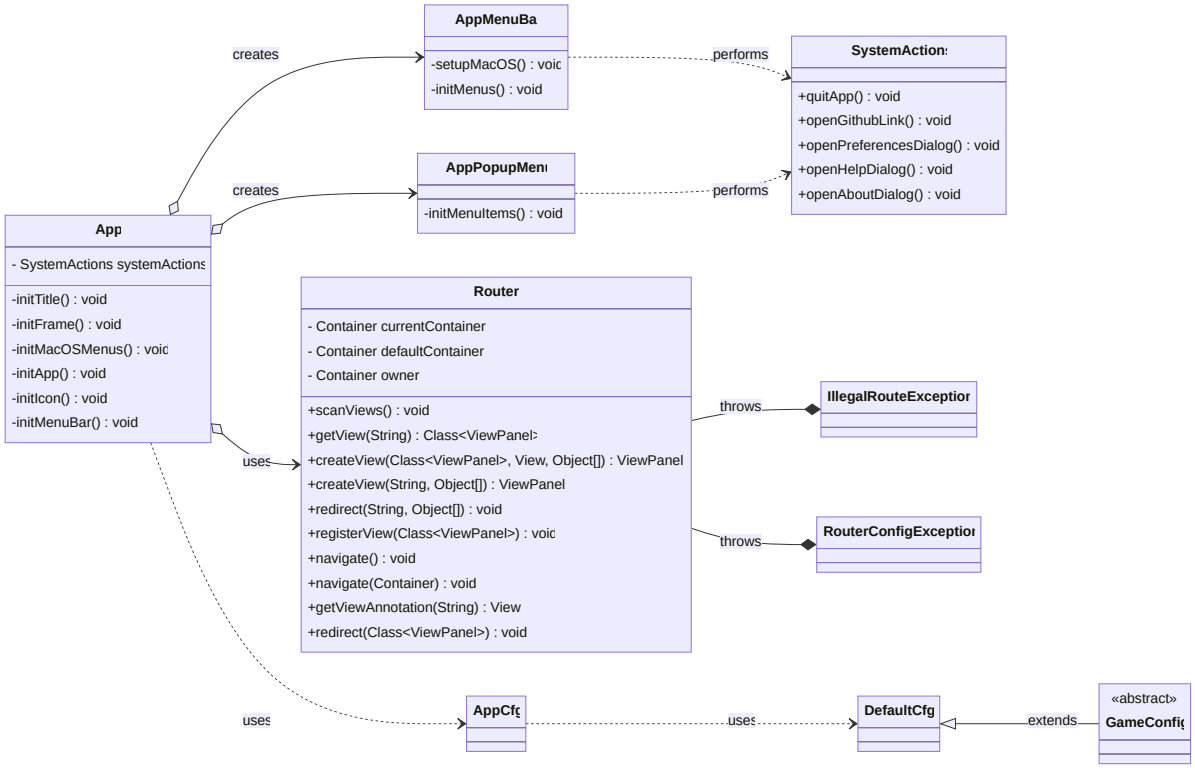
Class Diagrams

Application Classes

The app diagram is demonstrating the relationships between the main classes in the application and how they interact with each other to create the application window and handle user actions. The diagram also shows how the router handles routing for the application and how the menu bar and popup menu are created and used.

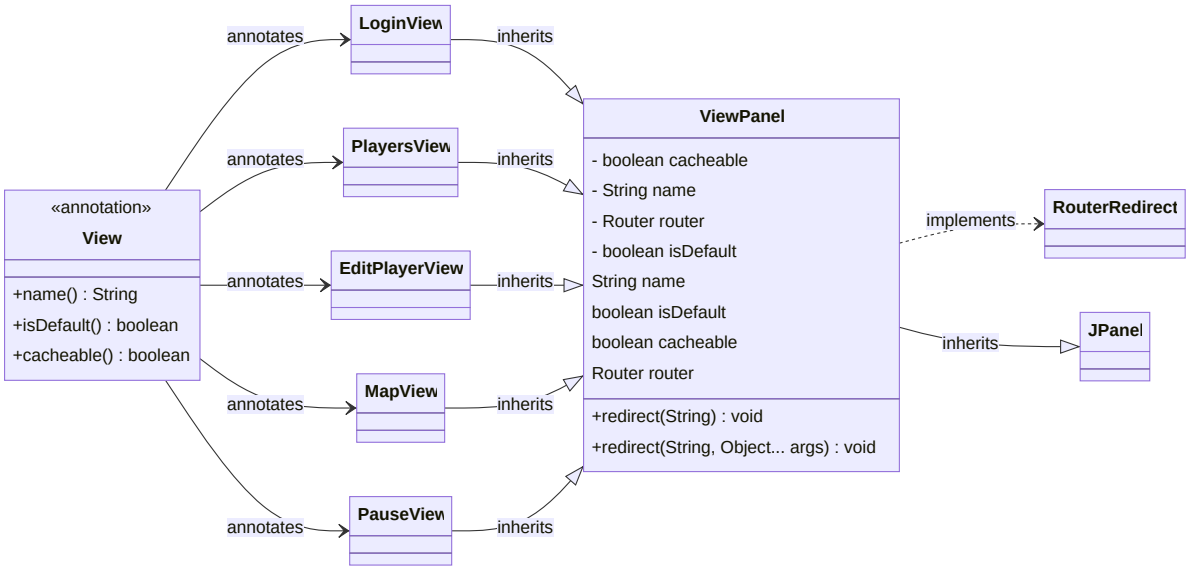
- `App`: The main class that creates the application window and contains the router, menu bar, and popup menu.
  - `AppMenuBar`: The menu bar class that creates the application's menu bar and its associated menus. Behaves differently
  - `AppPopupMenu`: The popup menu class that creates the MacOS application popup menu and its associated menu items.
- `Router`: The class that handles the routing for the application. It maps names to views and is responsible for loading and rendering the views on redirection. Router automatically registers all `ViewPanel` classes annotated with `View`.
- `SystemActions`: The class that handles system actions, such as quitting the application or opening the preferences dialog.



UI View Classes

The user interface views diagram is demonstrating the different views available in the application and their relationships. The diagram shows the names of the views and which views they inherit from.

- **From `app.router`:**
  - `View`: An annotation class that defines the basic properties of a view, including its name, whether or not it is the default view, and whether or not it is cacheable.
  - `ViewPanel`: An abstract class that defines the behavior of all application views.
- **From `ui.views`** — all the defined views for the application:
  - `LoginView`: A concrete class that inherits from `ViewPanel` and is responsible for rendering the login view.
  - `PlayersView`: A concrete class that inherits from `ViewPanel` and is responsible for rendering the player selection view.
  - `EditPlayerView`: A concrete class that inherits from `ViewPanel` and is responsible for rendering the player editing view.
  - `MapView`: A concrete class that inherits from `ViewPanel` and is responsible for rendering the game map view.
  - `PauseView`: A concrete class that inherits from `ViewPanel` and is responsible for rendering the pause menu view.

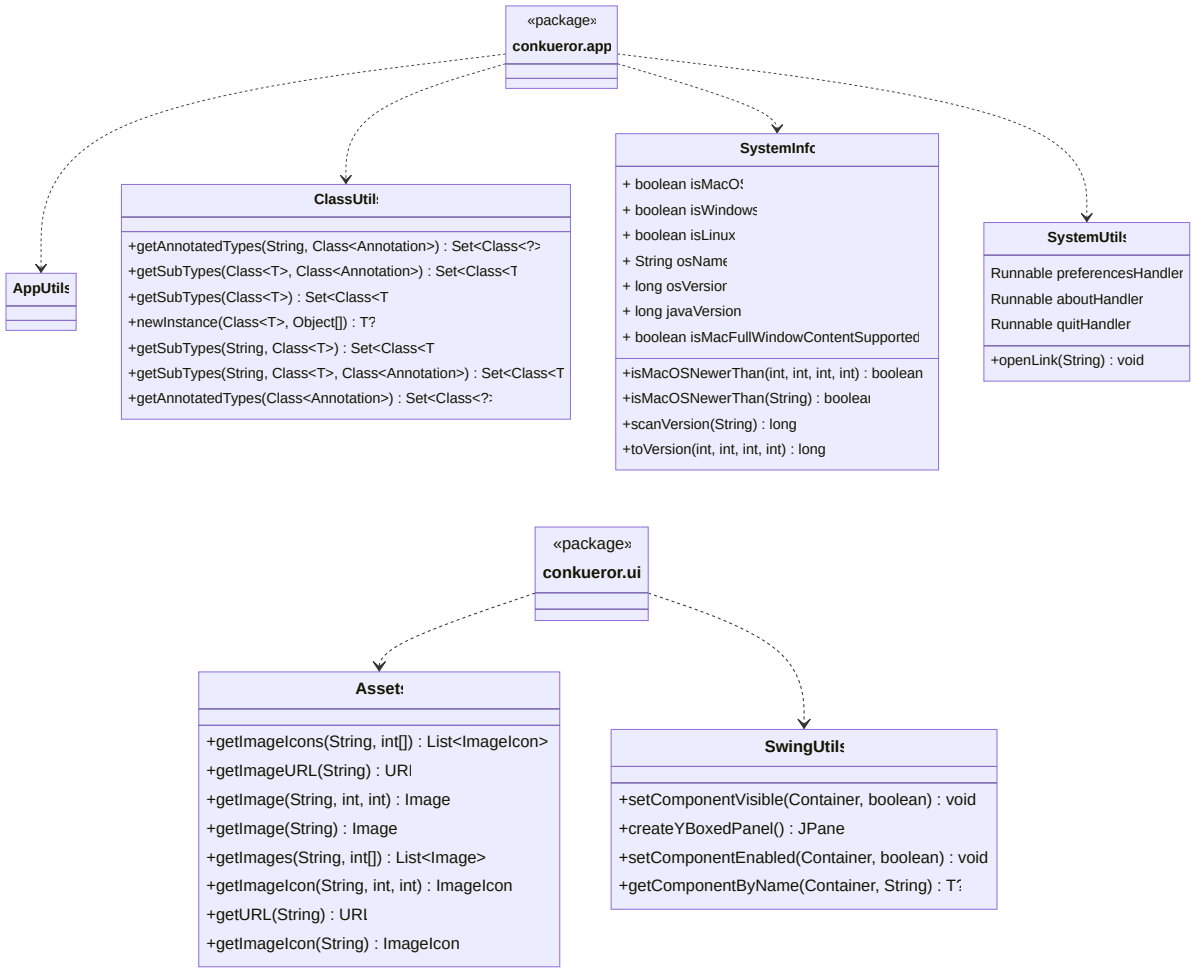


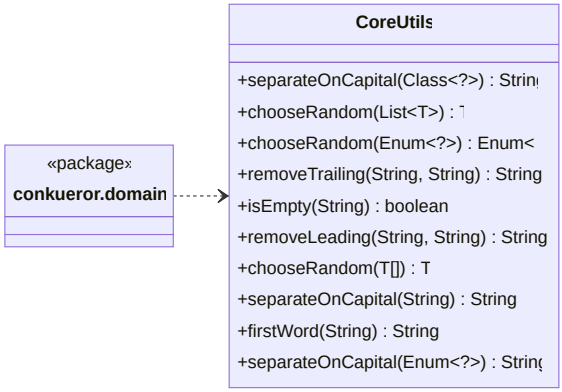
Utility Classes

The utility class diagram represents a set of utility classes that provide different functions to the application. For completeness, the diagram also shows packages that use the utility classes.

- `assets.Assets`: Contains methods for loading and retrieving assets, such as icons and images.

- `util.AppUtils`: Contains methods for performing various utility tasks in the application.
- `util.ClassUtils`: Contains methods for working with Java classes, such as getting annotated types and creating instances.
- `util.CoreUtils`: Contains various utility methods, such as separating strings on capital letters and choosing a random item from a list.
- `util.SwingUtils`: Contains utility methods for working with Swing components, such as setting component visibility and creating a boxed panel.
- `util.SystemInfo`: Contains information about the user's system, such as the operating system and Java version.
- `util.SystemUtils`: Contains methods for performing system-level tasks, such as opening a link in the default browser.



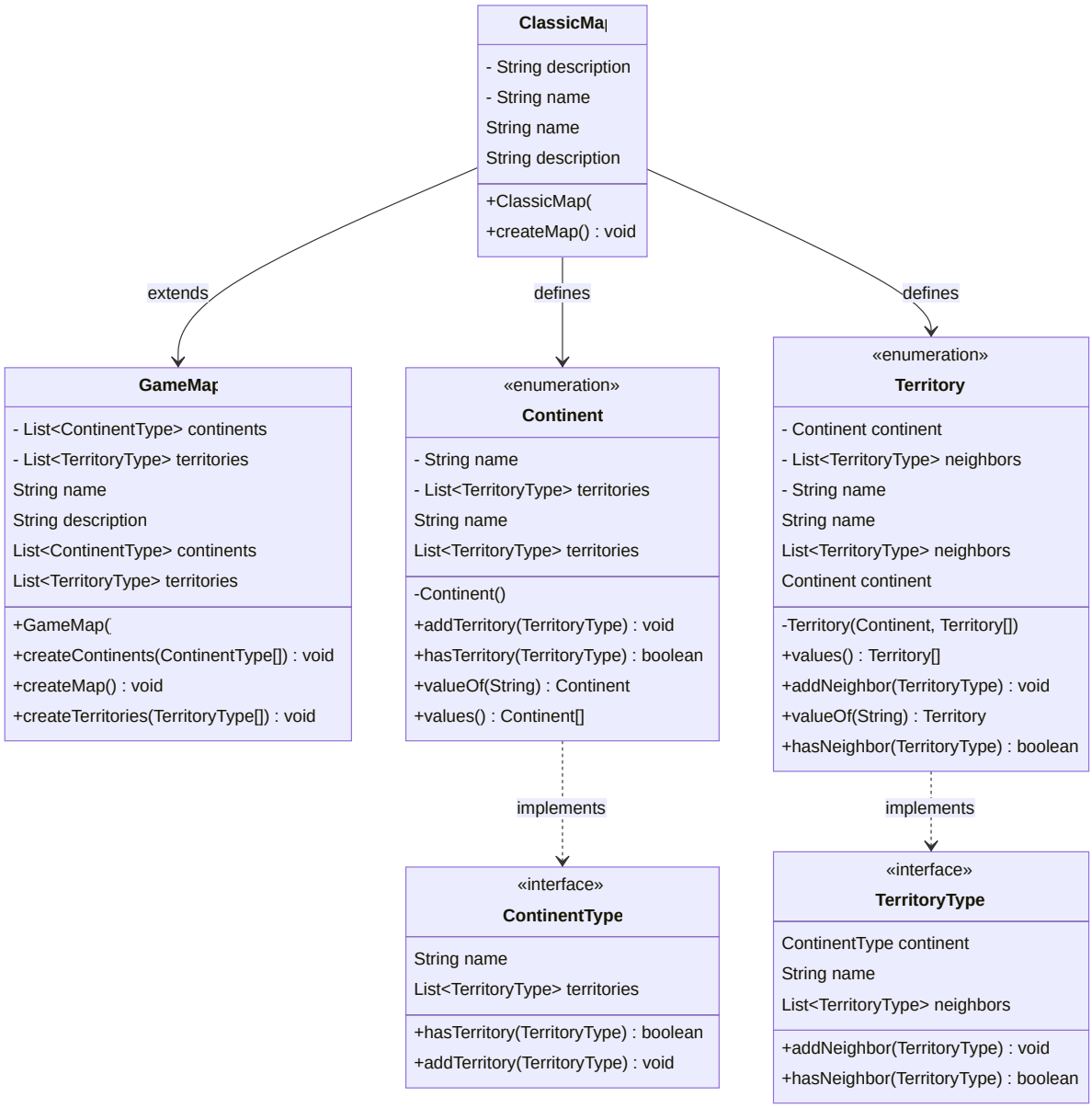


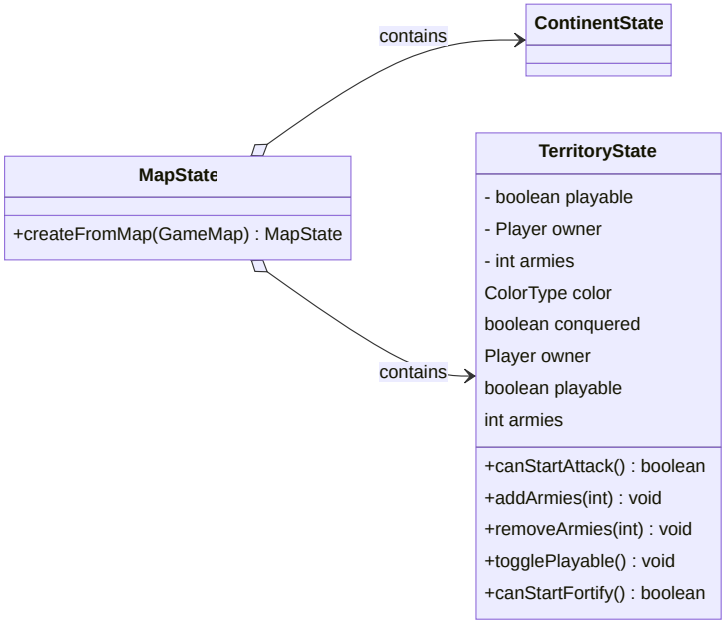
Map Classes

The map classes diagram shows the class hierarchy for the game's map system. The `ClassicMap` class extends the `GameMap` interface, which defines methods for creating a map and getting territories.

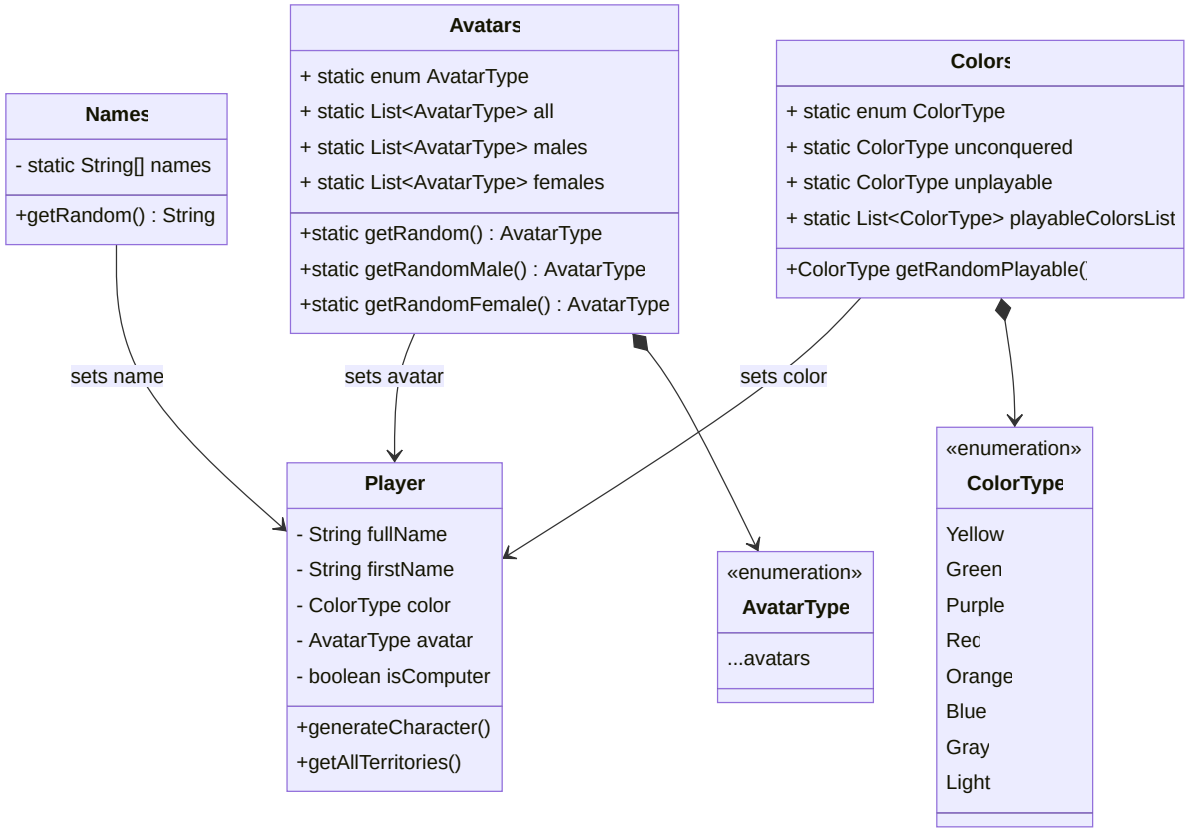
The `Continent` and `Territory` interfaces both contain methods for defining and manipulating territories and continents. The `Continent` interface implements the `ContinentType` interface and the `Territory` interface implements the `TerritoryType` interface. These interfaces define methods for adding and checking territories and neighbors.

In addition to these interfaces, the diagram also shows several classes that implement them. For example, the `DefaultWorldMap` class implements the `GameMap` interface and defines the game's default map. The `Continent` and `Territory` classes implement the `ContinentType` and `TerritoryType` interfaces, respectively, and contain additional methods for managing territories and their neighbors.

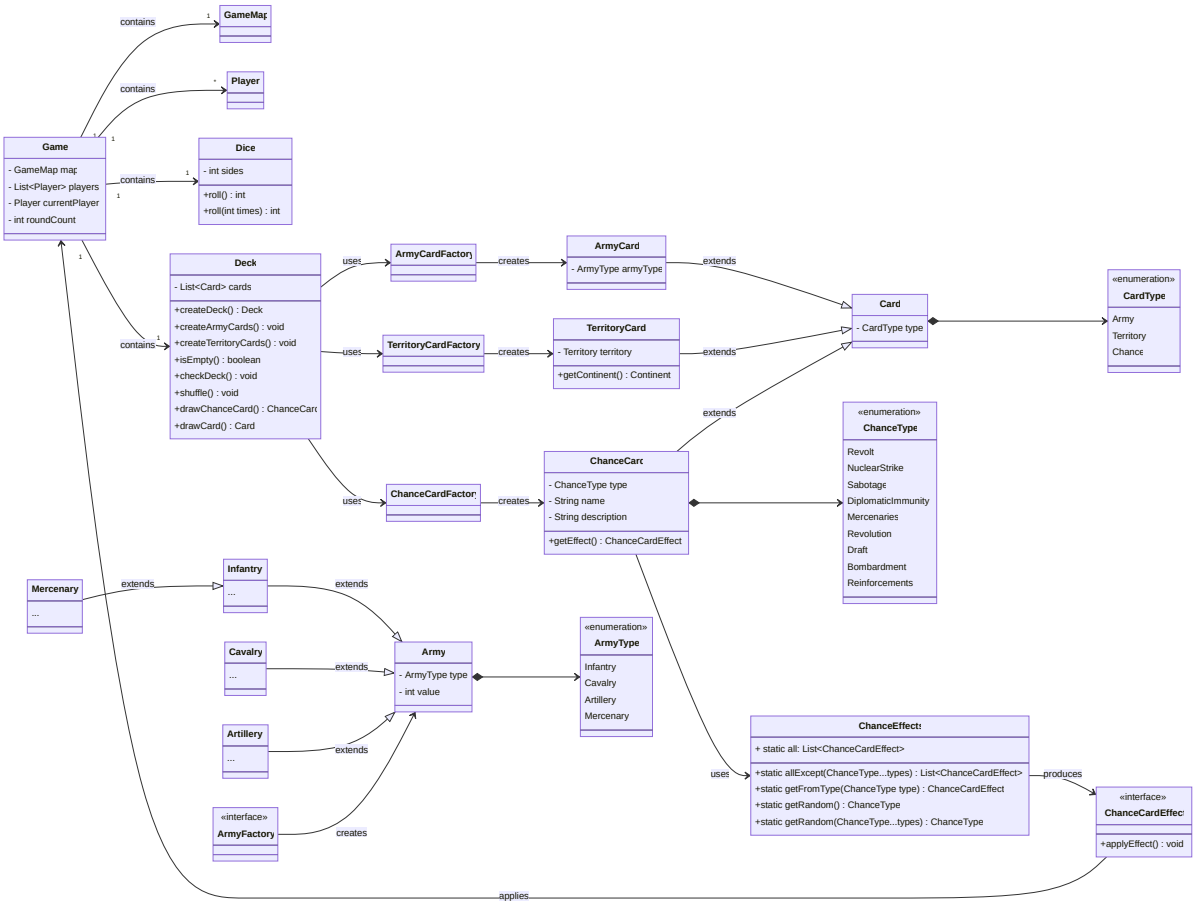

ClassicMa
- String description
- String name
String name
String description
+ClassicMap(
+createMap() : void
extends
defines
defines
GameMap
- List<ContinentType> continents
- List<TerritoryType> territories
String name
String description
List<ContinentType> continents
List<TerritoryType> territories
+GameMap(
+createContinents(ContinentType[]) : void
+createMap() : void
+createTerritories(TerritoryType[]) : void
«enumeration»
Continent
- String name
- List<TerritoryType> territories
String name
List<TerritoryType> territories
-Continent()
+addTerritory(TerritoryType) : void
+hasTerritory(TerritoryType) : boolean
+valueOf(String) : Continent
+values() : Continent[]
«enumeration»
Territory
- Continent continent
- List<TerritoryType> neighbors
- String name
String name
List<TerritoryType> neighbors
Continent continent
-Territory(Continent, Territory[])
+values() : Territory[]
+addNeighbor(TerritoryType) : void
+valueOf(String) : Territory
+hasNeighbor(TerritoryType) : boolean
implements
implements
«interface»
ContinentType
String name
List<TerritoryType> territories
+hasTerritory(TerritoryType) : boolean
+addTerritory(TerritoryType) : void
«interface»
TerritoryType
ContinentType continent
String name
List<TerritoryType> neighbors
+addNeighbor(TerritoryType) : void
+hasNeighbor(TerritoryType) : boolean

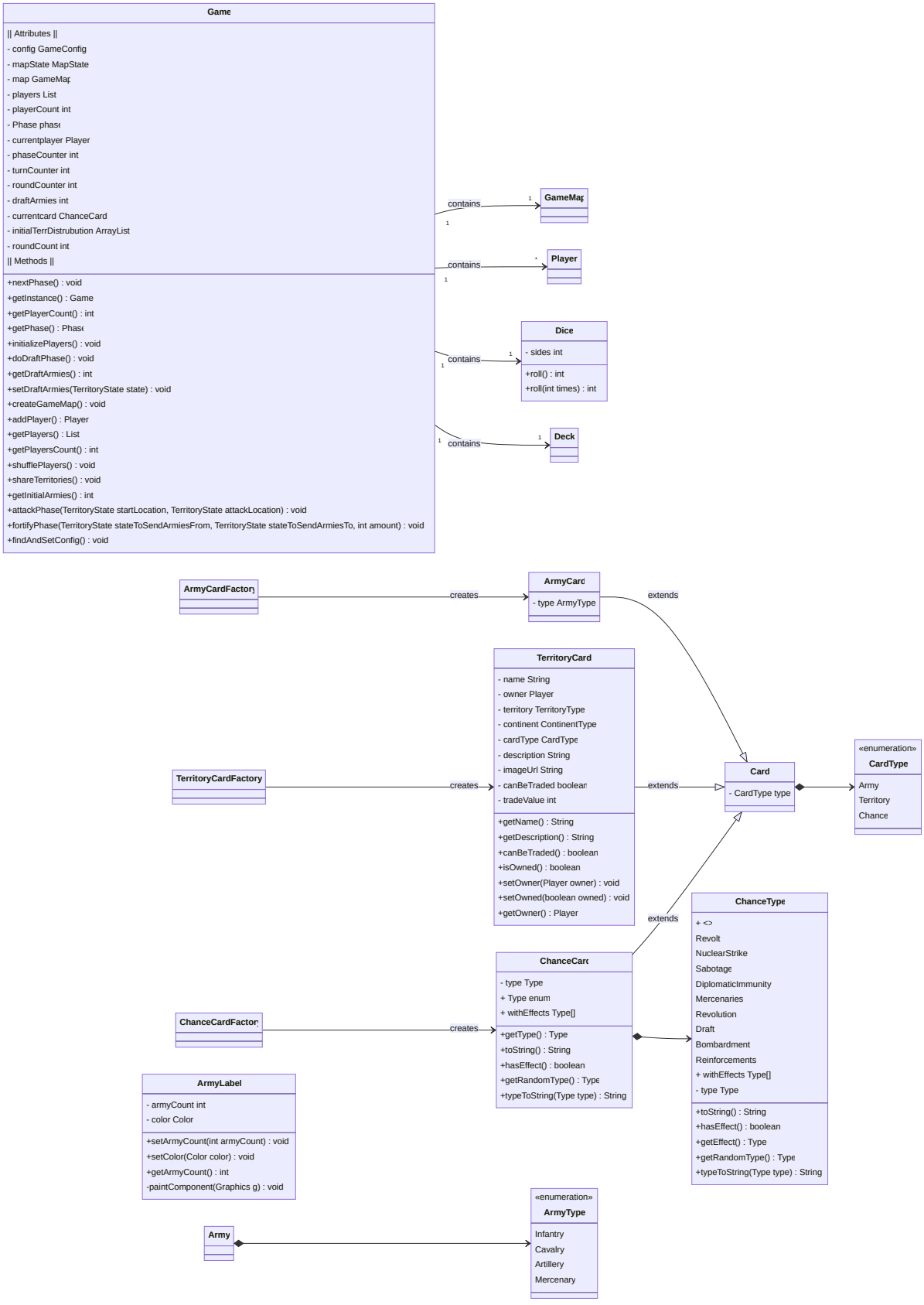

contains
ContinentState
MapState
+createFromMap(GameMap) : MapState
contains
TerritoryState
- boolean playable
- Player owner
- int armies
ColorType color
boolean conquered
Player owner
boolean playable
int armies
+canStartAttack() : boolean
+addArmies(int) : void
+removeArmies(int) : void
+togglePlayable() : void
+canStartFortify() : boolean

Player

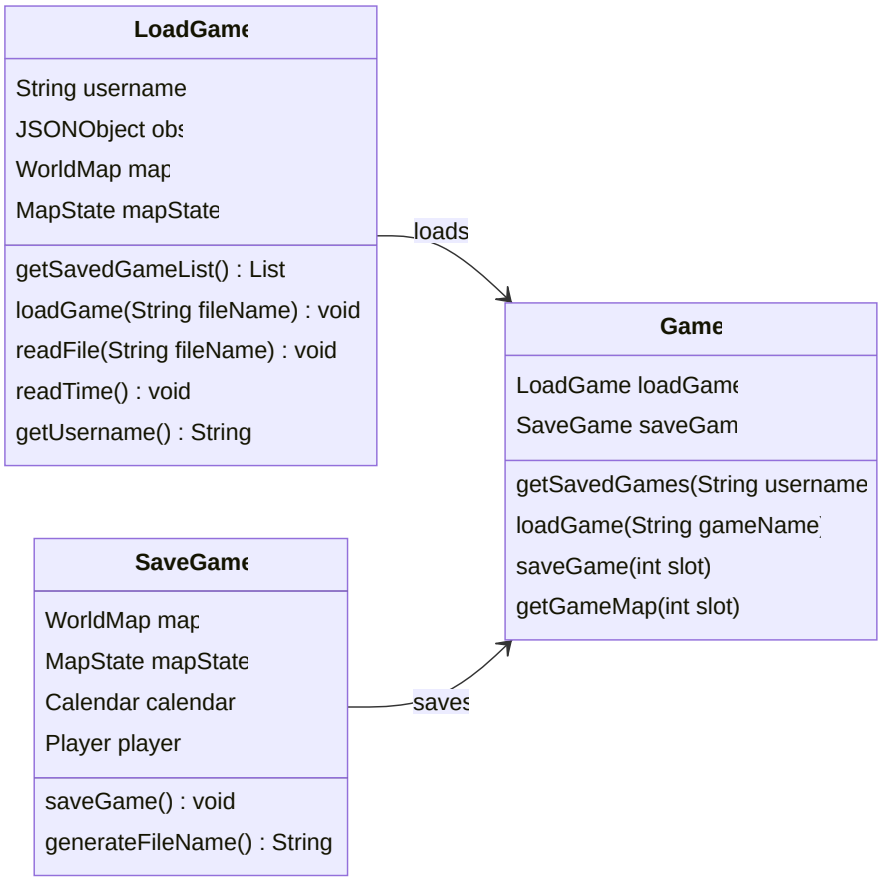

Avatars
+ static enum AvatarType
+ static List<AvatarType> all
+ static List<AvatarType> males
+ static List<AvatarType> females
+static getRandom() : AvatarType
+static getRandomMale() : AvatarType
+static getRandomFemale() : AvatarType
Colors
+ static enum ColorType
+ static ColorType unconquered
+ static ColorType unplayable
+ static List<ColorType> playableColorsList
+ColorType getRandomPlayable(
Names
- static String[] names
+getRandom() : String
sets name
sets avatar
sets color
Player
- String fullName
- String firstName
- ColorType color
- AvatarType avatar
- boolean isComputer
+generateCharacter()
+getAllTerritories()
«enumeration»
AvatarType
...avatars
«enumeration»
ColorType
Yellow
Green
Purple
Rec
Orange
Blue
Gray
Light

Game


contains
GameMap
contains
Player
Game
- GameMap map
- List<Player> players
- Player currentPlayer
- int roundCount
contains
Dice
- int sides
+roll() : int
+roll(int times) : int
contains
Deck
- List<Card> cards
+createDeck() : Deck
+createArmyCards() : void
+createTerritoryCards() : void
+isEmpty() : boolean
+checkDeck() : void
+shuffle() : void
+drawChanceCard() : ChanceCard
+drawCard() : Card
uses
ArmyCardFactory
creates
ArmyCard
- ArmyType armyType
extends
Card
- CardType type
«enumeration»
CardType
Army
Territory
Chance
uses
TerritoryCardFactory
creates
TerritoryCard
- Territory territory
+getContinent() : Continent
extends
extends
«enumeration»
ChanceType
Revolt
NuclearStrike
Sabotage
DiplomaticImmunity
Mercenaries
Revolution
Draft
Bombardment
Reinforcements
uses
ChanceCardFactory
creates
ChanceCard
- ChanceType type
- String name
- String description
+getEffect() : ChanceCardEffect
Mercenary
...
extends
Infantry
...
extends
Cavalry
...
extends
Artillery
...
extends
Army
- ArmyType type
- int value
«enumeration»
ArmyType
Infantry
Cavalry
Artillery
Mercenary
«interface»
ArmyFactory
creates
uses
ChanceEffects
+ static all: List<ChanceCardEffect>
+static allExcept(ChanceType...types) : List<ChanceCardEffect>
+static getFromType(ChanceType type) : ChanceCardEffect
+static getRandom() : ChanceType
+static getRandom(ChanceType...types) : ChanceType
produces
«interface»
ChanceCardEffect
+applyEffect() : void
applies



Database Load/Save


LoadGame
String username
JSONObject obs
WorldMap map
MapState mapState
getSavedGameList() : List
loadGame(String fileName) : void
readFile(String fileName) : void
readTime() : void
getUsername() : String
loads
Game
LoadGame loadGame
SaveGame saveGam
getSavedGames(String username
loadGame(String gameName
saveGame(int slot)
getGameMap(int slot)
SaveGame
WorldMap map
MapState mapState
Calendar calendar
Player player
saveGame() : void
generateFileName() : String
saves