Java

- Desarrollado por Sun 1991
- Basado en C++ (y algo en Smalltalk)
- Base para desarrollos en Internet
- Diversas versiones (1.0 hasta la 1.3)
  - En este curso veremos la versión 1.3
  - Comercialmente JDK 2.0
- Licencias para NetScape, IBM y Microsoft

Características de Java

- Simple
- Orientado a Objetos
- Distribuido
- Robusto
- Seguro
- Arquitectura neutral
- Portable
- Interpretado
- Alto rendimiento
- Multihebras
- Dinámico

Características de Java Simple

- Basado en C++ eliminando
  - Aritmética de punteros
  - Referencias
  - Registros y uniones
  - Definiciones de tipos y macros
  - Necesidad de liberar memoria
    - (Según dicen, estas razones son el 50% de los fallos en los programas de C y C++)

Características de Java Orienta a Objetos

- Encapsulación, herencia y polimorfismo
- Interfaces para suplir herencia múltiple
- Resolución dinámica de métodos
- Reflexión
- Serialización
- Una gran librería de clases estandarizadas

Características de Java Distribuido

- Extensas capacidades de comunicaciones
- Permite actuar con http y ftp
- El lenguaje no es distribuido, pero incorpora facilidades para construir programas distribuidos
- Se están incorporando características para hacerlo distribuido
  - RMI, COM, Corba

5

Características de Java Robusto

- Chequeos en tiempo de compilación
- Chequeos en tiempo de ejecución
  - Chequeo de punteros nulos
  - Chequeo de límites en vectores
  - Excepciones
  - Verificación de código binario
  - Recolección automática de basuras
- No es posible sobreescribir memoria o corromper datos

6

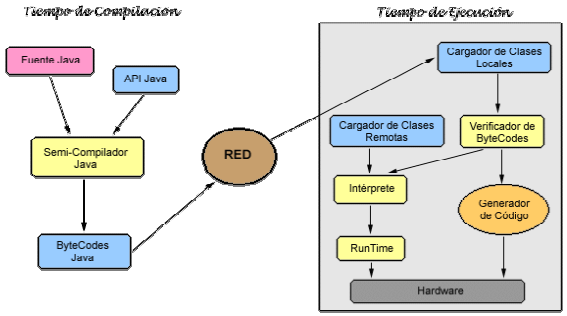
Características de Java Seguro

- No hay punteros
- El cast (promoción) hacia lo general debe ser implícito
- Los bytecodes pasan varios test antes de ser ejecutados
  - Varios mecanismos de seguridad

7

Características de Java Arquitectura neutral

- Genera .class



8

Características de Java
Portable

- WORA (Write Once Run Anywhere)
  - Mismo código en distintas arquitecturas
  - Define la longitud de sus tipos independiente de la plataforma
  - Construye sus interfaces en base a un sistema abstracto de ventanas

9

Características de Java
Interpretado

- Para conseguir la independencia del sistema operativo genera bytecodes
- El intérprete toma cada bytecode y lo interpreta
  - Mecanismos de seguridad
- El mismo intérprete corre en distintas arquitecturas

10

Características de Java
Alto rendimiento

- Cuando se necesite, el código puede compilarse
  - Existen compiladores del lenguaje
    - Asymetrix, Symantec, IBM
  - Compiladores Just in Time (JIT)
    - En la actualidad, casi todos los Plug-in

11

Características de Java
Multihebras

- Permite crear tareas o hebras (o hilos)
- Sincronización de métodos y bloques para exclusión mútua
- Comunicación de tareas

12

Características de Java Dinámico

- Conecta los módulos que intervienen en una aplicación en el momento de su ejecución
- No hay necesidad de enlazar previamente
- Compilación independiente

Características eliminadas de C++

- No hay typedef, defines ni preprocesamiento
- No hay estructuras ni uniones ni enum
- No hay funciones (sólo métodos en clases)
- No hay herencia múltiple
- No hay goto
- No hay sobrecarga de operadores
- No hay conversión automática (sólo seguras)
- No hay punteros
- No hay templates
- No hay que destruir los objetos inservibles

Aplicaciones Java

- Applet
  - Aplicaciones que pueden correr dentro de un navegador de internet
- Aplicaciones
  - Con o sin interfaz gráfico
- JavaBeans
  - Componentes combinables
- Servlets
  - Aplicaciones que corren en un servidor WWW

Programación en Java I

- Todo lo que se definen son clases o interfaces
- Todos los objetos son dinámicos
- Las declaraciones de acceso se hacen al nivel de miembro

  public, private, protected o por defecto (package)
- Dentro de un miembro se pueden utilizar

  `this` `super`
- Hay recolección automática de basuras

Programación en Java II

- Para enviar un mensaje

  objeto.mensaje(arg_1,arg_2,...,arg_n)
- Para enviar un mensaje a una clase

  NombreClase.mensaje(arg_1,arg_2,...,arg_n)

  objeto.mensaje(arg_1,arg_2,...,arg_n)
- Para acceder a una variable de instancia pública

  objeto.variable
- Para acceder a una variable de clase pública

  NombreClase.variableDeClase

  objeto.variableDeClase

Métodos y variables de clase

- Se declaran como static
- Los métodos de clase tienen acceso a las variables de clase
- Todos los herederos comparten la misma variable de clase

```
class Ejemplo {
   static private int a;
   static public int getSuma() {
      return a;
   }
   ...
}
```


Métodos y variables de instancia

- Un método de instancia tiene acceso
  - a las variables de clase y de instancia
  - cualificado a las variables de instancia de cualquier objeto de la misma clase

```
class Ejemplo {
  static private int a;
  private int b;
  public int getSuma() {
   return a+b;
  }
  public int getOtra(Ejemplo e) {
   return a+b+e.b;
  }
...
}
```


Ejemplo clase

```
class Urna {
    private int bolasBlancas;
    private int bolasNegras;
    public Urna(int bb, int bn) {
        bolasBlancas = bb;
        bolasNegras  = bn;
    }
    public void meterBola(char color) {
        if (color=='b') bolasBlancas++;
        else            bolasNegras++;
    }
    public boolean quedanBolas() {
        return (this.totalBolas() > 0);
    }
    ....
    private int totalBolas() {
        return bolasBlancas + bolasNegras;
    }
}
```

Compilación I

- Fichero fuente.
  - Debe tener la extensión .java.
- Para invocar al compilador de JDK

  javac Urna.java
- Si no hay errores, genera el fichero

  Urna.class
- Si hubiera más clases declaradas, generaría un fichero .class por cada clase.

Compilación II

- Si se van a utilizar librerías externas,
  - debe definirse en el entorno de ejecución la variable CLASSPATH
  - Las clases del sistema no necesitan incluirse

```
SET CLASSPATH=.;C:\CURSOS\JAVA\UTIL
```

- Indica dónde buscar las clases que necesita
- Crear las clases Relojes y Contenedores

Programación en Java Aplicación de consola

- Una aplicación de consola debe contener:

  `public static void main(String [] args)`
- Puede contener una clase public y otras clases
- El nombre del fichero debe coincidir con el de la clase public
- Una clase fundamental del sistema es System que tiene como miembros estáticos públicos

```
in   -- Entrada estándar
out  -- Salida estándar
err  -- Error estándar
```

- Ejemplo: para imprimir una cadena en out

  `System.out.println("hola mundo");`

El primer ejemplo

```
// Aplicación Hola mundo
class PrimerEjemplo {
   public static void main (String [] args) {
      System.out.println("Hola mundo");
   }
}
```

PrimerEjemplo.java

Ejecución de aplicación de consola

java PrimerEjemplo

- java contiene la máquina virtual que interpreta los bytecodes del programa.
- El intérprete busca el método de clase público `main` de la clase PrimerEjemplo y lo ejecuta.
- Se realizan ciertas comprobaciones de seguridad antes de ejecutar.
  - **Ejercicio**: comprobar el funcionamiento del ejemplo anterior

Sintáxis básica (como C y C++)

- Java es un lenguaje con sintaxis orientada a bloques.
  - Los bloques se delimitan entre "{" y "}"
  - Pueden declarar variables locales
    - ámbito: el bloque donde se declara
- Las sentencias del lenguaje terminan con ";"
- Las sentencias de selección e iteración son iguales que las de C y C++

  if , if else, switch, for, while, do while

Selección

```
if ( ExprBooleana )
   Sentencia;    o bien     BloqueSentencias

if ( ExprBooleana )
   Sentencia;    o bien     BloqueSentencias
else
   Sentencia;    o bien     BloqueSentencias

switch ( exprNumerable ) {
      case literal1 : Sentencias ; break;
      case literal2 : Sentencias ; break;
      ...
      default : Sentencias
}
```

- Ver Epoca.java

Iteración

```
for( Asignación ; Test ; Incremento)
      Sentencia;     o bien    BloqueSentencias

while ( ExpBooleana )
      Sentencia;     o bien    BloqueSentencias

do
      Sentencia;     o bien    BloqueSentencias
while ( ExprBooleana )
```

Comentarios

- Tres tipos de comentarios

```
// Comentarios de una sola línea

/* comentarios
de
varias lineas */

/** comentarios de documentación
  * de una o varias
  * líneas  */
```


Comentarios de documentación

- Se utilizan junto con la herramienta javadoc
- Genera una página html para la clase

```
/**
  * @see nombre de clase
  * @see nombre de clase # método

  * @version información versión

  * @author nombre autor
*/
```


Comentarios de documentación

```
/**
  * @param parámetro y descripción
  *   se utiliza en la descripción de métodos
  *   uno por cada parámetro

  * @return descripción
  *   lo que devuelve el método

  * @exception nombre de la excepción y descripción
  *   indica las clases de excepciones manejadas

 * @deprecated
*/
```

- Ver javamd.java y defecto.java

Identificadores, claves y tipos

- Identificadores
  - Comienzan con letra, _ o $, después pueden llevar dígitos y la longitud es ilimitada
- Claves
  - Palabras reservadas del lenguaje
- Tipos
  - Básicos y no básicos(objetos)

Palabras clave

```
abstract   continue   for          new          switch
boolean    default    goto         null         synchronized
break      do         if           package      this
byte       double     implements   private      threadsafe
byvalue    else       import       protected    throw
case       extends    instanceof   public       transient
catch      false      int          return       true
char       final      interface    short        try
class      finally    long         static       void
const      float      native       super        while
```

- Reservadas

```
cast       future     generic      inner
operator   outer      rest         var
```


Variables

- Una variable es una dirección de memoria
- No hay variables Globales. Todas se declaran dentro de una clase
- Java incorpora dos tipos de variables
  - Variable primitiva para datos de tipos básicos
    - Contienen al dato
  - Variable referencia para objetos
    - Referencian al objeto

Tipos básicos

- Hay 8 tipos básicos y uno especial
- Todos ellos conforman las constantes del lenguaje (literales)
  - boolean (true, false)
  - char (‘a’, ‘\123’, \u1234’,’\t’,’\n’,’\\’)
  - byte (8bits), short (16), int(32), long(64)
  - float (32bits)y double (64)
    - decimal, octal, hexadecimal
  - void

Tipos básicos. Ejemplos

```
byte b2;          // declaración
b2 = o145;        // asignación de valor octal
byte b1 = 0xfa;   // inicialización = declaración + asignación
                  //                  de valor
boolean sigo;
long total = 145L;
long checksum = 0xab01ffe2L;   // valor hexadecimal
float truncPi = 3.1415f;
double e = 2.71728d;
char c = ‘a’;
char tab = ‘\t’;
```

- String es un tipo no básico pero admite literales

```
String saludo = “hola que tal”;
```

Tipos básicos. Operaciones

- Las usuales (parecidas a C y C++)
  - Numéricas

    + - * / % += -= *= /= %= ++ --
  - Cadenas de caracteres (String) (No son tipos básicos)

    +
  - Expresiones relaciones

    < > <= >= == !=
  - Operadores lógicos

    && (y) || (o) ! (no)
- Asignación

  =

Promoción

- Todos los tipos básicos se promocionan a los superiores si es necesario

```
byte -> short -> int -> long -> float -> double
        char  ->
```

- En caso contrario hay que forzarlo (casting)

```
float f = 15;    // por defecto, es un int
double d = 17.6; // si no se indica nada se supone double
float h = 7.46;  // error, hay que forzarlo
float h = 7.45f; // Una manera de forzarlo
float h = (float) 7.45;    // otra manera de forzarlo
```

  - Ver Estadistica.java

Tipos no básicos Objetos

  - son dinámicos (Hay que crearlos)
  - son instancias de alguna clase
- Una variable puede
  - contener un valor de un tipo básico
    - La variable se declara como del tipo básico
  - referenciar a un objeto de una clase
    - La variables se declara como de la clase
- Un objeto se crea por medio del operador new salvo aquellas cadenas que actúan de literales

```
Punto p = new Punto(); // Inicialización
Punto q;               // Declaración
q = new Punto();       // Asignación de un objeto
```

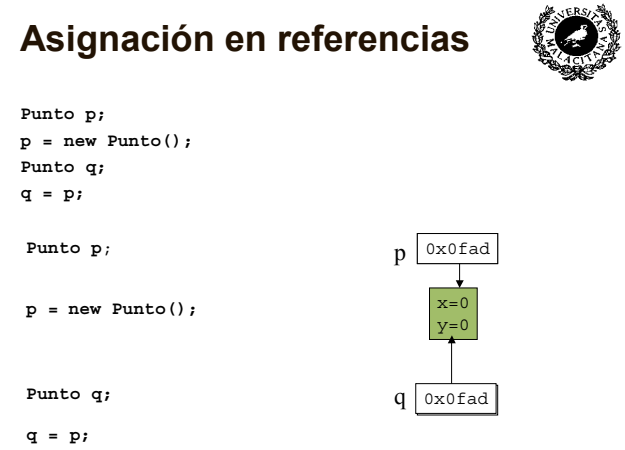

Creación de un Objeto

- Se hace a través del operador new
  - Declaración de una variable referencia
    - Se reserva memoria **SOLO** para referenciar al objeto

      ```
      Punto p;
      ```
  - Creación del objeto
    - Reserva de memoria para el objeto (heap)
    - Inicialización de las variables por defecto

      ```
      p = new Punto();
      ```
  - Posteriormente se mejorara la creación

Asignación en referencias



```
Punto p;
p = new Punto();
Punto q;
q = p;
```

```
Punto p;
```

```
p = new Punto();
```

```
Punto q;
```

```
q = p;
```


Igualdad de referencias a objetos

- Los operadores `==` y `!=` pueden usarse para comparar los valores de dos referencias a objetos
  - No comparan los objetos referenciados
- Una referencia no inicializada contiene el valor `null`
- Posteriormente veremos como comparar objetos

Clases

- Todo método debe indicar el tipo que devuelve
  - básico o clase;
- Si un método no devuelve nada, debe devolver el tipo `void`.
- La visibilidad se debe indicar en cada variable y método
- `this` es una autoreferencia dentro de los métodos y puede omitirse
- `super` es una autoreferencia (al padre) dentro de los métodos

La clase Punto

```
class Punto {
   double x,y;
   void valorInicial(double vx, double vy) {
      x = vx;      // es igual que   this.x = vx
      y = vy;
   }
   double distancia(Punto q) {
      double dx = x - q.x;
      double dy = y - q.y;
      return Math.sqrt(dx*dx+dy*dy);
   }
}
```

Uso de la clase Punto

```
class UsoPunto {
   public static void main(String [] args) {
      Punto p,q;
      double d;
      p = new Punto();
      p.valorInicial(5,4);
      q = new Punto();
      q = valorInicial(2,2);
      d = p.distancia(q);
      System.out.println("Distancia :" + d);
   }
}
```

- Ver UsoPunto.java

Arrays en Java I

- Grupo de elementos del mismo tipo (básico o no)

```
int diasDeMeses [];
int [] diasDeMeses;
```

  – Necesita que se le reserve memoria

```
diasDeMeses = new int[12];
```

  – y ya se puede utilizar

```
diasDeMeses[0]=31;   // Comienzan en 0
```

  - `ArrayIndexOutOfBoundsException` si hay un exceso o defecto de rango (no como C o C++)

  – También se pueden crear así:

```
int diasMeses[] =
         {31,28,31,30,31,30,31,31,30,31,30,31}
```


Arrays en Java II

- Grupo de elementos del mismo tipo (básico o no)

```
Punto arP [];
Punto [] arP;
```

  – Necesita que se le reserve memoria

```
arP = new Punto[2];
```

  – y hay que crear cada objeto que se inserte

```
arP[0]=new Punto();
arP[1]=new Punto();
```

  – También se pueden crear así:

```
Punto arP[] = {new Punto(), new Punto()};
```

  – La longitud de un array la proporciona la variable de instancia `length`

Argumentos en línea de comandos

```
public static void main(String [] args)
```

- El argumento de main es un array de String.
- El primer argumento es el 0, el segundo argumento es el 1, etc.
- Uso de los argumentos

```
java EjArg hola 45 "que pasa"
```

- En este caso, el array `args` tiene tres elementos

```
"hola" "45" "que pasa"
```

Argumentos en línea de comandos. Ejemplo

```
class EjArg {
   public static void main(String [] args) {
      for (int i=0; i<args.length ; i++)
            System.out.println(args[i]);
   }
}
```

- Una vez compilado, se ejecuta con

```
>java EjArg hola 45 "que pasa"
hola
45
que pasa
```


Arrays multidimensionales

- No existen como tal en Java pero pueden crearse de tres modos diferentes:

```
// Modo 1
int matriz[][]=new int[3][3];
matriz[1][2]=5;
// Modo 2
int matriz[][]=new int[3][];
matriz[0]=new int[6];
matriz[1]=new int[2];
matriz[2]=new int[7];
// Modo 3
int matriz[][] =
    {{1,2,3,4},{2,3,4,5},{3,4,5,6},{4,5,6,7}};
```


Constructores

- Métodos especiales invocados de forma automática cuando se crean objetos con new.
- Deben coincidir con el nombre de la clase
- Su prototipo no devuelve nada (ni void)
- Pueden tener visibilidad
- Si no se define ninguno, se crea uno por defecto sin argumentos que inicializa:
  - Las variables numéricas a 0
  - Los variables booleanos a false
  - Las variables de tipo caracter a '\u0000'
  - Las variables referencias a null

Constructores para la clase Punto

```
class Punto {
   double x,y;
   Punto() { // Como el por defecto
      x = 0; // es igual que this.x = 0
      y = 0;
   }
   Punto(double vx, double vy) {
      x = vx; // es igual que this.x = vx
      y = vy;
   }
   .....
}
```

Uso de constructores

```
class UsoPunto {
   public static void main(String [] args) {
      Punto p,q;
      double d;
      p = new Punto();
      q = new Punto(2,2);
      d = p.distancia(q);
      System.out.println("Distancia :" + d);
   }
}
```

Ejemplo: Urna, Primos, Listas de Enteros

Polimorfismo por sobrecarga

- Los nombres de los métodos pueden repetirse si varía el número o el tipo de los argumentos.
  - Válido también para los constructores
    - En la clase Punto, podemos definir la distancia a un par de coordenadas por medio del método distancia

```
class Punto {
      ...
      double distancia(double cx, double cy) {
            double dx = x - cx;
            double dy = y - cy;
            return Math.sqrt(dx*dx+dy*dy);
      }
}
```


Herencia I

- Una clase puede extender a otra ya existente

```
class <Derivada> extends <Base>
```

```
class Pixel extends Punto {
  byte color;
  Pixel(double vx, double vy, byte vc) {
       super(vx,vy);  // debe ser la primera sentencia
       color = vc
  }
  .....
}
```


Herencia II

- Una subclase hereda todas las variables y métodos de la clase base.
  - Los constructores no se heredan.
- Si una subclase redefine un método de una clase, éste queda oculto y sus objetos usarán el redefinido
  - Con super se puede acceder al método oculto.
- Las variables de clase (static) se comparten.

Redefinición de métodos

```
class Pixel extends Punto {
      ...
      double distancia(double cx, double cy) {
            double dx = Math.abs(x - cx);
            double dy = Math.abs(y - cy);
            return dx+dy; // distancia Manhattan
      }
}
```


Vinculación dinámica I

```
class PruebaPixel1 {
   public static void main(String [] args) {
       Pixel x = new Pixel(4,3,(byte)2);
       Punto p = new Punto(4,3);
       double d1,d2;
       d1 = x.distancia(1,1); // de pixel
       d2 = p.distancia(1,1); // de punto
       System.out.println("Distancia pixel:" + d1);
       System.out.println("Distancia punto:" + d2);
   }
}
```


Vinculación dinámica II

```
class PruebaPixel2 {
   public static void main(String [] args) {
       Punto x = new Pixel(4,3,(byte)2);
       Punto p = new Punto(4,3);
       double d1,d2;
       d1 = x.distancia(1,1); // de pixel
       d2 = p.distancia(1,1); // de punto
       System.out.println("Distancia pixel:" + d1);
       System.out.println("Distancia punto:" + d2);
   }
}
```


Acceso al método oculto

```
class Punto {
   ...
   void trasladar(double cx, double cy) {
       x += cx;
       y += cy;
   }
}

class Pixel extends Punto {
   ...
   void trasladar(double cx, double cy) {
       super.trasladar(cx,cy);
       color = (byte)0;
   }
}
```

Modificador final

- Un método o variable puede incorporar el modificador final
  - Se utiliza para indicar que no puede cambiarse su valor (variable) o redefinirse en la herencia (método)

```
class Punto {
       static final int VERDE = 1;
       static final int AZUL  = 2;
       ...
}
```

- Las variables final static son constantes

Clase Abstracta

- Se define mediante el cualificador abstract tanto para la clase como para los métodos sin cuerpo
- No se pueden crear objetos de clases abstractas
- Si se pueden crear referencias

```
abstract class Figura {
   abstract double perimetro();
   abstract double area();
}
```


Herencia y Clase Abstracta I

```
class Cuadrado extends Figura {
   double lado;
   Cuadrado(double l) {
          lado = l;
   }
   double perimetro() {
          return lado*4;
   }
   double area()     {
          return lado*lado;
   }
}
```


Herencia y Clase Abstracta II

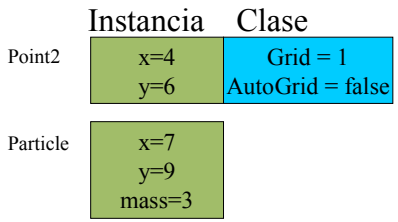
```
class Circulo extends Figura {
   double radio;
   Circulo(double r) {
          radio = r;
   }
   double perimetro() {
          return 2*Math.PI*radio;
   }
   double area() {
          return Math.PI*radio*radio;
   }
}
```

Herencia y Clase Abstracta III

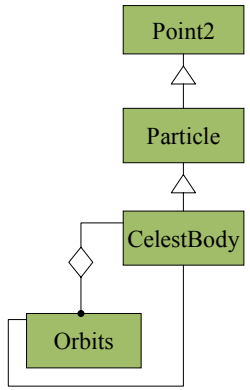
```
class EjHerencia {
   static public void main(String [] args) {
     Figura [] f = new Figura [2];
     f[0] = new Cuadrado(12);
     f[1] = new Circulo(7);

     for(int i=0;i<2;i++) {
       System.out.println("Figura "+i);
       System.out.println("\t"+f[i].perimetro());
       System.out.println("\t"+f[i].area());
     }
   }
}
```


Herencia y Variables de Clase



Cuerpos celestes



Interfaces

- Mecanismo incorporado por Java para suplir la herencia múltiple
- Una interface es un conjunto de declaraciones de funciones bajo un nombre.
- Una clase que implemente una interface, tiene la obligación de implementar las funciones que dicha interface declara.
- Es posible crear variables de tipo interface
  - Pueden recibir los mensajes definidos en la interface

Ejemplo Interfaz I

```
interfaz Agrandable {
       void zoom(int i);
}
class Circulo extends Figura implements Agrandable {
       ....
       public void zoom(int i) {
              radio *= i;
       }
       ....
}
```


Ejemplo Interfaz II

```
class Punto implements Agrandable {
       ....
       public void zoom(int i) {
              x *= i;
              y *= i;
       }
       ...
}
```

- Las clases Punto y Circulo no estan en la misma jerarquía de herencia

Ejemplo Interfaz III

```
class EjInterface {
   static public void main(String [] args) {
     Agrandable [] f = new Agrandable [2];
     f[0] = new Circulo(12);
     f[1] = new Punto(7,4);

     for(int i=0;i<2;i++) {
        f[i].zoom(3);
     }
   }
}
```


Herencia de interfaces

- Una interface puede extender a uno o varios interfaces más

```
public interface A extends B, C {
    ....
}
```

- Una clase puede implementar todos los interfaces que quiera

```
class ClaseA extends ClaseB
                    implements intA, IntB, IntC {
    ....
}
```

Clases Anidadas

- Clases que se definen en el contexto de otra clase
  - Clase (o Interface) Interna estática
  - Clase Interna no estática
  - Clase Local
  - Clase Anónima

Clase Interna (o Interface) estática (Poco interés)

- Declaradas como estáticas dentro de una clase
- Su nombre externo será:

  nombredelaclase.nombredelainterna
  - Para crear un objeto

```
new nombredelainterna
new nombredelaclase.nombredelainterna
```

  - Al compilar se creara

  nombredelaclase$nombredelainterna.class

Clase Interna no estática

- Una instancia de la interna puede solo existir en el contexto de una instancia de la clase
- Los métodos pueden referenciar a las variables de la clase que la contiene
- Puede llevar accesibilidad
- Para crear un objeto

```
new nombredelainterna
objeto.new nombredelainterna
```

- Al compilar

  nombredelaclase$nombredelainterna
  - Ejemplo:Ver Pila

Clases Locales (Poco interés)

- Se definen en el contexto de un bloque
  - Su visibilidad queda reducida al bloque que la define
  - No puede ser declarada static (Ni sus variables)
  - Sus variables no pueden tener accesibilidad
  - Tiene acceso a las variables y métodos de la clase que la define
  - Al compilar

    nombredelaclase1nombredelalocal.class

Clases Anónimas I

- Clases de las que sólo se crea una instancia
- Se definen en el momento de la creación de la instancia
  - Extendiendo otra clase
  - Implementando una interface
- Para ello, utilizan extensiones del operador new
- Al compilar se les da un numero correlativo

  nombredelaclase$numero.class

Clases Anónimas II Extendidendo una clase

```
new <superclase> (<Lista de parametros>) {<definición>}
```

- Ejemplo

```
class A {
  void metodo() {
    System.out.println("El metodo de A");
  }
}
class PAnonima {
  public static void main(String [] args) {
    A a = new A() {
              void metodo() {
                System.out.println("El metodo anonimo");
                super.metodo();
              }
    };
    a.metodo();
  }
}
```


Clases Anónimas III Implementando una interface

```
new <interface> () {<definición>}
```

- Ejemplo

```
interface I {
  public void metodo();
}

class PAnonima2 {
  public static void main(String [] args) {
    I i = new I() {
              public void metodo() {
                System.out.println("El metodo anonimo");
              }
    };
    i.metodo();
  }
}
```


Paquetes

- Conjunto de clases almacenadas todas en un mismo directorio
  - La jerarquía de directorios coincide con la jerarquía de paquetes
    - Los nombre de los paquetes deben coincidir con los nombre de los directorios (CLASSPATH)
  - El nombre completo de una clase es el del paquete seguido por el nombre de la clase
  - Para indicar la pertenencia a un paquete se utiliza la palabra reservada package

Cómo usar un paquete

- Se utiliza la palabra reservada `import` seguido del nombre del paquete

```
import java.util.Vector;
import java.awt.*;
import java.lang.*;  // por defecto siempre está
```

- Puede importarse una clase o el paquete
  - Si se quiere inportar un subpaquete hay que hacerlo explícitamente

```
import java.awt.*;
import java.awt.event.*;
```


Control de Acceso I

- Hay cuatro niveles para los miembros de una clase

Visibilidad en:

Nivel de Acceso	clase	subclase	paquete	todos
private	X			
protected	X	X	X	
public	X	X	X	X
package (nada)	X		X	

Control de Acceso II

- De otra forma

	private	package (nada)	protected	public
Misma clase	SI	SI	SI	SI
Subclase en paquete	NO	SI	SI	SI
Otra clase en paquete	NO	SI	SI	SI
Subclase en otro paquete	NO	NO	SI	SI
Otra clase en otro paquete	NO	NO	NO	SI

- Las clases pueden ser
  - public: accesible por todos
  - package (nada): accesible dentro del paquete

Crear un paquete

- Debe estar todos los ficheros que lo componen en un mismo directorio cuyo nombre coincidirá con el del paquete
- Deben comenzar con
  - package <nombre del paquete>
  - Por norma, los nombres de los paquetes empiezan por minúsculas
  - Si un paquete se encuentra en `formato\util` entonces el nombre del paquete es `formato.util`
- Ejercicio: crear un paquete con las listas

Paquetes básicos

- Java proporciona una librería completa de clases en forma de paquetes
  - `java.lang`: para funciones del lenguaje
  - `java.util`: para utilidades adicionales
  - `java.io`: para entrada y salida
  - `java.text`: para formato especializado
  - `java.awt`: para diseño gráfico e interaz de usuario
  - `java.awt.event`: para gestionar eventos
  - `javax.swing`: nuevo diseño de GUI
  - `java.net`: para comunicaciones
  - cientos más que van estandarizándose según versiones

El Paquete `java.lang`

- Siempre está incluido aunque no se diga
- Contiene a las clases
  - `Object`
  - `System`
  - `Math`
  - `Character`
  - `String` y `BufferString`
  - Envoltorios de tipos básicos

La clase Object

- Es la clase superior de toda la jerarquía de clases de Java
  - Si una definición de clase no extiende a otra, entonces extiende a Object
- Métodos de instancia importantes
  - `boolean equals(Object)` Hacer ejemplo
  - `String toString()` Hacer ejemplo
  - `void finalize()`
  - `Class getClass()`
  - ... (mirar documentación)

Reflexión. La clase Class

- Permite interrogar sobre caracteristicas de una clase
  - Métodos, constructores, interfaces, superclase, etc.
  - Conocer el nombre de la clase de un objeto
    - `String getName()`
  - Crear una instancia de esa clase
    - `Object newInstance()`
  - Saber si un objeto es de la familia de una clase
    - `boolean isInstance(Object)`
      `//  instanceOf`

La clase System

- Maneja particularidades del sistema
- Tres variables de clase (estáticos) públicas
  - `PrintStream out, err`
  - `InputStream in`
- Métodos de clase (estáticos) públicos
  - `void exit(int)`
  - `void gc()`
  - `void runFinalization()`
  - ... Mirar la documentación

La clase Math

- Incorpora como **métodos del clase (estáticos)** constantes y funciones matemáticas
- Constantes
  - `E, PI`
- Métodos (no se indican los argumentos ni tipos)
  - `abs(), sin(), cos(), tan(), asin(), acos(), atan()`
  - `max(), min(), exp(), pow(), sqrt(), round()`
  - `random()`, ... Mirar la documentación

La clase String

- Manipula cadena de caracteres **constantes**
- Permite la creación de literales sin mas que definirlos
- Métodos de instancia definidos
  - `char charAt(int),`
  - `boolean equalsIgnoreCase(String)` ....
  - `int length()`
  - El operador `+`
  - ... Mirar la documentación

La clase StringBuffer

- Manipula cadena de caracteres permitiendo su actualización
- Métodos de instancia
  - `StringBuffer append(...)`
  - `void setCharAt(int,char)`
  - `int length()`
  - `int capacity()`
  - el operador `+`
  - ... Mirar la documentación

Las clases envoltorios (wrappers)

- Supongamos que tenemos un array de tipo Object. ¿Qué podemos meter en el array?
  - Los tipos básicos no son objetos por lo que no pueden introducirse en ese array.
  - Para ello se utiliza un envoltorio

int Integer

Byte	Byte
float	Float
short	Short
double	Double
int	Integer
long	Long
char	Character
bool	Boolean

Los envoltorios numéricos

- Constructores crean envoltorios a partir de los datos númericos o cadenas de caracteres
- Métodos de clase para crear números y envoltorios a números a partir de cadenas de caracteres
  - `xxxx parseXxxx(String).`
  - `Exxxx valueOf(String)`
- Métodos de instancia para extraer el dato numérico del envoltorio
  - `xxxx xxxxValue()`

Ejemplos de envoltorios numéricos

```
int a = Integer.parseInt("34");

Double d   = new Double("-45.8989");
double dd  = d.doubleValue();

double ddd = Double.parseDouble("32.56");

Long l = Long.valueOf("27.98");
```

Se produce la excepción `NumberFormatException` si algo va mal

Ejercicio: Listas para cualquier objeto

Envoltorio `Boolean`

- Constructores crean envoltorios a partir de datos boleanos o cadenas de caracteres
- Método de clase para crear un envoltorio boleano a partir de cadenas de caracteres
  - `Boolean.valueOf(String) //si va mal, false`
- Métodos de instancia para extraer el dato booleano del envoltorio
  - `booleanValue()`

```
boolean b = (new Boolean("false")).booleanValue();
```

Envoltorio `Character`

- Constructor que crea un envoltorio a partir de un caracter
- Métodos de instancia para extraer el dato caracter del envoltorio
  - `caracterValue()`
- Métodos de clase para testear caracteres

```
isDigit(char)   isLetter(char)    isLowerCase(char)
  isUpperCase(char)     isSpaceChar(char)    ....
```

- Métodos de clase para convertir caracteres

```
toLowerCase(char)   toUpperCase(char)   ...
```


El Paquete `java.util`

- Contiene clases de utilidad
  - Las colecciones. Dos:
    - Arrastradas de versiones anteriores
    - Un nuevo marco introducido desde JDK1.2
  - La clase `StringTokenizer`
  - La clase `Random`
  - Otras ... Mirar documentación

El Paquete `java.util` Colecciones antiguas

- Java disponía de un pequeño conjunto de colecciones en el paquete `java.util`
- Vector
  - crece dinámicamente
  - `addElement(), contains(), indexOf(), insertElement()`
  - `elementAt(), firstElement(), lastElement(), isEmpty()`
- Stack
  - push(), pop(), empty()
- Hashtable
  - put(), get(), size(), isEmpty()

El Paquete `java.util` Ejemplo Colecciones

```
import java.util.Hashtable;

class EjHT {
   static public void main(String [] args) {
      Hashtable h = new Hashtable();
      String s;
      h.put("casa","home");
      h.put("copa","cup");
      h.put("lapiz","pen");
      s = (String)h.get("copa");
      System.out.println(h.get("copa"));
      System.out.println(h);
   }
}
```

El Paquete `java.util` Ejercicio Colecciones

- Dada una cadena como con una expresión prefija. Calcular su valor
  - `" 7 3 - 2 * 6 4 2 - + *" -> 64`
  - utilizar una pila para almacenar los números
  - `java Evalua 7 3 - 2 "*" 6 4 2 - + "*"`

El Paquete `java.util` Iteradores sobre colecciones

- Todas las colecciones implementan el método

```
Enumeration  elements()
```

- Enumeration es una interface

```
interfaz Enumeration {
      boolean hasMoreElements();
      Object nextElement();
}
```

- Una clase de usuario puede implementar lal interface `Enumeration`

El Paquete `java.util` Ejemplo `Enumeration`

```
class Evalua {
   ....

   static void listaPila() {
       Enumeration e = p.elements();
       System.out.print("< ");
       while (e.hasMoreElements())
          System.out.print(e.nextElement()+" ");
       System.out.println(">");
   }
}
```

- Ejercicio: Implementar `Enumeration` en las listas genéricas

El Paquete `java.util` `StringTokenizer`

- Permite la realización de un pequeño analizador para una cadena de caracteres dada
- En el constructor se proporciona la cadena y opcionalmente, los delimitadores
- Luego, se maneja a través de los métodos
  - `boolean hasMoreTokens()`
  - `String nextToken()`
- ver ST.java

El Paquete `java.util` Ejemplo `StringTokenizer`

```
import java.util.StringTokenizer;
class ST {
   static public void main(String [] args) {
       StringTokenizer st =
         new StringTokenizer("El agua:es:buena"," :");
       while (st.hasMoreTokens()) {
          System.out.println(st.nextToken());
       }
   }
}
```

- Ejercicio: Realizar la calculadora anterior pero con una llamada como
  - `java EvaluaST "7 3 - 2 * 6 4 2 - + *"`

El Paquete `java.util` `Random`

- Permite generar números aleatorios de diversas formas:
  - `float nextFloat()`
  - `double nextDouble()`
  - `int nextInt(int n)` `0 <= res < n`
  - `double nextGaussian()`

Excepciones

- Situaciones anómalas que aparecen durante la ejecución de un programa
  - No se trata de problemas externos (corte de luz, rotura del disco, etc)
  - Se trata de problemas debido al software (división por cero, conversión inadecuada, ficheros que no existen, etc)
- Java permite detectar (`try`) estas excepciones y manejarlas (`catch`)

Clases Excepciones

- `Throwable`
  - `Error (*)`
    - `....(*)`
  - `Exception`
    - `RuntimeException`
      - `ArithmeticException`
      - `IndexOutOfBoundsException`
        - » `ArrayIndexOutOfBoundsException`
      - `IllegalArgumentException`
        - » `NumberFormatException`
      - `NullPointerException`
    - `IOException (*)`
      - `FileNotFoundException (*)`
      - `MalformedURLException (*)`
    - `InterruptedException (*)`
    - `....  (*)`

(*) es obligatorio capturarlas

Throwable

- Esta clase y sus herederas disponen del constructor por defecto y otro que admite un argumento String
- Métodos de instancia

```
String getMessage()
void printStackTrace()
void printStackTrace(PrintWriter s)
void printStackTrace(PrintStream s)
```


Ejemplo sin excepciones

```
errorCodeType readFile {
    initialize errorCode = 0;
    abre el fichero;
    if (seAbre) {
        determina la longitud del fichero;
        if (conseguidaLongitud) {
            localiza suficiente memoria;
            if (conseguidaMemoria) {
                lee el fichero en memoria;
                if (fallaLectura) { errorCode = -1;} //bien
            } else { errorCode = -2;}
        } else { errorCode = -3;}
        cierra el fichero;
        if (noCierra && errorCode==0){errorCode = -4;}
        else { errorCode = errorCode and -4; }
    } else { errorCode = -5;}
    return errorCode;
}
```


Ejemplo con excepciones

```
readFile {
  try {
       abre el fichero;
       determina su tamaño;
       localiza suficiente memoria;
       lee el fichero en memoria;
       cierra el fichero;
  } catch (FileOpenFailed e) {
       ...;
  } catch (SizeDeterminationFailed e) {
       ...;
  } catch (MemoryAllocationFailed e) {
       ...;
  } catch (ReadFailed e) {
       ...;
  } catch (FileCloseFailed e) {
       ...;
  }
}
```


El bloque `try-catch`

- Si tenemos:

```
String s = "HOLA";
n = Integer.parseInt(s);
```

- Se produce una excepción. Para capturarla

```
try {
  n = Integer.parseInt(s);
} catch (NumberFormatException e) {
  System.err.println("Error en número. Tomo 0");
  n = 0;
}
```

- Ver ejemplo Cambio.java y CambioE.java

Lanzar una excepción

- Cuál podría ser la definición de la función `parseInt()`

```
public static int parseInt(String s) throws
   NumberFormatException {
   ......
   if ( hay error )
        throw new NumberFormatException();
   .....
}
```


Prototipo de funciones y captura de excepciones

- El prototipo de una función indica cuáles excepciones puede lanzar

```
public void miOperacion(String dato)
    throws NumberFormatException, ArithmeticException
```

- Igualmente, se pueden capturar varias excepciones en un mismo bloque (ver Excp1.java)

```
try {
   miOperacion(campo.getText())
} catch (ArithmeticException ae) {
  System.err.println("Error: División por cero");
} catch {NumberFormatException nfe) {
  System.err.println("Error: Mal numero");
}
```


Propagar excepciones

- Un método puede
  - capturar una excepción
  - capturar una excepción y propagarla

```
catch (ArithmeticException ae) {
   ....
   throw ae;
}
```

  - propagarla (ver Excp3.java)
    - no tratándola y definiendo en el prototipo que se puede lanzar

```
static public void miFun() throws
   InterruptedException {
   ...// código que puede provocar la interrupción
}
```


Generar excepciones

- Un método también puede generar una excepción

```
static public char miFun(String s) throws
   NullPointerException {
   ....
   if (s==null)
     throw new NullPointerException("String nulo");
   ....
}
```

- Ver Excp4.java

Excepciones propias

- Un usuario puede definir sus propias excepciones

```
class MiExcepcion extends Exception {
  MiExcepcion(int i) {
   super((new Integer(i)).toString)
  }
}
```

- Y ahora puede lanzarse como las demás

```
...
  throw new MiExcepcion(5);
...
```


Retraso en la ejecución

- La clase `Thread` entre otras cosas, permite producir retrasos en la ejecución

```
try {
  Thread.sleep(5000);
} catch (InterruptedException e) {
}
```

Produce un retraso de 5000 milisegundos

`finally`

- Puede indicarse un código que se ejecutará ocurra una excepción o no.

```
try {
  ....
} catch (Exception e) {
  ...
} finally {
  ...
}
```


Ejercicio Amortización

- Entrada:
  - CapitalInicial, InterésAnual y NúmeroDePagos

$$pagoMensual = \frac{CapitalInicial * InterésMensual}{1 - (1 + InteresMénsual)^{-NúmeroDePagos}}$$

interesPeriodo = capitalDeudor*interesMensual;
capitalAmortizadoPeriodo = pagoMensual-interesPeriodo;
capitalRestante = capitalDeudor - capitalAmortizadoPeriodo;
interesesAcumulados+=interesPeriodo;

nPago	CapDeudor	IntPeriodo	CapAmortPeriodo	CapRestante	IntAcumulados
1	3000000	25000	15000	2960000	25000
2	2960000	23000	17000	2920000	48000
...					

Clases en Amortización

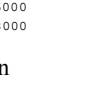
- EntradaTabla
  - Contiene la información de una línea
- Amortiza
  - En su constructor se les pasa
    - CapitalInicial, InterésAnual, NúmeroDePagos
  - Dispone de un método
    - Vector tablaCalculos()

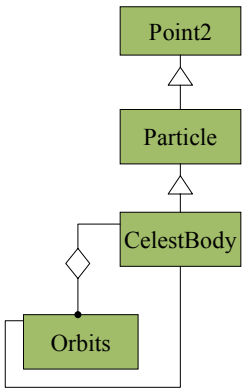
    que devuelve un vector con los objetos EntradaTabla necesarios
- AmortizaConsola
  - Toma los datos como argumentos y muestra el resultado

```
java AmortizaConsola 3000000 7 60
nPago    CapDeudor IntPeriodo CapAmortPeriodo CapRestante IntAcumulados
1        3000000   25000      15000           2960000     25000
2        2960000   23000      17000           2920000     48000
...
```

- Las entradas y salidas pueden ser formateadas con Terminal

Ejercicio Cuerpos Celestes





Point2D

Point2	
float x; float y;	float Grid; boolean AutoGrid;

float x();
float y();
float x(float vx);
float y(float vy);
float distance(Point2 p);
void move(float dx, float dy);
void rotate(float aNumber);
void rotateWrtCenter(float aNumber,Point2D center);
String toString();
	void grid(float aGrid);
	void enableGrid();
	void disableGrid();
	snapToGrid(float v);
	Point2(float vx, float vy);
	Point2();

```
public void rotate(float aNumber) {
    double nSin = Math.sin(aNumber);
    double nCos = Math.cos(aNumber);
    float tempX = (float) (x*nCos+y*nSin);
    float tempY = (float) (y*nCos-x*nSin);
    this.x(tempX);
    this.y(tempY);
}
```


Particle

Particle	
float mass;	

float mass();
String toString();
void mass(float vmass);
	Particle(float vx, float vy, float vmass)
	Particle(float vx, float vy)

CelestBody

CelestBody	
Vector orbits;	

void addWithPeriod(CelestBody aCB, int aPer);
void rotateWrtCenter(float aNumber,Point2D center);
String toString();
void tick();
void displayAtZoom(Graphics g,Point2 center,int zoom);
	CelestBody(float vx, float vy, float vmass);

```
public void tick(){
    Enumeration e = orbits.elements();
    while (e.hasMoreElements())
        ((Orbit)e.nextElement()).tick(this);
}

public void rotateWrtCenter(float aNumber, Point2D center) {
    super.rotateWrtCenter(aNumber,center);
    Enumeration e = orbits.elements();
    while (e.hasMoreElements()) {
        CelestBody satellite =
                    ((Orbit)e.nextElement()).satellite();
        satellite.rotateWrtCenter(aNumber,center);
    }
}
```


Orbit

Orbit	
int period; CelestBody satellite	
CelestBody satellite(); String toString(); void tick(Point2D center)	Orbit(CelestBody aCB, int aPer);

```
public void tick(Point2 center) {
    satellite.rotateWrtCenter((float)Math.PI*2/period,center);
    satellite.tick();
}
```


Cuerpos Celestes

```
import CelestBody;
class Simu1 {
  CelestBody sysMain;
  public static void main(String [] args) {
    (new Simu1()).exec();
  }
  Simu1() {
    sysMain = new CelestBody(0,0,22);
    sysMain.addWithPeriod(new CelestBody(30,10,6),40);
    sysMain.addWithPeriod(newCelestBody(20,30,10),90);
  }
  public void exec() {
    for(int tick=1;tick<10;tick++) {
       sysMain.tick();
       System.out.println(sysMain);
    }
  }
}
```