Requirement 1: Zombie attacks

- new interface Chanceable:

An interface for classes that model an action with a success probability. The interface has one abstract method `public boolean isSuccessful()`.

Using interface polymorphism, a code from another class can determine whether a `Chanceable` object is carried out or not without knowing / checking the objects' type and its success probability.

For example, `IntrinsicPunch` and `IntrinsicBite` implement `Chanceable` since they both have different success probabilities (or hit probabilities). Without the interface, we have to check whether the operation is a punch or a bite before determining the hit probability, which involves an `if-else` and repetitions (if Zombies' intrinsic ability extends to more than just bite and punch). That would also mean the hit probability has to be accessible from outside the `IntrinsicPunch` and `IntrinsicBite` classes.

Design principles: DRY, Declare things in the tightest possible scope.

- new classes IntrinsicPunch, IntrinsicBite extends from IntrinsicWeapon adding `HIT_PROBABILITY, isSuccessful(), heal()` **(for IntrinsicBite):**

The original design made Zombie's punch an `IntrinsicWeapon` object (it follows that the same should go for Zombie's bite). However, there is no attribute in `IntrinsicWeapon` to show the hit probability or how many health points the punch (or bite) redeems to the Zombie. Using this design, we can still implement these specifications by checking whether a punch or bite had occurred, although it is not ideal since a punch or bite should be responsible for their hit probability and redeeming health points.

Design principle: DRY, Classes should be responsible for their own property

Requirement 2: Beating up the zombie

- new members in `Zombie` **class:**

new attributes `private int numArms, private int numLegs` showing number of existing arms and legs of the zombie

new methods `getArms(), getLegs()` to get the number of arms and legs

`loseArms(int lostArms), loseLegs(int lostLegs)` update number of arms and legs when arms or legs are knocked off and if `lostArms` and `lostLegs` are within valid range

Design principle: Classes should be responsible for their own property, Declare things in the tightest possible scope

- new class LimbOffAction extends from AttackAction overriding `execute`**:**

`LimbOffAction` is the attack that specifically targets Zombies (which involves arms and legs knocked off and turning into `Items`) and replaces the `AttackAction` in Zombie's `getAllowableActions`.

The Zombie itself (rather than the AttackAction) should be in charge of what kind of attack it is vulnerable to. `AttackAction` targets not just Zombies but also Humans and Player. Hence to implement the limb-off effects, it must first check if the target is a Zombie. The problem would arise if later, other ZombieActors also need their own customization, which might eventually result in a long complex `if-else` structure and makes it harder to debug or extend.

Design principle: DRY, Classes should be responsible for their own property

- new class ZombieAttackBehaviour extends from AttackBehaviour overriding `getAction():`
Zombie has distinct mechanism for choosing his attack action (subject to his number of remaining arms). However, since `AttackBehaviour` is also currently shared by many ZombieActors, it follows from above that Zombie's behaviour should have his own class rather than nested in an `if-else` structure.

Design principle: DRY, Classes should be responsible for their own property

Requirement 3: Crafting Weapons

- new class CraftWeaponAction (extends Action class):
This class overrides the `execute` method and `menuDescription method of Action class`

A new action specific to the player, designed to craft weapons using zombie's dropped limbs. A dropped `ZombieLimb` can give the player the option of crafting a new weapon that deals large damage compared to normal punches. A crafted `ZombieClub` deals 30 damage while a crafted `ZombieMaze` deals 50 damage. The `execute` method checks if the player's inventory has either a zombie arm or leg and iterates using a `for loop` to create a new crafted weapon object and adds to the player's inventory. An `if-else` statement checks if the item is a `ZombieArm` or a `ZombieLeg.` During the player's `playTurn` method, this action is called if the player's inventory is not empty.

Design principle: DRY, Classes should be responsible for their own property

- new classes ZombieArm, ZombieLeg, ZombieClub, ZombieMaze (extends WeaponItem class):
The following classes are all weapons (2 of which are dropped by zombies and 2 crafted), having unique names, characters and damage points. Follows the same design as `Plank` class from the initial game classes. `ZombieArm` and `ZombieLeg` are items dropped at zombies location by a zombie when attacked (they are created in the LimbOffAction class during `execute`). If the player moves over the location of a dropped item, the player can pick up the zombie arm or leg. The player is then asked if he/she wants to craft a weapon using the collected item. `ZombieClub` and `ZombieMaze` are weapons crafted using a zombie arm or leg in the game. This is an action performed by the `CraftWeaponAction` class

Design principle: DRY, Classes should be responsible for their own property

Requirement 4: Rising from the dead

- new class Corpse **extends from** Item **overriding** tick**:**
A Corpse is created every time a Human is killed and lasts for 5-10 game turns before transitioning into a Zombie. To keep track of game turn, the tick method of Item is a perfect candidate as it is called once every turn. Since the method currently does nothing, overriding is needed to monitor the number of turns that have passed since the creation of the Corpse.
Design principle: DRY

Requirement 5: Farmers and Food

- new class Farmers (extends Human class) :
Has the same behaviours as normal humans. The exception is that if the Farmer is standing next to Dirt, it has the ability to sow a crop at the location of dirt. This implements a new Crop class and inside farmers playTurn, it checks if there is dirt next to the farmer's location and sows a crop at a 33% probability. Second feature of Farmer is that if there is an unripe crop at the farmer's location, the farmer can fertilize it. I.e it reduces the time to ripe by 10 turns. Last feature of the Farmer is that it is able to harvest crops. If Farmer is standing on or next to a riped crop, it removes the riped crop from its location and creates a food item at the crop's location. To achieve this feature, the Map class's methods are used.

Design principle: DRY, Classes should be responsible for their own property

- class Player (extends Human class) :
During the player's playTurn method, if the player is standing next to, it has the option to harvest food. Following the same implementation as the farmer's harvest feature, if the player harvests food the Food item is created and added to the player's inventory. If the player has sustained damage by a zombie, and has food in their inventory, the player can choose to recover some health (10 health points) using EatAction.

Design principle: DRY

- class Human (extends ZombieActor) :
During its playTurn method, if the human has item Food in its inventory, and has sustained damage, health is automatically recovered by eating the food (uses EatAction).

Design principle: DRY,

- new class Crop (extends Ground class) :
A class that extends the Ground class, similar to Dirt and Tree classes. Overrides the tick() method from Ground. Has an attribute called status that has a setter. This attribute determines if the crop is ripe or unripe. Extended implementation of tick (), counts the number of turns. Once the number of turns reaches 20, the crop ripes (i.e status changes to ripe). However, if the Farmer decides to harvest a crop, 10 turns are added to

the number of turns. If the number of turns is already greater than or equal to 10, the crop ripes.

Design principle: DRY, Classes should be responsible for their own property

- new class Food (extends PortableItem class) :
A portable item that the player can carry in his/her inventory. Has a recovery level of 10 that when consumed adds to the player's hit points.

Design principle: DRY, Classes should be responsible for their own property

- new class EatAction (extends Actionclass) :
Allows the player to heal if their current hitpoint is less than `maxHitPoints`. Uses the `heal` method of `Action` class. This action is called in the player's `playturn` method when the player has sustained damage and has food in his/her inventory.

Design principle: DRY, Classes should be responsible for their own property