COMP202
Instructor: Ladan Mahabadi
February 18, 2008

Professor Absem's Grading:

It is the end of term and Professor Absem needs to compile grades for his students. Each student's final grade can be only one of:

A+: > 90
A: 85 – 90
A-: 80 – 84.9
B+: 75 – 79.9
B: 70 – 74.9
B-: 65 – 69.9
C+: 60 – 64.9
C: 55 – 59.9
C-: 50 – 54.9
F: < 49.9

Write a program that allows the professor to enter 5 numerical values (corresponding to three projects, a midterm and a final exam) and outputs the corresponding letter grade. The five entered grades consist of grades for three projects (worth %30 of the final grade), a midterm exam (%20 of the final grade) and a final exam (worth %50 of the final grade).

```
import java.util.Scanner;

public class Grading
{
 enum Grade {AP, A, AM, BP, B, BM, CP, C, CM, F}

 public static void main (String[] args)
 {
  /**
   * Three projects are worth %30
   * Midterm is worth %20
   * Final exam is worth %50 of the final grade
   * */
  Scanner scan = new Scanner(System.in);
  double result = 0.0;
  Grade finalGrade = Grade.F;

  for (int i = 0; i < 3; i++)
  {
   System.out.println("Project " + (i+1) + " grade:");
   result += scan.nextDouble();
  }

  result *= 0.30;

  System.out.println("Midterm grade: ");
  result += scan.nextDouble() * 0.20;

  System.out.println("Final exam grade: ");
  result += scan.nextDouble() * 0.50;

  if (result >= 90)
   finalGrade = Grade.AP;

  if (result >= 85 && result <90)
   finalGrade = Grade.A;

  if (result >= 80 && result <85)
   finalGrade = Grade.AM;

  if (result >= 75 && result <80)
   finalGrade = Grade.BP;

  if (result >= 70 && result <75)
   finalGrade = Grade.B;

  if (result >= 65 && result <70)
   finalGrade = Grade.BM;

  if (result >= 60 && result <65)
```

```
  finalGrade = Grade.CP;

if (result >= 55 && result <60)
  finalGrade = Grade.C;

if (result >= 50 && result <55)
  finalGrade = Grade.CM;

if (result < 50)
  finalGrade = Grade.F;

switch(finalGrade)
{
  case AP:
    System.out.println("Final Grade: A+");
    break;

  case A:
    System.out.println("Final Grade: A");
    break;

  case AM:
    System.out.println("Final Grade: A-");
    break;

  case BP:
    System.out.println("Final Grade: B+");
    break;

  case B:
    System.out.println("Final Grade: B");
    break;

  case BM:
    System.out.println("Final Grade: B-");
    break;

  case CP:
    System.out.println("Final Grade: C+");
    break;

  case C:
    System.out.println("Final Grade: C");
    break;

  case CM:
    System.out.println("Final Grade: C-");
    break;

  case F:
    System.out.println("Final Grade: F");
    break;
```

```
    }
  }
}
```

BZ B

A (2-d) bee moves around a 10 by 10 2-d Grid pollinating the flowers located at discrete positions of the grid. At any time, the bee can move to the cell location directly above, below, to the left or to the right of his current position and will receive a reward associated with that cell. The bee starts from a random point on the grid, moves around (without falling off the grid) for a specific number of steps and accumulates pollen.

First, we need to create a class representing the points of the Grid:

```
public class GridPoint
{
 private int x;
 private int y;
 private int reward;

 public GridPoint(int inputX, int inputY)
 {
  //Set the the values of x and y to newX and newY respectively


 //Set reward to be the sum of x and y


 }

 /**
  * Accessor methods: getX, getY, getReward
  **/

 public int getX()
 {

 }

 public int getY()
 {

 }

 public int getReward()
 {

 }

 /**
  * Modifier methods: setX, setY, setReward
```

```
 **/

 public void setX(int newX)
 {

 }

 public void setY(int newY)
 {

 }

 public void setReward(int newReward)
 {

 }

}
```

Now, let's implement the Bzb Class:

```
import java.util.Random;

public class Bzb
{
  private GridPoint location;
  private int pollen;

  public Bzb()
  {
    //current location to a random point in the 10 X 10 range



    //initial pollen is zero

  }

  /**
   * Set the current position to a new position on the grid
   * */
  public void setLocation(int newX, int newY)
  {
    // If still within the 10X10 Grid, move to the new position; else, remain in position
```

```
}

/**
 * Get the current value of value
 * */
public int getPollen()
{

}

/**
 * Set pollen to a new value
 * */
public void setPollen(int newPollen)
{

}

/**
 * Pre: None
 * Post: Randomly move up, down, left or right
 *       update your location
 *       update pollen to the reward at that point
 * */

public void move()
{
  Random gen = new Random();
  int option = gen.nextInt(4);

  /**
   * 0 corresponds to Up
   * 1 corresponds to Down
   * 2 corresponds to Left
   * 3 corresponds to Right
   * */
```

```
    //Update Pollen

  }

  /**
   * Diplay where the bee is and how much he's accumulated
   * */
  public void print()
  {


  }

}
```

Now, use BzbTest to create a Busy bee:

```
import java.util.Scanner;

public class BzbTest
{
 public static void main(String[] args)
 {
   //Create a bee called Bob




   //Ask the user to enter the number of steps Bob will move around the grid









  /**
```

```
   * After bob's moved around enough, display his current position and
   * his total amount of pollen
   * */




  }
}
```

Now suppose that you'd like to take advantage of enumerated types, improve the method decomposition of the above program (Bzb Class) and implement the above by adding the following four methods:

```
Public void moveLeft()
Public void moveRight()
Public void moveUp()
Public void moveDown()
```

Also modify the move method so that:

```
/**
* It randomly chooses one of four possibilities: Left, Right, Up, Down
* Updates the current position using one of the above added methods
* Displays the direction taken (Left, Right, Up, or Down) to the screen
**/
```