Michael:

The Dinosaur parent class
Having Dinosaur as the parent abstract class and then having Stegosaur and Allosaur inherit it reduces the need for repeated code (Don't Repeat Yourself) and data members for all dinosaurs. Doing so will also future proof the design as adding news types of dinosaurs can be done by simply inheriting the Dinosaur class.

Dinosaurs growing
For dinosaurs to grow, a behaviour or action was not introduced. Instead there will be a boolean indicating if the dinosaur is an adult or not and after 30 turns the boolean will turn from false (for baby) to true (for adult). This reduces the need for extra classes, thus reducing dependencies.

Eating
SeekFoodBehaviour can be used by both carnivores and herbivores with the help of Capabilities. This reduces the need for separate classes and repeated code for the specific feeding type and even allows for omnivores where omnivore dinosaurs can take in both carnivore and herbivore capabilities (discussed in next section). This also reduces the need for Herbivore and Carnivore interfaces further reducing classes.
The use of Capabilities also helps us easily tag which food can be eaten by which dinosaur type and which dinosaurs can be or can’t be attacked.
Eggs also can’t be eaten by dinosaurs of the same type (Allosaurs can’t eat Allosaur Eggs). This is implemented by passing the dinosaur that laid the egg and storing it’s class as an instance variable. Classes of the dinosaur trying to eat the egg are then compared with the class stored inside the egg and if they are the same, then the egg cannot be eaten. The Class instance variable is also used to get the constructor with the getConstructors method to instantiate a new dinosaur once it is time for hatching. Doing this supports modularity.

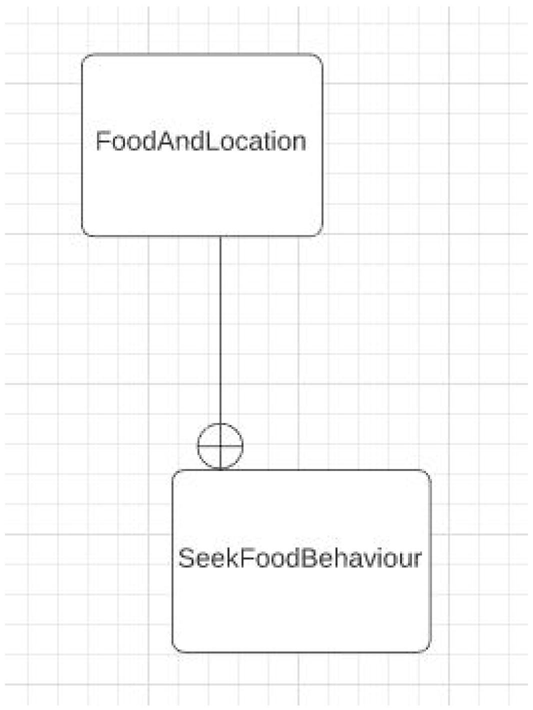
Using ArrayList to store carnivore, herbivore capability.
Instead of storing a single capability, an ArrayList was used to store one or two capabilities. Doing so allows for omnivores. Checking if the dinosaur could eat a certain food is done by checking if the ArrayList contains the capability. If the dinosaur was herbivorous, the ArrayList would contain just the capability for herbivores. Likewise for carnivores. If a dinosaur was omnivorous, the ArrayList would contain both the herbivore and carnivore capabilities. There was no need for other data sets like hash maps as the ArrayList was only storing two capabilities at most.

Storing and calling behaviours
Behaviours are kept in an array list and to be iterated over through, reducing the lines of code needed to call all the behaviours in the dinosaur class. Behaviours are customised within the subclasses of Dinosaur using static methods. The methods return an array list of behaviours customised to the specific dinosaur which is passed into the Dinosaur constructor, reducing the need to store behaviours and repeat the code for calling behaviours in the subclasses, instead calling on them in the Dinosaur parent class. The methods that are used to create and return the array list of behaviours are static (and

private) because it is the only way to call a method before an instance of a dinosaur is made (at least to my knowledge).

The use of private nested class in SeekFoodBehaviour
The nested private class FoodAndLocation is used as a return value for the findFood method within SeekFoodBehaviour. The reason for this is because I needed to return the food found and its location at the same time. Instantiating and returning an object which stores both of these was the best solution I could find. Since it is private and nested within SeekFoodBehaviour and to only be used in SeekFoodBehaviour, not making the instance variables private was not an issue and allowed for direct access to the variables rather than using accessors.

FoodAndLocation is shown in UML class diagram using the circle and cross notation like so:



Varying dinosaur corpse food level
To have varying food levels for dinosaur corpse depending on the type of dinosaur that died, the food value/level of the dinosaur if it was corpse is stored within the dinosaur itself and when it dies, DieAction passes that value into the DinosaurCorpse constructor
Removing grass from ground using GrassAsFood
When a dinosaur wants to graze on the grass, a GrassAsFood object is instantiated and passed into the EatAction constructor. GrassAsFood takes in the location of the grass and within its constructor, sets that location to dirt. This is done so that EatAction and SeekFoodBehaviouor would not be responsible for turning the ground from grass to dirt. GrassAsFood is different from Hay as they have different food level points and Hay is for when the player harvests.

Other random things

Classes are responsible for their own properties with most Behaviour and Action classes explicitly being responsible for certain behaviours and actions that dinosaurs can take.

Attributes only used in that class will only be declared in that class and variables used within methods only will only be declared within that method, declaring things within the tightest possible scope. This keeps classes very simple.

Instance variables are also declared private (except for the nested private class) or protected exposing as little data as possible to outside classes that do not need to know about it.

All behaviours also inherit from the CommonStuffBehaviour abstract class which implements the Behaviour interface, once again removing the need to repeat code in each behaviour class. CommonStuffBehaviour stores methods that are all used by the Behaviour classes that inherit it, reducing repeated code.

Dinosaur also inherits Actor can therefore have actions and behaviours called upon it with the use of polymorphism and casting. This follows the Liskov Substitution Principle and reduces the need to repeat code.

Excessive use of literals are also avoided with the help of constants.
Exceptions are also used in Dinosaur constructor to help with failing fast when there is an invalid input.

When increasing food or thirst level, if it exceeds 100, then it will be set to 100 rather than rejecting the food, following the specification of max food level being 100.

Archaeopteryx are unattackable in this game as they are flying units/dinosaurs with the only dinosaur they can attack currently being the Agilisaurus. However they can still eat dinosaur corpses of dinosaurs they can’t attack, like scavengers.