Ejercicio U5_B8B_E1:

```
class App {

    public static void main(String[] args) {
        try {
            System.out.println(dividir(4, 2));
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
        try {
            System.out.println(dividir(4, 0));
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
        try {
            System.out.println(dividir(6, 2));
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
    }

    static int dividir(int x, int y) throws Exception {
        int cociente;
        if (y == 0) {
            throw new Exception("no se puede dividir por 0");
        }
        cociente = x / y;
        return cociente;
    }
}
```

Ejercicio U5_B8B_E2:

```
abstract class Figura {

    protected String color;

    public Figura(String color) throws Exception {
        if (color.equals("blanco")) {
            throw new Exception("color blanco no válido");
        }
        this.color = color;
    }

    abstract public double area();
}

class Triangulo extends Figura {

    private double base;
    private double altura;

    public Triangulo(double base, double altura, String color) throws Exception {
        super(color);
        this.base = base;
        this.altura = altura;
    }

    @Override
    public double area() {
        return base * altura / 2;
    }
}

class Circulo extends Figura {

    private double radio;

    public Circulo(double radio, String color) throws Exception {
```

```
        super(color);
        this.radio = radio;
    }

    @Override
    public double area() {
        return Math.PI * radio * radio;
    }
}

class App {

    public static void main(String[] args) {
        try {
            Circulo c = new Circulo(2.0, "blanco");
            System.out.println("Area circulo " + c.area());
        } catch (Exception e) {
            //aquí podrían ir instrucciones para rectificar el color como corresponda
            //o simplemente avisamos que no puede crear el objeto...
            System.out.println("NO SE PUDO CREAR OBJETO: " + e.getMessage());
        }
        try {
            Triangulo t = new Triangulo(2.0, 3.0, "rojo");
            System.out.println("Area triangulo " + t.area());
        } catch (Exception e) {
            System.out.println("NO SE PUDO CREAR OBJETO" + e.getMessage());
        }
    }
}
```

Reflexiona: ¿Qué tal poner el try/catch en los constructores de Circulo y Triángulo?
La idea correcta es que App es cliente de la jerarquía y la jerarquía no trata excepciones, simplemente "avisa" a sus clientes de que se produjo una excepción, luego, el cliente App decide qué hacer cuando se produce una excepción por culpa del color blanco como por ejemplo:

- imprimir por pantalla "color invalido".
- volver a crear la figura con un color correcto
- lanzar un misil que destruya el universo.
- abortar el programa
- O cualquier cosa que App estime útil y oportuno para ella

La conclusión es: depende del contexto donde se debe capturar una excepción, pero normalmente se hace en la capa más externa.

Ejercicio U5_B8B_E3:

```
class FiguraException extends Exception{
   FiguraException(String msg){
      super(msg);
   }
}
abstract class Figura {

   protected String color;

   public Figura(String color) throws FiguraException {
      if(color.equals("blanco")){
         throw new FiguraException("color blanco no válido");
      }
       this.color = color;
   }

   abstract public double area();
}

class Triangulo extends Figura {
```

```
   private double base;
   private double altura;

   public Triangulo(double base, double altura, String color) throws FiguraException {
      super(color);
      this.base = base;
      this.altura = altura;
   }

   @Override
   public double area() {
      return base * altura / 2;
   }
}

class Circulo extends Figura {

   private double radio;

   public Circulo(double radio, String color) throws FiguraException  {
      super(color);
      this.radio = radio;
   }

   @Override
   public double area() {
      return Math.PI * radio * radio;
   }
}

class App {

    public static void main(String[] args) {
      try {
         Circulo c = new Circulo(2.0, "blanco");
         System.out.println("Area circulo " + c.area());
      } catch (FiguraException e) {
         //aquí podrían ir instrucciones para rectificar el color como corresponda
         //o simplemente avisamos que no puede crear el objeto...
         System.out.println("NO SE PUDO CREAR OBJETO: " + e.getMessage());
      }
      try {
         Triangulo t = new Triangulo(2.0, 3.0, "rojo");
         System.out.println("Area triangulo " + t.area());
      } catch (FiguraException e) {
         System.out.println("NO SE PUDO CREAR OBJETO" + e.getMessage());
      }
   }
}
```

En el enunciado se puso a proposito getMessagge(). Por error este método tiene un ‘g’ de más. Esto lo que hace es que al escribir el método simplemente se cree un nuevo método pero no se sobrescribe el del Padre porque baila una letra.

```
public String getMessage(){
      return "el color no puede ser blanco";
   }
```

observa que para conseguir la sobreescritura el compilador me indica la necesidad de que sea público.

```
class FiguraException extends Exception{


   @Override
   public String getMessage() {
      return "el color no puede ser blanco";
   }
```

```
}
abstract class Figura {

   protected String color;

   public Figura(String color) throws FiguraException {
      if(color.equals("blanco")){
         throw new FiguraException();
      }
       this.color = color;
   }

   abstract public double area();
}

class Triangulo extends Figura {

   private double base;
   private double altura;

   public Triangulo(double base, double altura, String color) throws FiguraException {
      super(color);
      this.base = base;
      this.altura = altura;
   }

   @Override
   public double area() {
      return base * altura / 2;
   }
}

class Circulo extends Figura {

   private double radio;

   public Circulo(double radio, String color) throws FiguraException  {
      super(color);
      this.radio = radio;
   }

   @Override
   public double area() {
      return Math.PI * radio * radio;
   }
}

class App {

     public static void main(String[] args) {
      try {
         Circulo c = new Circulo(2.0, "blanco");
         System.out.println("Area circulo " + c.area());
      } catch (FiguraException e) {
         //aquí podrían ir instrucciones para rectificar el color como corresponda
         //o simplemente avisamos que no puede crear el objeto...
         System.out.println("NO SE PUDO CREAR OBJETO: " + e.getMessage());
      }
      try {
         Triangulo t = new Triangulo(2.0, 3.0, "rojo");
         System.out.println("Area triangulo " + t.area());
      } catch (FiguraException e) {
         System.out.println("NO SE PUDO CREAR OBJETO" + e.getMessage());
      }
   }
}
```

Ejercicio U5_B8B_E4:

```
class Punto {
```

```
    int x = 0;
    int y = 0;
    Punto(int x, int y) throws Exception{
        if (x<0||y<0){
            throw new Exception("Al menos hay una coordenada negativa");
        }
        this.x = x;
        this.y = y;
    }
}
class Rectangulo {
    Punto origen;
    int ancho ;
    int alto ;
    Rectangulo(int x, int y, int w, int h) throws Exception{
        origen = new Punto(x,y);
        ancho = w;
        alto = h;
    }
}
class App{
    public static void main(String[] args){
        try{
            Rectangulo miRectangulo=new Rectangulo(-2,3,4,5);
        }catch(Exception ex){
            System.out.println(ex.getMessage());
        }

    }
}
```

Ejercicio U5_B8B_E5:

La nueva clase OrdenadorException

```
package ordenador;

public class OrdenadorException extends Exception{
    String nSerie;
    String msg;

    public OrdenadorException(String nSerie, String msg) {
        this.nSerie = nSerie;
        this.msg = msg;
    }

    @Override
    public String getMessage() {
        return nSerie+": "+msg;
    }

}
```

Constructor de la clase Ordenador reformado

```
public Ordenador(String nSerie,int capacidadMemoria,String tipMemoria,int velocidadMemoria,float capacidadDisco,String tipoDisco, String tipoProcesador, float velocidadProcesador,int precio) throws OrdenadorException {

        if(tipoProcesador.contains("i7")  &&  !tipoDisco.contains("SSD")){
            throw new OrdenadorException(nSerie,"i7 sin SSD no se monta");
        }
        if(nSerie.startsWith("HP") && capacidadMemoria<4 ){
            throw new OrdenadorException(nSerie,"Serie HP no puede tener menos de 4gb de memoria");
        }



        this.nSerie=nSerie;
        this.m= new Memoria(capacidadMemoria,tipMemoria,velocidadMemoria);
        this.d=new Disco(capacidadDisco, tipoDisco);
        this.p= new Procesador(tipoProcesador,velocidadProcesador);
    }
```

Ejercicio U5_B8B_E6:

```
package ordenador;

public class Portatil extends Ordenador {

    float peso;
    public Portatil(String nSerie, int capacidadMemoria, String tipMemoria, int velocidadMemoria, float capacidadDisco, String tipoDisco, String tipoProcesador, float velocidadProcesador, int precio, float peso) throws OrdenadorException {
        super(nSerie,capacidadMemoria,tipMemoria,velocidadMemoria,capacidadDisco,tipoDisco,tipoProcesador,velocidadProcesador,precio);
        if(peso>10){
            throw new OrdenadorException(nSerie,"portatil de más de 10 kg no se monta");
        }
        this.peso=peso;

    }

}
```

Ejercicio U5_B8B_E7:

Hay que hacer dos cosas:

1. Escribir la clase DNIException
2. Sustituir Exception por DNIException

```
package dni;

public class DNIException extends Exception{
    public DNIException(String mensaje){
        super(mensaje);
    }
}
```

La clase DNI hay que actualizarla de forma que
throw new Exception()
ahora es
throw new DNIException()

```
//DNI.java
package dni;
public class DNI {
    private String dni;//8 digitos + letra
    private static final String LETRAS_DNI = "TRWAGMYFPDXBNJZSQVHLCKE";

    public DNI(String dni) throws DNIException {
        //suponemos que un dni ES CORRECTO SI tiene  8 digitos + una letra. Sin guiones.
        //comprobamos si llega String null
        if (dni == null) {
            throw new DNIException("dni con valor nulo");
        }
        //comprobamos si la longitud del string no es exactamente 9
        if (dni.length() != 9) {
            throw new DNIException("la longinturd de "+ dni +" no es de 9 carateres");
        }

        //comprobamos que último caracter es letra
        //y la pasamos a mayuscula si no estuviera para simplificar ifs
        char letraDni = dni.charAt(dni.length() - 1);
        letraDni=Character.toUpperCase(letraDni);
```

```
        if (!Character.isLetter(letraDni)) {
            throw new DNIException("el último caracter de "+ dni +" no es letra ");
        }

        //comprobamos que los 8 primeros caracteres son números
        String parteNumero = dni.substring(0, dni.length() - 1);
        if (!esNumero(parteNumero)) {
            throw new DNIException("los 8 primeros caracteres de "+ dni +" no son todos números");
        }

        //compruebo el algortimo de la letra que detecta si hay error tanto en letra como en número
        char letraCorrecta = calcularLetra(parteNumero);
        if (letraDni != letraCorrecta) {
            throw new DNIException(" el dni  "+ dni +" no cumple algortimo de validación módulo 23");
        }
        this.dni = dni;
    }

    //algún posible método que podría interesar
    public String getDNI() {
        return this.dni;
    }

    public String getDNISoloNumero() {
        return this.dni.substring(0, this.dni.length() - 1);
    }

    public char getDNISoloLetra() {
        return this.dni.charAt(dni.length() - 1);
    }
    //aunque ahora todos los métodos son públicos las code conventions no obligan a que esten juntos
    //se pueden intercalar con los públicos sin problemas

    private boolean esNumero(String s) {
        for (char c : s.toCharArray()) {
            if (!Character.isDigit(c)) {
                return false;
            }
        }
        return true;
    }
    //con excepciones es menos eficiente
    private boolean esNumero2(String s) {
        int numero = 0;
        try {
            numero = Integer.parseInt(s);
        } catch (NumberFormatException e) {
            return false;
        }
        return true;
    }

    private char calcularLetra(String parteNumero) {
        int numero = Integer.parseInt(parteNumero);
        return LETRAS_DNI.charAt(numero % 23);
    }

}
```

Y App.java funcionaría tal cual estaba ya que en catch(Exception *e*) *e* puede "atrapar" un objeto DNIException, pero en cualquier caso sería más apropiado que App también manejara DNIException ya que así:

- podría acceder a métodos propios(no heredados) de DNIException (no los tenemos en el ejemplo)
- si en el try hubiera muchas excepciones se podrían producir muchos tipos de excepciones y puedo discriminar tratamiento por tipo de excepcion

Observa ahora el import (no era necesario para Exception, debes saber por qué)

```
import dni.DNIException;
public class App{
   public static void main(String[] args) {
      //metemos tb. un string vacio ""
      String[] dnis ={"12345678A","123456789A","123456789","0000001R","00000001R","","123abv11a"};
      for(String dni:dnis){
            try{
             System.out.print("\nintentando crear "+ dni+": ");
                  new dni.DNI(dni);
                  System.out.print(dni+ " icreado con exito!");
            }catch(DNIException e){
                  System.out.print(e.getMessage());
            }

      }
   }
}
```

Ejercicio U5_B8B_E8:

```
package personas;

import dni.DNI;
import dni.DNIException;

public class Empleado {
   DNI DNI;//Raro pero posible, atributo igual nombre que clase
   String nombre;
   double sueldo;

   public Empleado(String  DNI, String nombre, double sueldo) throws DNIException {
      this.DNI = new DNI(DNI);
      this.nombre = nombre;
      this.sueldo = sueldo;
   }
}
```

```
//App.java
import dni.DNIException;
import personas.Empleado;

public class App{
   public static void main(String[] args) {
      try {
```

```
            new Empleado("44444444H","YO",1000);
            System.out.println("44444444H es correcto");
        } catch (DNIException ex) {
            System.out.println(ex.getMessage());
        }
        try {
            new Empleado("44444444A","tu",2000);
            System.out.println("44444444A es correcto");
        } catch (DNIException ex) {
            System.out.println(ex.getMessage());
        }
    }
}
```

una excepción se lanza en un método y esta excepción va propagándose (throws) de método en método hasta encontrar una método que la capture(catch). Lo habitual es que la clase que capture la excepción es que sea la clase cliente "más externa" que en nuestro caso será la clase principal que contiene el main en los programas de consola.