

SECTION II: FREE RESPONSE

1. (a)

```
public boolean isValidGrid()
{
    Grid<Actor> gr = getGrid();
    if (gr == null)
        return false;

    ArrayList<Location> occupiedLocs =
        gr.getOccupiedLocations();
    for (Location loc : occupiedLocs)
    {
        Actor a = gr.get(loc);
        if (!(a instanceof Rock || a instanceof Flower)
            && a != this)
            return false;
    }
    return true;
}
```

(b)

```
public void moveLeft()
{
    setDirection(getDirection() + Location.LEFT);1
    move();
}
```

Notes:

1. Or:

```
setDirection(getDirection() - 90);
```

(c)

```
public boolean canMoveLeft()
{
    Grid<Actor> gr = getGrid();
    if (gr == null)
        return false;
    Location loc = getLocation();
    Location next =
        loc.getAdjacentLocation(getDirection() - 90);
    Location corner =
        loc.getAdjacentLocation(getDirection() - 135);1
    return gr.isValid(next) &&
        !(gr.get(next) instanceof Rock) &&
        (!gr.isValid(corner) ||
            gr.get(corner) instanceof Rock);2
}
```

Notes:

1. Or:

```
Location next =
    loc.getAdjacentLocation(getDirection() +
        Location.LEFT);
Location corner =
    loc.getAdjacentLocation(getDirection() +
        3 * Location.HALF_LEFT);
```

2. An alternative solution:

```
public boolean canMoveLeft()
{
    Grid<Actor> gr = getGrid();
    if (gr == null)
        return false;
    Location loc = getLocation();
    Location next =
        loc.getAdjacentLocation(getDirection() - 90);
    if (!gr.isValid(next) || gr.get(next) instanceof Rock)
        return false;
    Location corner =
        loc.getAdjacentLocation(getDirection() - 135);
    if (gr.isValid(corner) &&
        !(gr.get(corner) instanceof Rock))
        return false;
    return true;
}
```

(d)

```
public void turn()
{
    super.turn();
    super.turn();
}
```

2. (a)

```
public int distance(Student other)
{
    if (street.equals(other.getStreet()))
        return Math.abs(getNumber() - other.getNumber());
    else
        return 99999;
}
```

(b)

```
public class SchoolBus
{
    private int numSeats;
    private ArrayList<Student> students;

    public SchoolBus(int capacity)
    {
        numSeats = capacity;
        students = new ArrayList<Student>();1
    }

    public int getNumStudents() { return students.size(); }

    public boolean isFull()
    { return getNumStudents() == numSeats; }

    public Student getStudent(int i) { return students.get(i); }

    public boolean add(Student student)
    {
        if (!isFull())
        {
            students.add(student);
            return true;
        }
        return false;
    }

    public void printOut()
    {
        for (Student s : students)
            System.out.println(s);
    }
}
```

Notes:

1. Or:

```
students = new ArrayList<Student>(capacity);
```

(c)

```
boolean enroll(Student student)
{
    int minDistance = 10000;
    SchoolBus bestBus = null;

    for (SchoolBus bus : buses)
    {
        if (!bus.isFull())
        {
            for (int i = 0; i < bus.getNumStudents(); i++)
            {
                int d = student.distance(bus.getStudent(i));
                if (d < minDistance)
                {
                    minDistance = d;
                    bestBus = bus;
                }
            }
        }
    }

    if (bestBus != null)
    {
        bestBus.add(student);
        return true;
    }

    return false;
}
```