CSCI-UA-102-011-Spring-2025

Recitation - 1

- Rasmika Billa
(01/24/2025)

Overview

- Office hour + contact
  1. Office hour: Tuesday 1:00 - 3:00PM, Location: 60FifthAve Room 204
  2. Email: rb5719@nyu.edu

- Grading
  1. Mid-Term: 20%
  2. Final: 30%
  3. Quizzes: 30% (6-8)
  4. Programming: 20%

- Recitation logistics
  1. Attendance and participation are required. No recordings.
  2. Practice problems
     - Not graded. But it's supposed to help you with quizzes.
     - You should work out the practice problems in groups.
  3. Quizzes (20 minutes before the end of the recitation)
     - The content is going to be related to the materials of the week. I will not answer questions regarding the quiz before it takes place.
     - I will provide my answer before the end of day on Friday (I will try to provide it before 6 PM).
     - Quizzes should be done by individuals.

Agenda

1. Download Eclipse
   - https://www.eclipse.org/downloads/packages/

2. Groups (5 mins)
   - Form Groups of 4
   - Each group will be given a problem that should be solved among them
   - After discussing one of the group member has to present the solution

3. OOP - Inheritance

```
1  /** Generates a simple progression. By default: 0, 1, 2, ... */
2  public class Progression {
3
4    // instance variable
5    protected long current;
6
7    /** Constructs a progression starting at zero. */
8    public Progression( ) { this(0); }
9
10   /** Constructs a progression with given start value. */
11   public Progression(long start) { current = start; }
12
13   /** Returns the next value of the progression. */
14   public long nextValue( ) {
15     long answer = current;
16     advance( );      // this protected call is responsible for advancing the current value
17     return answer;
18   }
19
20   /** Advances the current value to the next value of the progression. */
21   protected void advance( ) {
22     current++;
23   }
24
25   /** Prints the next n values of the progression, separated by spaces. */
26   public void printProgression(int n) {
27     System.out.print(nextValue( ));           // print first value without leading space
28     for (int j=1; j < n; j++)
29       System.out.print(" " + nextValue( )); // print leading space before others
30     System.out.println( );                    // end the line
31   }
32 }
```

Code Fragment 2.2: General numeric progression class.

1. Encapsulation : current (Line 5)
2. Constructors : Progression() (Line 8,11)
   Progression(long start)

Line 2: Class begins.
Line 5: Variable 'current' is declared
has current value of the progression
Line 8-11: Constructor initialization
Line 14-17: nextValue()
returns the current value
advances the progression
Line 21-23: advance() increments current by 1
Line 26-31: printProgression()
` Prints first n values of the progression.

```
Progression prog = new Progression();
prog.printProgression(5);                    //0 1 2 3 4
```

```
1  public class ArithmeticProgression extends Progression {
2
3    protected long increment;
4
5    /** Constructs progression 0, 1, 2, ... */
6    public ArithmeticProgression( ) { this(1, 0); }       // start at 0 with increment of 1
7
8    /** Constructs progression 0, stepsize, 2*stepsize, ... */
9    public ArithmeticProgression(long stepsize) { this(stepsize, 0); }       // start at 0
10
11   /** Constructs arithmetic progression with arbitrary start and increment. */
12   public ArithmeticProgression(long stepsize, long start) {
13     super(start);
14     increment = stepsize;
15   }
16
17   /** Adds the arithmetic increment to the current value. */
18   protected void advance( ) {
19     current += increment;
20   }
21 }
```

Code Fragment 2.3: Class for arithmetic progressions, which inherits from the general progression class shown in Code Fragment 2.2.

Inheritance: extends Progression (Line 1)

Line 2: ArithmeticProgression class
Subclass of Progression.
Line 3: Variable increment
Store the step size.
Lines 6-9: Constructors
Sets a default increment 1, starting from 0.
Takes a custom step size and starts at 0.
Allows arbitrary start values, increments.
Line 13: Superclass (Progression) = super(start)
Lines 18-20: The advance() method is overridden to add the increment to the current value, advancing the progression by the step size.

```
ArithmeticProgression arithProg = new
ArithmeticProgression(3);
arithProg.printProgression(5);                    // 0 3 6 9 12
```

```
1  public class GeometricProgression extends Progression {
2
3    protected long base;
4
5    /** Constructs progression 1, 2, 4, 8, 16, ... */
6    public GeometricProgression( ) { this(2, 1); }          // start at 1 with base of 2
7
8    /** Constructs progression 1, b, b^2, b^3, b^4, ... for base b. */
9    public GeometricProgression(long b) { this(b, 1); }                  // start at 1
10
11   /** Constructs geometric progression with arbitrary base and start. */
12   public GeometricProgression(long b, long start) {
13     super(start);
14     base = b;
15   }
16
17   /** Multiplies the current value by the geometric base. */
18   protected void advance( ) {
19     current *= base;                        // multiply current by the geometric base
20   }
21 }
```

Code Fragment 2.4: Class for geometric progressions.

Inheritance: extends Progression (Line 1)

Line 2: GeometricProgression class
Inheriting from Progression.
Line 3: Variable base
Store the base (or multiplier)
Lines 6-9: Constructors
Sets to start at 1 with a base of 2.
Takes a custom base and starts at 1.
Allows both the base and starting value to be customized.
Line 13: Superclass (Progression) = super(start)
Lines 18-20: The advance() method multiplies the current value by base to move to the next term in the geometric progression.

```
GeometricProgression geoProg = new
GeometricProgression(3);
geoProg.printProgression(5);                    //1 3 9 27 81
```

Problem Statements (10-15 mins)

- Q2.10 - Group 1
- Q2.11 - Group 2
- Q2.12 - Group 3
- Q2.13 - Group 4
- Q2.17 - Group 5
- Q2.21 - Group 6
- Q2.24 - Group 7
- Q2.29 - Group 8