Dos & Don'ts:

- Pensar que, como tu profesor te da la impresión de ser muy serio, no está dispuesto a ayudarte en el proceso de aprendizaje. Puede que cuando él diga que por favor le preguntes todo lo que consideres oportuno, te esté diciendo la verdad.

- No valorar suficientemente la asistencia a clase. En las asignaturas presenciales (no virtuales) la clases aportan en general un gran valor añadido. En esta asignatura, el simplemente leer las transparencias no puede sustituir nunca al seguimiento presencial de la las clases.

- No intentar solucionar los errores relacionados con la programación por ti mismo antes de preguntar. No es que tu profesor esté intentando que preguntes menos, y por lo tanto trabajar menos, sino que considera que favorece mucho el aprendizaje dedicar un tiempo razonable depurando y estudiando código para encontrar bugs.

- No hacer *preguntas tontas*. Tu profesor no piensa que haya *preguntas tontas* si persisten después de una mínima reflexión. Si has pensado sobre alguna cuestión, y no encuentras la respuesta, entonces no es una *pregunta tonta*.

- Que te fastidie que tu profesor te conteste con otra pregunta. Tu profesor considera que dar sin más la respuesta a una duda tiene menos valor docente que intentar iniciar contigo un proceso de razonamiento que finalmente te lleve por ti mismo a la respuesta.

- No planificar el trabajo. Crear software no consiste en escribir código a toda velocidad, sino que demanda reflexión, por tanto no debes planificar simplemente el tiempo que tardarías en "escribir" el programa, sino en plantearte cómo hacerlo, pensando que además pueden surgir dudas y que quizás explores alternativas que debas corregir hasta dar con la respuesta más adecuada. Esta máxima no sólo sirve para las prácticas, sino también para la teoría: aprender PDOO requiere tiempo, no solo por la complejidad de los conceptos, sino porque es necesario reposarlos, practicarlos, y reflexionar sobre ellos según la experiencia que se haya tenido en la práctica. Esto significa que si dejas el estudio para última hora, será menos probable que hayas podido madurar bien lo aprendido

- Pensar que lo único importante en el software es el resultado que produce en un momento determinado. El diseño del software es igual o más importante que el resultado cuando se está aprendiendo a crear este tipo de productos. El software en general tiene que ser mantenido, modificado y ampliado y sin un diseño adecuado puede tener que ser rehecho desde el principio porque sea a todos los efectos no mantenible. Ten en cuenta que es muy habitual que durante la vida de un software intervengan personas distintas.

- Creer que se aprende a programar principalmente leyendo transparencias y estudiando el código de otras personas. Aunque en general cada persona es un mundo, pienso que es necesario dedicar un número no despreciable de horas de programación y depuración para aprender a programar bien. A conducir también se aprende conduciendo y no simplemente leyendo un manual o viendo como conduce otra persona mientras estamos sentados en el asiento del copiloto. Teniendo en cuenta que el supuesto de prácticas se reutiliza durante varios cursos, soy consciente que debe ser relativamente fácil conseguir una implementación decente de lo solicitado en prácticas sin programar una línea de código. Sin embargo, la mayor parte de las personas no está en condiciones de abstraer los conceptos de esta asignatura simplemente estudiando ese código descargado y creado por otra persona. Incluso partes de código que en apariencia son triviales, y aunque sean estudiadas con detalle, luego

no son correctamente reproducidas al requerir alguna leve variación en un examen.

- No entender el código que se está escribiendo. Debes entender totalmente el funcionamiento y propósito de cada una de las líneas de código que escribas. Tu profesor está para ayudarte en esta tarea. No es aceptable que en tu código ocurran las cosas “mágicamente”. Tampoco es recomendable dar por bueno código que se piensa que no debería funcionar correctamente pero que aparentemente sí lo hace.

- No seguir estrictamente lo indicado por los distintos diagramas suministrados. Aunque aún no conozcas esos diagramas, puedes considerarlos como algo equivalente a los planos de un edificio. Utilizando esa analogía, ¿ te parecería razonable que un trabajador de la construcción decidiera de forma autónoma cambiar una ventana de sitio o cambiar la planta en que se sitúa un cuarto de baño, incumpliendo lo que indican los planos ?
  Dicho esto, si crees que has encontrado un error en la especificación de las prácticas, o si crees que un diseño alternativo sería mejor, tu profesor estará encantado de debatir sobre diseños alternativos y su viabilidad o de confirmarte si efectivamente hay un error en algún diagrama.

- Repetir código. Si te encuentras en la situación en la que estás copiando y pegando código de una parte del proyecto a otra, posiblemente haya un problema de diseño. Tener el mismo código repetido varias veces implica que cualquier cambio en el mismo es necesario reproducirlo en todas las copias realizadas, y pasado un tiempo te habrás olvidado de las copias.

- Números mágicos. En general, el uso de literales numéricos en el código está altamente desaconsejado. Utiliza constantes que permitan el cambio de esas cifras si es necesario en el futuro y que aumentan además la legibilidad del código. Si esos números pueden ser obtenidos de forma automática, hazlo, guarda el valor en una variable y usa la variable (Ej: tamaño de una colección)

- Aunque posiblemente aún no sepas lo que es el patrón de diseño Modelo-Vista-Controlador, ya te adelanto que uno de los errores más habituales es introducir una cantidad importante de lo que se llama lógica del problema en la vista. De esta forma se introducen grandes dependencias entre la parte de código que soluciona el problema con la parte del código que debe mostrar los resultados y permitir al usuario interactuar con el mismo.

- También te adelanto que el sustituir el uso del mecanismo de herencia por herramientas que permiten obtener el tipo dinámico de una referencia también es un error grave. La herencia entre clases permite la creación de código genérico que a la vez permita abarcar diferentes modalidades o tipos de funcionamiento dentro del marco de un problema. El uso de la herencia no es algo obligatorio en cualquier programa, pero lo que no es aceptable es que sea sustituido o reemplazado por uso de operadores o métodos como instanceof, getClass, etc. asumiendo que se utiliza el lenguaje Java. En otros lenguajes este tipo de herramientas tienen nombres parecidos. Pregunta a tu profesor si no entiendes la razón por la que es peligroso el uso de este tipo de recursos. "*Anytime you find yourself writing code of the form 'if the object is of type T1, then do something, but if it's of type T2, then do something else,' slap yourself.* Effective C++. Second Edition. Página 176. Scott Meyers.