UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
Facultad de Ciencias

Equipo NullPointerException:
Adrián Aguilera Moreno - 421005200
Diego Angel Rosas Franco - 318165330
Marco Antonio Rivera Silva - 318183583

Modelado y programación

Práctica 4

Menciona los principios de diseño esenciales de los patrones Factory, Abstract Factory y Builder. Menciona una desventaja de cada patrón.

Factory

Principios de diseño:

- Se define una interfaz para crear objetos, pero permite decidir entre varias subclases cuál se usará para instanciar
- Se tiene un método conocido conceptualmente como "Factory Method" encargado de la creación de objetos segun las subclases.
- Encapsula las partes de código que cambian, de las que no cambian en un objeto sin exponer la lógica de creación al cliente.
- Ayuda que el código no dependa especificamente de alguna clase, lo cual ayuda a modularizar el proyecto.

Desventajas:

- Al extender el sistema, cuando se agrega una nueva clase, se deberá modificar el Factory Method considerando las nuevas clases. Esto es una desventaja pues estamos exponiendo el código a modificaciones inesperadas (por ejemplo).

Abstract Factory

Principios de diseño:

- El patrón funciona alrededor de una "fábrica" que crea "fábricas".
- Una interfaz es responsable de crear una fábrica de objetos relacionados sin especificar explícitamente las clases que involucran dichos objetos.
- Cada fábrica generada crea objetos como se establece en el patrón Factory.

Desventajas:

- Cada vez que necesitemos anexar otra entidad, necesitaremos una "Fábrica" extra, es decir, una interfaz que administre los objetos de este tipo de entidad.
- El sistema crece mucho si se abusa del patrón, y esto lo hace más complejo mediante la creacion de clases para administrar objetos.

Builder

Principios de diseño:

- Permite construír distintas representaciones de objetos complejos mediante el uso de constructores de objetos simples.
- La construcción de estos objetos complejos dependerá de un constructor o director que tratará a cada componente como una etapa de la construcción.
- Se pueden tener directores concretos que se dediquen a un objeto complejo específico o directores abstractos que permitirán la construcción cualquier combinación de componentes.

- Cada componente está modularizado, lo cual nos permite cambiar el orden de construcción con facilidad.
- Encapsula la forma en que se construye un objeto complejo.
- Permite que los objetos se construyan en un proceso de varios pasos y variable (a diferencia de las fábricas que es un solo paso).

Desventajas:

- La construcción de objetos requiere más conocimiento del dominio del cliente, ya que de no tener claro desde un inicio todas las posibilidades o la estructura final de los objetos, vamos a tener que estar modificando el código constantemente.

Instrucciones de instalación, compilación y ejecución.

Se dará por hecho que el usuario sabe moverse en terminal.

Requerimientos previos:

- Se debe contar con Java en su computadora. De preferencia la versión más reciente.

Ejecución del proyecto:

- Si está leyendo esto significa que desempaquetó con éxito el proyecto.
- Abra su terminal y diríjase a la ruta donde desempaquetó el proyecto.
- Una vez estando en la ruta `Practica04_NullPointerException`, diríjase a `Practica04_NullPointerException/sr`
- Ejecute: "`javac TallerNaves.java`", esto generará los .class del proyecto.
- Ejecute: "`java TallerNaves`", esto ejecutará el proyecto mostrándole el menú solicitado para la practica.

Justificación de uso de patrón:

Para esta práctica usamos el patrón **Builder** debido a que conciamos claramente los objetos simples (Compomentes de la nave) y el objeto complejo (Nave Espacial), además conocíamos el alcance que tendría, es decir, el número de componentes. A diferencia de Abstract Factory, donde tendríamos un sistema más abierto en ese sentido, decidimos usar Builder porque se adaptaba a la problematica que se nos pedía programar, además de que permitía escalar más facilmente los componentes que conformaban la nave. Y además Builder no nos límitaba a un número de pasos ni un orden en específico como Abstract Factory. Por otra parte, en Abstract Factory hubieramos necesitado más clases que representarían las Fábricas de cada componente, esto lo vimos innecesario ya que como mencionamos, ya conocíamos los componentes especificos que conformaban la nave y de que tipo serían. Y también builder ya nos decía como trabajar la parte de las naves ya construídas (las del catalogo). Es por esto que decidimos usar builder, pues se adaptaba muy bien a la problematica.

Otras justificaciones:

A pesar de que en el PDF no se indicaba, consideramos útil el que el usuario pudiera salir del sistema en ciertos puntos. Esto para no obligar al usuario a seguir ciertas opciones para poder salir (legalmente, sin un Ctrl+C) del sistema.

Tabla de Estadisticas de los Componentes de una Nave:

Tipo Componente	Precio	Peso	Ataque	Defensa	Velocidad
Laser Simple	79,834.99$	999.52 kg	7	2	-
Misiles de Plasma	298,777.99$	1,987.52 kg	23	2	-
Laser Destructor de Planetas	945,785.99$	10,000.52 kg	50	2	-
Blindaje Simple	50,000.50$	4,900.90 kg	-	5	-
Blindaje Reforzado	145,000.00$	9,975.87 kg	-	15	-
Blindaje Fortaleza	385.000.00$	49750.80 kg	-	50	-
Cabina Un Piloto	19,878.99$	1,986.10 kg	-	2	-
Cabina Tripulación Pequeña	69,876.00$	4,899.52 kg	-	5	-
Cabina Ejercito	199.789.99$	34,567.00 kg	-	10	-
Propulsión Viaje Intercontinental	105,000.00$	997.90 kg	-	1	100
Propulsión Viaje Interplanetario	294.000.00$	2,985.90 kg	-	3	500
Propulsión Viaje Intergalactico	875,000.00$	9,950.49 kg	-	5	1000

Diagrama UML: