# Assignment 4: Chapter 5, “Writing Classes” (8%)

This programming project should be completed and submitted by the end of Week 10, and is worth 8% of your final grade. Please refer to the “Assignment Instructions” for details on the marking rubric and submission instructions.

1. Design and implement a class called Bug, which represents a bug moving along a horizontal wire. The bug can only move in one direction for one unit of distance at a time. The bug can also turn to reverse direction. For your design, create a UML Class diagram similar to Figure 5.5 on page 180 of the textbook. Create a separate driver program that instantiates one or more Bug objects and demonstrates things like moving several times, then turning, then moving the same number of times (back to the original position).

**Hint:** Remember that a horizontal line has a zero position in the middle with positive to the right and negative to the left. Consider that a bug will land on the wire at some point before moving along the wire. Bug will require a toString method to return the current position along with which direction the bug is facing to the driver so it can be output.

1. Design and implement a class called Card that represents a standard playing card. Each card has a suit and face value. For your design, create a UML Class diagram similar to Figure 5.5 on page 180 of the textbook. Create a separate driver program that deals five random cards, then two specific cards that you choose. It is **not** necessary to ensure the cards being output are unique.

**Hint:** Represent the faces Ace thru King as 1 thru 13 and the suits as 1 thru 4. You need two constructors: one that receives a suit and face value as parameters, plus the default constructor where these values are randomly generated. The face and suit must have appropriate get\_ and set\_ methods for the numeric values, plus a get\_ for the textual equivalent (e.g. getFace() might return 13 while getFaceText() would return King). Your toString method should return a nice representation of the values like “Ace of Spades.”

|  |  |
| --- | --- |
| **Assignment Marking Criteria** | **Weighting** |
| **Correctness of solution:** Algorithm is implemented and produces correct results for the stated problem | /4 |
| **Testing:** Submission of test exhibits to indicate the solution works for a range of cases (e.g. minimum and maximum inputs) and handles unexpected exceptions | /2 |
| **Comments and documentation:** Source code contains comments that explain in plain English what the code is intended to do  **Note:** Javadoc style is **not** required. | /2 |