

- I. Complete pages 1-3 (see the directions below) 20 pts
 II. Do page 4 (see the directions on page 4) 5 pts

I. A ParentBunny and a BabyBunny both extends Actor. A ParentBunny gets the actors to be processed in the same way as a Critter (but it does not extend Critter). A ParentBunny only eats Flowers and will generate a BabyBunny if two ParentBunnys are next to each other and there is a grid cell for the BabyBunny to go into. A ParentBunny will move like a Critter but it will also turn 90 degrees right if it does generate a BabyBunny. A BabyBunny does nothing in its act method.

Complete the code for a ParentBunny on this page and the next two.
 Use BabyBunnyRunner as the client code to test it.

```
public class ParentBunny extends _____
{
    Boolean canBreed;
    Boolean didBreed;

    public ParentBunny()
    {
        setColor(Color.BLACK);
    }

    public ParentBunny(Color BunnyColor)
    {
        setColor(BunnyColor);
    }

    /**
     * A ParentBunny acts by getting a list of its neighbors, processing them,
     * breeding if possible, 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 = _____ ;
        processActors(actors);
        ArrayList<Location> moveLocs = getEmptyLocations();
        if (canBreed)
            _____ = breed( _____ );
        else
            didBreed = false;
        ArrayList<Location> newMoveLocs = _____ ;
        Location loc = _____ (newMoveLocs);
        makeMove(loc);
    }
}
```

over

```
/**
 * Processes the actors. Implemented to "eat" (i.e. remove) only Flowers
 * Precondition: All objects in actors are contained in the same grid as this ParentBunny.
 * @param actors the actors to be processed
 * Postcondition: canBreed is true if there are 2 ParentBunnys adjacent to each other
 */
```

```
public void processActors(ArrayList<Actor> actors)
{
    canBreed = false;
    for (Actor a : _____)
    {
        if (a instanceof _____)
            a.removeSelfFromGrid();
        if (a instanceof _____)
            canBreed = true;
    }
}
```

```
/** Attempts to breed into neighboring locations. If so return true. If not return false.
 * Precondition: All locations in <code>locs</code> are valid in the grid of this actor
 */
```

```
public Boolean breed(ArrayList<Location> breedLocs)
{
    // If there is nowhere to breed, then we're done.
    int n = _____.size();
    if (n == 0)
    {
        return false;
    }

    // Breed to empty neighboring location.
    Location loc = selectMoveLocation(breedLocs);
    generateChild( _____ );
    return true;
}
```

```
/** Creates a new BabyBunny which adds itself to the grid.
 * param loc location of the new BabyBunny
 */
```

```
public void generateChild(Location loc)
{
    Grid<Actor> gr = getGrid();
    BabyBunny baby = new BabyBunny();
    baby. _____ ;
}
```

continued

```

/**
 * Gets the actors for processing. The actors must be contained in the same
 * grid as this ParentBunny. Implemented to return the actors that occupy neighboring grid locations.
 * @return a list of actors that are neighbors of this ParentBunny
 */
public ArrayList<Actor> getActors()
{
    return getGrid()._____ (getLocation());
}

/**
 * Gets the possible locations for the next move. Implemented to return the empty neighboring locations.
 * Postcondition: The locations must be valid in the grid of this actor.
 * @return a list of possible locations for the next move
 */
public ArrayList<Location> getEmptyLocations() // was called getMoveLocations() in Critter
{
    return getGrid()._____ (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.
 * Precondition: All locations in <code>locs</code> are valid in the grid of this actor
 * @param locs the possible locations for the next move
 * @return the location that was selected for the next move.
 */
public _____ selectMoveLocation(ArrayList<Location> locs)
{
    int n = locs.size();
    if (n == 0)
        return getLocation();
    int r = (int) (Math.random() * n);
    return _____ .get(r);
}

/**
 * Moves this ParentBunny to the given location. Implemented to call moveTo.
 * If a ParentBunny did breed then turn the ParentBunny 90 degrees right
 * Precondition: <code>loc</code> is valid in the grid of this ParentBunny
 * @param loc the location to move to (must be valid)
 */
public void makeMove(Location loc)
{
    if ( _____ )
        setDirection( _____ + _____ );
    moveTo( _____ );
}
}

```

over

II. Modify the ParentBunny class and BabyBunny class:

- A) The bunnies will only breed if they are the same color as each other.
- B) If they breed the baby bunny will be the same color as the parent's color.
- C) Use BabyBunnyRunnerb as the client code to test it.

Save the project as GWch7lab7b

When perfect, show your teacher the coding and output (run)

(teacher signature)

Turn in this sheet to be graded!