

A `CornerBug` behaves like a `Bug` except that a `CornerBug` makes all turns at right angles rather than 45-degree angles. Of the following, which is the best design for the `CornerBug` class?

- (A) Create an abstract class called `RightAngleBug` that is a `Bug` that only turns 90 degrees, and then create a class `CornerBug` that inherits from `RightAngleBug`.
 - (B) Create an interface called `RightTurn` that includes the specification of a `turnRight` method, and then create a class `CornerBug` that implements `RightTurn`.
 - (C) Create a class `CornerBug` that inherits from `Bug` and adds a constructor that has an `int` parameter that determines if the bug should turn 90 degrees or 45 degrees.
 - (D) Create a class `CornerBug` that inherits from `Bug` and overrides the `Bug` `turn` method to turn the bug 90 degrees instead of 45 degrees.
 - (E) Create an interface `CornerBug` that includes the definition of a `turnRight` method that is automatically used by the `Bug` `act` method for objects that are instantiated as `CornerBug` objects.
-

Consider the following class that inherits from `Bug` and overrides the `turn` method.

```
public class LeftTurningBug extends Bug
{
    public void turn()
    {
        /* missing code */
    }
}
```

Which of the following can be used to replace `/* missing code */` so that a `LeftTurningBug` object will always turn 90 degrees to the left?

- I. `setDirection(Location.LEFT);`
- II. `setDirection(getDirection() + Location.LEFT);`
- III. `setDirection(Location.HALF_LEFT + Location.HALF_LEFT);`

- (A) I only
- (B) II only
- (C) III only
- (D) I and III only
- (E) I, II, and III

Consider the following class declaration.

```
public class MysteryBug extends Bug
{
    private int count;

    public MysteryBug()
    {
        setDirection(Location.NORTHEAST);
        count = 0;
    }

    public void act()
    {
        if (count < 3 && canMove())
        {
            move();
            count++;
        }
        else
        {
            turn();
            count = 0;
        }
    }
}
```

Assume that a `MysteryBug` object has been placed in the center of an otherwise empty grid that is 20 rows by 20 columns. What pattern of flowers is formed if the `act` method of the `MysteryBug` is called many times in a row?

- (A) No pattern is formed because the `MysteryBug` never puts any flowers into the world.
- (B) A square pattern is formed just as with a `BoxBug` object.
- (C) A diamond pattern is formed (a square that has been rotated 45 degrees).
- (D) A six-sided pattern is formed.
- (E) An eight-sided pattern is formed.

Consider the following code segment.

```
Grid<Actor> grid = new BoundedGrid<Actor>(10, 10);

Bug bug1 = new Bug();
bug1.setDirection(Location.EAST);
bug1.putSelfInGrid(grid, new Location(3, 5));

Bug bug2 = new Bug();
bug2.setDirection(Location.SOUTH);
bug2.putSelfInGrid(grid, new Location(2, 6));

bug1.act();
bug2.act();
```

What is the result of executing the code segment?

- (A) bug1 and bug2 will both occupy location (3, 6).
 - (B) bug1 will move into location (3, 6) and bug2 will turn.
 - (C) bug1 will move into location (3, 6) and bug2 will do nothing.
 - (D) bug1 will move into location (3, 6) and bug2 will throw an exception.
 - (E) bug1 will move into location (3, 6) but will be removed when bug2 moves into the same location, (3, 6).
-

Consider the following code segment.

```
Grid<Actor> grid = new BoundedGrid<Actor>(10, 10);
Bug ladybug = new Bug();
ladybug.putSelfInGrid(grid, new Location(2, 8));
ladybug.setDirection(Location.WEST);
ladybug.act();
```

Under which of the following circumstances can ladybug advance to location (2,7)?

- I. The location (2,7) contains another Bug.
 - II. The location (2,7) contains a Flower.
 - III. The location (2,7) contains a Rock.
- (A) I only
 - (B) II only
 - (C) III only
 - (D) I and II only
 - (E) I, II, and III