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.

 © 2006 by Carnegie Mellon University

Personal Software Process for Engineers: Part I

Assignment Kit for the Coding Standard

Overview

Overview

This assignment kit covers the following topics.

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

 © 2006 by Carnegie Mellon University

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.

 © 2006 by Carnegie Mellon University

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

 © 2006 by Carnegie Mellon University

Example C++ Coding Standard

Purpose	To guide implementation of C++ programs
Program Headers	Begin all programs with a descriptive header.
Header Format	/**/ /* Program Assignment: the program number */ /* Name: your name */ /* Date: the date you started developing the program */ /* Description: a short description of the program and what it does */ /**/
Listing Contents	Provide a summary of the listing contents
Contents Example	/**/ /* Listing Contents: */ /* Reuse instructions */ /* Modification instructions */ /* Compilation instructions */ /* Includes */ /* Class declarations: */ /* CData */ /* ASet */ /* Source code in c:/classes/CData.cpp: */ /* CData */ /* CData() */ /* Empty() */ /**/

(continued)

 © 2006 by Carnegie Mellon University

 © 2006 by Carnegie Mellon University

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.

 © 2006 by Carnegie Mellon University

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	/********** * Programa: NombreArchivo.java * Autor: Jared Fernández González * Fecha: 2025-11-27 * Descripción: <Breve descripción de lo que hace esta clase>. * <Si es necesario, dividir la descripción en> * <varias líneas alineadas>. **********/
Listing Contents	Proporciona un resumen de la lista de contenidos

 © 2006 by Carnegie Mellon University

Contents Example	/* * Contenido del archivo: * - Variables de entrada cruda: * + dataX, dataY, dataXk * - Arreglos normalizados: * + arrDataX * - Referencias a módulos: * + Input input * + Output output * + Data data * + EstimacionCorLineal est * - Constructor: * + Logic(Input input, Output output, Data data, EstimacionCorLineal est, String outFile) * - Métodos públicos: * + void logic1a() * + void setDataX(String dataX) * + void setDataY(String dataY) * + void setDataXk(String dataXk) */
Reuse Instructions	• 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. • Proporciona advertencias sobre valores no válidos, condiciones de error, o cualquier situación que pueda producir un funcionamiento incorrecto.

 © 2006 by Carnegie Mellon University

Reuse Example	``` /** * Reader utilizado opcionalmente para lectura buffered. * (No se usa directamente en esta implementación pero se deja * por consistencia con el diagrama UML.) */ private BufferedReader br = null; /** * Lee un archivo de texto usando codificación UTF-8 y devuelve su contenido * completo en un único String. * * @param inFile nombre del archivo a leer (por ejemplo "test.txt"). * @return el contenido completo del archivo. * @throws IOException si el archivo no existe o no puede leerse. */ public String readData(String inFile) throws IOException { ```
Identifiers	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.

 © 2006 by Carnegie Mellon University

Identifier Example	`// Correctos (descriptivos)` `private static final String INPUT_FILE  = "test.txt";` `private static final String OUTPUT_FILE = "test_resultados.txt";` `final List<String> x = new ArrayList<>();  // Lista de valores X` `final List<String> y = new ArrayList<>();  // Lista de valores Y` `private double dblSumX;  // ΣX — suma de los valores X` `private double dblSumY;  // ΣY — suma de los valores Y` `// Incorrectos (no descriptivos, a evitar)` `int x1;` `double y1;` `String s;`
Comments	• Documenta el código de forma que el lector pueda entender su funcionamiento. • Los comentarios deben explicar tanto el propósito como el comportamiento del código. • Comenta las declaraciones de variables cuando su propósito no sea obvio.
Good Comment	`/** ΣX — suma de los valores X */` `private double dblSumX;` `/** Número de pares válidos (X,Y) */` `private int intN;` `// Verifica si se ha proporcionado dataX antes de normalizarla` `if (dataX != null) {` `    arrDataX = data.saveData(dataX);` `}`
Bad Comment	`// suma X` `dblSumX = dblSumX + x;   // <-- comentario inútil, repite lo que ya se ve` `// if dataX es distinto de null` `if (dataX != null) {     // <-- comentario redundante` `    arrDataX = data.saveData(dataX);` `}`

 © 2006 by Carnegie Mellon University

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	Ejemplo real basado en Logic.logic1a() (adaptado): `/* ============================` `=== 1) Normalizar datos X ===` `============================ */` `if (dataX != null) {` `arrDataX = data.saveData(dataX);` `}` `/* ============================` `=== 2) Normalizar datos Y ===` `============================ */` `if (dataY != null) {` `String[] arrY = data.saveData(dataY);` `...` `}` `/* ============================` `=== 3) Calcular sumas y R ===` `============================ */` `// Llamadas a est.sumX(...), est.sumY(...), etc.`
Blank Spaces	• Escribe los programas con suficiente espacio en blanco para que no se vean amontonados. • Separa cada construcción del programa con al menos un espacio. • Ejemplo: if (dataX != null), for (int i = 0; i < n; i++). • Deja líneas en blanco entre bloques lógicos (por ejemplo, entre secciones de normalización, cálculo y salida en Logic.logic1a()).
Indenting	• Sangra (indenta) cada nivel de llaves respecto al anterior. • Las llaves de apertura y cierre deben estar en líneas propias y alineadas entre sí, siguiendo el estilo que usas en tu proyecto (Allman: llaves en la siguiente línea).

 © 2006 by Carnegie Mellon University

Indenting Example	Ejemplo en el mismo estilo que tus clases (Input, Output, Logic): `public void logic1a()` `{` `    try` `    {` `        /* ============================` `           === 1) Normalizar datos X ===` `           ============================ */` `        if (dataX != null)` `        {` `            arrDataX = data.saveData(dataX);` `        }` `        /* ============================` `           === 2) Normalizar datos Y ===` `           ============================ */` `        if (dataY != null)` `        {` `            String[] arrY = data.saveData(dataY);` `            ...` `        }` `    }` `    catch (IOException ex)` `    {` `        System.err.println("Error al procesar la lógica: " + ex.getMessage());` `    }` `}`
Capitalization	Escribe todas las constantes static final en mayúsculas con guiones bajos. • Usa camelCase para variables y métodos (dataX, readData, writeData, saveData, logic1a). • Los nombres de clase usan PascalCase (App, Input, Output, Data, Logic, EstimacionCorLineal). • Los mensajes mostrados al usuario pueden ir en mayúsculas y minúsculas mezcladas para una presentación más clara.

 © 2006 by Carnegie Mellon University

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("Leyendo archivo: " + INPUT_FILE);` `        Input input = new Input();` `        Output output = new Output();` `        Data data = new Data();` `        EstimacionCorLineal est = new EstimacionCorLineal();` `        Logic logic = new Logic(input, output, data, est, OUTPUT_FILE);` `        logic.logic1a();` `    }` `}`

 © 2006 by Carnegie Mellon University