APCS Gridworld:Chapter 4 Name: Per: F

Set 7, Page 30 The source for the `Critter` class is in Appendix C.

1. What methods are implemented in `Critter`?

2. What are the five basic actions common to all critters when they act?

3. Should subclasses of `Critter` override the `getActors` method? Explain.

4. Describe three ways that a critter could process actors.

5. What three methods must be invoked to make a critter move? Explain each of these methods.
   (a)

   (b)

   (c)

6. Why is there no `Critter` constructor?

Set 8, Page 33
The source code for the `ChameleonCritter` class is in Appendix C.

1. Why does `act` cause a `ChameleonCritter` to act differently from a `Critter` even though `ChameleonCritter` does not override `act`?

2. Why does the `makeMove` method of `ChameleonCritter` call `super.makeMove`?

3. How would you make the `ChameleonCritter` drop flowers in its old location when it moves?

4. Why doesn't `ChameleonCritter` override the `getActors` method?

5. Which class contains the `getLocation` method?

6. How can a `Critter` access its own grid?

Set 9, Page 35
The source code for the `CrabCritter` class is reproduced at the end of this part of GridWorld. This code is not required for the AP CS Exam, but working with the code is good practice as preparation for the exam.

1. Why doesnt `CrabCritter` override the `processActors` method?

2. (a) Describe the process a `CrabCritter` uses to find and eat other actors.

   (b) Does it always eat all neighboring actors? Explain.

3. Why is the `getLocationsInDirections` method used in `CrabCritter`?

4. If a `CrabCritter` has location (3, 4) and faces south, what are the possible locations for actors that are returned by a call to the `getActors` method?

5. What are the similarities and differences between the movements of a `CrabCritter` and a `Critter`?

6. How does a `CrabCritter` determine when it turns instead of moving?

7. Why don't the `CrabCritter` objects eat each other?

Exercises, page 35

1. Modify the `processActors` method in `ChameleonCritter` so that if the list of actors to process is empty, the color of the `ChameleonCritter` will darken (like a flower).

In the following exercises, your first step should be to decide which of the five methods`getActors`, `processActors`, `getMoveLocations`, `selectMoveLocation`, and `makeMove` should be changed to get the desired result.

2. Create a class called `ChameleonKid` that extends `ChameleonCritter` as modified in exercise 1. A `ChameleonKid` changes its color to the color of one of the actors immediately in front or behind. If there is no actor in either of these locations, then the `ChameleonKid` darkens like the modified `ChameleonCritter`.

3. Create a class called `RockHound` that extends `Critter`. A `RockHound` gets the actors to be processed in the same way as a `Critter`. It removes any rocks in that list from the grid. A `RockHound` moves like a `Critter`.

4. Create a class `BlusterCritter` that extends `Critter`. A `BlusterCritter` looks at all of the neighbors within two steps of its current location. (For a `BlusterCritter` not near an edge, this includes 24 locations). It counts the number of critters in those locations. If there are fewer than c critters, the `BlusterCritter`'s color gets brighter (color values increase). If there are c or more critters, the `BlusterCritter`'s color darkens (color values decrease). Here, c is a value that indicates the courage of the critter. It should be set in the constructor.

5. Create a class `QuickCrab` that extends `CrabCritter`. A `QuickCrab` processes actors the same way a `CrabCritter` does. A `QuickCrab` moves to one of the two locations, randomly selected, that are two spaces to its right or left, if that location and the intervening location are both empty. Otherwise, a `QuickCrab` moves like a `CrabCritter`.

6. Create a class `KingCrab` that extends `CrabCritter`. A `KingCrab` gets the actors to be processed in the same way a `CrabCritter` does. A `KingCrab` causes each actor that it processes to move one location further away from the `KingCrab`. If the actor cannot move away, the `KingCrab` removes it from the grid. When the `KingCrab` has completed processing the actors, it moves like a `CrabCritter`.