PRÁCTICA 4: EXCEPCIONES

EJERCICIO 1º: IMPLEMENTACIÓN DE UNA CLASE FECHA COMPLETAMENTE ROBUSTA

Diseñar una clase Fecha completamente robusta a partir del la siguiente API:

```
public class Fecha
{
	private int año = 0;
	private int mes = 1;
	private int dia = 1;

	public static final int MESES_AÑO = 12;

	public static boolean esBisiesto(int año)

	public Fecha()

	public Fecha(int año, int mes, int dia)

	public Fecha(Fecha f)

	public boolean esBisiesto()

	public void setAño(int año)

	public int getAño()

	public void setMes(int mes)

	public int getMes()

	public void setDia(int dia)

	public int getDia()

	public void set(int año, int mes, int dia)

	public void set(Fecha f)

	public String toString()

	public boolean equals(Object o)

}
```

Será necesaria una clase auxiliar FechaException de tipo implícita que deberá ser lanzada cuando cualquier usuario intente introducir en un objeto de tipo Fecha cualquier valor incorrecto:

a) Un mes menor que 1 o mayor que 12.

b) Un día incorrecto según el mes y el año que ya se tenga.

```
public class FechaException extends RuntimeException
{
      public FechaException()

      public FechaException(String message)

      public FechaException(String message, Throwable cause)

      public FechaException(Throwable cause)
}
```

Por último probar el correcto funcionamiento de la clase mediante el siguiente programa principal que intenta crear fechas de forma aleatoria capturando las posibles excepciones y mostrando los resultados correctos o incorrectos.

```
public class PruebaFecha
{
  public static final int MAX_FECHAS = 100;

  public static int random(int min, int max)
  {
    return (int)(Math.random() * (max - min + 1)) + min;
  }

  public static void main(String[] args)
  {
    for (int i = 0; i < MAX_FECHAS; i++)
    {
      try
      {
        Fecha f = new Fecha(random(1, 2007), random(1, Fecha.MESES_AÑO),
random(1, 31));
        System.out.println("Fecha correcta: " + f.toString());
      }
      catch(Exception e)
      {
        System.out.println("EXCEPTION: " + e.getMessage());
      }
    }
  }
}
```

EJERCICIO 2º: LECTURA POR LA ENTRADA ESTÁNDAR DEL SISTEMA

Para poder leer directamente de la entrada estándar se debe hacer uso del objecto System.in. Sin embargo, el tipo de datos de dicho objeto es demasiado simple para poder hacer una lectura de información lo suficientemente útil (sólo permite leer byte a byte) por lo que hay que hacer uso de otras clases de la API de Java. En definitiva, para poder leer de la entrada estándar del sistema (el teclado) hay que crear distintos objetos:

```
java.io.InputStreamReader isr = new java.io.InputStreamReader(System.in);
java.io.BufferedReader br = new java.io.BufferedReader(isr);
```

La clase BufferedReader tiene un método avanzado que permite leer cadenas de la entrada estándar del sistema.

```
String readLine()
```

Este método lanza una excepción explícita java.io.IOException.

Dado que tratar la introducción de datos por la entrada estándar es un poco compleja (hacer uso de distintas clases de la API de Java así como tratar una excepción) se pide implementar una clase llamada EntradaEstandar que contenga la siguiente declaración:

```
public class EntradaEstandar
{
	public static int leerInt(String mensaje)

	public static float leerFloat(String mensaje)

	public static String leerString(String mensaje)
}
```

Para poder resolver este problema deberán revisar ciertos métodos estáticos “parse” de las clases de boxing de los tipos de datos primitivos (Integer y Float). Además deberá saberse que estos métodos pueden lanzar una excepción implícita NumberFormatException que será lanzada si la cadena a convertir tiene un formato incorrecto.

Implementar una clase con un programa principal que permita la siguiente salida:

```
Operando entero 1: a
ERROR: El valor introducido no es correcto.
Operando entero 1: 1.4
ERROR: El valor introducido no es correcto.
Operando entero 1: 4
Operando entero 2: 7
Resultado: 4 + 7 = 11
Operando flotante 1: a
ERROR: El valor introducido no es correcto.
Operando flotante 1: 14.5
Operando flotante 2: 3.789
Resultado: 14.5 / 3.789 = 3.826867
Introduzca su nombre: Pedrito

Hola Pedrito. ¡Bienvenido a Java!
```

EJERCICIO 3º: PREGUNTAS

Responde a las siguientes preguntas:

1. ¿Qué habría que hacer si un método que queremos llamar indica que lanza una excepción explícita?
   a) Aunque se recomienda capturar la excepción, no hay por qué hacer nada si no se quiere. En dicho caso, si la excepción se lanza, la máquina virtual en última instancia será la encargada de tratarla.
   b) Hay que poner un bloque try y catch obligatoriamente.
   c) Se puede bien indicar que se relanza en el método que hace la llamada usando una cláusula throws o tratarla en dicho método mediante un bloque try y catch.
   d) Hay que indicar que la excepción se relanza obligatoriamente usando una cláusula throws.
2. ¿Qué opción escogerías en la pregunta anterior si la excepción fuera implícita?
3. ¿Se puede poner un bloque finally sin un bloque try?
4. ¿Se puede poner un bloque finally sin un bloque catch?