Práctico de Programación Defensiva
Taller de Programación I

Tomando como referencia los siguientes pseudocódigos, las clases definidas y el contrato especificado codifique en java los siguientes métodos con las pre, post condiciones e invariantes de clase necesarios.

Tenga en cuenta si corresponde agregar una excepción o un if de validación. O sea, complete agregando lo que sea necesario (pre, post, inv, if, excepción). También debe agregar el retorno de cada método.

Tener en cuenta que algunas clases son propias de la capa del modelo, y otras están más cerca de la capa de comunicación con el usuario. No en todas las capas debe ir todo, o sea, se debe determinar en dónde corresponde poner pre, post, inv, if, excepción

Para la codificación de los métodos tener en cuenta las siguientes recomendaciones:

Suponiendo un método metodo1 de la Clase1: Clase1→ método1(parámetro) se debe incluir la descripción del método la cual debe incluir:

- El contrato
- Teniendo en cuenta el paquete/capa (modelo, vista o negocio) a la cual pertenece el método, se debe completar el código con alguna, ninguna o todas estas opciones:
  - precondición
  - postcondición
  - invariante de clase
  - excepción (lanzamiento, propagación, try, catch)
  - if de validación.

En todos los casos agregar lo necesario y al decidir que va en cada método tener en cuenta la cadena de invocación Interfase de usuario (UI) → metodo1() capa de negocio → metodo1() capa de modelo.

Caso 1: Una cerveceria.

Requerimientos aplicables:

1. El local, para estar abierto, debe tener al menos una cantidad de mesas >= 1 y una
2. cantidad de productos >=1.
3. Al inicio de la jornada, todas las mesas deben estar desocupadas.
4. Al inicio de la jornada, la carta debe estar actualizada con todos los productos disponibles.
5. Al inicio de la jornada, se determina la cantidad de mesas que se van a habilitar. No necesariamente se deben habilitar todas las mesas.

Metodos a codificar:

Clase BeerHouse (Capa de Modelo)

BeerHouse→ abrirLocal(int cantMesas)

- cantMesas es la cantidad de mesas que se habilitan para la jornada.
- nicializa el estado de todos los atributos del local al inicio de la jornada.

BeerHouse → ocuparMesa(int nroMesa)

- Retorna la instancia de mesa de la lista de mesas que tiene el nroMesa indicado en el parámetro y establece el atributo uso como ocupada (‘O’)

BeerHouse → cerrarMesa(int nroMesa)

- Calcula y retorna el importe total consumido de la mesa indicada como parámetro y establece el atributo uso como libre (‘L’).

Clase Negocio (capa de negocio)

Negocio → abrirLocal(int cantMesas)

- Recibe de la interfaz de usuario la cantidad de mesas que se habilitan para la jornada sin ningún tipo de validación.

Negocio → ocuparMesa(int nroMesa)

- Recibe el nroMesa de la interfaz de usuario sin ningún tipo de validación. Retorna instancia de mesa correspondiente. Esto lo usará la interfaz de usuario.

Negocio → cerrarMesa(int nroMesa)

- Recibe el nroMesa de la interfaz de usuario sin ningún tipo de validación. Calcula y retorna el importe total consumido de la mesa indicada como parámetro.

Clase Interfaz de usuario (UI)

UI → abrirLocal()

- Lee un número de cantidad de mesas ingresado por el usuario.

UI → cerrarMesa()

- Lee un número de mesa ingresado por el usuario.

UI → ocuparMesa()

- Lee un nroMesa ingresado por el usuario.

Práctico de Programación Defensiva
Taller de Programación I

Caso 2 : Surtidor de Combustible

Requerimientos aplicables:

1. El surtidor está conectado a un deposito con capacidad máxima de 2000 litros, todo intento de cargar más de eso se aborta, no se realiza la carga.
2. El surtidor se inicia con una cantidad inicial de combustible>=1 y el resto de las variables (acumulado de ventas y última venta de cada mangueras) en cero.
3. El surtidor para estar en funcionamiento debe tener combustible (cantidad_Combustible>=1).
4. Cuando el surtidor se queda vacío, la única operación posible es la recarga de combustible.
5. El surtidor cuenta con dos mangueras que entregan combustible cuando el aperador acciona el gatillo de la boquilla, esta acción transcurre mientras hay combustible en el surtidorhasta que el operador libera el gatillo.
6. Las mangueras entregan combustible a razón de un litro por segundo, reduciendo el stock del surtidor y mostrando en la pantalla del surtidor el total de combustible restante en el surtidor y el entregado por cada manguera (acumulado) y en la última compra.

Métodos a codificar:

Clase Surtidor (Capa de Modelo)

Surtidor→ InicializarSurtidor(Real carga)

- carga es la cantidad de combustible inicial que se incorpora en cantidad_Combustible.
- Inicializa acumuladoManguera1, acumuladoManguera2, ultimaventaManguera1 y ultima venta Manguera2 en cero.

Surtidor → cargarSurtidor(Real carga)

- incorpora la cantidad que indica carga en el surtidor incrementando la variable cantidad_Combustible.

Surtidor → descargaManguera1();

- Recibe de la capa de negocio el pedido para que la Manguera1 comience a descargar, verifica si es posible, en ese caso comienza la descarga.
- Sigue descargando a razón de un litro por segundo hasta que se reciba la señal de desactivar la manquera o bien el deposito se vacíe. En ambos casos actualiza los acumulados de la manguera 1 y la existencia, e informa a la capa de negocio lo que acontece.
- Si la acción es imposible (informa a la capa de negocio.

Surtidor → activaManguera2()

- Recibe de la capa de negocio el pedido para que la Manguera1 comience a descargar, verifica si es posible, en ese caso comienza la descarga.

- Sigue descargando a razón de un litro por segundo hasta que se reciba la señal de desactivar la manquera o bien el deposito se vacíe. En ambos casos actualiza los acumulados de manguera 1 y la existencia, e informa a la capa de negocio lo que acontece.
- Si la acción es imposible (informa a la capa de negocio.

Surtidor → real getExistenciaDeposito();

- Devuelve la existencia del deposito desde la capa de modelo

Surtidor → real getAcumuladoManguera1()

- Devuelve el acumulado de la manguera 1, desde la capa del modelo

Surtidor → real getAcumuladoManguera2()

- Devuelve el acumulado de la manguera 2, desde la capa del modelo

Surtidor → real getUltimaVentaMG1();

- Devuelve la última venta de la manguera 1

Surtidor → real getUltimaVentaMG2();

- Devuelve la última venta de la manguera 2

Clase Negocio (capa de negocio)

Negocio→ Inicializa Surtidor (Real carga)

- Recibe de la interfaz de usuario la cantidad de litros de combustible a cargar en el depósito del surtidor sin ningún tipo de validación.
- Envía la orden a la capa del modelo e informa a la interfaz el éxito o fracaso de la operación.

Negocio→ cargaSurtidor(Real carga)

- Recibe de la interfaz de usuario la cantidad de litros de combustible a cargar en el depósito del surtidor sin ningún tipo de validación.
- Envía la orden a la capa del modelo e informa a la interfaz el éxito o fracaso de la operación.

Negocio→activaManguera1()

- Recibe de la interfaz de usuario el pedido para que la Manguera1 comience a descargar, la capa de negocio transfiere a la capa de modelo.
- Informa si la acción es posible o no (según informa la capa de modelo).

Negocio → activaManguera2()

- Recibe de la interfaz de usuario el pedido para que la Manguera2 comience a descargar, la capa de negocio transfiere a la capa de modelo.
- Informa si la acción es posible o no (según informa la capa de modelo).

Negocio → DesactivaManguera1()

- Recibe de la interfaz de usuario que se para la descarga de combustible.
- Informa a la interfaz que la operación se detiene porque no hay más combustible cuando le informa la capa de modelo y detiene la descarga automáticamente.

Negocio → DesactivaManguera2()

- Recibe de la interfaz de usuario el pedido para que la Manguera1 comience a descargar, la capa de negocio transfiere a la capa de modelo.
- Informa si la acción es posible o no (según informa la capa de modelo).

Negocio → real getExistenciaDeposito();

- Devuelve la existencia del deposito desde la capa de modelo

Negocio → real getAcumuladoManguera1()

- Devuelve el acumulado de la manguera 1, desde la capa del modelo

Negocio → real getAcumuladoManguera2()•

- Devuelve el acumulado de la manguera 2, desde la capa del modelo

Negocio → real getUltimaVentaMG1();

- Devuelve la última venta de la manguera 1

Negocio → real getUltimaVentaMG2();

- Devuelve la última venta de la manguera 2

Clase Interfaz de usuario (UI)

UI → inicializaSurtidor()

- Lee la cantidad de combustible a cargar en el surtidor por primera vez e inicializa el surtidor, solicita la acción a la capa de negocio.
- Indica si la operación fue exitosa o un fracaso (según informa la capa de negocio).

Práctico de Programación Defensiva
Taller de Programación I

UI → cargarSurtidor()

- Lee la cantidad de combustible y da la orden a la capa de negocio de cargar en el surtidor.
- Indica si la operación fue exitosa o un fracaso (según informa la capa de negocio).

UI → activaManguera1()

- Se indica a la capa de Negocio que la Manguera1 comienza a descargar combustible (acción del usuario).
- Indica si la acción es posible o no (según informa la capa de negocio).

UI → activaManguera2()

- Se indica a la capa de Negocio que la Manguera2 comienza a descargar combustible (acción del usuario).
- Indica si la acción es posible o no (según informa la capa de negocio).

UI → DesactivaManguera1()

- Indica a la capa de negocio cuando el usuario detiene la descarga de combustible.
- Indica que no hay mas combustible cuando le informa la capa de negocio y detiene la descarga automáticamente.
- Muestra en Pantalla los diferentes acumulados que le informa la capa de negocio.

UI → DesactivaManguera2()

- Indica a la capa de negocio cuando el usuario detiene la descarga de combustible.
- Indica que no hay más combustible cuando le informa la capa de negocios y detiene la descarga automáticamente
- Muestra en pantalla los diferentes acumulados que le informa la capa de negocio.

Práctico de Programación Defensiva
Taller de Programación I

Caso 3 : Certificado de notas

Requerimientos aplicables:

1. Solo se puede obtener un certificado de un alumno que existe.
2. Todo alumno debe tener un Legajo (numero entero de 4 cifras), junto al nombre y apellido.
3. Existen solo cuatro Materias (Historia, Matemática, Literatura y Física).
4. Todo alumno junto a sus datos de identificaciónn, registra el estado y notas de cada materias, las cuales pueden tener los siguientes estados : A cursar, Cursada y Aprobada.
5. Además por cada materia existen una notas (de 0 a 10), valida solo si la materia esta aprobada.
6. El Certificado debe encabezarse con los datos de Identificación del alumno y a continuación informar el estado y notas de cada materia, por último adicionar la condición de regular o irregular del alumno, que resulta de tener dos o más materias a cursar.

Métodos a Codificar :

Clase Certificado (Capa de Modelo)

Certificado → pedirCertificado(Integer Legajo)

- Se busca el alumno en base al legajo y se actualizan las variables internas con sus datos.
- Si la operación fracasa se informa a la capa de negocio.

Certificado → String traerApellidoyNombre()

- Si el legajo existe devuelve el dato en caso contrario informa del fracaso a la capa de negocio.

Certificado → String traerEstado(String Materia);

- Si el legajo existe devuelve el dato en caso contrario informa del fracaso a la capa de negocio.

Certificado → String traerNota(String Materia);

- Si el legajo existe devuelve el dato en caso contrario informa del fracaso a la capa de negocio.

Certificado → String traerCondición();

- Si el legajo existe devuelve el dato en caso contrario informa del fracaso a la capa de negocio.

Práctico de Programación Defensiva
Taller de Programación I

Clase Negocio (Capa de Negocio)

Negocio → pedirCertificado(Integer Legajo)

- La capa de negocio pasa a la capa de modelo el legajo del alumno.
- Si la capa de modelo informa que el legajo no existe informa a la UI.

Negocio → String traerApellidoyNombre()

- Se le pasa la solicitud al modelo de negocio y este envia el dato del alumno que se solicitó anteriormente, si no está disponible (no existe, no se pidió nada, etc), el modelo informa la novedad.

Negocio → String traerEstado(String Materia);

- Se le pasa la solicitud al modelo de negocio y este envia el dato del alumno que se solicitó anteriormente, si no está disponible (no existe, no se pidió nada, etc), el modelo informa la novedad.

Negocio → String traerNota(String Materia);

- Se le pasa la solicitud al modelo de negocio y este envia el dato del alumno que se solicitó anteriormente, si no está disponible (no existe, no se pidió nada, etc), el modelo informa la novedad.

Negocio → String traerCondición();

- Se le pasa la solicitud al modelo de negocio y este envia el dato del alumno que se solicitó anteriormente, si no está disponible (no existe, no se pidió nada, etc), el modelo informa la novedad.

Capa Interfase de Usuario (UI) :

UI → pedirCertificado(Integer Legajo)

- La UI pasa a la capa de negocio el legajo del alumno que le es ingresado a través de otros métodos y sin validar.
- Si la capa de negocio informa que el legajo no existe ó a ocurrido un error informa.

UI→ mostrarEstado(String Mensaje)

- Muestra el mensaje en la pantalla de haber ocurrido un error.

UI→ MostrarCertificado();

- Muestra el certificado con los últimos datos que resultan de pedirCertificado

Práctico de Programación Defensiva
Taller de Programación I

Caso 4 : Calculadora

Requerimientos aplicables:

1. Calculadora para números enteros positivos, cuyos resultados también son enteros positivos (>0).
2. Las operaciones se realizan ingresando dos operandos, la operación y nos debe mostrar el resultado.
3. Se consideran la suma (+), la resta (-), la división (/) y la multiplicación (*).

Métodos a Codificar :

Clase Calcular (Capa de Modelo)

Modelo → Calcular(Integer PrimerOperando, Integer SegundoOperando, String Operacion)

- Recibe de la Capa de negocio los parámetros de la operación
- Valida los parámetros y si hay un error lo informa a la capa de negocio
- Efectúa el cálculo y si este es imposible lo informa.

Modelo → Integer traerResultado();

- devuelve el resultado de la última operación exitosa.

Clase Monitor (Capa de Negocio)

Negocio → Calcular(Integer PrimerOperando, Integer SegundoOperando, String Operacion)

- Recibe de la UI los parámetros de la operación y los pasa sin validar a la capa de modelo.
- Si la capa de modelo informa de un error pasa a la UI la novedad;

Negocio → Integer traerResultado();

- Nos trae desde la capa de modelo el resultado de la última operación exitosa.

Capa Interfase de Usuario (UI) :

UI → Calcular(Integer PrimerOperando, Integer SegundoOperando, String Operacion)

- La UI pasa a la capa de negocio los parámetros de la operación que le ingresan (en otros métodos) sin validar.
- Si la capa de negocio informa que la operación es imposible sea por que los parámetros son incorrectos o el resultado imposible lo muestra.

UI → mostrarEstado(String Mensaje)

- Muestra el mensaje de error de haber ocurrido.

UI → MostrarResultado()

- Muestra el resultado del último calculo sin error.

Práctico de Programación Defensiva
Taller de Programación I

Caso 5: Juego para adivinar un número

Requerimientos aplicables:

1. Cada vez que se inicia un juego, la aplicación genera un número entero positivo aleatorio entre 1 y 100.
2. El jugador tiene 10 posibilidades para adivinar el número, sino lo logra pierde y la aplicación queda a la espera de un nuevo juego.
3. Con cada intento el jugador propone un número y la aplicación responde uno de los siguientes estados : bajo, acertó, alto, perdió.

Métodos a Codificar :

Clase Juego (Capa de Modelo)

Modelo → Inicializar el juego()

- Recibe de la Capa de negocio el pedido
- Genera un número aleatorio entre 1 y 100.
- Pone en 0 la cantidad de Intentos.

Modelo → Probar(Real Numero)

- Valida que el número se un entero positivo entre 1 y 100, si no es así informa el error
- Valida que el último estado no sea ‘perdió’ o acerto’, e informa que debe llamarse a un nuevo juego.
- incrementa en uno la cantidad de intentos.
- Si el número es valido establece el estado de la prueba :
  - ‘bajo’ si el número ingresado es menor que el generado
  - ‘alto’ si el número ingresado es mayor que el generado
  - ‘acertó’ si el número ingresado es igual al generado
  - ‘perdió’ si la cantidad de intentos es igual a 10 y el número ingresado no es igual al generado

Modelo → Integer traerIntentoss();

- devuelve el número de intentos hacia la capa de negocio.

Modelo → String traerEstado()

- Devuelve el último estado a la capa de negocios

Clase Monitor (Capa de Negocio)

Negocio → InicializarJuego()

- Recibe de la UI el pedido de un nuevo juego y lo pasa a la capa de modelo

Negocio→ ProbarNumero(Real Numero)

- Recibe de la UI el número a probar y lo pasa a la capa de negocio.
- Si sucede un problema con el número ingresado lo informa a la UI
- Si recibe de la capa del modelo que el juego terminó lo informa a la UI

Negocio → Integer traerIntentos();

- Nos trae desde la capa de modelo los intentos realizados.

Negocio → String traerEstado()

- Nos trae de la capa de modelo el resultado de la última Prueba.

Capa Interfase de Usuario (UI) :

La interfase tiene los siguientes componentes :

- Botón Juego nuevo: inicializa el juego
- Botón Probar : Prueba con el numero ingresado.
- Campo para ingresar el número
- Campo que indica el número de intentos hasta el momento
- Campo que indica el estado (bajo, acertó, alto ó perdió) según el resultado de la prueba.

Tomando en cuenta este diseño tener en cuanta los siguientes métodos:

UI → inicializarJuego()

- La UI pasa a la capa de negocio el pedido de un nuevo juego.

UI → probarNúmero()

- La UI pasa a la capa de negocio el número ingresado sin validar (en otro método), para que este sea probado
- Si sucede un error recibido desde la capa de negocio, lo informa, aborta el intento espera a que le pidan probar de nuevo.
- Si recibe de la capa de negocio que el juego termino, bloque todas las acciones salvo inicializarJuego.

UI → mostrarEstado()

- Trae el estado de la última prueba desde la capa de negocio y lo muestre en la interfase

UI → MostrarIntentos()

- Trae el número de intentos realizados desde la capa de negocios y lo muestra en la interfase.