Assignment Omega's submission

Author: **Faycal Kilali**

During Assignment Alpha, I did not include tests nor did I implement the MVC Architecture correctly. This time, I've implemented it much more robustly in comparison to assignment alpha.

The code also had various issues where sometimes Barbie and other important entities would fail to be placed. I've ensured that is resolved, which should fix the null pointer exceptions.

I also added test classes.

Refactorization

MVC

I've refactored the program entirely in order to adhere to the MVC pattern more closely and in order to reduce the responsibilities of the previous game logic class (which was the Model class, as I did not understand MVC) which had far too much responsibilities.

Design Patterns

Now we have a `ProgramController` exception class and a `ProgramFlow` model class. The `ProgramFlow` class essentially allows the entire game to be playable entirely by the Model and also serves as a **facade** for the rest of the model. the `ProgramController` co-serves as both a coordinator between controllers, the corresponding `ProgramView` utilizes the **composite** design pattern to *compose* the views in a ordered manner to effectively provide a fixed **update-able** UI. Where the updates occur through the `ProgramController` requesting updates from each of the other controllers.

SOLID

Dependency Inversion Principle

I've included various interfaces in order to align better with the dependency inversion principle. By definition, any class that relies on or uses another class should rely on its abstraction rather than concretion, hence every class that satisfies that requirement now has an interface and is being substituted for its interface through polymorphism. Some of the interfaces are `IGrid, IProgramFlow`, and so on...

Single-Responsibility Principle

I've also added a few other classes. The significant one being the **grid** class. This class has the responsibility of keeping track of the instances in the grid and maintaining the grid. This helps more closely satisfy SRP among

the ProgramFlow class which now serves as the class for the game logic. In the past, the grid and its entities were all stored in the `Model` class which was SRP-Violating.

Interface Segregation Principle

I've added several interfaces and ensured that the interfaces are as segregated as possible. No classes implementing interfaces are obligated to implement methods that they do not need or have no implementation of.

Open-Closed Principle

I've ensured the Grid class satisfies OCP. It requires no modification in order to store or maintain its entities and grid. The ProgramFlow class requires further work in order to ensure that OCP is also satisfied there.

The Views and Controllers all satisfy OCP currently. The OCP violation has been moved to the entity classes (Wall, Barbie, Zombie, Fruit) for now.

Code Smells

Removed **dead code** smell, reduced **comments** smell. Ensured that the message-chains code smell also no longer exists (limited method calls to 2 deep (2 method calls deep, 3 classes deep if we include caller), where `A → B → C`, where A is the initial caller object, B is the first method in class B, and C is the second method in the third class). This applied over to the majority of the `ProgramFlow` and `Grid` classes.

Other refactorization of note

I've reduced several instaneOf and casting issues. In the isValidZombieMove I've removed the instanceOf checks of Zombie and Barbie as those should already exist in isValidCell. In the future, I may use the visitor pattern to avoid having to use instanceOf in that context or attempt to re-structure the program further.

Furthermore: adjusted indentation, wrote much more appropriate javadoc, and commented out code.

I've also generated a UML Class Diagram as there was no explicit statement stating that we need to manually create a UML Class Diagram.