Assert use cases

Here are some common examples of using the `assert` keyword:

- pre-conditions (in private methods only) - the requirements which a method requires its caller to fulfill
- post-conditions - verify the promises made by a method to its caller
- class invariants - validate object state
- unreachable-at-runtime code - parts of your program which you expect to be unreachable, but which cannot be verified as such at compile-time (often `else` clauses and default cases in `switch` statements)

Example

```
import java.util.Random;

public final class Flower {

  public static void main(String... arguments) {
    final Flower tulip = new Flower("Tulip", 1);
    tulip.grow();
    tulip.grow();
    System.out.println(tulip);

    tulip.randomGrowOrWither();
    System.out.println(tulip);

    tulip.wither();
    tulip.wither();
    System.out.println(tulip);
  }

  /**
  * @param aSpecies must have content.
  * @param aInitialLength must be greater than 0.
  */
  public Flower(final String aSpecies, final int aInitialLength)
    //assert is NOT used to validate params of public methods
    if (!isValidSpecies(aSpecies)) {
      throw new IllegalArgumentException("Species must have conte
    }
    if (!isValidLength(aInitialLength)) {
      throw new IllegalArgumentException("Initial length must be
    }

    fSpecies = aSpecies;
    fLength = aInitialLength;

    //check the class invariant
    assert hasValidState(): "Construction failed - not valid stat
  }
```

```
public boolean isMature() {
  return fLength > 5 ;
  //not necessary to assert valid state here, since
  //the state has not changed.
}

/**
* Increase the length by at least one unit.
*/
public void grow(){
  //this style of checking post-conditions is NOT recommended,
  //since the copy of fLength is always made, even when
  //assertions are disabled.
  //See <tt>wither</tt> (below) for an example with an improved
  final int oldLength = fLength;
  fLength += getLengthIncrease(fLength);
  //post-condition: length has increased
  assert fLength > oldLength;

  //check the class invariant
  assert hasValidState(): this;
}

/**
* Decrease the length by one unit, but only if the resulting le
* will still be greater than 0.
*/
public void wither(){

  //this local class exists only to take a snapshot of the curr
  //although bulky, this style allows post-conditions of arbitr
  class OriginalState {
    OriginalState() {
      fOriginalLength = fLength;
    }
    int getLength() {
      return fOriginalLength;
    }
    private final int fOriginalLength;
  }
  OriginalState originalState = null;
  //construct object inside an assertion, in order to ensure th
  //no construction takes place when assertions are disabled.
  //this assert is rather unusual in that it will always succee
  //it has side-effects - it creates an object and sets a refer
  assert (originalState = new OriginalState()) != null;

  if (fLength > 1) {
    --fLength;
  }

  //post-condition: length has decreased by one or has remained
  assert fLength <= originalState.getLength();
```

```
  //check the class invariant
  assert hasValidState(): this;
}

/**
* Randomly select one of three actions
* <ul>
* <li>do nothing
* <li>grow
* <li>wither
* </ul>
*/
public void randomGrowOrWither() {
  //(magic numbers are used here instead of symbolic constants
  //to slightly clarify the example)
  Random generator = new Random();
  int action = generator.nextInt(3);
  //according to the documentation for the Random class, action
  //should take one of the values 0,1,2.
  if (action == 0) {
    //do nothing
  }
  else if (action == 1) {
    grow();
  }
  else if (action == 2) {
    wither();
  }
  else {
    //this is still executed if assertions are disabled
    throw new AssertionError("Unexpected value for action: " +
  }
  //check the class invariant
  assert hasValidState(): this;
}

/** Use for debugging only.  */
public String toString(){
  final StringBuilder result = new StringBuilder();
  result.append(this.getClass().getName());
  result.append(": Species=");
  result.append(fSpecies);
  result.append(" Length=");
  result.append(fLength);
  return result.toString();
}

// PRIVATE
private final String fSpecies;
private int fLength;

/**
* Implements the class invariant.
```

```
  *
  * Perform all checks on the state of the object.
  * One may assert that this method returns true at the end
  * of every public method.
  */
  private boolean hasValidState(){
    return isValidSpecies(fSpecies) && isValidLength(fLength);
  }

  /** Species must have content.  */
  private boolean isValidSpecies(final String aSpecies) {
    return aSpecies != null && aSpecies.trim().length()>0;
  }

  /** Length must be greater than 0.  */
  private boolean isValidLength(final int aLength) {
    return aLength > 0;
  }

  /** Length increase depends on current length. */
  private int getLengthIncrease(int aOriginalLength) {
    //since this is a private method, an assertion
    //may be used to validate the argument
    assert aOriginalLength > 0: this;
    int result = 0;
    if (aOriginalLength > 10) {
      result = 2;
    }
    else {
      result = 1;
    }
    assert result > 0 : result;
    return result;
  }
}
```

See Also :

Validate state with class invariants
Design by Contract