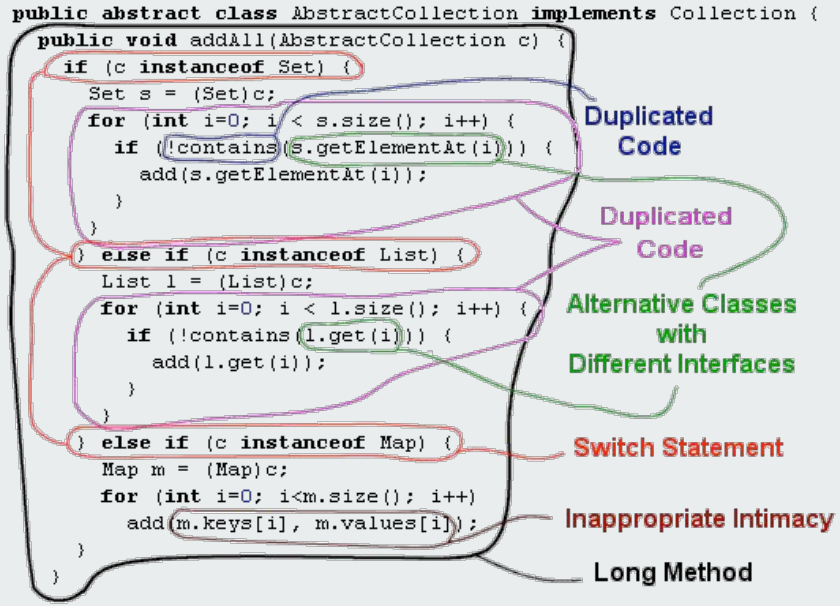
# Refactoring

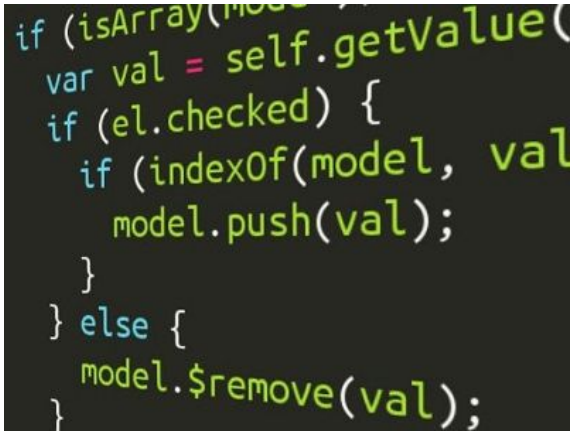


# ¿Qué es Refactoring?

- Una vez que has acabado de escribir un código, lo mejoras.
- Se añaden cambios para mejorar el código escrito:
  - Buenas prácticas
  - Clean Code
  - Comentarios

# Refactorización (Refactoring)

- Técnica de Ing de Software para optimizar un código escrito.
- Se hacen cambios internos que no deben afectar al funcionamiento externo.
- Se mejora el código sin cambiar funcionalidad
  - NO se añade funcionalidad: Eso es desarrollar
  - NO se arreglan errores: Eso es debug
- El objetivo es código limpio y modularizado

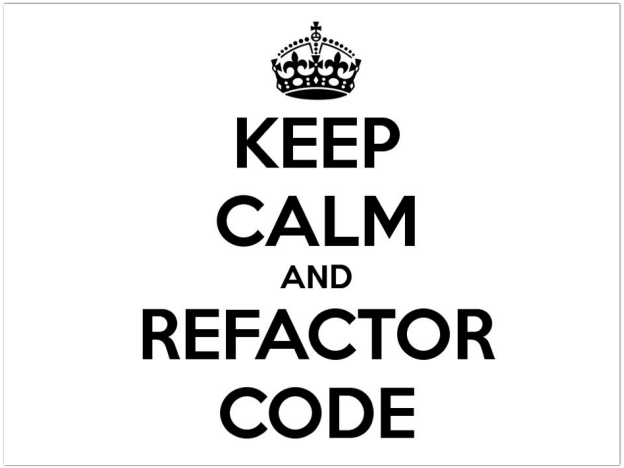


# Refactorizar

- Se mejora el código para facilitar nuevos desarrollos futuros simplificando el código que ya existe permitiendo que otras personas lo comprendan con más facilidad
- En la fase de refactorización es donde realmente aplicamos todos los aspectos vistos de código limpio: aumentar cohesión, dividir clases, modularizar aspectos, elegir nombres adecuados...
- Es muy difícil hacer un código limpio a la primera, así que es necesario refactorizar después de que el código funcione, y es necesario hacerlo antes de pasar a la siguiente tarea.

# Ventajas de Refactorizar

- Revisar diseño del sistema
- Plantearse cambios en el futuro
- Incremento de facilidad de lectura del código
- Mejor comprensión del código
- Detección y solución temprana de fallos
- Mayor velocidad de desarrollo
- No todo se puede refactorizar:
  - Bases de datos: No sería refactorizar, sino migrar estructura y datos



# ¿Cuándo refactorizar?: Malos olores (Bad Smells)

- Código duplicado (duplicated code):
  - si el mismo código está en más de un sitio, se mete en una función que se llama desde dos sitios: Aplicable a todo desde CSS en adelante
- Métodos muy largos (Long method):
  - Cuanto más largo es un método, más difícil es de entender.
  - Seguramente se pueda subdividir
  - Cuanto más corto es, más posibilidades tiene de reutilización.
- Lista de parámetros muy larga (Long parameter list):
  - Más de dos o tres parámetros son muchos
  - Se debe encapsular en clases toda esa información
  - Se reconoce porque hay parámetros que siempre van juntos => tienen entidad común

# ¿Cuándo refactorizar?: Malos olores (Bad Smells)

- Clases muy grandes (Large class):
  - Una clase con demasiados métodos o atributos
  - Tendrá demasiadas funcionalidades
  - Subdividir en clases, herencia…
- Cambio divergente (Divergent change):
  - Una clase que empezó siendo de una manera y ahora no la reconoce ni su madre
  - La clase no cumple su propósito original o incluso ningún propósito: obsoleta, desfasada
  - La clase se debe subdividir o eliminar
- Cirugía con escopeta (Shotgun surgery):
  - Si al modificar una clase hay que cambiar código en otros sitios
  - Mala granularización

# ¿Cuándo refactorizar?: Malos olores (Bad Smells)

- Envidia de funcionalidad (Feature envy):
  - Un método de una clase que utiliza muchos datos de otras clases
  - El método se debe mover a la clase que tanto usa y pasar la clase en la que está declarado como parámetro.
- Clase de sólo datos (Data class):
  - Una clase que sólo tiene atributos y métodos de acceso get/set
  - No tienen comportamiento
  - Se puede estudiar mover el contenido a otra clase
  - No siempre
- Herencia rechazada (Refused bequest):
  - Clases hija que no utilizan casi nada la clase padre
  - La herencia está mal diseñada