**Desarrollo de aplicaciones web**

**Desarrollo de aplicaciones multiplataforma**

***M03 – Programación II***

***Estructuras de datos avanzadas y control de excepciones***

Objetivos

* Escribir programas que manipulen información seleccionando y utilizando los tipos avanzados de datos facilitados por el lenguaje.
* Utilizar las clases básicas para almacenar y procesar información (listas y tablas de Hash).
* Implementar la gestión de excepciones en la utilización de clases facilitadas por el lenguaje.
* Implementar el lanzamiento de excepciones.

Actividad 1

***Agenda de contactos (HashMap)***

Crear un programa que simule una agenda de contactos. Cada contacto consta de un nombre y un número de teléfono. Se debe crear una clase Contacto que guarde la información especificada. Además, hay que sobrescribir los métodos equals y hashCode de la siguiente forma:

* Dos contactos se consideran iguales si tienen el mismo número de teléfono.
* El hashCode se calcula dividiendo el número de teléfono entre 100 y sumándole la longitud del nombre del contacto.

El programa principal debe hacer uso de un HashMap para almacenar los contactos. Implementar métodos para dar de alta contactos, eliminar, modificar y borrar la lista de contactos.

Actividad 2

***Garaje mecánico (ArrayList)***

Se desea realizar una aplicación que permita a los mecánicos de un garaje registrar, consultar y actualizar los trabajos (reparaciones y revisiones) que han sido realizados o que están en proceso de realización en el garaje.

Cada trabajo se identifica inequívocamente por su “identificador de trabajo”. El “identificador de trabajo” es un número que se asocia con el trabajo en el momento que se registra. El primer trabajo registrado tendrá el identificador 0, el segundo el 1 y así sucesivamente.

Los trabajos incluyen una pequeña descripción de la reparación o revisión a realizar.

Todos los trabajos incluyen el número de horas que van siendo necesarias para su realización. Al crear un trabajo el número de horas es 0.El número de horas irá aumentando a medida que los mecánicos van dedicando tiempo a realizar la reparación o la revisión. Cuando el trabajo se ha finalizado se marca como “finalizado” y el número de horas no puede volver a cambiarse.

Las reparaciones incluyen el precio del material utilizado (piezas o pintura). Al registrar una reparación el precio del material es 0 y va aumentando a medida que los mecánicos van utilizando material en la reparación. Una vez que la reparación se marca como “finalizada” no se puede cambiar el precio del material utilizado.

El precio a cobrar para cada trabajo se compone de una parte fija que resulta de multiplicar el número  de horas empleadas por 30€. Además, dependiendo del tipo de trabajo el coste varía de la siguiente manera:

* Reparación mecánica: su precio se calcula como fijo más el coste material multiplicado por 1.1.
* Reparación de chapa y pintura: su precio se calcula como fijo más el coste material multiplicado por 1.3.
* Revisión: su precio se calcula como fijo más extra independiente del número de horas de 20€.

La aplicación tendrá las siguientes funcionalidades:

* Registrar trabajo: se introduce el tipo de trabajo y su descripción. El mecánico introduce los datos y el programa añade el trabajo a la lista y muestra el identificador asignado al trabajo.
* Aumenta horas: el mecánico introduce el identificador del trabajo y el número de horas.
* Aumenta coste piezas: el mecánico introduce el identificador del trabajo y el coste de las piezas, y la aplicación aumenta el coste de las piezas al trabajo.
* Finaliza trabajo: el mecánico introduce el identificador del trabajo y la aplicación termina el trabajo.
* Muestra trabajo: el mecánico introduce el identificador del trabajo y la aplicación muestra el identificador, la descripción y el precio del trabajo.

Siempre que el mecánico introduzca un identificador incorrecto, el programa informará de este error.

Utilizar un ArrayList para almacenar los trabajos. Hay que hacer uso de clases y herencias.

Actividad 3

***Control de excepciones***

1. Crear un programa que pida los datos de dos alumnos, que son: nombre (letras), edad (entero) y altura (decimal). Se debe realizar un control de la entrada de datos, de tal forma que si el usuario introduce números en el nombre, o letras en la edad o la altura, se vuelva a pedir el dato correspondiente hasta que la entrada sea correcta.
2. Modificar el programa anterior de manera que si el usuario comete más de 5 errores al introducir datos, se muestre un mensaje informando de este hecho y se cierre el programa.
3. Escribir un programa que vaya pidiendo al usuario números y los vaya sumando. El programa pedirá números hasta que se hayan insertado cinco números o se introduzca una letra. El programa debe tener la siguiente estructura:
   * Habrá un método pedir número que devolverá el valor introducido por el usuario.
   * El método main llamará al método que pide números.

En esta primera versión, el control de excepciones se debe hacer de la siguiente forma:

* El método que pide números debe capturar las excepciones de tipo IOException y NumberFormatException.
* El método main captura las excepciones de tipo NumberFormatException. El resultado de la suma debe mostrarse en un mensaje ubicado en un bloque finally.

Siguiente la estructura anterior, ¿qué ocurre? ¿Funciona el programa tal y como debe?

1. Solucionar el programa anterior de forma que termine de las dos formas que se piden.
2. Crear una clase Alumno con los siguientes campos: nombre y edad. Se debe crear el constructor y los métodos get y set necesarios. Crear una excepción propia DemasiadosObjetos que muestre el mensaje “Se ha alcanzado el número máximo de elementos”. Crear otra clase con el método main que almacene en un ArrayList varios alumnos. Ir haciendo inserciones de forma que si se intenta insertar un sexto alumno, se lance una excepción de tipo DemasiadosObjetos.