

medición de tiempo de ejecución (ejemplo en Java)

```
class Timer {
    private long[] startTime;
    private long[] stopTime;

    Timer(int n) {
        startTime=new long[n];
        stopTime=new long[n];
    }
    public void Start(int i) {
        startTime[i]=System.nanoTime();
    }
    public void Stop(int i) {
        stopTime[i]=System.nanoTime();
    }
    public double Elapsed() {
        long minTime=startTime[0];
        for(int i=1; i<startTime.length; ++i)
            if(startTime[i]<minTime) minTime=startTime[i];
        long maxTime=stopTime[0];
        for(int i=1; i<stopTime.length; ++i)
            if(stopTime[i] > maxTime) maxTime=stopTime[i];
        return (maxTime-minTime)/1000000.0;
    }
}
```

El famoso *hola mundo* se programa en Java así:

```
class Hello {  
    public static void main(String[] args) {  
        System.out.println(  
            "Hello world, this is "+args[0]  
        );  
    }  
}
```

El programa principal se llama `main()` y tiene que ser declarado público y estático. No devuelve ningún valor (por eso se declara como `void`).

Los parámetros de la línea de comando se pasan como un vector de cadenas de letras (`String`).

```
javac Hello.java
```

```
java Hello primero segundo tercero
```

¿Qué se comenta?

- Se usa **doxygen** (o javadoc, u otro bueno) para generar automáticamente la documentación. Todos tienen unos comandos para aumentar la documentación.
- Existen varias posibilidades de escribir comentarios:

//	comentario de línea
/// ...	comentario de documentación
/* ... */	comentario de bloque
/** ... */	comentario de documentación

- Se documenta sobre todo lo que no es obvio, las interfaces (en el sentido amplio de la palabra), y los casos límite.
- Es decir: Los comentarios son las respuestas a preguntas del *¿Cómo?* y del *¿Por qué?*.

- Se usan los hilos para ejecutar varias secuencias de instrucciones de modo cuasi-paralelo.
- Es decir, la ejecución parece ser en paralelo, pero si es realmente paralelo (y no secuencial) depende del hardware (más preciso, del número de CPUs involucrados) que se está usando en cada momento.

creación de un hilo (para empezar)

- Se **crea** un hilo con
`Thread worker = new Thread()`
- Se inicializa el hilo y se define su comportamiento.
- Normalmente se usa una clase derivada o la interfaz `Runnable`.
- Se lanza el hilo con
`worker.start()`
- Pero en esta versión simple no hace nada. Hace falta sobrescribir el método `run()` especificando algún código útil.
- ¿Cuándo **arranca de verdad** este nuevo hilo?

- A veces no es conveniente extender la clase `Thread` porque se pierde la posibilidad de extender otro objeto.
- Es una de las razones por que existe la interfaz `Runnable` que declara nada más que el método `public void run()` y que se puede usar fácilmente para crear hilos trabajadores.

- Thread, Runnable
- join, isAlive, activeCount
- try-catch-finally
- volatile
- synchronized
- wait, notify, notifyAll
- finalize
- transient
- java.util.concurrent
- java.util.concurrent.atomic
- etc.

- Para facilitar la programación de casos excepcionales Java usa el concepto de lanzar excepciones.
- Una excepción es una clase predefinida y se accede con la sentencia

```
try { ... }  
catch(SomeExceptionObject e) { ... }  
catch(AnotherExceptionObject e) { ... }  
finally { ... }
```


- El bloque `try` contiene el código normal por ejecutar.
- Un bloque `catch (ExceptionObject)` contiene el código excepcional por ejecutar en caso de que durante la ejecución del código normal (que contiene el bloque `try`) se produzca la excepción del tipo adecuado.
- Pueden existir más de un (o ningún) bloque `catch` para reaccionar directamente a más de un (ningún) tipo de excepción.
- Hay que tener cuidado en ordenar las excepciones correctamente, es decir, las más específicas antes de las más generales.

- El bloque `finally` **se ejecuta siempre** una vez terminado o bien el bloque `try` o bien un bloque `catch` o bien una excepción no tratada o bien antes de seguir un `break`, un `continue` o un `return` hacia fuera de la sentencia `try-catch-finally`.
- Por eso es un buen punto donde ejecutar código en aplicaciones concurrentes que deben realizar cierto *trabajo* final, incluso en casos excepcionales.

Normalmente se extiende la clase `Exception` para implementar clases propias de excepciones, aún también se puede derivar directamente de la clase `Throwable` que es la superclase (interfaz) de `Exception` o de la clase `RuntimeException`.

```
class MyException extends Exception {  
    public MyException() { super(); }  
    public MyException(String s) { super(s); }  
}
```

- Entonces, una excepción no es nada más que un objeto que se crea en el caso de aparición del caso excepcional.
- La clase principal de una excepción es la interfaz `Throwable` que incluye un `String` para mostrar una línea de error legible.
- Para que un método pueda lanzar excepciones con las sentencias `try-catch-finally`, es imprescindible declarar las excepciones posibles antes del bloque de código del método con `throws`

```
public void myfunc(...) throws MyException {...}
```

- En C++ es al revés, se declara lo que se puede lanzar como mucho.

- Durante la ejecución de un programa se propagan las excepciones desde su punto de aparición subiendo las invocaciones de los métodos hasta que se haya encontrado un bloque `catch` que se ocupa de tratar la excepción.
- En el caso de que no haya ningún bloque responsable, la excepción será tratada por la máquina virtual con el posible resultado de abortar el programa.

- Se pueden lanzar excepciones directamente con la palabra `throw` y la creación de un nuevo objeto de excepción, por ejemplo:

```
throw new MyException("this is an exception");
```

- También los constructores pueden lanzar excepciones que tienen que ser tratados en los métodos que usan dichos objetos construidos.

- Además de las excepciones así declaradas existen siempre excepciones que pueden ocurrir en cualquier momento de la ejecución del programa, por ejemplo, `RuntimeException` o `IndexOutOfBoundsException`.
- La ocurrencia de dichas excepciones refleja normalmente un flujo de control erróneo del programa que se debe corregir antes de distribuir el programa a posibles usuarios.

- **Recomendación:** Se usan excepciones solamente para casos excepcionales, es decir, si pasa algo no esperado.
- Excepciones en programas concurrentes se convierten rápidamente en **pesadillas**.
Ya que aflorece el problema que hacer si muchos procesos empiezan a lanzar excepciones individualmente...
- Mirad **Safe (Disciplined) Exception Handling principle** con sus dos posibilidades de acción
 - fallo: re-establecer una invariante necesaria
 - re-intentar: re-hacer la operación con (quizá) cierta modificación