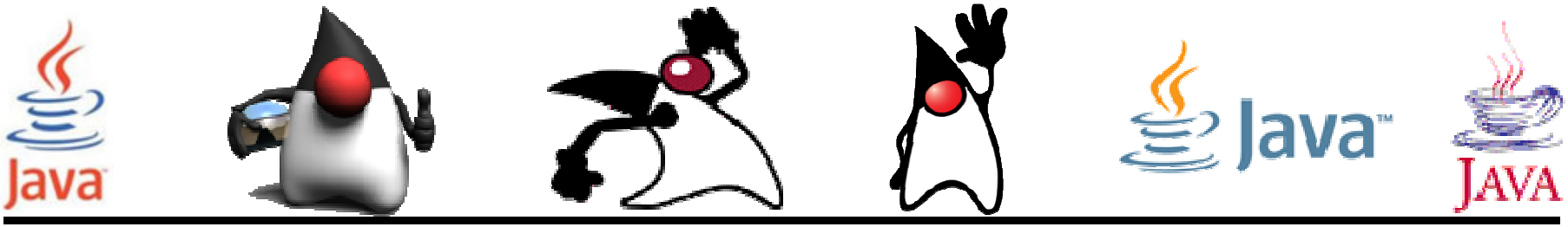

Uso de clases Java

Pedro Corcuera

Dpto. Matemática Aplicada y
Ciencias de la Computación

Universidad de Cantabria

corcuerp@unican.es

Objetivos

- Explicar los conceptos básicos de la programación orientada a objetos .
- Diferenciar entre clases y objetos.
- Diferenciar entre variables/métodos de instancia y variables/métodos de clase (static).
- Cast de tipos de datos primitivos y objetos.
- Comparar objetos y determinar la clase de un objeto.

Índice

- Programación orientada a objetos
- Clases y Objetos
- Tipos de métodos
- Tipos de variables
- Alcance de variables
- Casting de objetos
- Clases en java.lang.* y java.util.*

Programación Orientada a Objetos (POO)

- La programación estructurada se basa en el concepto de función (método), que modela las tareas en las que se descompone la solución del problema.
- La POO se basa en el concepto de objetos como elementos básicos de los programas.
- Los objetos se caracterizan por sus propiedades y comportamientos.
- Los objetos pertenecen a una clase, que agrupa objetos con mismas características y comportamientos.

Clases y Objetos

- En el mundo real el término:
  - Objeto representa una entidad o cosa individual.
  - Una clase representa un grupo de objetos que exhiben algunas características o comportamientos comunes.
  - Las clases son resultado de *clasificación*.
  - Ejemplos de objetos en el mundo real:
    - Estudiantes
    - Estudiantes Graduados
    - Estudiante AnteGrado
    - Estudiantes Master
    - Estudiantes de Doctorado

Clases y Objetos

- Una clase define un componente software autocontenido que representa una clase del mundo real.
- El diseño de clases debe seguir los principios básicos de la orientación a objetos:
  - modularidad
  - encapsulación
- Cada objeto es una instancia de su clase.
- Una clase define las características comunes de sus instancias

Clases y Objetos

- Una clase define las características comunes de sus instancias:
  - los comportamientos
  - las plantillas de los campos
- Cada objeto tiene su propio estado, que son los valores de los campos.
- Las clases se pueden relacionar (relaciones es-un, tiene-un) y organizar mediante la herencia.
  - subclase: especialización
  - superclase: generalización

Clases

- Las clases se pueden pensar como una plantilla o prototipo de un objeto.
- Es una estructura fundamental en la POO.
- Dos clases de miembros:
  - Campos (propiedades, variables). Especifican los tipos de datos definidos por la clase.
  - Métodos (comportamiento). Especifica las operaciones.
- Las clases permiten plasmar el concepto de reusabilidad.

Objetos

- Un objeto es una instancia de una clase (objeto de instancia).
- Los valores de las propiedades de un objeto es diferente de otros objetos de la misma clase.
- Los objetos de la misma clase comparten el mismo comportamiento (métodos).

Clases y Objetos: ejemplo

Clase Coche		Objeto Coche A	Objeto Coche B
Variable de Instancia	Matrícula	ABC 4321	XYZ 1234
	Color	Verde	Rojo
	Fabricante	Fiat	BMW
	Velocidad máx.	140	210
Métodos de Instancia	Acelerar		
	Girar		
	Frenar		

Creación de Objetos

- Para crear un objeto se usa el operador `new`.

```
Scanner in = new Scanner(System.in);
String str1 = new String("Hola Mundo!");
ArrayList<String> names = new ArrayList<String>();
```

Clase objeto

```
Scanner in = new..
```

- El operador `new` asigna memoria para el objeto y devuelve una referencia a la ubicación de memoria e invoca el constructor de clase.
- El constructor es un método para las inicializaciones.

Métodos

- Un método es una pieza separada de código que se puede llamar por cualquier método para realizar alguna función específica.
- Los métodos contienen el comportamiento de una clase o la lógica del negocio.
- Características de los métodos:
  - Pueden o no devolver valores
  - Pueden o no aceptar varios parámetros o argumentos
  - Después de terminar su ejecución vuelve al método que lo invocó.

Tipos de Métodos

- Métodos de instancia (no estáticos)
  - Son más comunes que los métodos estáticos
  - Se llaman después de haber creado un objeto de instancia
  - Sintaxis de la invocación:
    nombreObjeto.nombreMetodo(parametros);
- Métodos estáticos
  - Se llaman sin instanciar una clase (sin usar new).
  - Pertenecen a la clase y se distinguen por la palabra reservada static en su definición
  - Sintaxis de la invocación:
    NombreClase. nombreMetodo(parametros);

Tipos de Métodos: ejemplos

- Métodos de instancia (no estáticos)

```
String str1 = new String("Soy una instancia String");
char x = str1.charAt(2);
String str2 = new String("soy una instancia string");
boolean result = str1.equalsIgnoreCase( str2 );
```

- Métodos estáticos

```
System.out.println("10 ^ 3: " + Math.pow(10, 3));
int i = Integer.parseInt("10");
String hexEquiv = Integer.toHexString( 10 );
```

Paso de parámetros a métodos

- Paso por valor
  - el método hace una copia de la variable pasada al método, por lo que no puede modificar el argumento original.
  - todos los tipos de datos primitivos se pasan por valor en los métodos.
- Paso por referencia
  - se pasa una referencia al objeto al método invocado, por lo que el método puede modificar el objeto referenciado.
  - los arrays se pasan por referencia.

Tipos de variables

- Variables locales (automáticas)
  - declaradas dentro del cuerpo de un método
  - visibles sólo dentro del cuerpo de un método
  - se mantienen en una pila
- Variable de instancia
  - declaradas dentro de una clase pero fuera de los métodos
  - se crean por cada objeto instanciado
- Variables de clase (estáticas)
  - declaradas dentro de una clase pero fuera de los métodos
  - declaradas con el modificador static
  - compartidas por todas los objetos instanciados

Alcance de variables

- El alcance
  - determina dónde es accesible una variable en un programa
  - determina el tiempo de vida de una variable o cuánto tiempo existe una variable en memoria
  - es determinado por la ubicación de la declaración de la variable en el programa
    - Una variable declarada dentro de un bloque tiene como alcance desde su declaración hasta el final del bloque
    - Sólo se puede declarar una variable con un nombre dentro del mismo alcance, pero se puede tener dos variables con el mismo nombre si no están declaradas en el mismo bloque.

Alcance de variables

- Variables locales (automáticas)
  - sólo son válidas desde la línea en que son declaradas hasta la llave de cierre del método o bloque de código dentro del cual es declarada
- Variables de instancia
  - válida en tanto que el objeto está activo
- Variables de clase (estáticas)
  - están en alacance desde el punto en que se carga la clase en la JVM hasta que la clase es descargada. La clase se carga en la JVM la primera vez que se referencia la clase

Casting de tipos

- Asignación del tipo de un objeto a otro
- Casting de tipos primitivos
  - Permite convertir los valores de un dato de un tipo a otro tipo primitivo
  - Ocurre commúnmente entre tipos numéricos
  - Al tipo de dato boolean no se le puede hacer casting
  - Puede ser implícito: cuando se asigna una variable de un tipo de menor orden a otro de mayor orden (p.e. int y double)
  - Puede ser explícito: tiene la forma (tipo) valor

Casting de objetos

- Las instancias de clases también se les puede aplicar un cast, con una restricción:
  - Las clases origen y destino deben estar relacionadas por herencia; una clase debe ser una subclase de otra
- El casting de objetos es análogo a convertir un valor primitivo a un tipo de mayor orden, algunos objetos pueden no necesitan un cast explícito
- Sintaxis: `(nombreclase) objeto`
  - `nombreclase` es el nombre de la clase destino
  - `objeto` es la referencia al objeto fuente

Casting de objetos: ejemplo

- Se tiene una clase `Empleado` y otra subclase `VicePresidente`

```
Empleado emp = new Empleado();
VicePresidente vip = new VicePresidente();

emp = vip; // no requiere cast para ser mas general

vip = (VicePresidente)emp; //requiere cast explicito
```

Conversión de tipos primitivos a Objetos

- Algo que no se puede hacer es hacer un cast de un objeto a un dato de tipo primitivo.
- Como alternativa el paquete java.lang incluye clases que corresponden a cada tipo de dato primitivo. Estas clases se llaman clases Wrapper.
- La mayoría de estas clases tienen los mismos nombres que los tipos de datos, excepto que los nombres de las clases empiezan con la letra mayúscula.

Clases Wrapper

Integer	clase wrapper del tipo primitivo int
Double	clase wrapper del tipo primitivo double
Long	clase wrapper del tipo primitivo long

- Ejemplos:

```
// crea instancia de la clase Integer con valor 7801
Integer dataCount = new Integer(7801);
// convierte un objeto Integer al tipo int
int newCount = dataCount.intValue();
// convierte un String al tipo int
String cants = "65000";
int canti = Integer.parseInt(cants);
```

Comparación de Objetos

- Los operadores relacionales de comparación == (igual) y != (diferente) aplicados a objetos determinan si ambos lados del operador se refieren al mismo objeto y no el valor de los objetos.

```
String str1, str2;
str1 = "Universidad de Cantabria";
str2 = str1;
System.out.println("Mismo objeto? " + (str1 == str2));
str2 = new String(str1);
System.out.println("Mismo objeto? " + (str1 == str2));
System.out.println("Mismo valor? " + str1.equals(str2));
```

```
true
false
true
```

Determinación de la clase de un objeto

- Para conocer la clase de un objeto creado según

```
NombreClase key = new NombreClase();
```

- Hay dos métodos:
  - uso del método `getClass()` que retorna la clase de un objeto. Para obtener el nombre se usa el método `getName()` que devuelve la cadena que representa el nombre de la clase.

```
String name = key.getClass().getName();
```

Determinación de la clase de un objeto

- Uso del operador `instanceof` cuyo operando izquierdo es un objeto y el operando derecho el nombre de una clase.

  La expresión devuelve true o false según si el objeto es una instancia de la clase nombrada o cualquiera de las subclases.

  Ejemplo:

```
boolean ex1 = "Somo" instanceof String; // true
Object pt = new Punto(10, 10);
boolean ex2 = pt instanceof String; // false
```

Clases en java.lang.* y java.util.*

- Algunas de las clases más usadas son:

  java.lang.* (se importa automáticamente):

  - Clase Math
  - Clase String y StringBuffer
  - Clase Process y Runtime
  - Clase System
  - Clases Wrapper

  java.util.*

  - Clase Properties
  - Clase Date

Clase Math

- Ofrece constantes predefinidas y métodos para la realización de las operaciones matemáticas
- Documentación Java SE 6 API
  http://download.oracle.com/javase/6/docs/api/java/lang/Math.html

Clase Math: Ejemplo

```
class MathDemo {
  public static void main(String args[]) {
    System.out.println(" valor absoluto de -5: " + Math.abs(-5));
    System.out.println(" valor absoluto de 5: " + Math.abs(5));
    System.out.println("numero random (0-10): " + Math.random()*10);
    System.out.println("max de 3.5 y 1.2: " + Math.max(3.5,1.2));
    System.out.println("min de 3.5 y 1.2: " + Math.min(3.5,1.2));
    System.out.println("ceiling de 3.5: " + Math.ceil(3.5));
    System.out.println("floor de 3.5: " + Math.floor(3.5));
    System.out.println("e elevado a 1: " + Math.exp(1));
    System.out.println("log 10: " + Math.log(10));
    System.out.println("10 elevado a 3: " + Math.pow(10,3));
    System.out.println("valor redondeado de pi: " +
                        Math.round(Math.PI));
    System.out.println("raiz cuadrada de 5 = " + Math.sqrt(5));
    System.out.println("10 radian = " + Math.toDegrees(10)+" grados");
    System.out.println("sin(90): " + Math.sin(Math.toRadians(90)));
    }
}
```

Clase String

- Representa combinaciones de caracteres literales
- En Java, las cadenas de caracteres se pueden representar usando:
  - Arrays de caracteres
  - La clase String

  Un objeto String es diferente de un array de caracteres
- Dispone de varios constructores
- Documentación Java SE 6 API

  http://download.oracle.com/javase/6/docs/api/java/lang/String.html

Clase String: Ejemplo$_1$

```
class StringConstructorsDemo {
   public static void main(String args[]) {
      String s1 = new String();           // string vacio
      char chars[] = { 'h', 'e', 'l', 'l', 'o'};
      String s2 = new String(chars);   // s2="hello";
      byte bytes[] = { 'w', 'o', 'r', 'l', 'd' };
      String s3 = new String(bytes);   // s3="world"
      String s4 = new String(chars, 1, 3);
      String s5 = new String(s2);
      String s6 = s2;

      System.out.println(s1);
      System.out.println(s2);
      System.out.println(s3);
      System.out.println(s4);
      System.out.println(s5);
      System.out.println(s6);
  }
}
```

Clase String: Ejemplo$_2$

```
class StringDemo {
  public static void main(String args[]) {
    String name = "Juan";
    System.out.println("name: " + name);
    System.out.println("3r caracter de name: " + name.charAt(2));
    // el caracter primero alfabeticamente tiene menor valor unicode
    System.out.println("comp Juan con Sara: "+ name.compareTo("Sara"));
    System.out.println("cm Sara con Juan: "+ "Sara".compareTo("Juan"));
    // 'J' tiene menor valor unicode comparado a 'j'
    System.out.println("cm Juan con juan: " + name.compareTo("juan"));
    System.out.println("Juan comparado" +
                       name.compareToIgnoreCase("juan") );
    System.out.println("es Juan igual a Juan? " + name.equals("Juan"));
    System.out.println("es Juan igual a juan? " + name.equals("juan"));
    System.out.println("es Juan igual a juan (ignorar may/min)? " +
                       name.equalsIgnoreCase("juan"));
    char charArr[] = "Hi XX".toCharArray();
    "Juan".getChars(0, 2, charArr, 3);
    System.out.print("getChars metodo: ");
```

Clase String: Ejemplo$_2$

```
        System.out.println(charArr);
        System.out.println("Long. de name: " + name.length());
        System.out.println("Remplaza a's con e's en name: " +
                           name.replace('a', 'e'));
        // se requiere añadir 1 al parametro endIndex de substring
        System.out.println("substring de name: " + name.substring(0, 2));
        System.out.println("Trim \"  a b c d e f   \": \"" +
                           "  a b c d e f   ".trim() + "\"");
        System.out.println("representa String de expresion boolean 10>10: "
                           + String.valueOf(10 > 10));
        // metodo toString se llama en el metodo println
        System.out.println("representa String de expresion boolean 10<10: "
                           + (10<10));
        // Notar que no hay cambio en el nombre del objeto String
        System.out.println("name: " + name);
    }
}
```

Clase StringBuffer

- El problema con los objetos String es que una vez creados no se pueden modificar (es una clase final)
- Un objeto StringBuffer
  - es similar a un objeto String
  - puede ser modificado en longitud y contenido mediante métodos de la clase
- Documentación Java SE 6 API

http://download.oracle.com/javase/6/docs/api/java/lang/StringBuffer.html

Clase StringBuffer: Ejemplo

```
class StringBufferDemo {
  public static void main(String args[]) {
    StringBuffer sb = new StringBuffer("Juan");
    System.out.println("sb = " + sb);
    System.out.println("capacidad de sb: "+ sb.capacity()); /*pd 16 */
    System.out.println("anadir \'O\' to sb: " +  sb.append('O'));
    System.out.println("sb = " + sb);
    System.out.println("3r caracter de sb: " + sb.charAt(2));
    char charArr[] = "Hi XX".toCharArray(); sb.getChars(0,2,charArr,3);
    System.out.print("metodo  getChars: ");
    System.out.println(charArr);
    System.out.println("Inserta \'jo\' en 3r pos: "+sb.insert(2,"jo"));
    System.out.println("Borra \'jo\' en 3r pos: " + sb.delete(2,4));
    System.out.println("long de sb: " + sb.length());
    System.out.println("remplaza: " + sb.replace(3, 9, " Ong"));
    System.out.println("substring (1r dos caracteres): "
                          + sb.substring(0, 3));
    System.out.println("toString() implicito : " + sb);
   }
}
```

Clases Wrapper

- Los tipos de datos primitivos no son objetos y por tanto no pueden acceder a los métodos de la clase *Object*
- Las clases Wrapper permiten la representación en objetos de los tipos primitivos

Clases Wrapper

Tipo primitivo	Clase Wrapper	Argumentos Constructor
byte	Byte	byte o String
short	Short	short o String
int	Integer	int o String
long	Long	long o String
float	Float	float, double o String
double	Double	double o String
char	Character	char
boolean	Boolean	boolean o String

Clase Wrapper Boolean: Ejemplo

```
class BooleanWrapper {
  public static void main(String args[]) {
    boolean booleanVar = 1 > 2;
    Boolean booleanObj = new Boolean("True");
    /* primitivo a objeto; tambien se puede usar metodo valueOf */
    Boolean booleanObj2 = new Boolean(booleanVar);

    System.out.println("booleanVar = " + booleanVar);
    System.out.println("booleanObj = " + booleanObj);
    System.out.println("booleanObj2 = " +  booleanObj2);
    System.out.println("compare 2 wrapper objects: "
                       + booleanObj.equals(booleanObj2));

    /* objeto a primitivo */
    booleanVar = booleanObj.booleanValue();
    System.out.println("booleanVar = " + booleanVar);
  }
}
```

Clase Process

- Representa los programas en ejecución
- Ofrece métodos para la gestión de procesos
  - Ejecutar un proceso
  - Matar un proceso
  - Comprobar el estado de un proceso
- Documentación Java SE 6 API

http://download.oracle.com/javase/6/docs/api/java/lang/Process.html

Clase Runtime

- Representa el entorno de ejecución
- Algunos de los métodos importantes son:

  static Runtime getRuntime(): Returna el objeto runtime asociado con la aplicación en curso Java

  Process exec (String command): Ejecuta el comando especificado en un proceso separado

- Documentación Java SE 6 API

http://download.oracle.com/javase/6/docs/api/java/lang/Runtime.html

Clases Process y Runtime: Ejemplo

```
class RuntimeDemo {
  public static void main(String args[]) {
    Runtime rt = Runtime.getRuntime();
    Process proc;
    try {
      proc = rt.exec("regedit");
      proc.waitFor(); //probar ejecutar sin esta linea
    } catch (Exception e) {
      System.out.println("regedit es comando desconocido");
    }
  }
}
```

Clase System

- Ofrecen muchos campo y métodos útiles
  - Entrada standard
  - Salida standard
  - Métodos de utilidad para copia rápida de una parte de un array
- Documentación Java SE 6 API

http://download.oracle.com/javase/6/docs/api/java/lang/System.html

Clase System: Ejemplo

```
import java.io.*;

class SystemClass {
  public static void main(String args[]) throws IOException {
    int arr1[] = new int[1050000];
    int arr2[] = new int[1050000];
    long startTime, endTime;
    /* initializa arr1 */
    for (int i = 0; i < arr1.length; i++) {
       arr1[i] = i + 1;
    }
    /* copiado manual */
    startTime = System.currentTimeMillis();
    for (int i = 0; i < arr1.length; i++) {
       arr2[i] = arr1[i];
    }
    endTime = System.currentTimeMillis();
    System.out.println("Tiempo para copiado manual: " +
                       (endTime-startTime) + " ms.");
```

Clase System: Ejemplo

```
    /* uso de la utilidad de copia ofrecido por clase System */
    startTime = System.currentTimeMillis();
    System.arraycopy(arr1, 0, arr2, 0, arr1.length);
    endTime = System.currentTimeMillis();
    System.out.println("Tiempo usando utilidad copy: " +
                        (endTime-startTime) + " ms.");
    System.gc();      //fuerza a trabajar el garbage collector

    System.setIn(new FileInputStream("temp.txt"));

    // imprime las caracteristicas del sistema
    Properties p1 = System.getProperties();
    p1.list(System.out);

    System.exit(0);
  }
}
```

Clase Properties

- La clase Properties representa un conjunto persistente de propiedades
- La propiedades se pueden almacenar en o cargar desde un stream, típicamente un fichero
- Cada campo y su correspondiente valor en la lista de propiedades es una cadena de caracteres
- Documentación Java SE 6 API

http://download.oracle.com/javase/6/docs/api/java/util/Properties.html

Clase System: Ejemplo

```
import java.io.FileInputStream;
import java.io.IOException;
import java.util.Properties;

public class SetSystemProperties {
  public static void main(String[] args) {
     FileInputStream propFile = null;
     Properties p = null;

     try { // configura nuevas propiedades desde "myProperties.txt"
        propFile = new FileInputStream("myProperties.txt");
        p = new Properties(System.getProperties());
        p.load(propFile);
     } catch (IOException e) { System.out.println(e); }
     // asigna propiedad con metodo setProperty()
     p.setProperty("myKey1", "myValue1");
     // asigna propiedades del sistema
     System.setProperties(p);
     // muestra nuevas propiedades
     System.getProperties().list(System.out);
  }
}
```

Clase Date

- Representa un momento preciso de tiempo a nivel de milisegundos
- Los datos de fecha se representan como un tipo long que cuenta el número de milisegundos desde el 1 de Enero 1970 (Greenwich)
- Documentación Java SE 6 API

  http://download.oracle.com/javase/6/docs/api/java/util/Date.html

Clase Date: Ejemplo

```
import java.util.Date;

public class DateTest {
  public static void main(String[] args) {
     // Imprime el numero de milisegundos que toma
     // la ejecucion de un trozo de codigo
     Date d1 = new Date();

     // codigo a ser medido
     for (int i=0; i<10000000; i++) {
         int j = i;
     }

     Date d2 = new Date();
     long elapsed_time = d2.getTime() - d1.getTime();

     System.out.println("Duracion " +elapsed_time+" milisegundos");
  }
}
```