Chapter 2 (review)

- Class definitions
- Instance variables
- Constructors
- Methods, accessor methods

package edu.colorado.simulations;
public class Throttle {
  private int top;      // The topmost position of the throttle
  private int position; // The current position of the throttle
  public Throttle(int size) {
    if (size <= 0)
      throw new IllegalArgumentException("Size <= 0: " + size);
    top = size;
    position = 0;
    // No assignment needed for position -- it gets the default value of 0.
  }
  public double getFlow() {
    return (double)position / (double)top;
  }
  public boolean isOn() {
    return (position > 0);
  }
  public void shift(int amount) {
    if (amount > top - position)
      // Adding amount would put the position above the top.
      position = top;
    else if (position + amount < 0)
      // Adding amount would put the position below zero.
      position = 0;
    else
      // Adding amount puts position in the range [0...top].
      position += amount;
  }
  public void shutoff() {
    position = 0;
  }
}
• parameters
• equals()
• clone()
• toString()
• class Object

package edu.colorado.geometry;

public class Location implements Cloneable {
    private double x; // The x coordinate of this Location
    private double y; // The y coordinate of this Location
    public Location(double xInitial, double yInitial) {
        x = xInitial;
        y = yInitial;
    }
    public Location clone() {
        // Clone a Location object.
        Location answer;
        try {
            answer = (Location) super.clone();
        } catch (CloneNotSupportedException e) {
            // This exception should not occur. But if it does, it would probably
            // indicate a programming error that made super.clone unavailable.
            // The most common error would be forgetting the "Implements Cloneable"
            // clause at the start of this class.
            throw new RuntimeException("This class does not implement Cloneable.");
        }
        return answer;
    }
    public static double distance(Location p1, Location p2) {
        double a, b, c_squared;
        // Check whether one of the locations is null.
        if ((p1 == null) || (p2 == null))
            return Double.NaN;
        // Calculate differences in x and y coordinates.
        a = p1.x - p2.x;
        b = p1.y - p2.y;
        // Use Pythagorean Theorem to calculate the square of the distance.
        // between the locations.
        c_squared = a*a + b*b;
        return Math.sqrt(c_squared);
    }
}
public boolean equals(Object obj) {
    if (obj instanceof Location) {
        Location candidate = (Location) obj;
        return (candidate.x == x) && (candidate.y == y);
    } else
        return false;
}

public double getX() { return x; }
public double getY() { return y; }

public static Location midpoint(Location p1, Location p2) {
    double xMid, yMid;
    //Check whether one of the locations is null.
    if ((p1 == null) || (p2 == null))
        return null;
    // Compute the x and y midpoints.
    xMid = (p1.x/2) + (p2.x/2);
    yMid = (p1.y/2) + (p2.y/2);
    // Create a new location and return it.
    Location answer = new Location(xMid, yMid);
    return answer; // returning an object reference
}

public void rotate90() {
    double xNew;
    double yNew;
    // For a 90 degree clockwise rotations, the new x is the original y
    // and the new y is -1 times the original x.
    xNew = y;
    yNew = -x;
    x = xNew;
    y = yNew;
}

public void shift(double xAmount, double yAmount) {
    x += xAmount;
    y += yAmount;
}

public String toString() {
    return "(x=" + x + "  y=" + y + ")";
}
Specifying a Java Method

Preconditions and Postconditions are a method of specifying what a method accomplishes. They communicate what a method accomplishes, without any indication of how the method does its work.

The precondition statement indicates what must be true before the method is called.

The postcondition statement indicates what will be true when the method finishes its work.

The programmer who calls the method is responsible for ensuring that the precondition is valid when the method is called.

/**
* Convert a temperature from Celsius degrees to Fahrenheit degrees.
* @param c a temperature in Celsius degrees
* @precondition c >= -273.16.
* @return the temperature c converted to Fahrenheit
* @throws java.lang.IllegalArgumentException Indicates that c is less than the smallest Celsius temperature (-273.16).
**/
public static double celsiusToFahrenheit(double c) { … }

See Appendix H on Javadoc.
The TemperatureConversion Java application prints a table
converting Celsius to Fahrenheit degrees.

public class TemperatureConversion {
  // The main method prints a Celsius to Fahrenheit conversion table.
  // The bounds of the table range from -50C to +50C in 10 degree increments.
  public static void main(String[] args) {
    final double TABLE_BEGIN = -50.0; // The table's first Celsius temperature
    final double TABLE_END = 50.0; // The table's final Celsius temperature
    final double TABLE_STEP = 10.0; // Increment between temps in table
    double celsius; // A Celsius temperature
    double fahrenheit; // The equivalent Fahrenheit temperature
    System.out.println("TEMPERATURE CONVERSION");
    System.out.println("----------------------");
    System.out.println("Celsius     Fahrenheit");
    for (celsius = TABLE_BEGIN; celsius <= TABLE_END; celsius += TABLE_STEP) {
      fahrenheit = celsiusToFahrenheit(celsius);
      System.out.printf("%6.2fC", celsius);
      System.out.printf("%14.2fF\n", fahrenheit);
    }
    System.out.println("----------------------");
  }
  /**
   * Convert a temperature from Celsius degrees to Fahrenheit degrees.
   * @param c
   *   a temperature in Celsius degrees
   * @precondition
   *   c >= -273.16.
   * @return
   *   the temperature c converted to Fahrenheit
   * @throws java.lang.IllegalArgumentException
   *   Indicates that c is less than the smallest Celsius temperature (-273.16).
   **/
  public static double celsiusToFahrenheit(double c) {
    final double MINIMUM_CELCIUS = -273.16;
    if (c < MINIMUM_CELCIUS)
      throw new IllegalArgumentException("Argument " + c + " is too small.");
    return (9.0/5.0)*c + 32;
  }
}
**Assert statements.**

Assert statements document conditions that the programmer intends to be valid at particular locations in the program.

```java
assert (size > 0 : "size is not greater than 0.");
```

Assertions are not normally checked during execution.

Assertion checking can be turned on using the `–enableAssertions` (or `–ea`) option:

```bash
java –ea foo
```

Should you assert preconditions? Assertions can be turned off. Throw an exception if the precondition is not satisfied.

**Static Checking Tools**

Static checking tools analyze the source code (i.e., without executing the program).

FindBugs: http://findbugs.sourceforge.net/
public class AssertExamples {
    // The main method demonstrates calls to maxOf3 and maxOfArray.
    public static void main(String[] args) {
        int[] array = { 10, 20, 30, -40 };  
        System.out.println("Max of 3 (should be 22): "+ maxOf3(10, 2, 22));  
        System.out.println("Max of array (should be 30): " + maxOfArray(array));  
    }
    // This private method returns true if the specified value appears somewhere
    // in the array a; otherwise the return value is false.
    static boolean contains(int a[], int value) {
        for (int item : a) {
            if (item == value)
                return true;
        }
        return false;
    }
    // This private method returns true if the specified value is greater than
    // or equal to every element in the array a; otherwise the return value is false.
    static boolean greaterOrEqual(int a[], int value) {
        for (int item : a) {
            if (item > value)
                return false;
        }
        return true;
    }
    // Returns the largest of three int values. The return value is the largest of the three arguments
    // a, b and c.
    * @example
    * maxOf3(2, -8, 1); // Returns 2
    ***/
    public static int maxOf3(final int a, final int b, final int c) {
        int answer;
        answer = a;
        // Change answer to b if b is bigger:
        if (b > answer) answer = b;
        // Change answer to c if b is bigger:
        if (c > answer) answer = c;
        assert (answer == a) || (answer == b) || (answer == c)
            : "maxOf3 answer is not equal to one of the arguments";
        assert (answer >= a) && (answer >= b) && (answer >= c)
            : "maxOf3 answer is not equal to the largest argument";
        return answer;
    }
}
/**
 * Returns the largest value in an array
 * @param a
 *   an array of int values
 * @precondition
 *   a.length > 0
 * @throws java.lang.ArrayIndexOutOfBoundsException
 *   Indicates that a.length <= 0.
 * @return
 *   The return value is the largest value in a.
 **/

public static int maxOfArray(final int a[]) {
    int answer; // The answer will be stored here.
    int i;      // Array index
    answer = a[0];
    for (i = 1; i < a.length; i++) {
        if (a[i] > answer) {
            answer = a[i];
        }
    }
    assert contains(a, answer)
        : "maxOfArray answer is not in the array";
    assert greaterOrEqual(a, answer)
        : "maxOfArray answer is less than an element of the array";
    return answer;
}