# 

[**INTRODUCCIÓ**](#_jcu0i7razkol) **3**

[**OBJECTIU GENERAL**](#_dnxqfcc8373t) **3**

[**OBJECTIUS ESPECÍFICS**](#_w03shok7agbn) **3**

[**CONTINGUT TÈCNIC**](#_7f5gbrbjz6uq) **3**

[DIAGRAMA D'APLICACIONS](#_d76h4z878aw4) 4

[DESCRIPCIÓ DE LES CLASSES I INTERFACES IMPLEMENTADAS A L'APLICACIÓ](#_4qsfkzf0sshs) 8

[DIAGRAMA DE LA NAVEGACIÓ DE L'APLICACIÓ](#_v5cidn5x8mwg) 9

[**GLOSSARI**](#_hs0th3smc8sz) **10**

[**FAQ**](#_c70mmkpt1p0t) **11**

[**CONCLUSIONS**](#_c9i24dt7v18m) **11**

[**Webgrafía / Bibliografía**](#_2hlo8lhvvc05) **11**

# 

# INTRODUCCIÓ

Aquesta aplicació està realitzada sota la plataforma Java a Eclipse 4.22.0, el nostre joc està basat en un joc popular de navegador, ogame. Serà una versió força simplificada.

# OBJECTIU GENERAL

Desenvolupar una Aplicació a la plataforma de java a Eclipse utilitzant el mode gràfic i la Metodologia orientada a objectes, que contribueixi amb l'aprenentatge de l'assignatura.

# OBJECTIUS ESPECÍFICS

* Crea una aplicación en Java/Eclipse que permita al usuario jugar a un juego parecido a ogame pero de una forma más específica.

* implementar a l'aplicació Herències, Classes Abstractes, Interfícies.
* Realitzar dins de l'aplicació reportis de diferents atacs de les naus enemigues

# CONTINGUT TÈCNIC

En aquesta secció del manual es troba detallada el funcionament i el desenvolupament de l'aplicació…

## DIAGRAMA D'APLICACIONS



Podem veure que es té un arbre d'herència definit, on la classe **main**, és la super clase o classe **Pare** de **MissileLauncher**, planet, **ResourceException** aquestes classes filles o sub/classe implementen la interfícies.

Una de les regles en treballar amb classes abstractes o interfícies és que totes les classes concretes que en baixin estan obligades a implementar els seus mètodes.

A la nostra aplicació la classe veurem com la classe **main** no està obligada a fer-ho, però les seves filles sí.

## DESCRIPCIÓ DE LES CLASSES I INTERFACES IMPLEMENTADAS A L'APLICACIÓ

### **Classe Planet**

* Necesitaremos una clase **Planet**, que representará nuestro planeta.

### **Classe ResourceException**

* Para gestionar las excepciones lanzadas cuando queramos subir un nivel de alguna tecnología y no dispongamos de los recursos suficientes, vamos a crear nuestra propia excepción, la clase **ResourceException**, que extenderá de Exception, y nos mostrará un mensaje descriptivo del motivo de la excepción.

### **Classe Ship**

* Para crear nuestra flota, es decir, unidades militares no defensivas, vamos a crear la clase abstracta **Ship**. Esta clase tendrá como propiedades: int *armor*; int *initialArmor*; int *baseDamage*;

### **Classe LigthHunter**

* Para cada tipo de unidad de nuestra flota, crearemos una clase que heredará de la clase Ship. Todas ellas dispondrán de dos constructores.

### **Classe HeavyHunter**

* Para cada tipo de unidad de nuestra flota, crearemos una clase que heredará de la clase Ship. Todas ellas dispondrán de dos constructores.

### **Classe BattleShip**

* Para cada tipo de unidad de nuestra flota, crearemos una clase que heredará de la clase Ship. Todas ellas dispondrán de dos constructores.

### **Classe ArmoredShip**

* Para cada tipo de unidad de nuestra flota, crearemos una clase que heredará de la clase Ship. Todas ellas dispondrán de dos constructores.

### **Classe Defense**

* Para crear nuestra defensa, vamos a crear la clase abstracta **Defense**. Esta clase tendrá como propiedades: int armor; int initialArmor; int baseDamage;

### **Classe MissileLauncher**

* Para cada tipo de unidad de nuestra defensa, crearemos una clase que heredará de la clase **Defense**. Todas ellas dispondrán de un único constructor al que le pasaremos como argumentos int armor, int baseDamage. El funcionamiento del constructor de nuestras defensas, será igual que el de nuestra flota. Se establecerán el blindaje y la armadura dependiendo del nivel de nuestras tecnologías. Cada una de estas clases tendrá que implementar los métodos definidos en la interfaz MilitaryUnit.

### **Classe IonCannon**

* Para cada tipo de unidad de nuestra defensa, crearemos una clase que heredará de la clase **Defense**. Todas ellas dispondrán de un único constructor al que le pasaremos como argumentos int armor, int baseDamage. El funcionamiento del constructor de nuestras defensas, será igual que el de nuestra flota. Se establecerán el blindaje y la armadura dependiendo del nivel de nuestras tecnologías. Cada una de estas clases tendrá que implementar los métodos definidos en la interfaz MilitaryUnit.

### **Classe PlasmaCannon**

* Para cada tipo de unidad de nuestra defensa, crearemos una clase que heredará de la clase **Defense**. Todas ellas dispondrán de un único constructor al que le pasaremos como argumentos int armor, int baseDamage. El funcionamiento del constructor de nuestras defensas, será igual que el de nuestra flota. Se establecerán el blindaje y la armadura dependiendo del nivel de nuestras tecnologías. Cada una de estas clases tendrá que implementar los métodos definidos en la interfaz MilitaryUnit.

### **Classe Battle**

* La clase **Battle** nos servirá para gestionar el desarrollo de las batallas. Ésta será posiblemente la clase más compleja de implementar del proyecto.

### **Clase Main**

* Desde la clase **Main** podremos acceder a diferentes opciones como crear unidades en nuestro planeta, ya sean defensivas como flota, mejorar nuestras tecnologías, ver los reportes de batalla de al menos las últimas 5 batallas, o en caso de haber una flota enemiga a punto de atacarnos, ver la flota enemiga.

### **Interface MilitaryUnit**

* **MilitaryUnit** será una interfaz que implementan todas nuestras unidades (defensas y naves), y que nos servirá para dos cosas, para gestionar todas las unidades militares por igual, y para obligarnos a implementar los métodos que serán comunes a todas las unidades militares.

### **Interface Variables**

* Esta será una interfaz que nos permitirá parametrizar el juego, de forma que después de testearlo varias veces, encontremos unos valores equilibrados.

## DIAGRAMA DE LA NAVEGACIÓ DE L'APLICACIÓ

# GLOSSARI

* **Programa:** conjunt d’instruccions, escrites en un llenguatge de programació, que descriuen el tractament que cal donar a unes dades inicials (entrada) per aconseguir el resultat esperat (sortida).
* **Barra d’eines:** Conjunt de botons que representen les opcions de menú més comunes.
* **Clic:** Polsar un botó del ratolí.
* **Panel:** Un contenedor para uno o más gráficos relacionados con una tabla.
* **Base de datos:** El término colectivo para todas las tablas.
* **Dependencia:** Las dependencias se crean cuando los elementos de los componentes dependen unos de otros para funcionar.
* **Entidad:** Una entidad es la forma clásica de describir una tabla.
* **Heredado:** Esto se refiere a características que han quedado en desuso o a la forma en que se crearon, se ha trasladado a una experiencia más moderna, por ejemplo una interfaz.
* **Clase:** Una clase es una plantilla que define la forma de un objeto. Especifica los datos y el código que operará en esos datos. Java usa una especificación de clase para construir objetos. Los objetos son instancias de una clase.

# FAQ

# CONCLUSIONS

# Webgrafía / Bibliografía

* <https://www.delftstack.com/es/howto/java/java-end-program/>