Do you know:

Set 7:

1. What methods are implemented in Critter?

Answer: getActors, processActors, getMoveLocations, selectMoveLocation, makeMove and act.

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

   Answer: getActors, processActors, getMoveLocations, selectMoveLocation, makeMove

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

   Answer: Yes. Different type of critter have different behavior details. And if it need to get its actor in different way(or it want to get specify actor). It need to override the getActhor method.

4. Describe the way that a critter could process actors

   Answer: Critter first gets a list of actors to process, and then processes those actors like change their color and so on(call processActors method).

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

   Answer: getMoveLocations, selectMoveLocation, makeMove. First, act method call getMoveLocations to find a list of valid locations. And then call selectMoveLocation to select one of the location from the list to move. At last, act method call makeMove to move a critter.

6. Why is there no Critter constructor?

   Answer: Critter extends Actor, and Actor class has its own constructor. If you don't write a constructor for Critter, java will create a default constructor for it. And this constructor will call super method to call Actor's constructor. Critter don't need to initialize its object with special behavior, so we don't need to write a constructor for it.

Set 8:

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

   Answer: The act method call getActors, processActors, getMoveLocations, selectMoveLocation, makeMove methods. And ChameleonCritter override processActors and makeMove methods and them have different behavior from processActors and makeMove methods in Critter. As a result, calling act for a ChameleonCritter object will have different behavior from Critter object.

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

   Answer: When a ChameleonCritter moves, it first turns toward the new location and then move. Except for change the direction, ChameleonCritter has the same behavior as Critter when calling makeMove method. And call super.makeMove can make ChameleonCritter move like a Critter.

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

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

   Answer: We can override the makeMove method. After call moveTo method, add a flower in its previous location. The code is as follows:

```
public void makeMove(Location loc)
{
        Location preLoc = getLocation();
        setDirection(preLoc.getDirectionToward(loc));
        super.makeMove(loc);
        Flower flower = new Flower(getColor());
        flower.putSelfInGrid(getGrid(), preLoc);
}
```

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

Answer: Because ChameleonCritter doesn't have new behavior when calling getActors method. The Actor it want to get is the same as Critter's. As a result, it can inherit Critter's getActors method.

5. Which class contains the getLocation method?

Answer: Actor class contains this method. And because Critter extends Author and other critter class extends Critter. So the subclasses inherit this method.

6. How can a Critter access its own grid?

Answer: Call the getGrid method which inherit from Actor.

Set 9:

1. Why doesn't CrabCritter override the processActors method?

   Answer: Because CrabCritter have the same behavior as Critter when calling processActors method. It will inherit processActors method from Critter to process these actors.

2. Describe the process a CrabCritter uses to find and eat other actors. Does it always eat all neighboring actors? Explain.

   Answer: First call getActors method to find neighbors that are immediately in front, to the right-front, or the left-front of CrabCritter. And then call processActors method to eat these actor (if not Rock or critter).
   No . The actors in other locations will be ignore (not immediately in front, to the right-front, or the left-front of CrabCritter).

3. Why is the getLocationsInDirections method used in CrabCritter?

   Answer: The parameter of getLocationsInDirections method is a list of three angle that is in front, the right-front, or the left-front of critter. critter need getLocationsInDirections to get the valid locations in front, to the right-front, or to the left-front of it.

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?

   Answer: (4, 4), (4, 3), (4, 5)

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

   Answer:
   Similarities: Both CrabCritter and Critter do not change their direction and only call moveTo method. They both chose their next location randomly from the list of candidate location.

   Differences: A CrabCritter can move only to the right or to the left while Critter can move to an arbitrary direction. If a CrabCritter cannot move, then it turns 90 degrees, randomly to the left or right. While a Critter will not turn.

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

   Answer: If both left and right side of CrabCritter can not move, CrabCritter will need to turn. From the code we can see that CrabCritter will need to turn when the parameter loc (which it should move to) is equal to its location.

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

   Answer: CrabCritter extends Critter, and the processActors method in Critter can't eat Rock or other critter. And the processActors method in CrabCritter is inherit from Critter. So CrabCritter objects don't eat each other.

Exercise:

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).

   Answer: The answer is in code.

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.

   Answer: We need to override getAthors method to get the actors immediately in front or behind of critter. And to make this we should rewrite getLocationsInDirections to get the locations immediately in front or behind of critter.

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.

Answer: We need to override the processActors method.

```
public class RockHound extends Critter
{
   /**
    * Remove any rocks in that list from the grid
    */
   public void processActors(ArrayList<Actor> actors)
   {
         for (Actor actor : actors) {
               if (actor instanceof Rock) {
                     actor.removeSelfFromGrid();
               }
         }
   }
}
```

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.

Answer: We need to override processActors and getActors. And add two private method darken, bright to set the color of critter.

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.

Answer: We need to override the getMoveLocations method. In this method, we should found whether the left and right location is available. If not, we should call CrabCritter's getMoveLocations method. To do this, I add a ifLRCanMove method to judge whether the left and right location is available.

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.

Answer: We need to override the processActors method. To find a further location, add a method getLocFuther. To get the instance between two location, add a method getDistance.