Critter.java

```
package info.gridworld.actor;

import info.gridworld.grid.Location;
import java.util.ArrayList;

/**
 * A Critter is an actor that moves through its world, processing
 * other actors in some way and then moving to a new location.
 * Define your own critters by extending this class and overriding any methods of this class except for act.
 * When you override these methods, be sure to preserve the postconditions.
 * The implementation of this class is testable on the AP CS A and AB Exams.
 */
public class Critter extends Actor
{
  /**
   * A critter acts by getting a list of other actors, processing that list, getting locations to move to,
   * selecting one of them, and moving to the selected location.
   */
  public void act()
  {
    if (getGrid() == null)
      return;
    ArrayList<Actor> actors = getActors();
    processActors(actors);
    ArrayList<Location> moveLocs = getMoveLocations();
    Location loc = selectMoveLocation(moveLocs);
    makeMove(loc);
  }


  /**
   * Gets the actors for processing. Implemented to return the actors that occupy neighboring grid locations.
   * Override this method in subclasses to look elsewhere for actors to process.
   * Postcondition: The state of all actors is unchanged.
   * @return a list of actors that this critter wishes to process.
   */
  public ArrayList<Actor> getActors()
  {
    return getGrid().getNeighbors(getLocation());
  }
  /**
   * Processes the elements of actors. New actors may be added to empty locations.
   * Implemented to “eat” (i.e., remove) selected actors that are not rocks or critters.
   * Override this method in subclasses to process actors in a different way.
   * Postcondition: (1) The state of all actors in the grid other than this critter and the
   * elements of actors is unchanged. (2) The location of this critter is unchanged.
   * @param actors the actors to be processed
   */
  public void processActors(ArrayList<Actor> actors)
  {
    for (Actor a : actors)
    {
      if (!(a instanceof Rock) && !(a instanceof Critter))
        a.removeSelfFromGrid();
    }
  }


  /**
   * Gets a list of possible locations for the next move. These locations must be valid in the grid of this critter.
   * Implemented to return the empty neighboring locations. Override this method in subclasses to look
   * elsewhere for move locations.
   * Postcondition: The state of all actors is unchanged.
   * @return a list of possible locations for the next move
   */
  public ArrayList<Location> getMoveLocations()
  {
    return getGrid().getEmptyAdjacentLocations(getLocation());
  }


  /**
   * Selects the location for the next move. Implemented to randomly pick one of the possible locations,
   * or to return the current location if locs has size 0. Override this method in subclasses that
   * have another mechanism for selecting the next move location.
   * Postcondition: (1) The returned location is an element of locs, this critter's current location, or null.
   * (2) The state of all actors is unchanged.
   * @param locs the possible locations for the next move
   * @return the location that was selected for the next move.
   */
  public Location selectMoveLocation(ArrayList<Location> locs)
  {
    int n = locs.size();
    if (n == 0)
      return getLocation();
    int r = (int) (Math.random()  * n);
    return locs.get(r);
  }
```

```
    /**
     * Moves this critter to the given location loc, or removes this critter from its grid if loc is null.
     * An actor may be added to the old location. If there is a different actor at location loc, that actor is
     * removed from the grid. Override this method in subclasses that want to carry out other actions
     * (for example, turning this critter or adding an occupant in its previous location).
     * Postcondition: (1) getLocation() == loc.
     * (2) The state of all actors other than those at the old and new locations is unchanged.
     * @param loc the location to move to
     */
    public void makeMove(Location loc)
    {
        if (loc == null)
            removeSelfFromGrid();
        else
            moveTo(loc);
    }
}
```

ChameleonCritter.java

```
import info.gridworld.actor.Actor;
import info.gridworld.actor.Critter;
import info.gridworld.grid.Location;

import java.util.ArrayList;

/**
 * A ChameleonCritter takes on the color of neighboring actors as it moves through the grid.
 * The implementation of this class is testable on the AP CS A and AB Exams.
 */
public class ChameleonCritter extends Critter
{
    /**
     * Randomly selects a neighbor and changes this critter's color to be the same as that neighbor's.
     * If there are no neighbors, no action is taken.
     */
    public void processActors(ArrayList<Actor> actors)
    {
        int n = actors.size();
        if (n == 0)
            return;
        int r = (int) (Math.random()  * n);

        Actor other = actors.get(r);
        setColor(other.getColor());
    }


    /**
     * Turns towards the new location as it moves.
     */
    public void makeMove(Location loc)
    {
        setDirection(getLocation().getDirectionToward(loc));
        super.makeMove(loc);
    }
}
```