Simulacro del Primer Parcial

1. Dado el siguiente fragmento de programa, indicar:

   a. Indicar cuál es la relación entre `Personaje` y `Pantalla`.

   b. Indicar si viola algún principio de diseño y, en caso afirmativo, indique cuál(es) y explique por qué lo hace.

```
public interface Personaje {
  int getPosicionX();
  int getPosicionY();
  void caminarALaDerecha();
  void caminarALaIzquierda();
  void saltar();
  void disparar();
}

public class Pantalla {
  public void mostrar(Personaje p) {
    System.out.println("El personaje está en las coordenad
        p.getPosicionX(),
        p.getPosicionY()
      );
  }
}
```

métodos de más (al menos con respecto a la pantalla)

a) Relación entre Personaje y Pantalla:
La Pantalla USA un personaje, la relación es DEPENDENCIA

b) ISP: tengo métodos que no son necesarios para Pantalla (los que tienen que ver con movimientos)

2) Explicar en palabras cuál es la relación entre las clases A y B. ¿Qué diferencia habría si el rombo no estuviera pintado?

Relación: COMPOSICIÓN

La clase A se compone de cero o muchos B

Si A deja de existir, "los" de B también dejarían de haberlo.

ROMBO SIN PINTAR: AGREGACIÓN

La clase A tiene 1 o más B y temporalmente será dueño de las instancias de B's.

ejemplo: un auto con 4 ruedas.

3. Dado el siguiente programa:

a. Indicar si es factible escribir pruebas unitarias para la función `torresDeHanoi`. En caso negativo indicar cuál es la razón y proponer un cambio para que la función sea *testeable*.

b. Proponer la menor cantidad de pruebas unitarias que deberían considerarse para cubrir todas las clases de equivalencia posibles.

lo único que hace es imprimir cosas en pantalla. (no puedo verificar con asserts)

```
class TDH
{
    public static void torresDeHanoi(int n, char desde_tor
    {
        (1) if (n < 1) throw new IllegalArgumentExcept
        (2) if (n > 20) throw new IllegalArgumentExcep
        if (n == 1) (3)
        {
            System.out.println("Mover el disco 1 desde la
            return;
        } (4)
        torresDeHanoi(n-1, desde_torre, aux_torre, hacia_t
        System.out.println("Mover el disco " + n + " desde
        torresDeHanoi(n-1, aux_torre, hacia_torre, desde_t
    }

    public static void main(String args[])
    {
        int n = 4; // Cantidad de discos
        torresDeHanoi(n, 'A', 'C', 'B');  // A, B y C son
```

DIP en vez de depender de la implementación concreta depende de la abstracción

(clases de equivalencia)

cobertura: verificar todas las ramificaciones

ith Super

la función no es testeable a partir de pruebas unitarias pues para ello debería ingresar un input conocido y que el método pueda devolver un output conocido y evaluar la salida.

n =

(1) -1

(2) 25

1

15

Deberíamos obtener excepción excepción

4. Dado el siguiente fragmento de programa, indicar:

a. Dibujar el diagrama de clases

b. Indicar si está basado en algún patrón de diseño, y cuál.

c. Indicar si el código es candidato a ser refactorizado mediante algún otro patrón de diseño y, en caso afirmativo, indicar mediante cuál(es) y explicar cómo lo haría.

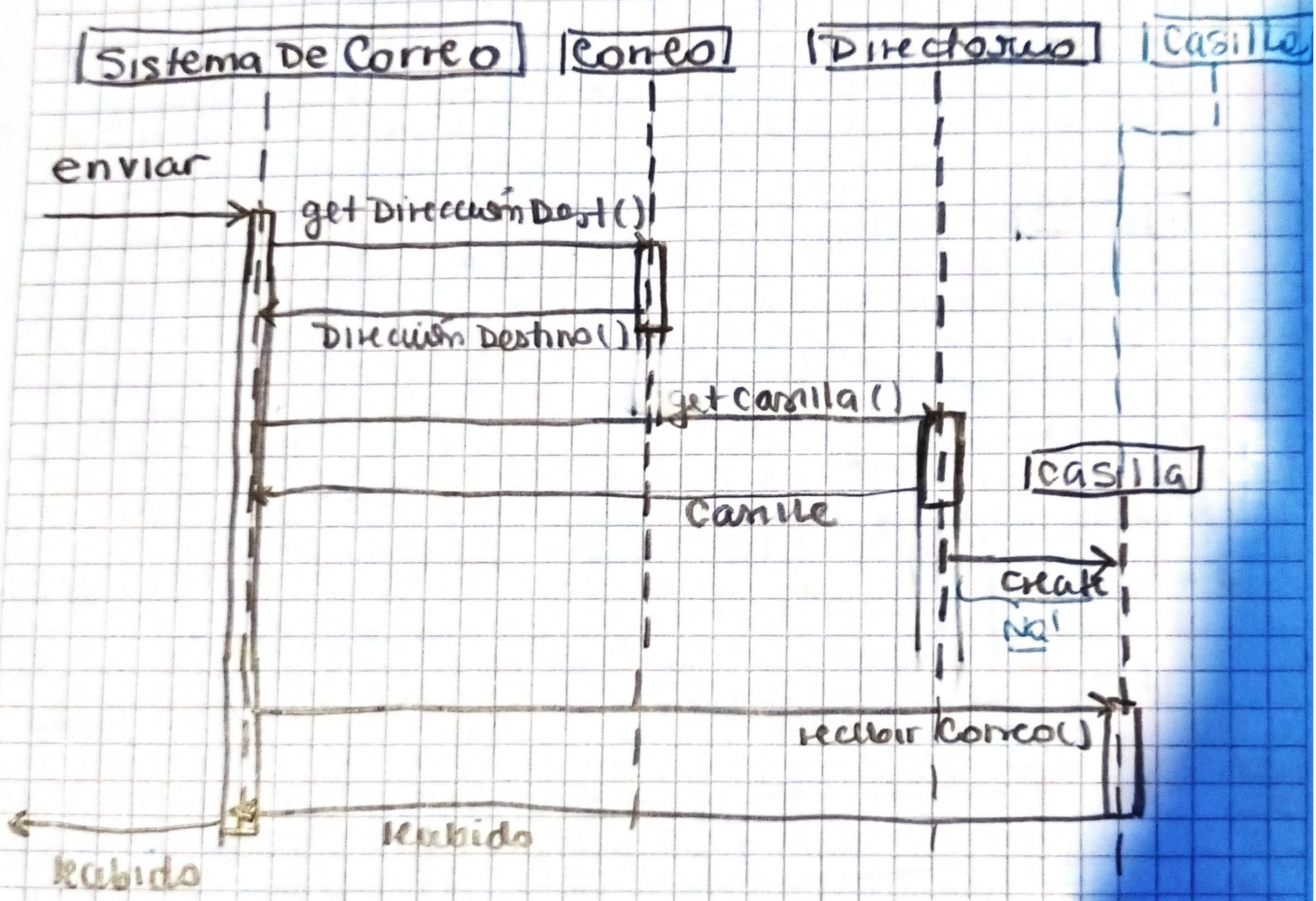
```
public interface FabricaDeFiguras {
  Triangulo crearTriangulo();
  Rectangulo crearRectangulo();
}

public interface Triangulo { ... }
public interface Rectangulo { ... }

public class FabricaDeFigurasRojas implements FabricaDeFigu
  @Override Triangulo crearTriangulo() { return new Triangu
  @Override Rectangulo crearRectangulo() { return new Rect
}

public class FabricaDeFigurasAzules implements FabricaDeFi
  @Override Triangulo crearTriangulo() { return new Triangu
  @Override Rectangulo crearRectangulo() { return new Rect
}

public class TrianguloRojo implements Triangulo { ... }
public class RectanguloRojo implements Rectangulo { ... }

public class TrianguloAzul implements Triangulo { ... }
public class RectanguloAzul implements Rectangulo { ... }
```

b) Está basado en el abstract factory

c) como tenemos explosión de clases, conviene BRIDGE.

5. Dibujar un diagrama de secuencia resumiendo el siguiente fragmento de programa:

```
class SistemaDeCorreo {
  public bool enviar(Correo correo, Directorio d) {
    String direccionDestino = correo.getDireccionDestino();
    Casilla destino = d.getCasilla(direccionDestino);
    bool recibido = destino.recibirCorreo(correo);
    return recibido;
  }
}
```



Si en el código veo new → sí pongo el create(). y sobre justo le lugar donde lo creo