

```

package Testing.Cuenta;

import Testing.Exceptions.CuentaException;

public class Cuenta
{
    private IAdminCuenta admin;
    private HistorialCuenta hist;

    private String IDCliente;
    private int fondosDisponibles;
    private int umbral;

    public Cuenta(String id, IAdminCuenta admin) {
        this.admin = admin;
        IDCliente = id;
        fondosDisponibles = 0;
        hist = new HistorialCuenta();
        umbral = admin.calcularUmbral(hist, fondosDisponibles);
    }

    public void ingresar(String fecha, int cantidad) {
        if(cantidad ≤ 0) throw new CuentaException("El valor debe ser positivo.");
        fondosDisponibles+=cantidad;
        hist.addIngreso(fecha, cantidad);
        umbral = admin.calcularUmbral(hist, fondosDisponibles);
    }

    public void retirar(String fecha, int cantidad) {
        if(cantidad ≤ 0) throw new CuentaException("El valor debe ser positivo.");
        cantidad = (fondosDisponibles-cantidad < 0) ? fondosDisponibles : cantidad;
        fondosDisponibles-=cantidad;
        hist.addGasto(fecha, cantidad);
        umbral = admin.calcularUmbral(hist, fondosDisponibles);
    }

    public HistorialCuenta getHistorial() {
        return hist;
    }

    public String getID() {
        return IDCliente;
    }

    public int getFondosDisponibles() {
        return fondosDisponibles;
    }

    public int getUmbral() {
        return umbral;
    }
}

////////////////////////////////////
package Testing.Cuenta;

public interface IAdminCuenta
{
    public int calcularUmbral(HistorialCuenta hist, int disponible);
}

////////////////////////////////////
package Testing.Exceptions;

public class CuentaException extends RuntimeException{

    public CuentaException() {
        super();
    }
}

```

```

        public CuentaException(String msg) {
            super(msg);
        }
    }

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
package Testing.Cuenta;

import java.util.ArrayList;
import java.util.List;

public class HistorialCuenta {
    private List<String> ingresos;
    private List<String> gastos;

    public HistorialCuenta() {
        ingresos = new ArrayList<String>();
        gastos = new ArrayList<String>();
    }

    public List<String> getIngresos() {
        return ingresos;
    }

    public List<String> getGastos() {
        return gastos;
    }

    public void addIngreso(String fecha, int abs) {
        ingresos.add("Ingreso "+abs+" "+fecha);
    }

    public void addGasto(String fecha, int abs) {
        gastos.add("Gasto "+abs+" "+fecha);
    }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
package Testing.CuentaTest;

import Testing.Cuenta.*;
import Testing.Exceptions.CuentaException;

import static org.junit.Assert.*;

import org.junit.Before;
import org.junit.Test;

import static org.mockito.Mockito.mock;
import static org.mockito.Mockito.when;

import static org.mockito.Mockito.any;
import static org.mockito.Mockito.eq;

public class CuentaTest
{
    private String id;
    private Cuenta c;
    private HistorialCuenta hc;
    private IAdminCuenta admin;

    @Before
    public void inicializar() {
        id = "001";
        admin = mock(IAdminCuenta.class);
        c = new Cuenta(id, admin);
        hc = new HistorialCuenta();
    }
}

```

```

@Test
public void constructorCreaUnaCuentaCorrectamente() {
    when(admin.calcularUmbral(any(), eq(0))).thenReturn(0);

    assertEquals(hc.getGastos(), c.getHistorial().getGastos());
    assertEquals(hc.getIngresos(), c.getHistorial().getIngresos());
    assertEquals(id, c.getID());
    assertEquals(0, c.getFondosDisponibles());
    assertEquals(0, c.getUmbral());
}

@Test
public void ingresarIncrementaFondosDisponiblesCorrectamente() {
    when(admin.calcularUmbral(any(), eq(100))).thenReturn(30);
    c.ingresar("TODAY", 100);
    hc.addIngreso("TODAY", 100);

    assertEquals(hc.getGastos(), c.getHistorial().getGastos());
    assertEquals(hc.getIngresos(), c.getHistorial().getIngresos());
    assertEquals(id, c.getID());
    assertEquals(100, c.getFondosDisponibles());
    assertEquals(30, c.getUmbral());
}

@Test
public void retirarDecrementaFondosDisponiblesCorrectamente() {
    when(admin.calcularUmbral(any(), eq(100))).thenReturn(30);
    when(admin.calcularUmbral(any(), eq(50))).thenReturn(15);
    c.ingresar("TODAY", 100);
    c.retirar("TODAY", 50);
    hc.addIngreso("TODAY", 100);
    hc.addGasto("TODAY", 50);

    assertEquals(hc.getGastos(), c.getHistorial().getGastos());
    assertEquals(hc.getIngresos(), c.getHistorial().getIngresos());
    assertEquals(id, c.getID());
    assertEquals(50, c.getFondosDisponibles());
    assertEquals(15, c.getUmbral());
}

@Test
public void intentarRetirarMasDeLoQueHaySoloRetiraHastaCero() {
    when(admin.calcularUmbral(any(), eq(100))).thenReturn(30);
    when(admin.calcularUmbral(any(), eq(0))).thenReturn(0);
    c.ingresar("TODAY", 100);
    c.retirar("TODAY", 200);
    hc.addIngreso("TODAY", 100);
    hc.addGasto("TODAY", 100);

    assertEquals(hc.getGastos(), c.getHistorial().getGastos());
    assertEquals(hc.getIngresos(), c.getHistorial().getIngresos());
    assertEquals(id, c.getID());
    assertEquals(0, c.getFondosDisponibles());
    assertEquals(0, c.getUmbral());
}

@Test (expected = CuentaException.class)
public void ingresarConValorNegativoLanzaCuentaException() {
    c.ingresar("TODAY", -1);
}

@Test (expected = CuentaException.class)
public void retirarConValorNegativoLanzaCuentaException() {
    c.retirar("TODAY", -1);
}
}

```