Retrospectiva 1:  
  
Retrospectiva

1. ¿Cuáles fueron los mini-ciclos definidos? Justifíquenlos.

Primer ciclo: Implementar las clases base (Position, Robot, Store).

Segundo ciclo: Crear la lógica de cálculo de profit con ProfitCalculator.

Tercer ciclo: Integrar SilkRoad con DayManager y MessageHandler para simular varios días.

Cuarto ciclo: Ajustar la visualización y resultados, verificando con ejemplos de entrada/salida.

2. ¿Cuál es el estado actual del proyecto en términos de mini-ciclos? ¿Por qué?

El proyecto está casi finalizado porque completamos todos los mini-ciclos y los resultados coinciden con los ejemplos oficiales excepto uno.

3. ¿Cuál fue el tiempo total invertido por cada uno de ustedes? (Horas/Hombre)

El tiempo total invertido fue de 12 horas Sergio Buitrago y 10 horas Santiago Garcia.

4. ¿Cuál consideran fue el mayor logro? ¿Por qué?

El mayor logro fue lograr que los resultados de profits fueran correctos en todas las pruebas, mostrando que el diseño de clases y la integración fueron acertados.

5. ¿Cuál consideran que fue el mayor problema técnico? ¿Qué hicieron para resolverlo?

El mayor problema fue el cálculo duplicado de profits con varios robots, que resolvimos simplificando la lógica para asignar cada tienda al robot más conveniente.

6. ¿Qué hicieron bien como equipo? ¿Qué se comprometen a hacer para mejorar los resultados?

Trabajamos bien organizándonos en distintas tareas pero siempre comunicandonos y haciendo todo juntos.

7. Considerando las prácticas XP incluidas en los laboratorios, ¿cuál fue la más útil? ¿Por qué?

La práctica más útil fue la programación incremental con pruebas continuas porque permitió detectar y corregir errores rápidamente.

8. ¿Qué referencias usaron? ¿Cuál fue la más útil? Incluyan citas con estándares adecuados.

Usamos Objects First with Java, la documentación oficial de Java y el proyecto shapes de BlueJ.

Retrospectiva ciclo 2:

1. ¿Cuáles fueron los mini-ciclos definidos? Justifíquenlos.

1. CICLO DE CREACIÓN DE OBJETOS

1.1 Mini-Ciclo de Inicialización del Mapa

Secuencia: SilkRoad() → ok() → makeVisible()

Propósito: Crear la estructura base del mapa con dimensiones válidas

Funciones: Inicializar la matriz del mapa, validar dimensiones mínimas/máximas, crear la cuadrícula de coordenadas

Validación: Verificar que el mapa tenga dimensiones apropiadas y estructura correcta

1.2 Mini-Ciclo de Posicionamiento del Robot

Secuencia: placeRobot() → getBestRobot() → reboot()

Propósito: Ubicar y configurar el robot en su posición inicial

Funciones: Colocar robot en coordenadas de inicio, validar que la posición esté libre, inicializar atributos del robot

Validación: Confirmar que el robot esté correctamente posicionado y configurado

1.3 Mini-Ciclo de Distribución de Tiendas

Secuencia: placeStore() → getBestStore() → stores()

Propósito: Distribuir estratégicamente las tiendas en el mapa

Funciones: Calcular posiciones óptimas, evitar solapamiento, asegurar accesibilidad desde el camino principal

Validación: Verificar que todas las tiendas sean accesibles y estén bien distribuidas

1.4 Mini-Ciclo de Inicialización de Recursos

Secuencia: resupplyStores() → removeStore() → robots()

Propósito: Configurar los recursos y precios iniciales del sistema

Funciones: Asignar productos a tiendas, establecer precios base, configurar inventarios

Validación: Confirmar que todos los recursos estén correctamente asignados

2. CICLO DE MOVIMIENTO OPTIMIZADO

2.1 Mini-Ciclo de Análisis del Entorno

Secuencia: robotsIn() → stores() → moveRobot()

Propósito: Analizar el entorno inmediato del robot para toma de decisiones

Funciones: Identificar robots en ubicación, listar tiendas disponibles, preparar movimiento básico

Validación: Asegurar que el análisis sea preciso y completo

2.2 Mini-Ciclo de Cálculo de Rutas

Secuencia: calculateOptimalPath() → moveRobotOptimized() → returnHome()

Propósito: Determinar y ejecutar la mejor ruta considerando múltiples factores

Funciones: Aplicar algoritmos de pathfinding, ejecutar movimiento optimizado, regresar a casa

Validación: Verificar que la ruta seleccionada sea óptima y factible

2.3 Mini-Ciclo de Gestión de Posición

Secuencia: removeRobot() → dropRobot() → finish()

Propósito: Gestionar cambios de posición y finalización de tareas

Funciones: Remover robot de posición actual, soltar robot en nueva ubicación, finalizar operación

Validación: Confirmar que los cambios de posición se ejecuten correctamente

2.4 Mini-Ciclo de Visualización Dinámica

Secuencia: moveRobotOptimizedColored() → makeVisible() → makeInvisible()

Propósito: Controlar la visualización del movimiento del robot

Funciones: Ejecutar movimiento con colores, mostrar elementos, ocultar elementos

Validación: Verificar que la visualización sea clara y funcional

3. CICLO DE CONSULTA

3.1 Mini-Ciclo de Procesamiento de Consultas

Secuencia: consult() → consultSilkRoad() → ok()

Propósito: Procesar y validar las consultas del usuario

Funciones: Realizar consulta básica, consultar información específica del camino de seda, validar operación

Validación: Asegurar que la consulta sea válida y procesable

3.2 Mini-Ciclo de Análisis de Rendimiento

Secuencia: calculateMaxProfit() → getBestRobot() → getBestStore()

Propósito: Analizar el rendimiento económico y seleccionar mejores opciones

Funciones: Calcular ganancia máxima, identificar mejor robot, identificar mejor tienda

Validación: Confirmar que los análisis sean matemáticamente correctos

3.3 Mini-Ciclo de Análisis de Recursos

Secuencia: getBestStore() → stores() → robots()

Propósito: Analizar y consultar el estado de recursos del sistema

Funciones: Identificar mejor tienda, listar todas las tiendas, listar todos los robots

Validación: Verificar que los datos de recursos sean precisos y actuales

3.4 Mini-Ciclo de Gestión del Sistema

Secuencia: create() → ok() → reboot()

Propósito: Gestionar operaciones generales del sistema

Funciones: Crear nuevos elementos, verificar estado del sistema, reiniciar sistema

Validación: Confirmar que las operaciones del sistema funcionen correctamente

2. ¿Cuál es el estado actual del proyecto en términos de mini-ciclos? ¿por qué?

Estado actual: Casi COMPLETADO

Funcionalidad básica implementada (creación de robots/stores) Extensiones de requisitos funcionales (10, 11, 12, 13) implementadas Integración con BlueJ funcional Todas las clases modificadas y compatibles

3. ¿Cual fue el tiempo total invertido por cada uno de ustedes? (Horas/Hombre)

Invertimos 8 horas Sergio Buitrago y 8 horas Santiago Garcia

4. ¿Cuál consideran fue el mayor logro? ¿Por qué?

Mayor logro: Implementación completa de los requisitos funcionales extendidos

5. ¿Qué consideran que es el mayor problema técnico? ¿Qué hicieron para resolverlo?

Mayor problema: Compatibilidad con BlueJ y gestión de tipos de datos Soluciones implementadas:

Problema de getPosition(): Creé métodos específicos (getPositionX()) para evitar conflictos Integración de objetos: Modifiqué métodos para retornar objetos (Robot, Store) directamente Manejo de estados: Implementé tracking de estados con isRemoved(), isActive() Estructuras de datos: Uso de HashMap y List para estadísticas eficientes

6. ¿Qué hicieron bien como equipo? ¿Qué se comprometen a hacer para mejorar los resultados?

Lo que hicimos bien:

Análisis completo de requisitos antes de implementar Código modular y bien documentado Pruebas incrementales de cada funcionalidad Mantenimiento de compatibilidad con código original

Compromisos de mejora:

Implementar más casos de prueba automatizados Mejorar documentación con ejemplos de uso Optimizar algoritmos de pathfinding para mejor rendimiento

7. Considerando las instancias XP incluidas, ¿cuál fue la más difícil?, ¿cuál la más útil?

Más difícil: Integración continua con BlueJ

Requirió múltiples iteraciones para que los objetos aparezcan correctamente Manejo de tipos de retorno y compatibilidad de métodos

Más útil: Refactoring iterativo

Permitió mejorar el diseño sin romper funcionalidad existente

8. ¿Qué referencias usaron? ¿Cuál fue la más útil? Incluyan citas con estándares adecuados.

Referencias principales:

Oracle Java Documentation (2023). Java Collections Framework. Oracle Corporation.

BlueJ Documentation (2023). BlueJ User Manual. University of Kent.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.

Retrospectiva Ciclo 3: Resolución de Problemas y Simulación

**1. ¿Cuáles fueron los mini-ciclos definidos? Justifíquenlos.**

CICLO 1: RESOLUCIÓN DEL PROBLEMA DE LA MARATÓN

1.1 Mini-Ciclo de Análisis de Entrada

Secuencia: SilkRoadSolver() → validación de datos → solve()

Propósito: Validar y preparar los datos de entrada para la resolución

Funciones: Verificar que posiciones, tenges y robots sean válidos; normalizar formato de entrada; inicializar estructuras internas

Justificación: La calidad de la resolución depende directamente de la validez de los datos entrada. Este ciclo asegura que errores de entrada se detecten temprano

Validación: Confirmar que datos nulos, vacíos o mal formados se rechacen apropiadamente

1.2 Mini-Ciclo de Cálculo de Profits

Secuencia: Generar pares (robot, tienda) → Calcular profit para cada par → Almacenar en estructura

Propósito: Calcular la matriz de profits para todas las combinaciones posibles

Funciones: profit = tenges\_tienda - distancia(robot, tienda) para cada par; crear instancias de ProfitAssignment; manejar profits negativos

Justificación: Este es el corazón del algoritmo. Necesita ser preciso porque todas las decisiones posteriores dependen de estos cálculos

Validación: Verificar que cada profit se calcule correctamente y que se identifiquen casos con profit negativo

1.3 Mini-Ciclo de Ordenamiento Greedy

Secuencia: Ordenar matriz de profits descendente → Seleccionar asignaciones óptimas → Construir solución

Propósito: Aplicar el algoritmo greedy para encontrar la asignación óptima

Funciones: Ordenar por profit descendente; validar restricciones (cada robot/tienda una sola vez); acumular profit total

Justificación: El ordenamiento garantiza que se seleccionen los pares más rentables primero, maximizando la utilidad total respetando restricciones

Validación: Confirmar que la solución no viola restricciones y que el profit total es correcto

1.4 Mini-Ciclo de Salida de la Solución

Secuencia: getOptimalAssignments() → getSolutionAsMatrix() → getMaxProfit()

Propósito: Retornar la solución en múltiples formatos para consumo por diferentes componentes

Funciones: Convertir asignaciones a lista, matriz y valores escalares; mantener consistencia entre formatos

Justificación: Diferentes partes del sistema necesitan la solución en diferentes formatos (lista para iterate, matriz para retorno, escalar para análisis)

Validación: Verificar que todos los formatos contengan exactamente la misma información

CICLO 2: SIMULACIÓN DE LA SOLUCIÓN

2.1 Mini-Ciclo de Validación de Precondiciones

Secuencia: simulate(solver) → Validar solver no nulo → Validar robots/tiendas existen

Propósito: Verificar que el estado de SilkRoad es válido antes de ejecutar la simulación

Funciones: Verificar que solver no sea nulo; que existan robots y tiendas; que sus índices sean válidos

Justificación: Prevenir errores de ejecución por precondiciones no cumplidas

Validación: Confirmar que excepciones se manejen gracefully

2.2 Mini-Ciclo de Ejecución de Movimientos

Secuencia: Obtener asignaciones → Para cada asignación: mover robot → recolectar tenges → registrar profit

Propósito: Ejecutar físicamente cada asignación de la solución en el simulador

Funciones: robot.moveToNewPos(store.position) ; robot.collectFromStore(store) ; robot.recordProfit(collected)

Justificación: Simular fielmente la solución calculada permite validar que los cálculos teóricos son correctos en la práctica

Validación: Verificar que cada robot llegue a la tienda correcta y recolecte exactamente lo esperado

2.3 Mini-Ciclo de Actualización de Estado

Secuencia: Cada colección → Incrementar tiendas vaciadas → Actualizar profits por movimiento → Actualizar tenges totales

Propósito: Mantener el estado del sistema consistente con los movimientos ejecutados

Funciones: store.collect() incrementa timesEmptied; robot.recordProfit() registra ganancia; robot.collect() acumula totales

Justificación: El estado del sistema es el reflejo de la historia de movimientos; debe estar siempre sincronizado

Validación: Verificar que profit total acumulado == maxProfit de solver al finalizar

2.4 Mini-Ciclo de Consistencia Final

Secuencia: Finalizar simulación → Validar que sum(robot.collected) == solver.maxProfit → Reportar resultado

Propósito: Verificar que la simulación reprodujo correctamente la solución teórica

Funciones: Sumar todos los tenges recolectados; comparar con maxProfit; generar reporte

Justificación: La simulación no tiene sentido si no reproduce exactamente la solución. Esta validación es crítica

Validación: Confirmar equivalencia numérica exacta

CICLO 3: INTEGRACIÓN SOLVE + SIMULATE

3.1 Mini-Ciclo de Flujo Completo

Secuencia: solver.solve() → silkRoad.simulate(solver) → silkRoad.consult()

Propósito: Permitir el flujo completo de resolver y simular en una operación cohesiva

Funciones: Mantener referencia entre solver y simulador; permitir que simulate acceda a asignaciones de solve

Justificación: El usuario debe poder resolver un problema, simular la solución y consultar resultados en secuencia natural

Validación: Verificar que los datos fluyan correctamente sin pérdida de información

3.2 Mini-Ciclo de Diferenciación solve vs moveRobots

Secuencia: moveRobots() (greedy automático) vs solve() + simulate() (óptimo prescrito)

Propósito: Mantener dos estrategias distintas para permitir experimentación y comparación

Funciones: moveRobots() toma decisiones locales en tiempo real; solve()+simulate() ejecuta plan óptimo predeterminado

Justificación: La separación permite comparar calidad de soluciones y entender diferencia entre greedy automático vs óptimo calculado

Validación: Verificar que ambas estrategias coexistan sin conflictos

**2. ¿Cuál es el estado actual del proyecto en términos de mini-ciclos? ¿Por qué?**

Estado del laboratorio: Casi completo ya que faltaría agregar pruebas conjuntas y pulir algunos detalles.

**3. ¿Cuál fue el tiempo total invertido por cada uno de ustedes? (Horas/Hombre)**

El tiempo total invertido fue de 15 horas Buitrago y 15 horas Garcia.

**4. ¿Cuál consideran fue el mayor logro? ¿Por qué?**

Mayor logro: Separación clara entre resolución y simulación

Justificación:

Arquitectura limpia: SilkRoadSolver (resuelve) y SilkRoad.simulate() (simula) son responsabilidades perfectamente separadas, siguiendo principios SOLID

Flexibilidad: Permite que usuarios puedan:

Resolver el problema sin simular

Simular soluciones diferentes

Comparar estrategias (greedy automático vs óptima)

**5. ¿Cuál consideran que fue el mayor problema técnico? ¿Qué hicieron para resolverlo?**

El mayor problema técnico fue asegurar que simulate() reprodujera exactamente la solución calculada por solve(). Esto requería garantizar que cada robot llegara a la tienda correcta y recolectara el amount exacto esperado. Se resolvió implementando validación de índices y estado de activos, verificación post-simulación que sum(robot.collected) == solver.maxProfit(), y pruebas unitarias específicas que detectaran cualquier desviación entre la solución teórica y su ejecución. Este enfoque proporciona garantía de fidelidad: simulate(solve()) es una ejecución exacta de la solución calculada, no una aproximación.

**6. ¿Qué hicieron bien como equipo? ¿Qué se comprometen a hacer para mejorar los resultados?**

Lo que hicimos bien:

Separación clara de responsabilidades

SilkRoadSolver conoce cómo resolver

40 casos de prueba cubriendo todos los requisitos

Documentar casos de uso comunes

**7. Considerando las prácticas XP incluidas en los laboratorios. ¿Cuál fue la más útil? ¿Por qué?**

Práctica más útil: Test-Driven Development (TDD) ya que fuimos declarando pruebas sobre funcionalidades que deberia de tener el codigo de esta forma adecuar el codigo para que tenga las pruebas aseguradas junto con las funcionalidades.

**8. ¿Qué referencias usaron? ¿Cuál fue la más útil? Incluyan citas con estándares adecuados.**

Referencias utilizadas:

Oracle Java Documentation (2023). Java Collections Framework. Oracle Corporation.

BlueJ Documentation (2023). BlueJ User Manual. University of Kent.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.