T4 - REFACTORIZACIÓN Y DOCUMENTACIÓN

DEFINICIÓN
- modificación del cod. sin alterar su funcionalidad
- Objetivo → crear un cod + { claro / sencillo } { eliminar cod. redundante / limpiar cod / modularizar }

Proceso
- cambios simples y peq. { mover propiedades / crear nuevos métodos }
- evitar introducir errores al reescribir/modificar partes del cod.
- realizar como paso aislado en el diseño del programa.

1° Ver cod. funciona (pruebas) → 2° Cambios → 3° Pruebas

(=)

BAD SMELLS

- **DUPLICATED CODE** → bloques cod repetido / muy parecido
  sol → extraer cod = método
- **LONG METHOD** → dificultan comprensión y mantenución
  sol → dividir método → peq responsabilidades
- **LARGE CLASS** → muchas respons., métodos y atributos.
  sol → dividir " "
- **ATRIBUTO PÚBLICO EN UNA CLASE** → privado { gets / sets }
  públicos SOLO! → { atbos ctes }
- **CLASE DE DATOS** → atbos + gets + sets → NO COMPORTAMIENTO
  cuestionarse ?? → innecesarias
- **DIVERGENT CHANGE** → clase q necesita modificarse frecuentemente x razones diversas
  indica → clase demasiadas tareas = DIVIDIDA / ELIMINADA
- **SHOTGUN SURGER** → ocurre cuando un cambio requiere varias modificaciones adicionales en dif. partes del cod. x ser compatible.
- **FEATURE ENVY** (Envidia de funcionalidad) → una clase q utiliza + métodos/datos de otra clase que de la propia
- **REFUSED BEQUEST** (Legado Rechazado) → subclase hereda de otra pero utiliza pocas características de la superclase, lo q sugiere un error en la jerarquía de clases.

- CÓDIGO NO IDENTADO
- NOMBRES DE VARIABLES Y MÉTODOS → deben proporcionar info. sobre su función
  ↳ convención de nombres
  ↳ comentar < mins < variables / métodos
  MAYUS → clases

MOMENTOS PARA REFACTORIZAR (debería ser habitual)

↳ DESPUÉS DE APLICAR CAMBIOS AL SOFTWARE → ayuda a comprender mejor cod y mejorar su estructura
↳ AL RESOLVER UN FALLO
↳ DURANTE LAS REVISIONES DE COD

MOMENTOS PARA NO REFACTORIZAR

1. Cuando cod NO FUNCIONA = Reescribir
2. CERCA DE UNA ENTREGA

↳ PATRONES COMUNES DE REFACT.

- RENOMBRADO → nombres +significativos
- SUSTITUIR BLOQ. COD x UN MÉTODO = reutilización y legibilidad
- CAMPOS ENCAPSULADOS < gets / sets
- MOVER LA CLASE → si se necesita en otro dsp. de cod < nuevo paquete / proyecto ...
- BORRADO SEGURO → todas las referencias del cod borrado
- CAMBIAR LOS PARÁMETROS DEL PROYECTO → menos param. y cambiar mod. de acceso
- EXTRAER LA INTERFAZ →
- MOVER DEL INTERIOR A OTRO NIVEL

ANALIZADORES DE COD.

↳ ANÁLISIS ESTÁTICO → evalúa si softw sigue las normas de estilo predefinidas
facilita incorporación nuevos miembros + mantenimiento
usa herramientas → LINTER
analizadores léxicos y sintácticos, reglas...

1º PROCESAN COD FUENTE
2º DETECTAN ESTRUCT. MEJORABLES
3º INDICAN MEJORAS ÁREAS Y POSIBLES SOLUCIONES

- FindBugs en Netbeans
- Manualmente
- JAVA = PDM

REFACTORIZACIÓN EN LOS ENTORNOS DE DESARROLLO

↳ HERRAMIENTAS → todo lo anterior se puede hacer manualmente = Errores
x ello los entornos de desarrollo implementan herr.

menos pérdida tiempo ← → menos errores / redundancia

1. RENAME
2. EXTRACTING → EXTRACT INTERFACE → metodos clase → interfaz (def metodos de una clase para los desarrollo)
→ EXTRACT SUPERCLASS →
3. CHANGE A METHOD SIGNATURE → cambiar def. de un método → param. entrada, salida, nombre
4. MOVE → lo lee = cambia referencias
5. COPY
6. SAFELY DELETE → borra (clase, fichero) → busca sus referencias en otras clases → pedir borrarlos
7. INLINE → ajustar referencia a una variable → linea de cod.

```
int x = 1;
int j = 2;        ⟹   int x = 1;
x = x + j               x = x + 2;
```

8. ENCAPSULAR CAMPOS
9. CONVERT A LOCAL VARIABLE IN FIELD
10. PULL UP, PUSH DOWN → mover métodos en jerarquía de clases.

DESARROLLO GUIADO POR PRUEBAS (TDD)

- Programadores implementan pruebas unitarias antes escribir cod. real
↳ desarrollan peq unidades y pasan