CMPT-225 Jan Manuch

Recommended Labs – Monday, July 10, 2006

The goal of this lab is to get hands on an abstract base class.

The task is to design several classes generating different types of numeric progressions: arithmetic, geometric, Fibonacci, etc. The common part of these classes is captured in the following abstract class Progression:

```
public abstract class Progression {
	protected long first;
	protected long current;

	public Progression(long start)
	{
		current=first=start;
	}

	public long firstValue()
	// Reset the progression to the first value
	// and return that value.
	{
		current=first;
		return current;
	}

	public abstract long nextValue();
	// Step the progression to the next value and return that value.

	public void printProgression(int length)
	// Print the first 'length' values of the progression.
	{
		System.out.print(firstValue());
		for (int i=2; i<=length; i++)
			System.out.print(" "+nextValue());
		System.out.println();
	}
}
```

The class knows what should be the first value generated, but it doesn't know how to generate a next value, that's why the method is left abstract and is to be overridden in the non-abstract subclasses.

Use inheritance to design the following subclasses:

1. ArithProgression; the constructor should take two arguments: first value and increment. For instance, for values 2 and 5, the generated sequence of length 5 should be 2, 7, 12, 17, 22. It should be enough to provide the constructor and override nextValue() method.

2. GeomProgression; the constructor should take two arguments: first value and base (of multiplication). For instance, for values 2 and 5, the generated sequence of length 5 should be 2, 10, 50, 250, 1250
3. FibonacciProgression; the constructor should take two arguments: first value and the second value. Besides changing the first two value, the sequence should behave as normal Fibonacci sequence. For instance, for values 2 and 5, the generated sequence of length 5 should be: 2, 5, 7, 12, 19. For Fibonacci sequence it might be necessary to override method firstValue() as well (depending on your implementation).

Test you progressions with the following code:

```
class Tester {
	public static void main() {
		Progression list[3];
		list[0]=new ArithProgression(2,7);
		list[1]=new GeomProgression(2,7);
		list[2]=new FibonacciProgression(2,7);

		for (int i=0; i<3; i++) {
			list[i].printProgression(5);
			System.out.println("The sum: "+
				sumProgression(list[i],5));
		}
	}

	public static int sumProgression(Progression P,int limit)
	// PRE: ASSUME limit>=1
	{
		int sum=P.firstValue();
		for (int i=2; i<=limit; i++)
			sum += P.nextValue();
		return sum;
	}
}
```

Note that the method pringProgression() provided in the abstract base class Progression will call nextValue() method from the appropriate class (depending on the type of object). The same will happen in the new method sumProgression().