Ejercicios de prueba:

1.- Prueba el programa siguiente:

```
public class Excepcion1{
    public static void main(String args[])
    {
        int numero[]=new int[5];
        numero[7]=0;
    }
}
```

Este programa genera una excepción al momento de su ejecución. A continuación, agrega el código para el manejo de la excepción.

2.- Prueba el programa siguiente

```
class Excepcion2{
    void miMetodo(){
        int numero[]=new int[5];
        try{
                System.out.println("Accesando a una posicion fuera del
                vector"); numero[7]=0;
        }
        catch (ArrayIndexOutOfBoundsException excep)
        {
                System.out.println ("Ocurrio una excepcion");
        }
    }
}

public class PruebaExcepcion2{
     public static void main (String args[]){
                GeneraExcepcion2 objeto =new GeneraExcepcion2();
                objeto.miMetodo();
      }
}
```

¿En qué método se genera la excepción?
¿Qué método la captura?
¿Vuelve el control del programa al main, después de la captura?

3.- Prueba el programa siguiente

```
class Excepcion3{
    static void divide(){
            int num[]={4,8,16,32,64,128,256};
            int den[]={2,0,4,4,0,8,16};
```

```
            for (int i=0;i<num.length;i++){
                try{
                        System.out.println(num[i]+ "/" + den[i]+"=" + num[i]/den[i]);}
                catch(java.lang.ArithmeticException excepcion){
                        System.out.println("Dividiendo por cero");
                 }
             }
        }
}


public class PruebaExcepcion3{
         public static void main (String args[]){
                  ExcepcionContinua.divide();
          }
}
```

¿Qué ocurre después de que el programa responde a un error?

4.- En un mismo segmento de código se pueden generar más de una excepción por diferentes motivos, por lo tanto dicho segmento puede tener un bloque catch para cada excepción. Completa los dos bloques match, el primero captura la excepción que se produce al dividir por cero y el segundo al acceder a una posición fuera del vector.

```
public class Excepcion4 {

 static void divide(){
 int num[]={4,8,16,32,64,128,256};
 int den[]={2,0,4,4,0,8};
 for (int i=0;i<num.length+1;i++){
 try{
 System.out.println(num[i]+ "/" + den[i]+"=" + num[i]/den[i]);  }
 catch . . . . . . . {
                              . . . . .
 }
 catch . . . . . . . {
                              . . . . .
 }
 }
 }
 }

public class PruebaExcepcion4 {
 public static void main (String args[]){
 Excepcion4.divide();
 System.out.println("FIN");
 }
 }
```

5.- catch específicos y genéricos.

En el ejemplo siguiente se deberán completar los dos bloques catch, el primero atrapa específicamente la excepción que se produce cuando se accede fuera del límite del vector, el segundo atrapa cualquier excepción que se produzca en la superclase throwable.

```
public class Excepcion5 {

        public static void main(String args[]){
         int num[]= {4,8,16,32,64,128,256,512};
         int den[]= {2,0,4,4,0,8};
         for(int i=0;i<num.length;i++){
         try{
         System.out.println(num[i]+"/"+den[i]+"="+ num[i]/den[i]);  }
         catch . . . . . . . {
                              . . . . .
         }
         catch . . . . . . . {
                              . . . . .
         }

         }
         } // fin de main
}
```

6.- Los bloques try se pueden anidar y así permitir diferentes niveles de errores, que serán manejados de diferentes formas. Algunos de estos errores son menores y pueden arreglarse fácilmente, pero otros son catastróficos y no se pueden corregir. Un método comúnmente empleado por los programadores es usar **un bloque externo try** para atrapar los **errores graves** y **bloques internos try para el control de los errores sencillos**. Prueba el programa siguiente que usa bloques try anidados:

```
class Excepcion6 {
 public static void main(String args[]){
 int num[]= {4,8,16,32,64,128,256,512};
 int den[]= {2,0,4,4,0,8};

 try
 {
  for (int i=0;i<num.length;i++)
 {
 try //try interno
 {
 System.out.println(num[i]+"/"+den[i]+"="+ num[i]/den[i]);  }
 catch (ArithmeticException Excep)
 {
 System.out.println("División por cero "+ i);
 }
 }
 } //try externo
```

```
catch (Throwable Excep)
{
System.out.println("Ocurrio una excepcion fatal ");
}
System.out.println("El programa continu aqui");

} // fin de main}
}
```

7.- Completa los dos bloques catch del ejemplo siguiente. El primero atrapa la excepción que se produce cuando se divide entre cero, el segundo atrapa la excepción que se produce cuando se accede a una posición fuera del vector.
Diseña el bloque finally que se ejecuta siempre tanto si se produce excepción como no. Este bloque contiene la orden: **System.out.println("Ejecutando código de limpieza");**

```
public class Excepcion7 {

public static void generaExcepcion(int i){
int t;
int num[] = {2,4,6};
System.out.println("Recibiendo "+ i);
try
{
switch(i)
{
case 0: t=10/i; //division por cero
break;
case 1:num[4]=4; //genera un error
break;
case 2: return;
}
}

    catch . . . . . {
                . . . . .
            }
    catch . . . . . {
                . . . . .
            }

     finally
     {
     . . . . . .
     }

} //fin de metodo
}//clase
```