declaraciones

```
double d;          //tipo primitivo
Double D;          //Clase - objeto
```

Conversiones de Clase/Cadena

```
String toString()
static Double valueOf(String s)
static double parseDouble(String s);
```

Conversiones

```
int intValue()
long longValue()
float floatValue()
double doubleValue()
```

Comparaciones

```
boolean equals(Object obj)
int compareTo(Double anotherDouble)
```


LA CLASE DOUBLE


```
//Ejemplo Clase Double
import java.io.*;

class ClaseDouble
{
    public static void main( String args[] ) throws IOException
    {
        BufferedReader entrada=
                new BufferedReader(new InputStreamReader(System.in));

        double var = 3.14; //Variable basada en el tipo primitivo double
        Double obj1  = new Double(5.5); //Objeto basado en la clase Double
        Double obj2; //Objeto basado en la clase Double
        int i = obj1.intValue();
        String cad = "3.5";

        cad = Double.toString(var);
        obj2 = new Double(cad);
        var = obj2.doubleValue();
```

LA CLASE DOUBLE


```
        obj1 = new Double(7.5);     obj2 = new Double(7.5);

        //Comprobación de contenidos
        if (obj1.compareTo(obj2)==0 && obj1.equals(obj2))
                        System.out.println( "Los dos objetos contienen lo mismo." );
        //Comprobación de referencias
        if (obj1 == obj2)
                System.out.println( "obj1 y obj2 apuntan a la misma posición de memoria." );
        else
                System.out.println( "obj1 y obj2 NO apuntan a la misma posición de memoria." );

        obj2 = obj1; //ahora apuntan a la misma posición de memoria

        if (obj1 == obj2)
                System.out.println( "obj1 y obj2 apuntan a la misma posición de memoria." );
        else
                System.out.println( "obj1 y obj2 NO apuntan a la misma posición de memoria." );

        //Pedimos un número por pantalla al usuario
        System.out.println( "\nEscribe un número: " );
        var = Double.parseDouble(entrada.readLine());
        System.out.println("\nEl cuadrado del número es: " + Math.pow(var,2.0) );
    }
}
```


CONVERSIÓN CADENA - NÚMERO

Clases Integer/String	Clases Double/String
Método: *static int parseInt(String s)* Ejemplo: String cad = "2"; int num = Integer.parseInt(cad);	Método: *static double parseDouble(String s)* Ejemplo: String cad = "2.5"; double num = Double.parseDouble(cad);
Método: *int intValue()* Ejemplo: String cad = "2"; Integer objint = new Integer(cad); int num = objint.intValue();	Método: *double doubleValue()* Ejemplo: String cad = "2.5"; Double objdouble = new Double(cad); double num = objdouble.doubleValue();
Método: *static Integer valueOf(String s)* Ejemplo: String cad = "2"; Integer objint = Integer.valueOf(cad); int num = objint.intValue();	Método: *static Double valueOf(String s)* Ejemplo: String cad = "2.5"; Double objdouble = Double.valueOf(cad); double num = objdouble.doubleValue();

LA CLASE CHARACTER

- **Declaraciones**

```
char c;          //tipo primitivo
Character C;     //Clase - objeto
```

- **Comprobaciones**

```
static boolean isLowerCase(char ch)
static boolean isUpperCase(char ch)
static boolean isDigit(char ch)
static boolean isLetter(char ch)
static boolean isSpaceChar(char ch)
```

- **Transformación de caracteres**

```
static char toLowerCase(char ch)
static char toUpperCase(char ch)
```

- **Otros métodos de la clase Character**

```
char charValue()
String toString()
boolean equals(Object obj)
int compareTo(Character anotherCharacter)
```


LA CLASE CHARACTER


```
//Ejemplo Clase Character
import java.io.*;
class ClaseCharacter {
    public static void main( String args[] ) throws IOException {
        BufferedReader entrada=new BufferedReader(new InputStreamReader(System.in));
        char var = 'Z';  //Variable basada en el tipo primitivo char
        Character obj = new Character('A'); //Objeto basado en la clase Character
        var = obj.charValue();
        if (var == 'A') System.out.println( "Ahora var contiene: "+var );
        System.out.print( "\nEscribe una letra: " );
        var = entrada.readLine().charAt(0);
        if (Character.isDigit(var)) System.out.println( "El carácter es un número." );
        else
        if (Character.isLowerCase(var)) {
                System.out.println( "La letra esta en minúsculas" );
                var = Character.toUpperCase( var );
                System.out.println( " y ahora en mayúsculas: " + var );
        } else {
                System.out.println( "La letra esta en mayúsculas" );
                var = Character.toLowerCase( var );
                System.out.println( " y ahora en minúsculas: " + var );
        }
    }
}
```

LA CLASE RANDOM

- La clase **Random** es un generador de números pseudo-aleatorios. De un objeto *Random* se pueden extraer varios tipos de números aleatorios a través de sus métodos.

- **Métodos de Random**

  – nextInt()
  Extrae un int , distribuido uniformemente a lo largo del rango de los enteros

  – nextLong()
  Extrae un long , distribuido uniformemente a lo largo del rango de long

  – nextFloat()
  Extrae un float , distribuido uniformemente entre 0.0 y 1.0

  – nextDouble()
  Extrae un double , distribuido uniformemente entre 0.0 y 1.0

LA CLASE RANDOM

- **Ejemplos Clase Random:**

```
import java.util.*;
class Aleatoria
{
    public static void main( String args[] )
    {
         int x;
         float y;
         double z;

         Random rand = new Random();
         x = rand.nextInt(50);
         y = rand.nextFloat();
         z = rand.nextDouble();

         System.out.println( x );
         System.out.println( y );
         System.out.println( z );
    }
}
```

```
import java.util.*;
class Primitiva
{
    public static void main( String args[] )
    {
         int primi[] = new int[6];

         Random rand = new Random();

         for (int i=0; i<6; i++)
         {
                 primi[i] = rand.nextInt(49) + 1;
                 //Da valores del 1 al 49
         }
         //Imprimimos
         System.out.println( "\nCombinación :" );
         for (int i=0; i<6; i++)
         {
                 System.out.println(" "+primi[i]);
         }
    }
}
```

LA CLASE STRING

- **Constructores**

```
String();
String( String value );
String( char value[] );
String( byte bytes[] );
String( StringBuilder builder );
```

- **Funciones Básicas**

La primera devuelve la longitud de la cadena y la segunda devuelve el carácter que se encuentra en la posición que se indica en índice:

```
int length();
char charAt( int indice );
```

- **Funciones de Comparación de Strings**

```
boolean equals( Object obj );
boolean equalsIgnoreCase( Object obj );

int compareTo( String str2 );
```

Devuelve un entero menor que cero si la cadena es léxicamente menor que str2. Devuelve cero si las dos cadenas son léxicamente iguales y un entero mayor que cero si la cadena es léxicamente mayor que str2.

LA CLASE STRING

- **Funciones de Comparación**

– Comprueba si una región de esta cadena es igual a una región de otra cadena.

```
boolean regionMatches(int thisoffset,String s2,int s2offset,int len);
boolean regionMatches(boolean ignoreCase,int thisoffset,String s2,
                      int s2offset,int len );
```

– Devuelve verdadero si esta cadena comienza o termina con un cierto prefijo o sufijo comenzando en un determinado desplazamiento.

```
boolean startsWith( String prefix );
boolean startsWith( String prefix,int offset );
boolean endsWith( String suffix );
```

– Devuelve el primer/último índice de un carácter/cadena empezando la búsqueda a partir de un determinado desplazamiento.

```
int indexOf( int ch ); int indexOf( int ch,int fromindex );
int lastIndexOf( int ch ); int lastIndexOf( int ch,int fromindex );
int indexOf( String str ); int indexOf( String str,int fromindex );
int lastIndexOf( String str ); int lastIndexOf( String str,int fromindex );
```

- **Funciones para manipulación de cadenas**

– Extracción de subcadenas, reemplazo de caracteres, paso a mayúsculas y minúsculas, ajuste de los espacios en blanco, conversión de la cadena en array de caracteres.

```
String substring( int beginindex ); String substring( int beginindex,int endindex );
String replace( char oldchar,char newchar );
String toLowerCase(); String toUpperCase();
String trim(); char[] toCharArray();
```

LA CLASE STRING


```
//Ejemplo Clase String
import java.io.*;
class ClaseString {
      public static void main( String args[] ) throws IOException {
            String cad;
            int i;
            BufferedReader entrada=new BufferedReader(new InputStreamReader(System.in));

            cad = "Hola Mundo!";
            int l = cad.length();
            System.out.println("Longitud de cad: " + l );

            char c;
            c = cad.charAt(3);
            System.out.println("Caracter 4º de cad: " + c );
            i = cad.indexOf("u");
            System.out.println("Índice de u: " + i );

            String s1 = "Hola Mundo!", s2 = "Hola Mundo!", s3 = "hola mundo!";
            if (s1.equalsIgnoreCase(s3)) System.out.println( "s1 y s3 son iguales." );
            if (s1.regionMatches(0,s2,0,2)) System.out.println( "s1 y s2 coinciden." );

            System.out.println( "Escribe una cadena: " ); cad = entrada.readLine();
            System.out.println("Cadena invertida: ");
            for (i=cad.length()-1;i>=0;i--) System.out.print(String.valueOf(cad.charAt(i)));
      }
}
```


LA CLASE STRINGBUILDER

- **Un objeto *StringBuilder* representa una cadena cuyo tamaño puede variar en tiempo de ejecución.**
- Constructores:

```
StringBuilder();
StringBuilder( int len );
StringBuilder( String str );
```

- Funciones para manipular el tamaño de las cadenas:

```
int length();
char charAt( int index );
String toString();
void setLength( int newlength );  void setCharAt( int index,char ch );
int capacity();  int reverse();
```

- Para cambiar el contenido de un StringBuilder, se pueden utilizar dos métodos: ***append()*** que permite añadir al final de la cadena e ***insert()*** que inserta en una posición determinada.
- Ejemplo:

```
public class CadenaVar {
  public static void main( String args[] ) {
          StringBuilder str = new StringBuilder("Hola ");
          str.append( " todos" );
          str.insert( 5, "a");
          System.out.println( str );
  }
}
```

LA CLASE STRINGTOKENIZER

- La clase **StringTokenizer** proporciona un analizador gramatical de una cadena de entrada.
- Recibe un String de entrada y un String de delimitación. Los delimitadores marcan la separación entre los símbolos que se encuentran en la cadena de entrada. El conjunto de delimitadores por defecto son los caracteres de espacio.
- Métodos: *nextToken()* para ir extrayendo los símbolos consecutivamente; *hasMoreTokens()* devuelve *true* cuando todavía quedan símbolos por extraer.

```
import java.util.StringTokenizer;

class Tokens {
  static String cadena = "Esto,es,una,cadena";

  public static void main( String args[] ) {
          StringTokenizer st = new StringTokenizer( cadena,"," );

          while( st.hasMoreTokens() ) {
                  String token = st.nextToken();
                  System.out.println( token );
          }
  }
}
```

- La misma funcionalidad se puede conseguir utilizando el método ***split*** de la clase String.

```
String cadena = "Esto,es,una,cadena";
String[] tokens = cadena.split(“,”);
```


FECHAS

- La clase **Date** se utiliza para representar una fecha y una hora.
- Los constructores de la clase Date son: Date(long date) siendo el parámetro el número de milisegundos transcurridos desde el 1 de enero de 1970; o Date() que nos devuelve la fecha actual.
- Las fechas se pueden formatear mediante la clase **DateFormat** para adaptarlas a una localización determinada.
- La clase **Calendar** nos permite manipular todos los elementos de una fecha dándonos acceso a todos sus campos como YEAR, MONTH, DAY, HOUR, MINUTE, etc.
- La clase **GregorianCalendar** es una subclase de Calendar que proporciona una implementación del calendario gregoriano.

```
//Fechas con Calendar
import java.util.Calendar;
public class FechaCalendar {

   public static void main(String args[]) {
      Calendar hoy = Calendar.getInstance();
      int año = hoy.get(Calendar.YEAR);
      int mes = hoy.get(Calendar.MONTH) + 1;
      int dia = hoy.get(Calendar.DAY_OF_MONTH);
      System.out.println("Año: " + año);
      System.out.println("Mes: " + mes);
      System.out.println("Dia: " + dia);
   }
}
/*****************************************************/

//Fechas con Date
import java.util.Calendar; import java.text.DateFormat;
public class FechaDate {
   public static void main(String args[]) throws Exception {
      Date hoy = new Date();
      DateFormat df = DateFormat.getDateTimeInstance(DateFormat.SHORT,DateFormat.SHORT);
      String hoystr = df.format(hoy);
      System.out.println("Hoy: " + hoystr);
      Date fecha = df.parse("01/01/1970 00:00");
      String fechaTxt = df.format(fecha);
      System.out.println("Fecha: " + fechaTxt);
   }
}
```



```
//Fechas con GregorianCalendar

import java.text.DateFormat;
import java.util.Calendar;
import java.util.Date;
import java.util.GregorianCalendar;

public class FechaGregorianCalendar {
   public static void main(String[] args) {
      GregorianCalendar hoy = new GregorianCalendar();
      Date hoyD = hoy.getTime();
      String hoystr =
     DateFormat.getDateInstance(DateFormat.LONG).format(hoyD);
      System.out.println("Hoy: " + hoystr);

      GregorianCalendar diaHora =
            new GregorianCalendar(2010, Calendar.MARCH, 25, 10, 59, 59);
      diaHora.set(Calendar.MONTH, Calendar.JUNE);
      diaHora.set(Calendar.HOUR, 12);
      Date diaHoraD = diaHora.getTime();
      String diaHorastr =
     DateFormat.getDateTimeInstance(DateFormat.LONG,DateFormat.SHORT).format(diaHoraD);
      System.out.println("DiaHora: " + diaHorastr);
   }
}
```

- En la versión 1.8 de Java se incorpora el paquete java.time.* que permite tratar fechas, tiempos, instantes y duraciones.

- Las clases más interesantes son las siguientes:
  - LocalDateTime: permite almacenar una fecha con hora.
  - LocalTime: permite almacenar una hora sin fecha tal como: 12:30:16
  - LocalDate: permite almacenar fecha sin hora, tal como 2016-03-01
  - Period: permite saber la cantidad de tiempo entre dos fechas.
  - ChronoUnit: representa unidades de tiempo.


```
//Fechas con java.time.*

import java.time.*; import java.time.format.DateTimeFormatter; import java.time.temporal.ChronoUnit;

public class JavaTimeFechas {

   public static void main(String[] args) {
      // La clase LocalDateTime maneja la fecha completa.
      LocalDateTime fechaCompleta = LocalDateTime.now();
      System.out.println("La fecha de hoy completa es: " + fechaCompleta);
      // La clase LocalTime maneja la hora.
      LocalTime hora = LocalTime.now();
      System.out.println("La hora actual es: " + hora);
      // La clase LocalDate maneja la fecha, pero no la hora como Date. Sirve para manejar fechas específicas.
      LocalDate hoy = LocalDate.now();
      System.out.println("La fecha de hoy es: " + hoy);
      // Cambiamos el formato
      DateTimeFormatter formatoCorto = DateTimeFormatter.ofPattern("dd/MM/yyyy");
      System.out.println("Hoy es (formato corto)" + hoy.format(formatoCorto));
      DateTimeFormatter formatoLargo = DateTimeFormatter.ofPattern("EEEE, dd 'de' MMMM 'de' yyyy");
      System.out.println("Hoy es (formato largo)" + hoy.format(formatoLargo));
      LocalDate fecNacim = LocalDate.of(2000, Month.JANUARY, 1);
      System.out.println("Fecha de nacimiento: " + fecNacim.format(formatoCorto));
      Period p = Period.between(fecNacim, hoy);
      long dias = ChronoUnit.DAYS.between(fecNacim, hoy);
      System.out.println("Eres " + p.getYears() + " años, " + p.getMonths()
            + " meses, y " + p.getDays()
            + " días más viejo. (" + dias + " días en total)");
   }

}
```

APLICACIÓN DE MÉTODOS

- Para utilizar los métodos de una clase hay que fijarse en varios aspectos:

1. Si el método es estático hay que anteponer el nombre de la clase. Ejemplo:

```
String cadena = String.valueOf('A');
```

2. Si no es estático debe aplicarse a un objeto. Ejemplo:

```
String cadena = "Hola";
cadena = cadena.toUpperCase();
```

3. Se pueden anidar llamadas a métodos de forma que el resultado de un método sea lo esperado por otro. Ejemplo:

```
String cadena = " Hola ";
cadena = cadena.trim().toUpperCase();
```