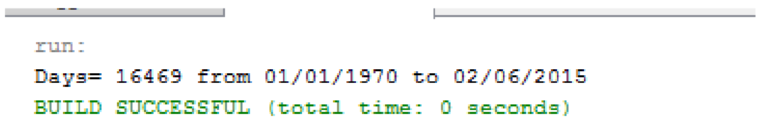
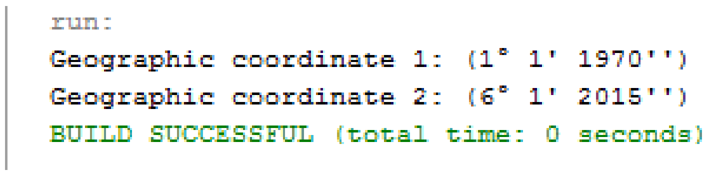
1. Create an implementation Date2.java that represents a date a single integer that counts the number of days since January 1, 1970. Compare to Date.java.



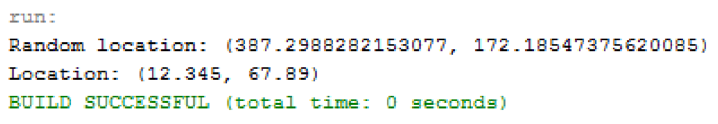
```
run:
Days= 16469 from 01/01/1970 to 02/06/2015
BUILD SUCCESSFUL (total time: 0 seconds)
```

2. Create a data type `GeographicCoordinate` that represents a geographic coordinate either in (degrees, minutes, seconds, sign) or in floating point.



```
run:
Geographic coordinate 1: (1° 1' 1970'')
Geographic coordinate 2: (6° 1' 2015'')
BUILD SUCCESSFUL (total time: 0 seconds)
```

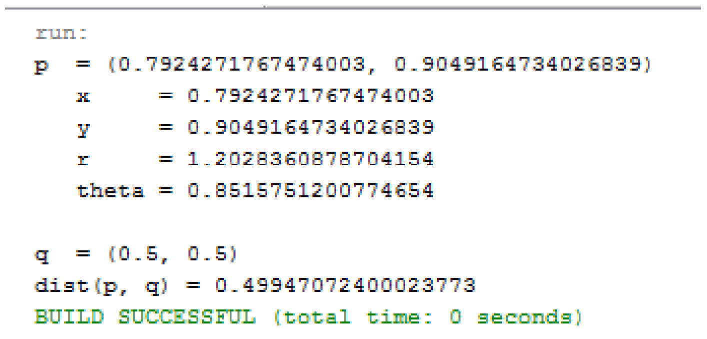
3. Create a data type `Location` for dealing with locations on Earth using spherical coordinates (latitude/longitude). Include methods to generate a random location on the surface of the Earth, parse a location "25.344 N, 63.5532 W" and compute the great circle distance between two locations.



```
run:
Random location: (387.2988282153077, 172.18547375620085)
Location: (12.345, 67.89)
BUILD SUCCESSFUL (total time: 0 seconds)
```

4. **Polar representation of points.** Point.java and PointPolar.java implement the following point interface using rectangular and polar coordinates, respectively.

```
Point()
Point(double, double)
double x()
double y()
double r()
double theta()
double distance(Point)
public String toString()
```



```
run:
p  = (0.7924271767474003, 0.9049164734026839)
   x     = 0.7924271767474003
   y     = 0.9049164734026839
   r     = 1.2028360878704154
   theta = 0.8515751200774654

q  = (0.5, 0.5)
dist(p, q) = 0.49947072400023773
BUILD SUCCESSFUL (total time: 0 seconds)
```

5. **Encapsulation.** Why does the following break encapsulation, even though all instance variables are declared `private`.

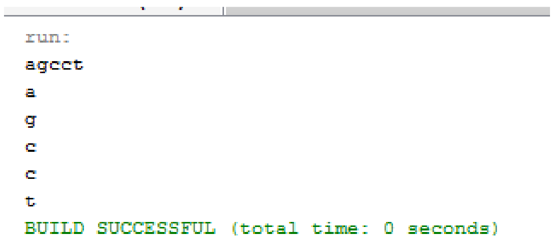
```
public class Appointment {
    private Date date;
    private String customer;
    public Appointment(Date date) {
        // check that date is in some legal range
        this.date = date;
    }
    public Date getDate() { return date; }
```

Answer: The reason is that the class `Date` is mutable. The method `setDate(seconds)` changes the value of the invoking date to the number of milliseconds since January 1, 1970, 00:00:00 GMT. This has the unfortunate consequence that when the function `d = getDate()` returns the date, the client program can invoke `d.setDate()` and change the date in an `Appointment` object type, perhaps setting it to an illegal value for a member of `Appointment`. Must not let references to mutable objects escape since caller can then modify its state. One solution is to create a *defensive copy* of the `Date` before returning it using `new Date(date.getTime())`; also need to do a defensive copy when storing it via `this.date = new Date(date.getTime())`. Many programmers regard the mutability of `Date` as a design flaw. (`GregorianCalendar` is a more modern Java library for storing dates; but it is mutable too.)

6. **Genome.** Implement a data type to store the genome of an organism. Biologists often abstract away the genome to a sequence of nucleotides (A, C, G, or T). The data type should support the method `addNucleotide`, `nucleotideAt(int i)`, and `doSomeComputation`. Perhaps change to `addCodon`. Advantages of encapsulation: can check that only legal nucleotides are added, can change to more time or memory efficient implementation without affecting client.

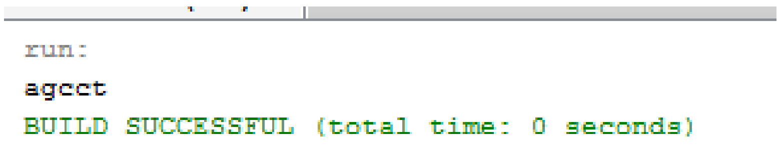
   - StringGenome.java has one instance variable of type `String`. It implements `addNucleotide` with string concatenation. Each method call takes time proportional to the size of the current genome. Not practical spacewise either for large genomes since nucleotide is stored as a 16-bit `char`.

Activity 6



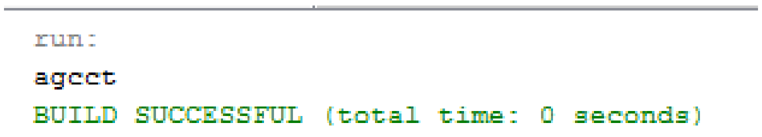
```
run:
agcct
a
g
c
c
t
BUILD SUCCESSFUL (total time: 0 seconds)
```

- Genome.java implements a genome as an array of characters. The size of the array is doubled when the array fills up. The method `addNucleotide` is now constant time. Space consumption is still 16 bits per nucleotide.



```
run:
agcct
BUILD SUCCESSFUL (total time: 0 seconds)
```

- CompactGenome.java implements a genome as boolean array. We need to use two bits per nucleotide since there are 4 different nucleotides. As in the previous implementation, we use a dynamic array with repeated doubling. Now, each nucleotide consumes 2 bits of storage (instead of 16).



```
run:
agcct
BUILD SUCCESSFUL (total time: 0 seconds)
```