Tema 5: Diseño y realización de pruebas

Errare Humanum Est

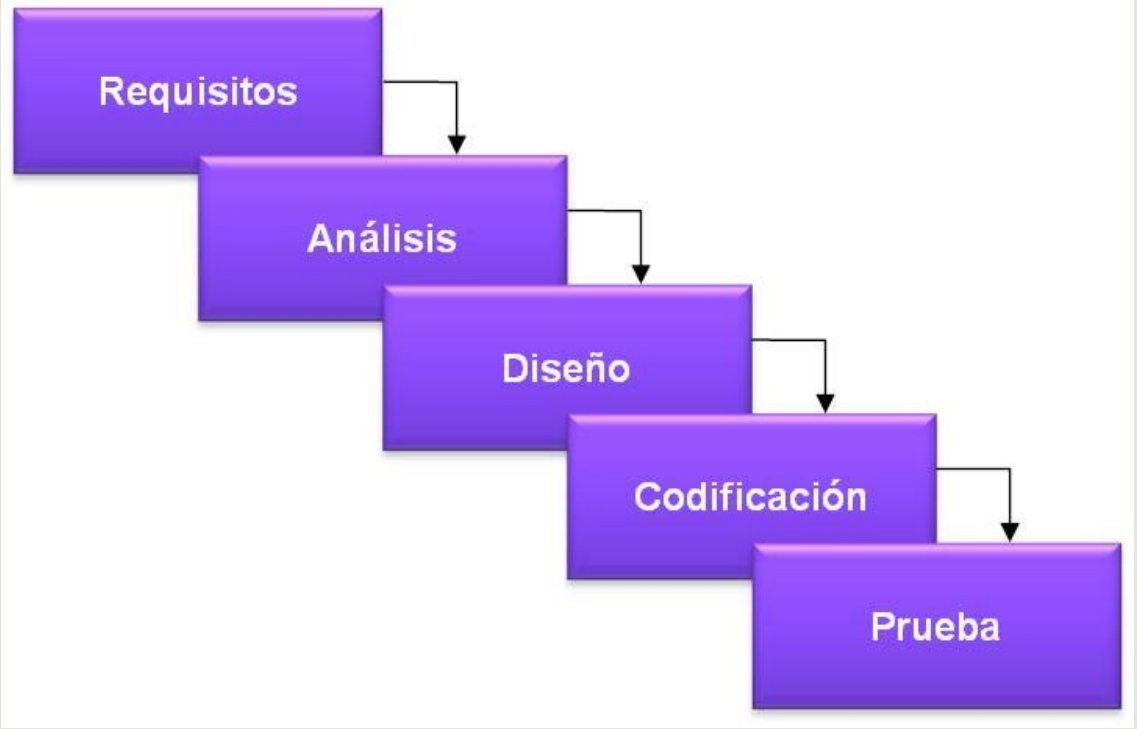
Introducción

- Validacion y verificación. Ciclo de vida
- Caja negra y caja blanca
- Pruebas funcionales, estructurales y de regresión

Introducción

La prueba es el proceso de ejecutar un programa con el objetivo de encontrar errores

Un caso de prueba es bueno cuando su ejecución conlleva una probabilidad elevada de encontrar un error y tiene éxito cuando lo detecta.



Introducción

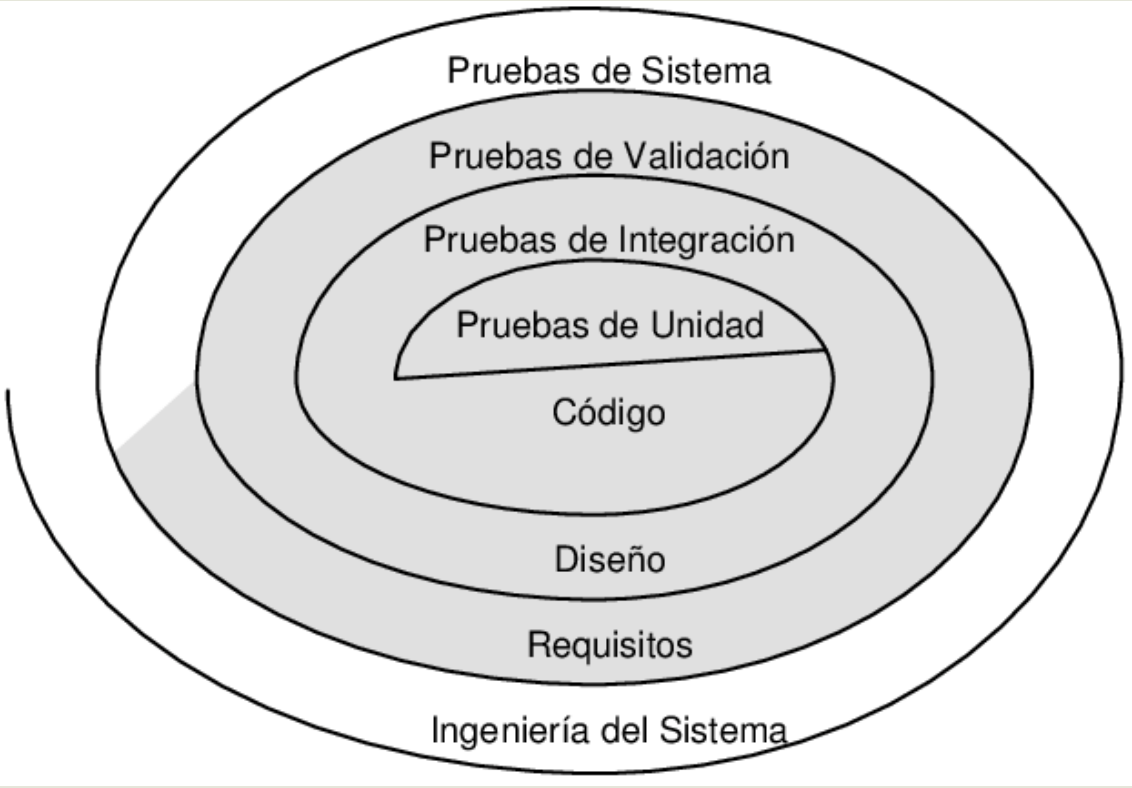
- Mediante la realización de pruebas se produce la verificación y la validación del software
- Verificación: *¿Estamos construyendo el producto correctamente?* ¿Cumple los requerimientos especificados?
- Validación: *¿Estamos construyendo el producto correcto?* ¿El producto funciona según lo esperado?

Introducción

Para hacer las pruebas se necesita llevar a cabo alguna estrategia. Una de ellas es utilizar el Modelo en Espiral.

Para ello nuestra aplicación debe pasar secuencialmente:

- **Pruebas de unidad**
- **Pruebas de Integración**
- **Pruebas de Validación**
- **Pruebas de Sistema**



Introducción: Modelo en Espiral

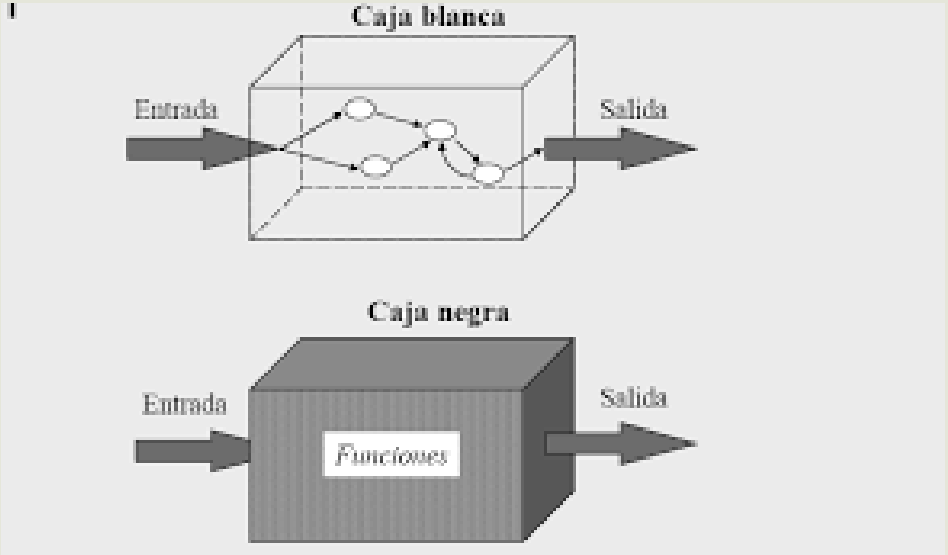
- Pruebas de Unidad: En esta parte se realizan pruebas sobre cada módulo, función, clase, etc. De nuestro programa. Su objetivo es eliminar errores en la lógica interna.
- Para ello se utilizan técnicas de Caja Negra y Caja Blanca.
- Se realizan pruebas de unidad sobre:
  - La interfaz
  - Estructuras de datos, para asegurarse que mantienen su integridad
  - Condiciones límite
  - Caminos de la estructura de control

Introducción: Modelo en Espiral

- Pruebas de integración: se observa como interactúan los distintos módulos. El ensamblaje de los módulos se puede dar todos a la vez (no incremental) o por pequeños segmentos (incremental)
- Pruebas de validación: se comprueba que el software funciona según lo esperado. Pruebas Alfa y Pruebas Beta.
- Pruebas del sistema: comprueba el funcionamiento total del software y otros elementos del sistema
- Mantenimiento

Tipos de prueba

- A la hora de realizar un caso de prueba necesitamos saber ciertas cosas: necesitamos identificar unos valores de entrada y conocer el comportamiento esperado con dichos valores. Según su tipo tenemos dos grandes casos.
- Pruebas de Caja Negra (Funcionales): Son pruebas que se realizan sobre la aplicación sin conocer su funcionamiento interno. Lo fundamental es comprobar si los resultados coinciden con los esperados.
- Pruebas de Caja Blanca (Estructurales): Son pruebas que se realizan dentro, siguiendo la lógica de la aplicación.



Tipos de prueba

- Un ejemplo de estas dos pruebas para una calculadora podría ser:
- **Pruebas de caja negra**: meter en la calculadora los valores 3+7,8/2,$\sqrt{25}$ y comprobar que da los valores esperados.
- **Pruebas de caja blanca**: ir al código asociado a cada operación y seguir paso a paso los pasos que realiza para cada valor.
- En las primeras pruebas no nos preocupamos del algoritmo y su eficiencia, o de si hay código innecesario, si ocupa mucha memoria, etc. Es lo estudiamos en las segundas.

Tipos de prueba: Pruebas Funcionales (Caja Negra)

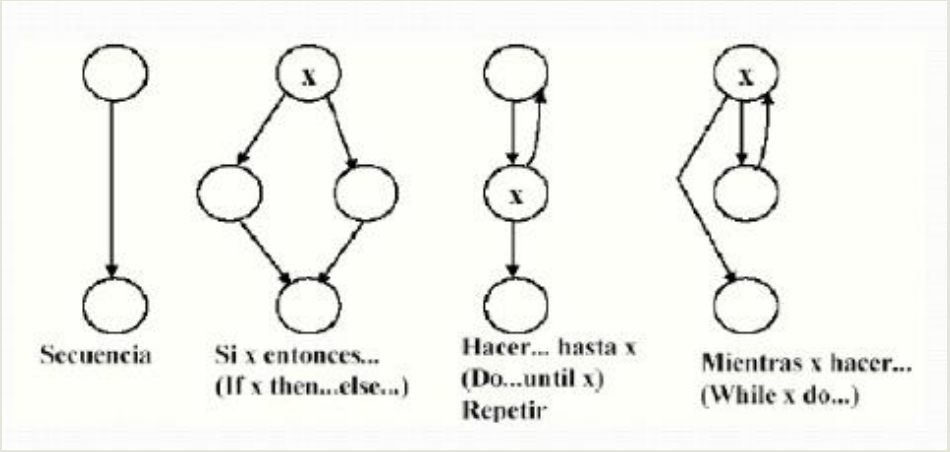
- En las pruebas funcionales: Intentamos comprobar si las salidas que devuelve la aplicación son las esperadas. ¿Puede el usuario hacer esto? ¿Funciona esta utilidad? No nos preocupamos de la estructura interna de la aplicación. Tenemos principalmente tres tipos de pruebas funcionales:
- **Particiones equivalentes**: se trata de realizar el mínimo número de pruebas agrupando los casos de prueba en clases equivalentes. La verificación de un caso de prueba para una clase serviría también para verificar el funcionamiento del resto de los casos. EJ: hacer 6-3 es parecido a hacer 10-4 y 16-9 (todos dan un resultado positivo), mientras que hacer 7-11 es lo mismo que 14-23 (resultado negativo?

Tipos de prueba: Pruebas Funcionales (Caja Negra)

- **Análisis de valores límite**: trata de probar los casos extremos a las clases de equivalencia. PJ: comprobar la división entre cero -> 16/0=?
- **Pruebas aleatorias**: consiste en generar aleatoriamente entradas para la aplicación que hay que comprobar. Se suele utilizar generadores de pruebas que son capaces de crear un volumen de casos de prueba.
- Existen más tipos de pruebas pero todas tienen el mismo objetivo: comprobar el funcionamiento usando solo la interfaz.

Tipos de prueba: Pruebas Estructurales (Caja Blanca)

- Las Pruebas estructurales tienen en cuenta el código de la aplicación y se fijan en los caminos que puede seguir. No pretenden comprobar la corrección de los resultados, su función es comprobar que se ejecutan todas las instrucciones del programa, no hay código sin usar, se recorren todos los caminos, etc.
- Para ello se siguen unos criterios de cobertura lógica como se ven a continuación.



Tipos de prueba: Pruebas Estructurales (Caja Blanca)

- **Cobertura de sentencias:** hay que asegurarse de que tenemos casos suficientes como para que cada instrucción sea ejecutada al menos una vez.
- **Cobertura de decisiones:** cada prueba lógica debe evaluarse al menos una vez a cierto y una vez a falso.
- **Cobertura de condiciones:** cada condición se debe evaluar a falso y verdadero
- **Cobertura de condiciones y decisiones:** realizar las dos a la vez
- **Cobertura de caminos:** el más importante. Se trata de recorrer al menos una vez cada secuencia de sentencias encadenadas, desde el principio del programa al final. A la secuencia de sentencias se le conoce como camino.

Tipos de prueba: Pruebas de Regresión

- Un proceso de prueba será exitoso y encontramos algún error. Como consecuencia esto provocará una modificación de una parte del código, esta modificación puede generar errores que antes no existían. Es por ello que debemos repetir pruebas que antes habíamos realizado.
- Este es el objetivo de las pruebas de regresión. Se deben llevar a cabo cuando se modifica el sistema y se deben llevar a cabo en todos los componentes, no solo en los componentes modificados.
- Normalmente se realiza repitiendo las pruebas ya realizadas previamente. Para ello se utilizan métodos se generación de pruebas automáticos. Para ser eficiente, se deben realizar el conjunto de pruebas que traten uno más errores en cada una de las funciones. No es práctico realizar todas las pruebas de cada función después de un cambio.

Pruebas de Código

- Consiste en la ejecución de un programa con el objetivo de encontrar errores en el código. Necesitamos definir unos casos de entrada y resultados esperados.
- Se utilizan tres enfoques:
  - **Estructural**: Nosotros usaremos pruebas de camino básico
  - **Funcional:** Se centra en valores límite y clases de equivalencia
  - **Aleatorio:** utilizando generadores automáticos de casos de prueba.

Pruebas de Código: Camino básico

- Permite medir la complejidad lógica del diseño y usar esa medida como guía para la definición de unos caminos básicos a seguir. Los caminos obtenidos aseguran que cada secuencia de sentencias se ejecuta al menos una vez.
- Para conseguir la complejidad lógica (complejidad ciclomática) se utilizará un grafo de flujo
- Las estructuras de control vienen a continuación

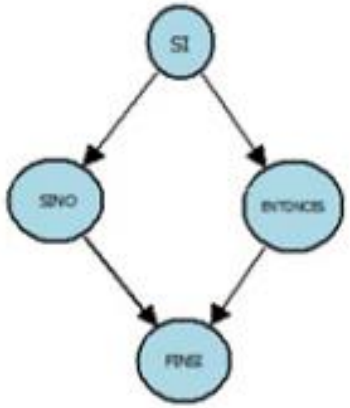
Pruebas de Código: Camino básico

SECUENCIAL

Instrucción 1
Instrucción 2
...
Instrucción n

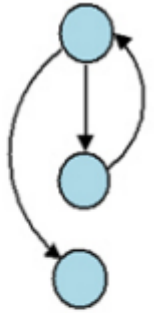
CONDICIONAL

SI <condición> ENTONCES
    <Instrucciones>
SINO
    <Instrucciones>
FIN SI



HACER MIENTRAS

MIENTRAS <condicion> HACER
    <Instrucciones>
FIN MIENTRAS



REPETIR HASTA

REPETIR
    <Instrucciones>
HASTA QUE <condicion>

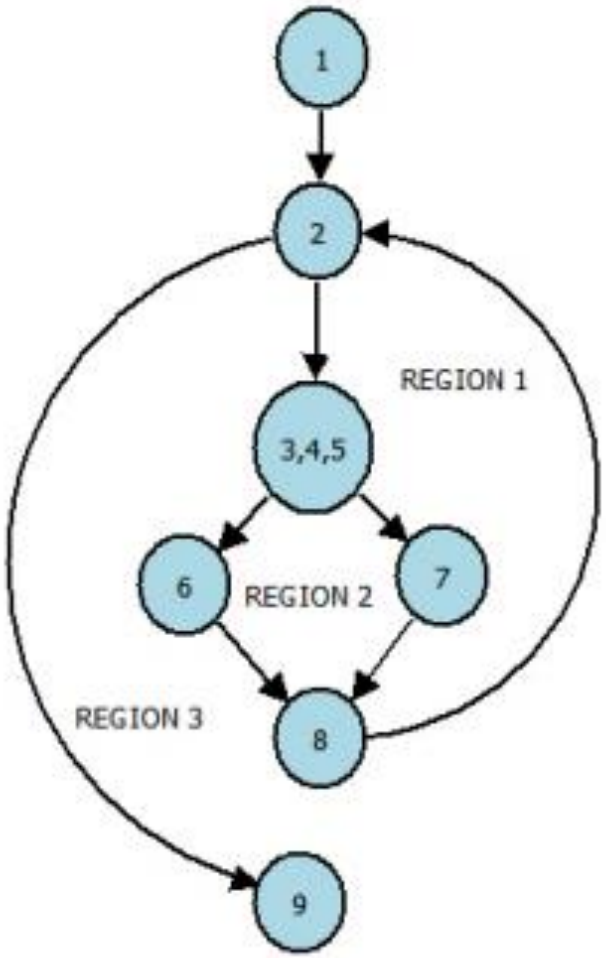
Pruebas de Código: Camino básico

- Cada círculo se llama **nodo** y representa uno o más sentencias.
- Las fechas se llaman **aristas** y representan el flujo de control
- Un nodo que contiene una decisión o condición se llama **nodo predicado** y se caracterizan porque de el salen más de una arista.

Pruebas de Código: Camino básico

En el grafo de al lado se pueden ver 7 nodos, con 8 aristas.

Tiene dos nodos predicados, el “3,4,5” y el “2” ya que de ellos SALEN más de una arista.

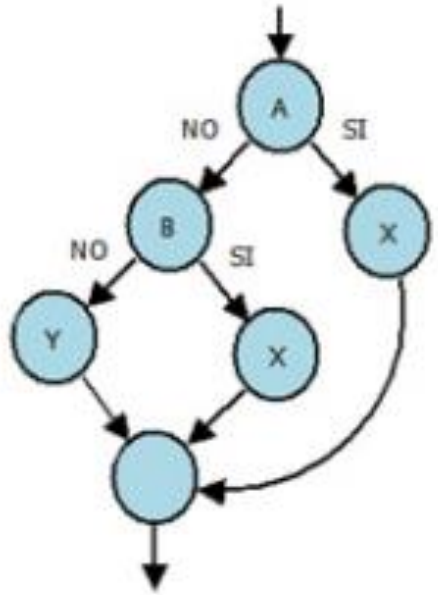


Pruebas de Código: Camino básico

- Cuando nos encontramos con una condición compuesta. Se crea un nodo para cada condición. Por ejemplo aquí si tiene una condición de OR

```
if A or B then
    sentencia X
else
    sentencia Y
end
```

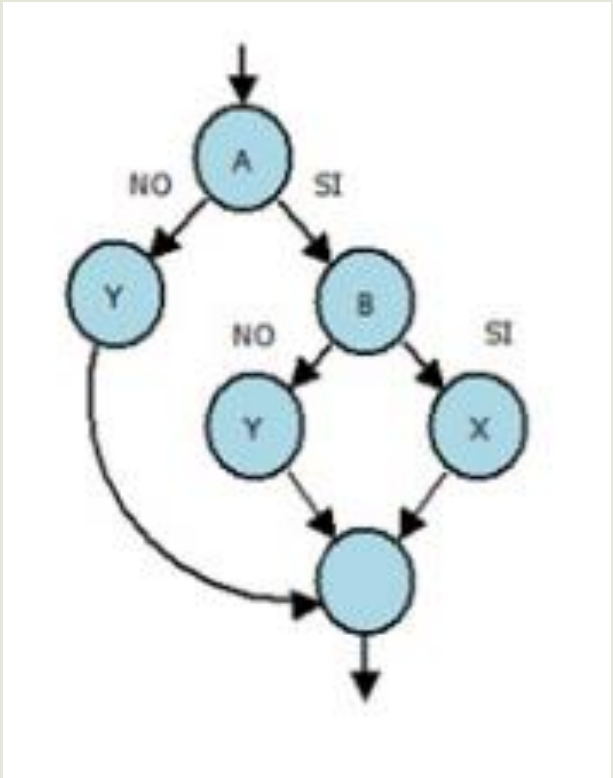
```
SI A OR B ENTONCES
    PROC X
SINO
    PROC Y
FIN SI
```



Pruebas de Código: Camino básico

- Aquí se tiene una condición AND

```
if A and B then
     sentencia X
else
     sentencia Y
end
```



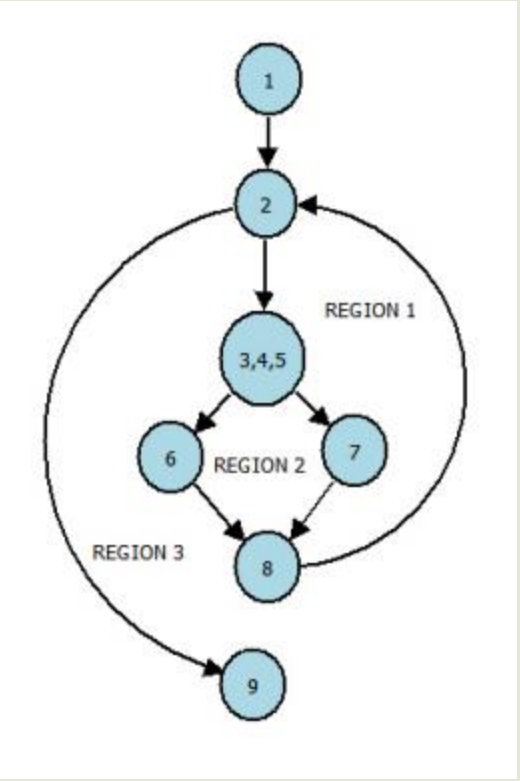
Pruebas de Código: Camino básico

- En un grafo se puede medir la complejidad ciclomática y sirve para medir de cuantas maneras se puede recorrer el grafo. La fórmula para calcular la complejidad es
- V(G)=Nº de aristas – Nº de nodos +2
- V(G)=Nº de nodos predicado + 1
- La complejidad nos dice cuántos casos de prueba se necesitan hacer para ese segmento de código.

Pruebas de Código: Camino básico

- En este ejemplo la complejidad es 3

- V(G)=Nº de aristas – Nº de nodos +2=
  =8-7+2=3
- V(G)=Nº de nodos predicado + 1=2+1=3



Pruebas de Caja Negra: Clases de equivalencia

- Las clases de equivalencia es una estrategia de pruebas de caja negra que pretende cubrir el mayor número de entradas posibles. Consiste en dividir todos los casos de entrada posibles en grupos llamados clases de equivalencia. Hacer la prueba con un valor de una clase de equivalencia equivaldría a hacerlo con cualquier otro valor de la clase.
- Las clases de equivalencia deben comprender **entradas válidas** y **no válidas**. Así pues, tenemos varias opción según a donde pertenezca una condición de entrada.
  - Pertenece a un valor específico, debemos crear un valor válido y otro no válidos
  - Pertenece a un rango, debemos crear tres clases: por debajo, en el rango y por encima.
  - Si la prueba es lógica debemos crear dos pruebas: una falsa y otra cierta.

Pruebas de Caja Negra : Clases de equivalencia

- Ejemplo:

```
public static void calculaEdad(int edad){
        if(edad<18) System.out.println("Eres menor de edad");
        if(edad>=18 && edad<=65) System.out.println("Estás en edad de trabajo");
        if(edad>65) System.out.println("Estás en edad de jubilación");
}
```

- Este es un programa que te calcula en que etapa de tu vida estás. Hay **tres** tipos de resultados posibles y por tanto podemos crear **tres** clases de equivalencia distintas.
  - Clase 1: cualquier número menor que 18
  - Clase 2: números entre 18 y 65 (ambos incluidos)
  - Clase 3: números mayores que 65
- Hacer una prueba con cualquier elemento de la clase 1, equivale a hacerlo con los demás. Es decir, si hago la prueba con el número 3 es lo mismo que si la hago con el número 8,13,17,2,11,etc. Puesto que todos pertenecen a la clase 1.
- Pregunta: ¿Cómo tratamos a las edades negativas?

Pruebas de Caja Negra : Valores límite

- El análisis de los valores límites se basa en que los valores tienden a producirse más en los valores extremos de entrada. Estos valores extremos se encuentran justo por encima y por debajo de las clases de equivalencia.
  - Si una clase requiere un rango de valores se debe aplicar casos de prueba para los límites del rango y para los valores justo por encima y justo por debajo del rango. Por ejemplo si una condición requiere valores de entrada entre 1 y 10, debemos probar los valores: 1, 10, 0 y 11.
  - Si una condición de entrada especifica una cantidad de valores, debemos hacer pruebas con el valor máximo, mínimo, y los valores justo por encima del máximo y por debajo del mínimo. Por ejemplo, si un programa requiere entre 10 y 100 valores para hacer una estimación, debemos hacer pruebas pasándole 10, 100, 9 ,101 valores.