# Package: actors

## Player (Modified)

### Attributes

- No new attributes

### Methods

- public purchase(VendingMachine vending machine, Item item): void
    - Calls purchaseItem() method from VendingMachine
    - Adds item in player's inventory
- public harvest(): void
    - Creates an instance of HarvestAction, and calls its execute() method
- public feed(Dinosaur dinosaur): void
    - Player searches for food item in inventory
    - Feeds the given stegosaurus using its increaseHunger() method
    - Food item is discarded from inventory
- public attack(Dinosaur dinosaur): void
    - Creates an instance of AttackAction
- public dropItem(): void
    - Iterate over all items in inventory (to get the current items)
    - Creates a menu interface to select which available item to drop
    - Creates an instance of DropItemAction, and calls its execute() method
- public pickUpItem(Item item): void
    - Creates an instance of PickUpItemAction, and calls its execute() method

### How does this implementation follow good design principles?

Whilst methods for each relevant action are present in this class, the implementation of these actions are delegated to separate classes (e.g. AttackAction, HarvestAction, etc.). As mentioned before, this prevents the Player class from having too many responsibilities.

### What does this class interact with in the system?

- Player purchases Item from VendingMachine
- Player harvests Hay from Dirt, and harvests Fruit from Tree
- Player feeds Dinosaur with Food

- Player attacks Dinosaur with WeaponItem or IntrinsicWeapon
- Player adds and removes Item from its own inventory

### Alternative implementations considered

None.

## Dinosaur (New)

- Abstract class
- Extends Actor class

### Attributes

- private int: age
- private int: hungerLevel
- private int: daysUnconscious
- private String: sex
- private int: MINIMUM_HUNGER_LEVEL = 0
- private int: MAXIMUM_HUNGER_LEVEL = 100

### Methods

- public Dinosaur(String sex)
    - Default constructor for when game starts
        - Required so that there are two adult (age 30) Stegosaurus' of opposite sex at the start of the game
    - this.age set to 30 (by default)
    - hungerLevel set to 50 (by default)
    - daysUnconscious set to zero
    - sex set by the argument passed by constructor
- public Dinosaur()
    - Constructor for when Egg hatches
    - this.age set to 0 (by default)
    - hungerLevel set to 10 (by default)
    - daysUnconscious set to zero
    - sex is set randomly between "Male" or "Female"
- public playTurn(Actions actions, Action lastAction, GameMap map, Display display): Action

    - FollowBehaviour class implements dinosaurs following a food source and also a potential mate (when food level is greater than 50)
    - Decreases hunger by 1 on every turn
    - Uses isHungry() when hungerLevel is zero
        - If true, should stop wandering (no longer moves). At this point, they would be more vulnerable to Allosaur attacks.
        - When the dinosaur is hungry, it displays a suitable message
- private isHungry(): boolean
    - Helper method to increase daysUnconscious
        - If daysUnconscious == 20, then dies
    - Returns true if hungerLevel == 0
- public increaseHunger(int hunger): void
    - Increases the hunger level based on given integer input.
    - Two precondition checks, one that sends a message if hunger is already 100, and another that increases up to 100 and excess hunger is not counted
- private resetDeath(): void
    - Used to reset daysUntilDeath when hungerLevel is positive again
- public mate(): Egg
    - Abstract method. Return value differs between subclasses.
- private isOppositeSex(Dinosaur dinosaur): void
    - Checks whether the given dinosaur is the opposite of the current dinosaur
    - Helper method for mate()
- public die(): void
    - Kills the dinosaur. Stops wandering.
    - Removes dinosaur from the system

## How does this implementation follow good design principles?

Creating this as an abstract class follows the DRY principle, because this would allow creating subclasses that share attributes and methods between dinosaurs. Thus, we would not need to recreate the same methods for Stegosaurus and Allosaurus.

Furthermore, this reduces dependencies by using constant static values for the hunger levels. This abstract class also follows the Fail Fast principle, since we will use the MINIMUM_HUNGER_LEVEL and  MAXIMUM_HUNGER_LEVEL to ensure the dinosaur's hunger level does not go out of bounds.

### What does this class interact with in the system?

This class doesn't necessarily interact with the system because it's an abstract class, but Stegosaurus and Allosaurus classes should be able to extend this class.

### Alternative implementations considered

Initially implemented dinosaurs that extend Actor without this abstract class, but this would violate the DRY principle (as all dinosaurs have common behaviour that can be grouped).

## Stegosaurus (Modified)

- Extends Dinosaur class

### Attributes

- No new attributes

### Methods

- public playTurn(Actions actions, Action lastAction, GameMap map, Display display): Action
    - Uses super() to copy Dinosaur abstract class' method
    - Calls graze() method
- public graze(Ground ground): void
    - Checks if on grass
    - If yes, then eats grass (calls removeGrass() method) and hunger level increases by 5
- public mate(Stegosaurus stegosaurus): StegosaurusEgg
    - Checks if stegosaurus is of the opposite sex using isOppositeSex()
    - Checks if age of both stegosaurus' is >= 30
    - Both stegosaurus' stop wandering
    - Then ten turns later, the female of the pair lays an egg by calling layEgg()
- private layEgg(): StegosaurusEgg
    - Helper function to lay egg when ten turns of mating has passed

### How does this implementation follow good design principles?

By extending the abstract class Dinosaur, this follows the Do not Repeat Yourself principle, because this Stegosaurus class inherits the methods related to Dinosaur.

### What does this class interact with in the system?

- Stegosaurus eats grass from Dirt
- Player feeds Stegosaurus with Hay, Vegetarian Food Kits
- Stegosaurus breeds with other Stegosaurus of the opposite sex.
- Allosaurus attacks and eats Stegosaurus

### Alternative implementations considered

We were planning to implement egg-laying within the mate() method, but decided to create a helper function instead for clarity.

## Allosaurus (New)

- Extends Dinosaur class

### Attributes

- No new attributes

### Methods

- public attack(Stegosaurus stegosaurus): void
    - Attacks the stegosaurus using AttackAction
- public eat(Stegosaurus stegosaurus): void
    - Eats the stegosaurus
- public mate(Allosaurus allosaurus): AllosaurusEgg
    - Checks if allosaurus is of the opposite sex using isOppositeSex()
    - Checks if age of both allosaurus' is >= 30
    - Both allosaurus' stop wandering
    - Then ten turns later, the female of the pair lays an egg by calling layEgg()
- private layEgg(): AllosaurusEgg
    - Helper function to lay egg when ten turns of mating has passed

### How does this implementation follow good design principles?

By extending the abstract class Dinosaur, this follows the Do not Repeat Yourself principle, because this Allosaurus class inherits the methods related to Dinosaur.

### What does this class interact with in the system?

- Player feeds Stegosaurus with carnivore meal kits

- Allosaurus breeds with other Allosaurus. But they need to be opposite sex.
- Allosaurus attacks and eats Stegosaurus

## Alternative implementations considered

Details for alternative implementations for Allosaurus are the same as for the Stegosaurus class.

# Package: ground

## Dirt (Modified)

### Attributes

- private Boolean: grass = false
    - Different from hay. Hay is a PortableItem object that is instantiated when a player harvests grass.
- static final int: POINTS_WHEN_HARVEST_OR_GROW_GRASS = 1

### Methods

- public tick(Location location): void
    - Executes only if grass is set to false
        - Calls chanceOfGrass
        - Value returned from chanceOfGrass is passed into growGrass
        - If the return value of growGrass is true, change the displayChar attribute (to “^”) and the grass attribute to true
- public hasGrass(): Boolean
    - Queries grass attribute of a Dirt object
- private chanceOfGrass(Location location): int
    - Returns probability of grass growing for a Dirt object
    - Instantiates ScanSurrounds and calls the numberOfGrass() and numberOfTree() methods to determine the chance of grass growing (scenarios with higher percentage considered first; if percentage is equal, prioritise the rule involving other terrain). Evaluates return value accordingly.
- private growGrass(int probability): Boolean
    - Based on the given probability, return true (indicating that grass should grow) or false (indicating no change)
    - Instantiates EcoPointsSystem and calls earn() method if return value is true
- public harvestGrass(): Hay
    - Change the displayChar attribute (to “.”) and the grass attribute to false
    - Instantiates EcoPointsSystem and calls earn() method
    - Creates an instance of Hay and returns this (to the player)

- public removeGrass(): void

    - Change the displayChar attribute (to “.”) and the grass attribute to false

### How does this implementation follow good design principles?

The code follows good design principles of abstraction; the only thing that needs to be called is the tick() method to implement the functionality of growing grass. Again, the responsibility of checking the surroundings for a given Dirt object is delegated to the ScanSurrounds class.

### How does this class interact with the rest of the system?

- After each turn, the game engine calls tick() on each Dirt object. The grass attribute may be set to true if it is currently false.

### Alternative implementations considered

We considered creating a Grass class and instantiating Grass objects to exist in the same location as Dirt objects, but decided to implement a Dirt attribute instead for simplicity. With the existing game engine, you also can’t store a Dirt and Grass object in the same location.

## Tree (Modified)

### Attributes

- private int: DROP_CHANCE = 5

### Methods

- public tick(Location location): void
    - Calls dropFruit()
- private dropFruit(Location location): void
    - Using DROP_CHANCE and the Random class, calculates whether or not a Fruit object is initialised
    - Call addItem() method in Location
- public harvestFruit(): Fruit
    - Uses Random class and calculates whether a Fruit can be found in a tree (40% success rate)
    - If successful, returns Fruit
    - If unsuccessful, print a suitable message

### How does this implementation follow good design principles?

The code follows good design principles of abstraction; the only thing that needs to be called is the tick() method to implement the functionality of dropping fruit and removing dropped fruit that rots with the passing of time.

### How does this class interact with the rest of the system?

- Fruits can be dropped onto the Location of the Tree
- After each turn, the game engine calls tick() on each Tree object. This can change the fruits attribute of Tree (in that a Fruit object may be added or removed if rotten).

### Alternative implementations considered

Previously had an ArrayList<Fruit> attribute to keep track of the fruit that drops to the base of a tree (thus creating an association between the Tree and Fruit class). This was not necessary, because a dropped item can be stored in the Location class.

## ScanSurrounds (New)

This is not a subclass of Ground, but it has been included in this package because of its close relation to the Ground object, in that it helps to determine the likelihood of grass growing on a square of dirt.

### Attributes

- No attributes

### Methods

- public static numberOfTrees(Location location): int
  - Returns the number of trees around the location (using an accumulator)
- public static numberOfGrass(Location location): int
  - Returns the number of Dirt objects with hasGrass attribute set to true around the location (using an accumulator)
- private static isGrass(Location location): Boolean
  - Returns true if given location is Dirt object (determined using instanceof operator) with hasGrass attribute set to true
- private static isTree (Location location): Boolean
  - Returns true if given location is Tree object using instanceof operator

## How does this implementation follow good design principles?

ScanSurrounds is delegated (by the Ground class) the task of checking if the surrounding locations of a given location have objects of particular types. By doing so, these static methods would be used by Ground types to apply Do not Repeat Yourself principle.

## How does this class interact with the rest of the system?

- Used to assist classes with querying their surroundings

## Alternative implementations considered

We thought about expanding this class to account for other scenarios such as recognising if an actor is next to another actor, but the game engine provides an implementation for this already (which was not appropriate for implementing the grass-growing functionality).

# Package: portables

## PortableItem (Modified)

Abstract class extending Item

### Attributes

- private int: COST

### Methods

- public getCost(): int
    - Returns value of COST attribute

### How does this implementation follow good design principles?

By creating an abstract class for portable items, this creates a shared attribute and method between all portable items, because these can be bought from the vending machine. By doing so, we are applying the DRY principle.

### How does this class interact with the rest of the system?

- Extends Item
- Abstract class for items that are portable

### Alternative implementations considered

Initially implemented food items (that extend PortableItem) without this abstract class, but this would violate the DRY principle (as all food items have common behaviour that can be grouped).

## Food (New)

Abstract class extending PortableItem.

### Attributes

- private int: FILL
- private int: COST

### Methods

- public feed(Dinosaur dinosaur): void

    - Abstract method

### How does this implementation follow good design principles?

Defines common behaviour (e.g. feed method) for all items that can be fed to actors in the game.

### How does this class interact with the rest of the system?

- Extends PortableItem
- Acts as a “template” for items that can be eaten by dinosaurs

### Alternative implementations considered

Initially implemented food items (that extend PortableItem) without this abstract class, but this would violate the DRY principle (as all food items have common behaviour that can be grouped).

## Fruit (New)

- Extends Food abstract class
- Fruit objects in the player’s inventory will not rot because the tick() method is only called on fruit that is in a location

### Attributes

- private int: age
    - Keep track of when the fruit rots away, once fallen onto floor
- static final int: FILL = 30
- static final int: COST = 30
- static final int: POINTS_WHEN_FRUIT_FED = 15
- static final int: ROT_AGE = 20

### Methods

- public Fruit()
    - Set age to zero
- public tick(Location location): void
    - Increments age attribute by +1
    - If age >= ROT_AGE, remove the item from location.items
- public getAge(): int

        - Returns the current age of the fruit
- private getCost(): int
    - Returns value of COST attribute
- public feed(dinosaur:Dinosaur): void
    - Checks if dinosaur is Stegosaurus
        - Else throws error saying it can’t eat this
    - Increases dinosaur’s hunger
    - Instantiates EcoPointsSystem and calls earn() method

### How does this implementation follow good design principles?

The class is isolated and self-contained, so it does not have any “knowledge” of other classes. Thus, discussion of design principles is not applicable.

### How does this class interact with the rest of the system?

- Extends Food abstract class
- “Stored” as part of a Tree (when dropped on floor and not picked up) or Actor (when in inventory) object

### Alternative implementations considered

None.

## Hay (New)

Extends Food abstract class.

### Attributes

- private int: COST = 20
- private int: FILL = 20
- static final int: POINTS_WHEN_HAY_FED = 10

### Methods

- private getCost(): int
    - Returns value of COST attribute
- public feed(dinosaur:Dinosaur): void
    - Checks if dinosaur is Stegosaurus, else throws error saying it can’t eat this
    - Increases dinosaur’s hunger

    - Instantiates EcoPointsSystem and calls earn() method

### How does this implementation follow good design principles?

The class is isolated and self-contained, in that it does not have any “knowledge” of other classes. Nor does it require a specific implementation; it just exists as another item that needs to be kept track of for the game. Thus, discussion of design principles is not applicable.

### How does this class interact with the rest of the system?

- Extends Food abstract class
- “Stored” as part of Actor (when in inventory) object
- Hay is instantiated and added to inventory attribute of an Actor when an Actor decides to harvest grass

### Alternative implementations considered

None.

## MealKit (New)

Extends Food abstract class.

### Attributes

- private int: FILL = 100

### Methods

- public feed(dinosaur:Dinosaur): void
    - Abstract method. Code logic differs between its subclasses.

### How does this implementation follow good design principles?

By creating this as an abstract class, this follows the Do not Repeat Yourself principle since subclasses can inherit shared attributes and methods. Furthermore, the feed() method as an abstract method follows the Do not Repeat Yourself method since this shared method would be inherited by subclasses.

### What does this class interact with in the system?

- Because this is an abstract class, this can be extended by other MealKit subclasses
- Extends Food abstract class

### Alternative implementations considered

Initially implemented meal kits (that extended PortableItem rather than Food) without this abstract class, but this would violate the DRY principle (as all meal kits have common behaviour that can be grouped).

## VegetarianMealKit (New)

Extends MealKit class.

### Attributes

- private int: COST = 100

### Methods

- private getCost(): int
  - Returns value of COST attribute
- public feed(dinosaur: Dinosaur): void
  - Checks if dinosaur is Stegosaurus, else throws error saying it can't eat this
  - Increases dinosaur's hunger to MAXIMUM_HUNGER_LEVEL

### How does this implementation follow good design principles?

By inheriting the MealKit abstract class, this follows the Do not Repeat Yourself principle since it will inherit the shared attributes and methods of MealKit. This also follows the Fail Fast principle because it checks the precondition that the dinosaur needs to be a Stegosaurus.

### What does this class interact with in the system?

- VegetarianMealKit can be bought from VendingMachine by Player
- VegetarianMealKit can be fed to Stegosaurus by Player
- Extends MealKit class

### Alternative implementations considered

None.

## CarnivoreMealKit (New)

Extends MealKit class.

### Attributes

- private int: COST = 500

### Methods

- private getCost(): int
  - Returns value of COST attribute
- public feed(dinosaur: Dinosaur): void
  - Checks if dinosaur is Allosaurus, else throws error saying it can't eat this
  - Increases dinosaur's hunger to MAXIMUM_HUNGER_LEVEL

### How does this implementation follow good design principles?

By inheriting the MealKit abstract class, this follows the Do not Repeat Yourself principle since it will inherit the shared attributes and methods of MealKit. This also follows the Fail Fast principle because it checks the precondition that the dinosaur needs to be an Allosaurus.

### What does this class interact with in the system?

- CarnivoreMealKit can be bought from VendingMachine by Player
- CarnivoreMealKit can be fed to Allosaurus by Player
- Extends MealKit class

### Alternative implementations considered

None.

## Egg (New)

- Abstract class
- Extends PortableItem

### Attributes

- private int: timeAlive
- static final int: POINTS_WHEN_HATCH

### Methods

- public tick(Location location): void
  - Increments timeAlive by +1

    - If timeAlive = 10, call hatch()
- public hatch(Location location): Dinosaur
    - Abstract method

### How does this implementation follow good design principles?

By creating this as an abstract class, this follows the Do not Repeat Yourself principle since this creates a ‘template’ with shared attributes and methods for eggs. Therefore, any subclasses of Egg would inherit its attributes without having to rewrite in subclasses.

### What does this class interact with in the system?

Since this is an abstract class, it does not interact with the system, but its subclasses (e.g. eggs for specified dinosaur breeds) will.

### Alternative implementations considered

None.

## StegosaurusEgg (New)

- Extends Egg class

### Attributes

- private int: COST = 200
- static final int: POINTS_WHEN_HATCH = 100

### Methods

- public hatch(): Dinosaur
    - If timeAlive is equal to 10 turns, returns a Stegosaurus object with age zero
    - Instantiates EcoPointsSystem and calls earn() method
    - Removes StegosaurusEgg object from the system
- private getCost(): int
    - Returns value of COST attribute

### How does this implementation follow good design principles?

By extending the Egg class, this inherits the shared attributes and methods of Egg. This means that attributes and methods don’t have to be rewritten.

### What does this class interact with in the system?

- StegosaurusEgg hatches into Stegosaurus
- Extends Egg class

### Alternative implementations considered

None.

## AllosaurusEgg (New)

- Extends Egg class

### Attributes

- private int: COST = 1000
- static final int: POINTS_WHEN_HATCH = 1000

### Methods

- public hatch(): Dinosaur
  - If timeAlive is equal to 10 turns, returns an Allosaurus object with age zero
  - Instantiates EcoPointsSystem and calls earn() method
  - Removes AllosaurusEgg object from the system
- private getCost(): int
  - Returns value of COST attribute

### How does this implementation follow good design principles?

By extending the Egg class, this inherits the shared attributes and methods of Egg. This means that attributes and methods don't have to be rewritten.

### What does this class interact with in the system?

- AllosaurusEgg hatches into Allosaurus
- Extends Egg class

### Alternative implementations considered

None.

## LaserGun (New)

Extends WeaponItem.

### Attributes

- private int: COST = 500

### Methods

- public LaserGun()
  - Calls super()
- private getCost(): int
  - Returns value of COST attribute

### How does this implementation follow good design principles?

The class is isolated and self-contained, in that it does not have any “knowledge” of other classes. Nor does it require a specific implementation; it just exists as a weapon to keep track of and potentially use. Thus, discussion of design principles is not applicable.

### How does this class interact with the rest of the system?

- Extends WeaponItem
- When purchased from the vending machine, it is added to player’s inventory
  - When instantiated, damage should be set to 100 so that it can kill a dinosaur instantly

### Alternative implementations considered

None.

# Package: purchasing

## EcoPointsSystem

A single global EcoPointsSystem object will exist in the entire game. Players access this to spend points, and the engine will access this to earn points for the Player.

### Attributes

- private int: ecoPoints = 0

### Methods

- public getPoints(int points): int
  - Returns the eco-points that the Player can spend
- public earn(int points): void
  - Called by other classes in the system for the Player to earn eco-points
  - Precondition: points must be positive integer
  - This method is only used to earn money
- public spend(int points): void
  - Called by VendingMachine to spend eco-points
  - Precondition: points must be positive integer
  - Postcondition: ecoPoints cannot be lower than zero
  - This method is only used to spend money

### How does this implementation follow good design principles?

This class follows the Reduce Dependencies principle since the Grass and Egg classes would not need to know about the Player class for the Player to earn points. The above preconditions and postconditions listed follows the Fail Fast principle, since this ensures that the game runs properly and debugging becomes easier.

### What does this class interact with in the system?

- Eco-points are earned when Dirt grows grass
- Eco-points are earned when Player feeds Dinosaur with Hay or Fruit
- Eco-points are earned when Egg hatches
- Eco-points are spent by the Player for the VendingMachine

### Alternative implementations considered

Initially thought of assigning an ecoPoints attribute to Player, but this made earning points when Dirt grows grass or Egg hatches very difficult. This is because there was no method in the game engine to return the Player at any point of the map.

# VendingMachine (New)

A single VendingMachine object is to be added to the game map instantiated in the main() method of the Application class.

### Attributes

- private HashMap<String, Item>: items

### Methods

- public purchaseItem(String item): Item
  - Checks whether item exists in items
  - If so, creates instance of item and returns it
  - Instantiates EcoPointsSystem and calls spend() method

### How does this implementation follow good design principles?

Since this is a very small and isolated class, a discussion on design principles is not applicable.

### What does this class interact with in the system?

- Players can purchase Item objects from VendingMachine

### Alternative implementations considered

None.

# Package: actions

The Behaviour interface, FollowBehaviour class and WanderBehaviour classes will all be stored in this package as well.

## HarvestAction (New)

- Extends Action

### Attributes

- private Ground: target

### Methods

- public HarvestAction(Ground ground)
    - Sets target to ground
- public execute(Player player, GameMap gamemap): String
    - Overrides method in Action abstract class
    - If the player is standing on grass, calls harvestGrass() method to harvest the grass and turn it into hay
        - A Hay object will then be stored in the player's inventory
    - If player standing at the base of a tree, calls harvestFruit() to try and pick a fruit
        - If successful, a Fruit object will be stored in the player's inventory
- public menuDescription(Actor actor): String
    - Overrides method in Action abstract class
    - Description of action to display in the menu (e.g. "Player harvests the grass")

### How does this implementation follow good design principles?

Introducing a separate class for the action of harvesting crops of any form adheres to the single-responsibility principle, as it ensures that classes housing the crops only need to implement the functionality of growing the crop itself.

### How does this class interact with the rest of the system?

- Extends Action abstract class
- Allows a player to pick fruit from a tree and take hay from grass

## Alternative implementations considered

We did not create a method in Player to implement all this functionality because we wanted to avoid having lengthy methods in Player for each relevant action. This would make the class bloated (i.e. too many responsibilities).