Trabajo Práctico N° 4: Diseño por Contratos

1. Defina los contratos de las siguientes clases:

a. La clase Fecha representa una fecha en el formato dia, mes y año. Especificar los contratos para asegurar el manejo correcto de la clase Fecha.

b. En un sistema Java que gestiona los usuarios de un servicio telemático se utiliza una clase Usuario. Especificar los contratos para asegurar una facturación correcta.

```
class Usuario {
  private String nombre, dni;
  private int cuenta;

Usuario (String d, String n)
{ nombre = n, dni=d; cuenta=0;}

void conexion (int s) // Contabiliza 's' segundos en la cuenta
{ cuenta = cuenta + s;}

double calculaFacturación() //Calcula el importe facturable
{ return cuenta * 0,32; }

void reset() // Pone a cero la cuenta
{ cuenta = 0;}
}
```

c. En una Cuenta de Ahorro el saldo mínimo es de $ 0,00. Las extracciones nunca pueden superar el saldo disponible y los depósitos válidos son importes superiores a $ 0,00.
Definir los contratos y completar la implementación propuesta.

```
class CuentaDeAhorro {
   private int numero;
   private float saldo;

  CuentaDeAhorro(int numero)
  {  this.numero=numero;}
  void extraer(double importe)
  {saldo = saldo - importe;}
  void depositar(double importe)
  {saldo = saldo + importe;}
  float getSaldo()
  {return saldo;}
  int getNumero()
  {return numero;}
}
```

d. Una Cuenta Corriente es un tipo de cuenta bancaria muy parecida a una Cuenta de Ahorro. El descubierto es una cantidad de dinero limite que el banco autoriza para que se puedan realizar extracciones cuando el saldo = 0,00.
Definir los contratos y completar la implementación propuesta.

```
class CuentaCorriente {
   private int numero;
   private float saldo;
   private float descubierto;

  CuentaCorriente(int numero, float descubierto)
  {  this.numero=numero;
     this.descubierto=descubierto;}
  void extraer(double importe)
  {saldo = saldo - importe;}
  void depositar(double importe)
  {saldo = saldo + importe;}
  float getSaldo()
```

```
  {return saldo;}
  float getDescubierto()
  {return descubierto;}
    int getNumero()
  {return numero;}
}
```

e. En un Video Club cuando se realiza el alquiler de una película se registra el id de la película y la fecha de alquiler. Mientras la película esta alquilada el estado es verdadero. Cuando se produce la devolución de la película se registra la fecha de devolución y el estado del alquiler pasa a falso. Solo se calcula la facturación de los alquileres cuyo estado es falso. La facturación se obtiene aplicando la siguiente formula: *(Hasta – Desde) * coef*. Donde *coef = 0,27* si el alquiler no supero los 7 días; *coef = 0,39* si el alquiler superó los 7 días pero no los 15 días y *coef = 0,55* para el resto de los casos.
Definir los contratos y completar la implementación propuesta.

```
class Alquiler {
  private String Id_pelicula
  private Fecha Desde;
  private Fecha Hasta;
  private boolean estado;
  Alquiler(String id_pelicula, Fecha desde)
  {}
  void devolucion(..){}
  boolean getEstado(){}
  float facturarAlquiler(){}
}
```

f. Un Plazo Fijo es una operación bancaria de inversión que permite a los clientes obtener un beneficio (interés) económico por prestarle al banco una determinada cantidad de dinero en un periodo de tiempo estipulado. El beneficio de un plazo fijo se calcula según la siguiente tabla. (M es monto).

Días	M < 10000	M < 50000	M >=50000
7	0,05	0,05 + (0,8% del interés)	0,05 + (1% del interés)
14	0,075	0,075 + (1% del interés)	0,075 + (1,2% del interés)
30	0,15	0,15+ (1,2% del interés)	0,15 + (1,4% del interés)
60	0,30	0,30+ (1,4% del interés)	0,30 + (1,6% del interés)
90	0,60	0,60+ (1,6% del interés)	0,60 + (1,8% del interés)
120	1,2	1,2+ (1,8% del interés)	1,2 + (2% del interés)

```
class PlazoFijo {
  private int DNI_Cliente;
  private float monto;
  private int dias;

  PlazoFijo(..){}
  float calcularBeneficio(){}
  void renovar(){} // la renovación implica redefinir el mismo plazo fijo con el
monto anterior, mas los beneficios obtenidos. La cantidad de días puede ser
diferente.
  int getDias(){}
}
```

Definir los contratos y completar la implementación propuesta.

2. La clase Rectángulo tiene los atributos base y altura, los cuales se utilizan por ejemplo para el cálculo de superficie y perímetro. Los asertos `base > 0` y `altura > 0` deben ser pre-condiciones de los métodos *Superficie()* y *Perímetro()* o invariantes de clase????
Crear los contratos para la clase Rectángulo.

3. Construir la clase “Fraccion” y sus contratos para el tratamiento de Fracciones que realice las siguientes operaciones:

- Constructor
- Acceso al numerador y al denominador

- Operaciones aritméticas entre fracciones, tales como: suma, resta, producto, cociente y simplificar una fracción. (en todos los casos devuelve otra fracción como resultado)
- Operaciones de comparación de fracciones, tales como: igualdad, mayor, menor, mayor o igual y menor o igual.
- Operaciones de lectura y escritura de fracciones en la entrada/salida estándar.

4. Construir la clase "Conjunto" y sus contratos para tratar la abstracción matemática **Conjunto**, que maneje conjuntos de números naturales del 1 al 100.

Características:

- Descripción por enumeración, por ejemplo {1,4,6}
- No hay orden asociado.
- No hay elementos repetidos.
- No tiene sentido manipular los componentes individuales.
- Sugerencia: implementar con un array de boolean

Operaciones:

- Crear el conjunto vacío.
- Insertar un elemento.
- Borrar un elemento.
- Comprobar si el conjunto está vacío, si un elemento pertenece a un conjunto y si dos conjuntos con iguales.
- Unión de conjuntos (devuelve el resultado en un 3er conjunto)
- Intersección de conjuntos (devuelve el resultado en un 3er conjunto)
- Resta de conjuntos (devuelve el resultado en un 3er conjunto)
- Complementario de un conjunto (devuelve el resultado en un 3er conjunto)
- Lectura y escritura de conjuntos en la entrada/salida estándar.

5. La Batalla Naval es un juego que consiste en hundir los barcos enemigos cuya localización inicialmente se desconocen. La simulación del juego puede realizarse de manera muy sencilla con una matriz (que representa el mar), en la cual ciertas celdas se identifiquen de manera tal que representan un barco.
Teniendo en cuenta que los barcos se pueden ubicar en forma vertical u horizontal, y que en una posición no pueden haber más de un barco, se deben ubicar 6 barcos: uno de longitud 4, dos de longitud 3 y tres de longitud 2 en la matriz de 15 x 15. En una celda no pueden superponerse dos barcos.
Luego comienza el juego en el cual se realizan una serie de disparos hasta hundir los 6 barcos. Solo el primer disparo que acierta en una posición que ocupa un barco se contabiliza como acierto. Es decir, el número total de aciertos será de 16. Al finalizar el juego se muestran la cantidad de disparos efectuados.
A continuación se propone el código preliminar del juego.

```
class BatallaNaval {
   private int mar[15][15];
   private int aciertos;
   private int disparos;

   BatallaNaval()
   {// inicializar atributos}
   void ubicarBarco(…)
   { // se ubica en mar un barco de longitud n, en forma vertical u horizontal}
   void disparo(…)
   { // se chequea si el disparo efectuado acierta en algún barco
     // se incrementa aciertos si corresponde
     // se incrementa disparos }
   int getAciertos()
   { return aciertos;}
   int getDisparos()
   { return disparos;}
}

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

```
  // Comienza el Juego !!!
  BatallaNaval bn = new BatallaNaval();

  // Ubicar los 6 barcos
  bn.UbicarBarco(..) // ubicar barco 1

  bn.UbicarBarco(..) // ubicar barco 6

  // Hundir los barcos
  do {
    // toma datos para efectuar el disparo
    Ba.disparo(..)
  } while ( bn.getAciertos() <= 16)

  // mostrar resultados
}
```

Se pide:
Defina los contratos de Batalla Naval
Complete la implementación propuesta.
(*) usar números ramdon.

6. Los robots son utilizados para realizar tareas de alto riesgo para los seres humanos. El robot DM, tiene la peligrosa misión de hallar una ruta segura por la cual los seres humanos puedan atravesar el campo minado. Para esto, DM debe previamente recorrer el campo y desactivar las minas que ha ocultado el enemigo en su camino. El robot DM recorre el campo paso a paso. Cada paso se realiza en dirección: adelante, atrás, izquierda o derecha. Antes de realizar un movimiento, el robot debe chequear si en esa próxima ubicación no hay una mina. Esta detección la realiza con un sensor que lleva incorporado. Si en dicha posición hay una mina, el robot DM puede: a) desactivarla para avanzar hacia esa ubicación; o b) intentar moverse a otra ubicación. Cada vez que DM desactiva una mina, gasta una carga de explosivo.
El campo se representa con una matriz de 8 x 8 y DM comienza en [0][0], hasta llegar a [7][7]. En el campo hay esparcidas un total de 25 minas. DM solo tiene 9 cargas de explosivo.
A continuación se propone el código preliminar del juego.

```
class Campo {
 prívate boolean [8][8] campo;
 Campo()
 {// ubica las 25 minas en campo (*)}
 booolean hayMina(fila, columna)
 { // chequea si hay mina en fila, columna }
 void desactivarMina(fila, columna)
 { // desactiva la mina de fila, columna}
}

class Robot{
 private int posición_fila;
 private int posición_columna;
 private int cargas;

 Robot()
 {inicializa cargas}
 void Inicio()
 { //avanza a la posición 0,0}
 void pasoAdelante(){// avanza una fila}
 void pasoAtras(){ // retrocede una fila}
 void pasoIzquierda(){//retrocede una columna}
 void pasoDerecha(){// avanza una columna}
 desactivarMina(){// gasta una carga}
 int getPosicionFila(){// completar}
 int getPosicionColumna(){// completar}
}

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

```
  Campo c = new Campo();
  Robot DM = new Robot();

  if (c.hayMina(0,0))
  {  DM.desactivar();
     c.desactivarMina();}
  DM.inicio();
  do {
   // decide hacia donde avanzar (*)
   // chequea si hay mina
   // si no hay mina: avanza
   // si hay mina: decide si a) desactiva o b) busca nueva posición (*)
   // a) desactiva y avanza
   // b) lo hace en la proxima pasada x el bucle
  } while (DM.getPosicionFila==7 && DM.getPosicionColumna==7 || DM.getCargas()==-1)
}
// mostrar camino seguro.
```

Se pide:
Defina los contratos del sistema
Complete la implementación propuesta, teniendo en cuenta que falta registrar el camino seguro.
(*) usar números random para