**EXAMEN ENTORNO DE DESARROLLO**

1. En este ejercicio aplicaremos la técnica "Extract Method". El código mantiene su funcionalidad, pero ahora resulta más legible. Al extraer el método, este se vuelve reutilizable en otras partes del sistema.

public class Actividad1 {

void imprimirFactura() {

imprimirEncabezado();

imprimirDetalles(getCantidad());

}

void imprimirDetalles(Double cantidad) {

System.out.println("Nombre: " + nombre);

System.out.println("Cantidad: " + cantidad);

}

}

1. En este ejercicio aplicaremos "Inline Method". Es fundamental eliminar el método original una vez que su contenido ha sido incorporado directamente, ya que no será utilizado nuevamente.

public class Actividad2 {

int obtenerCalificaciones() {

return (numeroDeRetrasos > 5) ? 2 : 1;

}

}

1. Una estructura condicional permite seleccionar diferentes comportamientos según el tipo de un objeto. La solución consiste en convertir la estructura condicional en un método mediante "Extract Method". Cada opción condicional se transformará en una versión sobrescrita del método dentro de la clase derivada correspondiente.

abstract class Empleado {

abstract double getSalario();

}

class Ingeniero extends Empleado {

double getSalario() {

return salarioBase + productividad;

}

}

class Vendedor extends Empleado {

double getSalario() {

return salarioBase + ventas \* comision;

}

}

class Director extends Empleado {

double getSalario() {

return salarioBase + bonificacion + dietas;

}

}

1. Cuando se emplean tipos enumerados para diferenciar variantes de un tipo, puede ser conveniente mejorar el código mediante subclases.

public class Shape {

private double size;

public Shape(double size) {

this.size = size;

}

public double getSize() {

return size;

}

}

public class Square extends Shape {

public Square(double size) {

super(size);

}

public double area() {

double size = getSize();

return size \* size;

}

}

public class Circle extends Shape {

public Circle(double size) {

super(size);

}

public double area() {

double size = getSize();

return Math.PI \* size / 4.0;

}

}

1. La solución a este ejercicio es la misma que en prácticas, haciendo uso de etiquetas como:
   * @see para referenciar la documentación de otra clase o método.
   * @version para indicar la versión de una clase.
   * @author para registrar el nombre del autor.
   * @since para especificar la fecha de incorporación de una clase.
   * @param para describir los parámetros de un método.
   * @return para indicar el valor devuelto por un método.
   * @throws para documentar excepciones lanzadas.
   * @deprecated para señalar métodos obsoletos.
2. Renombrado: Asignar nombres más descriptivos a paquetes, clases, métodos o campos.
3. **Sustitución de bloques de código por métodos:** Transformar fragmentos de código en métodos reutilizables.
4. **Encapsulación de campos:** Implementar métodos getter y setter para un acceso controlado a los campos de una clase.
5. **Reubicación de clases:** Mover clases entre paquetes o proyectos para evitar duplicaciones.
6. **Eliminación segura:** Borrar código innecesario asegurándose de eliminar todas sus referencias.
7. **Modificación de parámetros:** Ajustar parámetros de métodos y sus modificadores de acceso.
8. **Extracción de interfaz:** Crear una interfaz a partir de métodos seleccionados en una clase.

7. Los analizadores de código incluyen:

* 1. Analizadores léxicos.
  2. Analizadores sintácticos.
  3. Reglas aplicables a ciertas estructuras.

Sus funciones principales son:

* 1. Identificar fragmentos de código que reduzcan el rendimiento.
  2. Detectar errores potenciales en el software.
  3. Evaluar la complejidad del código.
  4. Analizar el flujo de datos.
  5. Identificar vulnerabilidades de seguridad.

8.

* El **análisis estático** evalúa el software sin ejecutarlo.
* Se realiza sobre el código fuente para detectar mejoras sin modificar su semántica.
* Se utilizan herramientas específicas llamadas **analizadores estáticos**.
* Un analizador estático de código:
  + Recibe el código fuente.
  + Lo procesa para determinar su funcionalidad.
  + Sugiere mejoras o identifica problemas.
* El análisis puede realizarse de forma automática o manual.
* **PMD** es un analizador de código estático que detecta errores y vulnerabilidades de manera preventiva.
* **Repositorio:** Espacio de almacenamiento para datos de proyectos.
* **Módulo:** Sección específica dentro del repositorio, que puede representar una parte o la totalidad del proyecto.
* **Revisión:** Versión parcial o modificación del código en el repositorio.
* **Etiqueta:** Información textual añadida a archivos o módulos para destacar datos importantes.
* **Rama:** Línea de desarrollo paralela que permite realizar cambios sin afectar la versión principal.
* **Identificación:** Establecer estándares de documentación y nomenclatura.
* **Control de cambios:** Evaluar y registrar modificaciones en la configuración del software.
* **Auditorías de configuración:** Garantizar la correcta implementación de cambios mediante revisiones técnicas.
* **Generación de informes:** Documentar la evolución de los distintos elementos del software para asegurar su consistencia.