

# Assignment Kit for Coding Standard



---

## Personal Software Process for Engineers: Part I

The Software Engineering Institute (SEI)  
is a federally funded research and development center  
sponsored by the U.S. Department of Defense and  
operated by Carnegie Mellon University.

This material is approved for public release.  
Distribution limited by the Software Engineering Institute to attendees.

# Personal Software Process for Engineers: Part I

## Assignment Kit for the Coding Standard

### Overview

---

#### Overview

This assignment kit covers the following topics.

Section	See Page
Prerequisites	2
Objectives	2
Coding standard requirements	3
Example coding standard	4
Evaluation criteria and suggestions	7
Coding standard template	8

---

#### Prerequisites

- Prerequisites
- Read Chapter 4
  - Complete Size Counting Standard
- 

#### Objectives

- The objectives of the coding standard are to
- establish a consistent set of coding practices
  - provide criteria for judging the quality of the code that you produce
  - facilitate size counting by ensuring your programs are written so they can be readily counted
  - for LOC counting, require that there be a separate physical line for each logical line of code
-

## Coding standard requirements

---

### **Coding standard requirements**

Produce, document, and submit a completed coding standard that calls for quality coding practices.

For LOC counting, ensure that a separate physical source line is used for each logical line of code.

Submit the coding standard with your program 2 assignment package.

---

## Example coding Standard

---

### Coding standard example

Pages 5 and 6 of this workbook contain an example C++ coding standard.

Notes about the example

- Since it is an example, tailor it to meet your personal needs.
- If you have an existing organizational standard, consider using it for the PSP exercises.

---

*Continued on next page*

## Evaluation criteria and suggestions

---

**Evaluation  
criteria**

Your standard must be

- complete
  - legible
- 

**Suggestions**

Keep your standards simple and short.

Do not hesitate to copy or build on the PSP materials.

---

## Coding Standard Template

Purpose	Guiar el desarrollo de programa 3a en Java para que sea legible, mantenible y consistentes, (clases App, Input, Output, Data, Logic, EstimacionCorLineal).
Program Headers	Todos los programas comienzan con una cabecera descriptiva
Header Format	<pre> /***** *****  * Programa 4a: Simpson Integration of t Distribution * Clase:      &lt;NombreDeLaClase&gt; * Autor:      Diego Noe Roldan Vivanco * Fecha:      AAAA-MM-DD * * Descripción: *   &lt;Breve descripción de lo que hace esta clase&gt;. *   &lt;Si es necesario, dividir la descripción en varias líneas alineadas&gt;. * *   - (Opcional) Lista de ideas clave o variables importantes.  *****/ </pre>

Listing Contents	Proporciona un resumen de la lista de contenidos
Contents Example	<pre> /*  * Contenido del archivo:  *   - Atributos principales:  *       + intNumSeg          : número de segmentos de Simpson  *       + dblW               : ancho de subintervalo  *       + dblE               : error aceptable  *       + intDOF             : grados de libertad  *       + dblX               : límite superior de integración  *       + dblTotXi           : arreglo de puntos Xi  *       + dblFirstBaseTerms : términos base (1 + Xi^2 / dof)  *       + dblFxi             : valores de f(Xi)  *       + dblFinalTerms      : términos finales con multiplicadores 1,4,2,...  *       + dblFinalValue      : valor final de la integral  *   - Dependencias:  *       + GammaFunction gammaFunction  *   - Constructor:  *       + SimpsonIntegration(int intDOFIn, double dblXIn)  *   - Métodos públicos principales:  *       + void computeW(int intNumSegIn, double dblXIn)  *       + void computeXi(int intNumSegIn)  *       + void computeFirstBaseTerms(int intNumSegIn, double[] dblXi, int intDOFIn)  *       + void computeExponent(int intDOFIn)  *       + void computeCoefficient(int intDOFIn)  *       + void computeFxi(int intNumSegIn)  *       + void computeFinalTerms(int intNumSegIn)  *       + double computeFinalValue(int intNumSegIn)  *       + double getFinalValue()  */ </pre>
Reuse Instructions	<ul style="list-style-type: none"> <li>• Describe cómo se utiliza el programa o la clase. Proporciona el formato de declaración, los valores y tipos de parámetros y los límites de esos parámetros.</li> <li>• Proporciona advertencias sobre valores no válidos, condiciones de error, o cualquier situación que pueda producir un funcionamiento incorrecto.</li> </ul>

Reuse Example	<p>Ejemplo basado en el método principal de integración en SimpsonIntegration:</p> <pre> /**  * Método principal de esta clase: calcula el valor  * final de la integral  * con la regla de Simpson para un número de segmentos  * dado.  * &lt;p&gt;  * Sigue la secuencia:  * &lt;ol&gt;  *   &lt;li&gt;computeW&lt;/li&gt;  *   &lt;li&gt;computeXi&lt;/li&gt;  *   &lt;li&gt;computeFirstBaseTerms&lt;/li&gt;  *   &lt;li&gt;computeExponent&lt;/li&gt;  *   &lt;li&gt;computeCoefficient&lt;/li&gt;  *   &lt;li&gt;computeFxi&lt;/li&gt;  *   &lt;li&gt;computeFinalTerms&lt;/li&gt;  *   &lt;li&gt;Suma y multiplicación final por (w / 3)&lt;/li&gt;  * &lt;/ol&gt;  *  * @param intNumSegIn número de segmentos (debe ser  * par) .  * @return valor aproximado de la integral desde 0 hasta  * x.  */ public double computeFinalValue(int intNumSegIn) { </pre>
Identifiers	<p>Utiliza nombres descriptivos para todas las variables, nombres de métodos, constantes y otros identificadores. Evita abreviaturas crípticas o variables de una sola letra, excepto en bucles sencillos.</p>

Identifier Example	<p>Ejemplos tomados de GammaFunction y SimpsonIntegration:</p> <pre>// Correctos (descriptivos) private int    intNumSeg;           // número de segmentos de Simpson  private double dblW;               // ancho de cada subintervalo  private double dblE;               // error aceptable  private int    intDOF;             // grados de libertad  private double[] dblTotXi;         // puntos Xi  private double[] dblFirstBaseTerms; // términos base (1 + Xi^2 / dof)  private double[] dblFxi;           // valores de f(Xi)  private double[] dblFinalTerms;    // términos finales ponderados  private double  dblFinalValue;     // valor final de la integral  private double  gammaValue;        // último Gamma(x) calculado  // Constantes en App (correctas) private static final String INPUT_FILE  = "test.txt"; private static final String OUTPUT_FILE = "test_resultados.txt";  // Incorrectos (no descriptivos, a evitar) int n1; double v; String s;</pre>
Comments	<ul style="list-style-type: none"> <li>• Documenta el código de forma que el lector pueda entender su funcionamiento.</li> <li>• Los comentarios deben explicar tanto el propósito como el comportamiento del código.</li> <li>• Comenta las declaraciones de variables cuando su propósito no sea obvio.</li> </ul>

Good Comment	<pre> /** Número de segmentos usados en la integración de Simpson. */ private int intNumSeg;  /** Ancho de cada subintervalo: <math>w = x / \text{num\_seg}</math>. */ private double dblW;  /** Grados de libertad de la distribución t. */ private int intDOF;  /**  * Calcula el ancho de subintervalo w en función del número  * de segmentos  * y del límite superior x.  *  * @param intNumSegIn número de segmentos (debe ser par para  * Simpson).  * @param dblXIn límite superior de integración.  */ public void computeW(int intNumSegIn, double dblXIn) {     this.intNumSeg = intNumSegIn;     this.dblX      = dblXIn;     this.dblW      = dblXIn / (double) intNumSegIn; } </pre>
Bad Comment	<pre> // calcula w dblW = dblX / (double) intNumSeg;    // &lt;-- comentario redundante  // for recorre todos los términos finales for (int i = 0; i &lt;= intNumSeg; i++) { // &lt;-- el for ya se entiende solo     dblSum += dblFinalTerms[i]; } </pre>

## Coding Standard Template (continued)

Major Sections	Precede las secciones principales del programa con un bloque de comentarios que describa el procesamiento que se va a realizar a continuación.
Example	<p>Ejemplo basado en la estructura de SimpsonIntegration:</p> <pre> /*-----*/ -----*/  /*  Attributes */  /*-----*/ -----*/  /** Número de segmentos usados en la integración de Simpson. */ private int intNumSeg;  /** Ancho de cada subintervalo: w = x / num_seg. */ private double dblW;  /*-----*/ -----*/  /*  Public Services */  /*-----*/ -----*/  /**  * Calcula el ancho del subintervalo w y guarda x y num_seg.  */ public void computeW(int intNumSegIn, double dblXIn) {     ... }  /**  * Calcula el valor final de la integral usando la regla de Simpson.  */ public double computeFinalValue(int intNumSegIn) {     ... } </pre>
Blank Spaces	<ul style="list-style-type: none"> <li>• Escribe los programas con suficiente espacio en blanco para que no se vean amontonados.</li> <li>• Separa cada construcción del programa con al menos un espacio.</li> <li>• Ejemplo: if (intNumSeg &gt; 0), for (int i = 0; i &lt;= intNumSeg; i++).</li> <li>• Deja líneas en blanco entre bloques lógicos (por ejemplo, entre cálculo de Xi, cálculo de términos base y suma final en SimpsonIntegration).</li> </ul>

Indenting	<ul style="list-style-type: none"> <li>• Sangra (indenta) cada nivel de llaves respecto al anterior.</li> <li>• Las llaves de apertura y cierre deben estar en líneas propias y alineadas entre sí, siguiendo el estilo que usas en tu proyecto</li> </ul>
Indenting Example	<pre> public double computeFinalValue(int intNumSegIn) {     double dblSum = 0.0;      /* Secuencia lógica de cálculo siguiendo el diagrama */     computeW(intNumSegIn, dblX);     computeXi(intNumSegIn);     computeFirstBaseTerms(intNumSegIn, dblTotXi, intDOF);     computeExponent(intDOF);     computeCoefficient(intDOF);     computeFxi(intNumSegIn);     computeFinalTerms(intNumSegIn);      // Sumar todos los términos finales     for (int intI = 0; intI &lt;= intNumSegIn; intI++)     {         dblSum += dblFinalTerms[intI];     }      // Aplicar el factor final de Simpson     dblFinalValue = (dblW / 3.0) * dblSum;     return dblFinalValue; } </pre>
Capitalization	<ul style="list-style-type: none"> <li>• Escribe todas las constantes static final en mayúsculas con guiones bajos.</li> <li>• Usa camelCase para variables y métodos (intNumSeg, dblFirstBaseTerms, computeFinalValue, computeDblGamma).</li> <li>• Los nombres de clase usan PascalCase (App, GammaFunction, SimpsonIntegration, Output, Logic).</li> <li>• Los mensajes mostrados al usuario pueden ir en mayúsculas y minúsculas mezcladas para una presentación más clara.</li> </ul>

**Capitalization  
Example**

Ejemplo basado en App.java:

```
public class App
{
    private static final String INPUT_FILE = "test.txt";
    private static final String OUTPUT_FILE =
"test_resultados.txt";

    public static void main(String[] args)
    {
        System.out.println("Ejecutando programa 5a:
integración de t.");

        // Creación de módulos
        Output output = new Output();
        GammaFunction gammaFunction = new GammaFunction();
        SimpsonIntegration simpson = new
SimpsonIntegration(10, 1.10);

        // Aquí iría la lógica de orquestación (Logic) para
        // calcular la probabilidad y escribir el resultado:
        // Logic logic = new Logic(output, gammaFunction,
simpson);
        // logic.run();
    }
}
```