GUIA EJERCICIOS POO AVANZADO

1. En equipos electrónicos es habitual representar el color por medio de 3 componentes: rojo (R), verde (G) y azul (B). Así un color queda definido por medio de tres valores, RGB. En este ejercicio a cada componente se le asigna un valor real entre 0.0 y 1.0, de forma que algunos colores típicos quedan definidos como en la siguiente tabla:

|  |  |
| --- | --- |
| (R, G, B) | Color |
| (1.0, 0.0, 0.0) | Rojo |
| (0.0, 1.0, 0.0) | Verde |
| (0.0, 0.0, 1.0) | Azul |
| (1.0, 1.0, 1.0) | Blanco |
| (0.0, 0.0, 0.0) | Negro |

Se pide desarrollar una aplicación con los siguientes requerimientos:

1. Al crear un color se deben especificar los valores de los tres componentes los cuales se ingresan a través de controles jTextField. Estos no pueden ser negativos ni mayores que 1. Considerar además que el usuario puede introducir valores no numéricos por lo que se genera excepción de tipo NumberFormatException. Para cada caso ya descrito lanzar la excepción correspondiente
2. La aplicación debe permitir aplicar un filtro a cada componente. Para ello crear el método “filtro(cr, cg, cb)” que realiza la operación ya descrita. A este método se le pasan tres coeficientes el de rojo, el de verde y el de azul. El método multiplica cada componente por el coeficiente correspondiente: así por ejemplo, filtro(1.0,0.0,0.0) sería un filtro de rojo, que sólo deja pasar el componente R. Los valores de los coeficientes no pueden ser menores a cero, en este caso se debe lanzar la excepción correspondiente indicando cuál componente presenta el error. Si el resultado de multiplicar un componente por su coeficiente es mayor a 1.0, el valor debe acotarse a 1.0.

Se pide implementar en java la situación planteada aplicando manejo de excepciones personalizadas para todos los casos que lo requieran. La gestión de las excepciones debe realizarse en el Main

1. Dadas las excepciones Exc0 que hereda de Exception y Exc1 que hereda de Exc0. ¿Cuál es el resultado de la ejecución del siguiente código? Justifica la respuesta.

public class Examen {

public static void main (String[] args){

try{

throw new Exc1();

}catch (Exc0 e1){

System.out.println(“Capturada Exc0”);

}catch(Exception e2){

System.out.println(“Capturada Exception”);

}

}

}

1. Indicar cuál es la salida del siguiente programa y explicar por qué.

class A {

public static void main (String args[]) throws X {

try {

f ();

throw new Z ();

}

catch (Y ex) {

System.out.println (“Y” + ex);

}

catch (X ex) {

System.out.println (“X” + ex);

}

}

static f () throws X {

try {

throw new Y ();

} catch (X ex) {

g ();

}

}

static g () throws X {

try {

throw new X ();

} catch (Y ex) {

}

}

}

class X extends Exception {

public String toString () { return "X"; }

}

class Y extends X {

public String toString () { return "Y"; }

}

class Z extends Y {

public String toString () { return "Z"; }

}

1. Cuál es la salida del siguiente programa:

class Ejercicio {

public static void main (String [] args) {

Clase1 obj1= new Clase1(7);

Clase1 obj2= new Clase1(9);

obj2=obj1;

System.out.print(obj1.valorprop()+" ");

System.out.print(obj2.valorprop()+" ");

}//fin main

}

class Clase1{

static int prop=0;

public Clase1 (int valor){ prop=valor;}

public int valorprop(){ return prop; }

}

1. Implemente en lenguaje de diseño la clase genérica Cola, con todas las operaciones necesarias
2. Desarrollar una clase vector que permita realizar operaciones básicas de vectores con números enteros. Los métodos a añadir son los siguientes:

**Suma():** suma dos vectores y retorna el vector resultante

**Acumula():** suma un vector con el vector actual. El resultado queda almacenado en el vector actual

Considerar todas las condiciones de error que se podrían dar tanto en los métodos ya implementados como en los que ud implementará.

Notificar dichos errores utilizando excepciones propias (donde crea conveniente)

A continuación se presenta la clase Vector parcialmente implementada:

/\*\*

\* Clase para almacenar un vector de nºs enteros y hacer operaciones con él.

\*/

public class Vector {

// componentes del vector

private int[] componentes;

/\*\*

\* Constructor que crea un vector de la dimension indicada, inicialmente con todas sus componentes a cero.

\*

\* @param dimension

\* El numero de componentes del vector

\*/

public Vector(int dimension) {

componentes = new int[dimension];

}

/\*\*

\* Este método cambia la componente i-ésima del vector por el valor pasado

\* como argumento.

\*

\* @param pos

\* El indice de la posición cuyo valor se va a cambiar.

\* @param valor

\* El nuevo valor que le damos a la componente i-ésima del

\* vector.

\*/

public void cambiaComponente(int pos, int valor) {

componentes[pos] = valor;

}

}

/\*\*

\* Prueba de la clase Vector

\*

\*/

public class PruVector {

/\*\*

\* Método principal de prueba de la clase Vector

\*/

public static void main(String[] args) {

// crea vector (1, 2, 3)

Vector v3a = new Vector(3);

v3a.cambiaComponente(0, 1);

v3a.cambiaComponente(1, 2);

v3a.cambiaComponente(2, 3);

// crea vector (10, 20, 30)

Vector v3b = new Vector(3);

v3b.cambiaComponente(0, 10);

v3b.cambiaComponente(1, 20);

v3b.cambiaComponente(2, 30);

// crea vector (0, 0, 0, 0)

Vector v4a = new Vector(4);

}

}

1. Todas las clases en Java heredan de la clase base Object, por lo que se puede emplear Object para implementar algoritmos genéricos. Considerando lo anterior: Diseñe e implemente una clase Caja que pueda guardar objetos de cualquier tipo(object). Realizar Main de prueba respectivo.
2. Transformar la clase Caja anterior, en una clase Generica utilizando el parámetro formal del tipo <T>. Realizar Main de prueba respectivo.
3. Utilizando la clase ArrayList, escribe un programa que realice los siguientes pasos:
   1. Guarde en el array 4 objetos de la clase Caja que contengan: “medias”, 3.5, 7, Otra caja.
   2. Recorra el array y muestre por pantalla el contenido de todas las cajas.
   3. Realizar Main de prueba respectivo.
4. Se pide implementar una clase genérica PAR que guarda el par llave-valor, donde los tipos de la llave y el valor son especificados por los parámetros L y V respectivamente. Esta clase debe permitir actualizar y obtener los datos de los atributos. En el main se pide crear dos instancias de esta clase: una para un par String-Integer (por ejemplo, para guardar un tipo de dimensión y su valor) y otro para un par Estudiante-double (por ejemplo, para guardar la calificación de un estudiante).

La clase estudiante se define como sigue:

Public class Estudiante{

String nombre;

int edad;

public Estudiante(String n, int e){

nombre=n;

edad=e;

}

}

1. Se tiene la implementación en JAVA el manejo de colas con arreglos como se indica a continuación:

|  |  |  |
| --- | --- | --- |
| public class Cola{  private int [] cola;  private int frente, fin;  private int MAX\_ELEM=3;  public Cola()  {  cola=new int[MAX\_ELEM];  frente=0;  fin=0;  }    public boolean estaVacia() {  return (frente==fin);  }  public boolean estaLlena() {  return (frente==(fin+1)%MAX\_ELEM);  } | public boolean encolar(int x)  {  boolean exito;  fin=(fin+1)%MAX\_ELEM;    if (frente==fin) {  exito=false;  if (fin==0)  fin=MAX\_ELEM-1;  else  fin=fin-1;  }  else{  cola[fin]=x;  exito=true;  }  return exito;    } | public int desencolar() {  int x=-1;  if (estaVacia()) {  System.out.println("Cola vacia");  }else{  frente=(frente+1)%MAX\_ELEM;  x=cola[frente];  }  return x;  }      public void imprimir (){  while (!estaVacia()){  System.out.println("\t"+ desencolar());  }  } |

Se pide:

* Implementar el manejo de excepciones en los siguientes casos:
  + Insertar elementos en cola llena
  + Sacar elementos de cola vacia
* Capturar y gestionar las excepciones en el main.