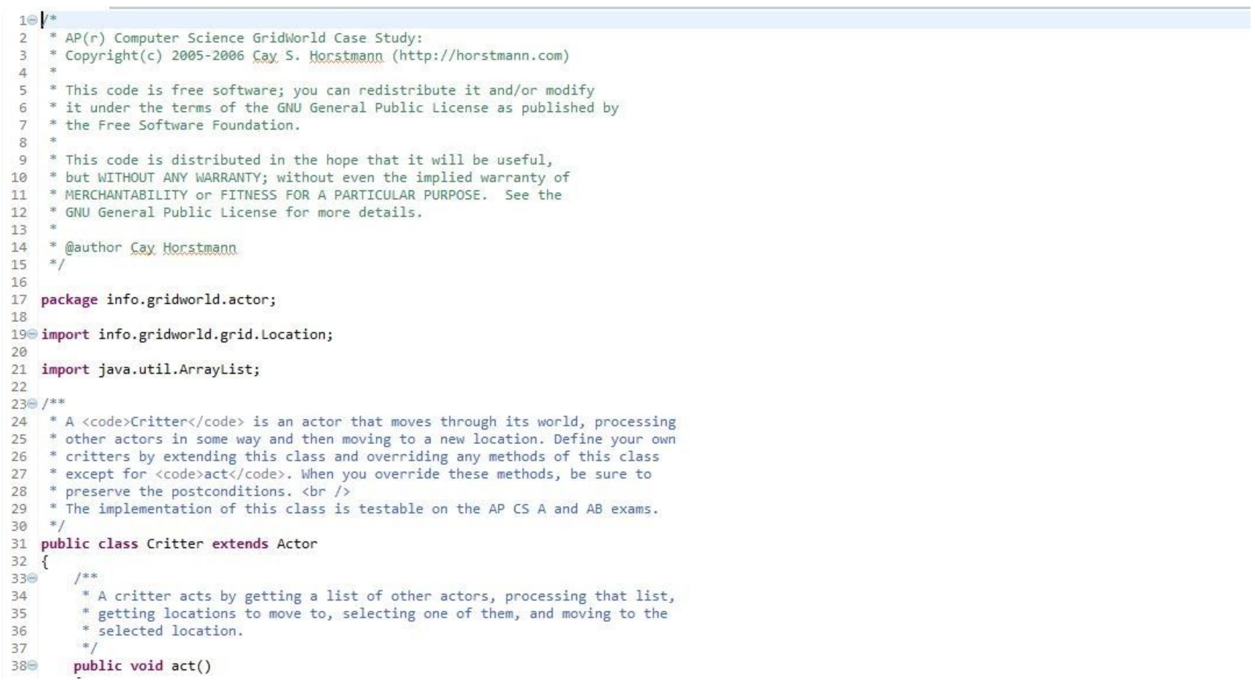
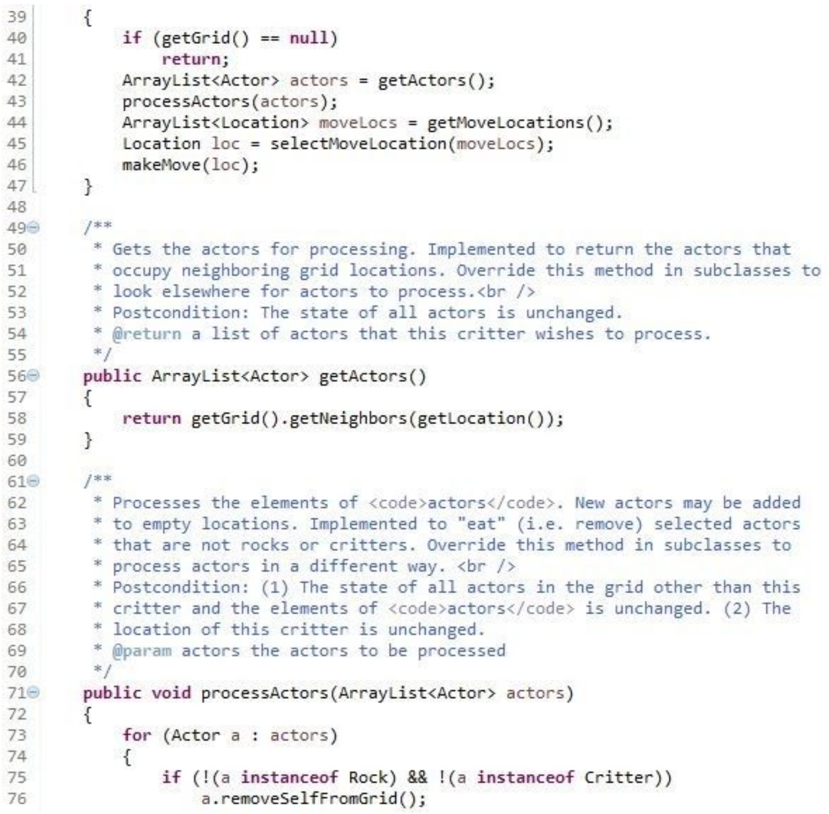
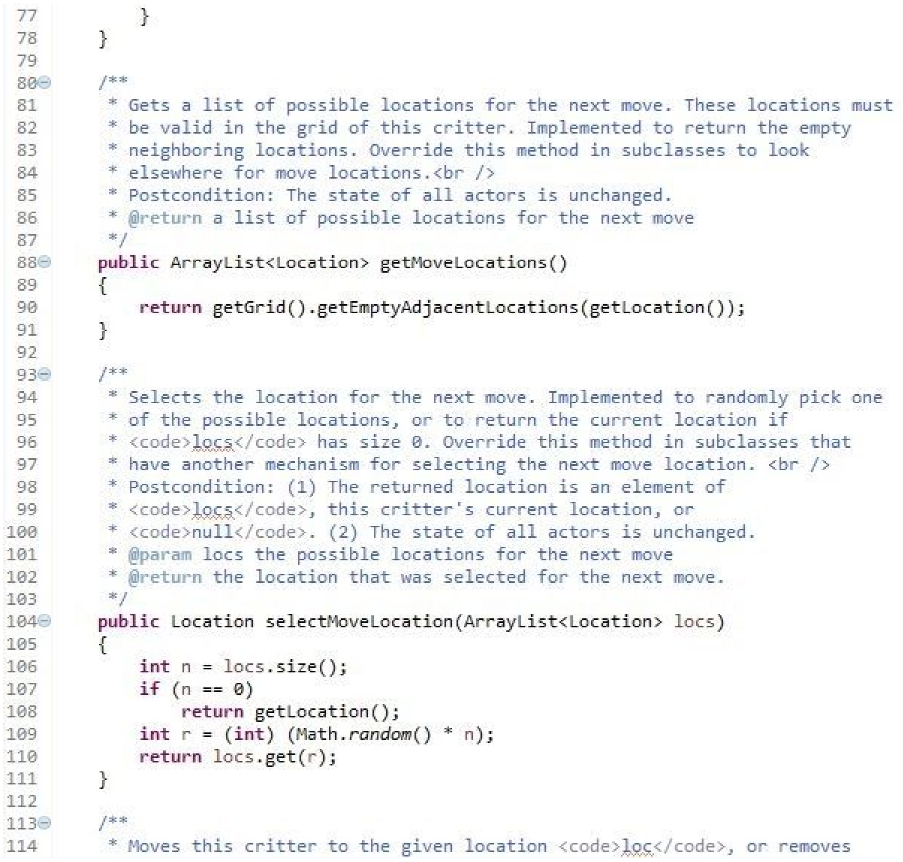


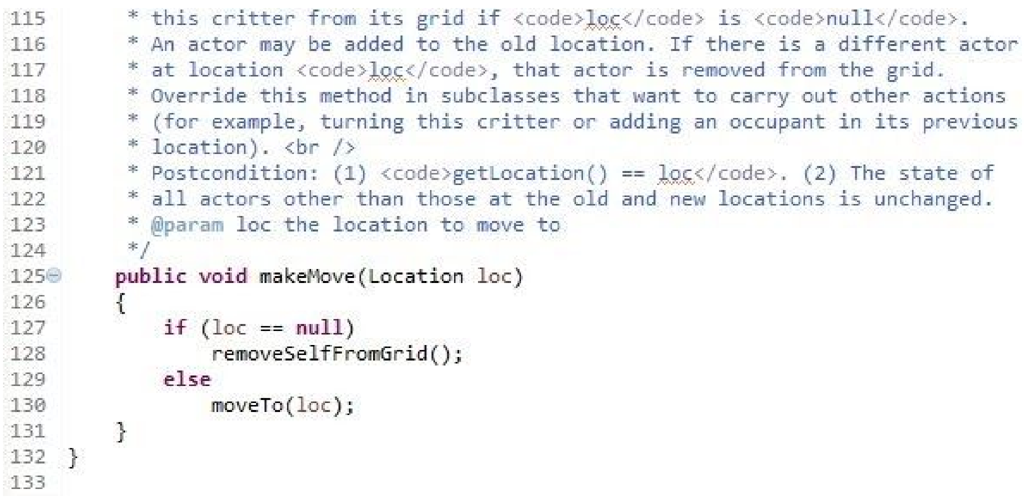
```java
/*
 * AP(r) Computer Science GridWorld Case Study:
 * Copyright(c) 2005-2006 Cay S. Horstmann (http://horstmann.com)
 *
 * This code is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation.
 *
 * This code is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * @author Cay Horstmann
 */

package info.gridworld.actor;

import info.gridworld.grid.Location;

import java.util.ArrayList;

/**
 * A <code>Critter</code> 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 <code>act</code>. When you override these methods, be sure to
 * preserve the postconditions. <br />
 * 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()
```



```java
    {
        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.<br />
     * 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 <code>actors</code>. 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. <br />
     * Postcondition: (1) The state of all actors in the grid other than this
     * critter and the elements of <code>actors</code> 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();
```



```java
        }
    }

    /**
     * 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.<br />
     * 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
     * <code>locs</code> has size 0. Override this method in subclasses that
     * have another mechanism for selecting the next move location. <br />
     * Postcondition: (1) The returned location is an element of
     * <code>locs</code>, this critter's current location, or
     * <code>null</code>. (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 <code>loc</code>, or removes
```



```java
     * this critter from its grid if <code>loc</code> is <code>null</code>.
     * An actor may be added to the old location. If there is a different actor
     * at location <code>loc</code>, 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). <br />
     * Postcondition: (1) <code>getLocation() == loc</code>. (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);
    }
}
```