Canonical Forms, Unit Testing

CS151

Chris Pollett

Oct. 12, 2005.

Outline

- Canonical Forms for Classes
- Unit Testing
- JUnit

Canonical Forms for Classes

- Classes designed for general use should provide the following elements:
  - A public no-arg constructor.
  - Overriden equals() and hashCode() methods
  - Overriden toString() methods
  - Overriden clone() method. (Need to implement Cloneable)
  - Overriden readObject() and writeObject() if instances of the class might need to saved or written across a network.(Implement java.io.Serializable to do this)
- Classes which provide the above are said to be of the **canonical form** for public classes.

No-Argument Constructor

- One useful feature of Java is the ability to dynamically at runtime load a class with a line like:

  Class t = Class.forName(someString);

- Then create instances of it at runtime with a line like

  t.newInstance();

- For this to work, we need a no-argument constructor. So that's why we have it for our canonical form.

Object Equality

- The equals() method defines the equality of object states on a per-class basis.
- The default implementation in Object only tests the equality of Objects. So o1.equals(o2) are equal iff o1 and o2 refer to the same object.
- The contract of the equals() method is that all implementations must satisfy the following conditions:
  - Reflexivity: x.equals(x) is always true.
  - Symmetry: x.equals(y) iff y.equals(x)
  - Transitivety: x.equals(y) and y.equals(z) iff x.equals(z)
  - Consistency: repeated calls to x.equals(y) should always return the same answer if the states of x and y are unchanged.
  - Nonnullity: x.equals(null) should always be false.

Template for equals()

```
public boolean equals(Object other)
{
      if(this==other) return true;
      if(other instanceof C)
      {
          C otherObj=(C)other;
          /*compare each field and return false if not equal*/

          return true;
      }
      return false;
}
```

To check fields:

if p is of primitive type:

```
if(p !=otherObj.p) return false;
```

if r is of reference type:

```
if(r == null ? otherObj.r != null : !r.equals(otherObj.r)) return false;
```

Hash Code of Objects

- The hashCode() method, which return an int, is used by collection classes that implement hash tables such as HashMap and HashSet,
- Overriding the equals() method requires also overriding hashCode().
- This is because the contract of hashCode() requires if two objects are equals() they must have the same hash code.
- The general way to create a hash code is to create a hash code for each significant field. (That is, for each field which is checked by the equals() method).
- To implement hashCode(), for each field, if it is of primitive type one can convert it to an int; otherwise, if it is of reference type and nonnull, one can call its hashCode() method.
- After we have made the hash codes of each of these fields we combine them by either adding (hash = hash << n | c) or OR ing (hash = hash * p + c) them together to get a final integer code.

Cloning Objects

- The clone() method returns a copy of the object itself. It is similar to a C++ copy constructor.
- The contract of the clone() method is:
  - The cloned object is not the same object as the original object. i.e., o.clone() != o.
  - The cloned object and the original object are of the same class.
  - The cloned object must equal the original object. That is, o.clone().equals(o);
- The Object implementation of clone:
  - throws a CloneNotSupportedException if the class does not implement Cloneable
  - creates a shallow copy of the original object
- A *shallow copy* means that the values of each field are copied from then original object to the copy. i.e., reference fields are not themselves cloned.
- If we implement clone() by recursively cloning reference fields when doing the copying we get a so-called *deep copy*.

Using Clones in Assertions

- Using the clone() method, we can assert postconditions involving object in the prestate:

```
/**
    @post  x.stuff() == x@pre.stuff()+1
 */
 void myMethod()
{
    X xpre = x.clone();
    //do some code
    assert x.stuff() == xpre.stuff()+1;
}
```

String Representation of Objects

- The result of the toString() method should include all fields of the object:
- For example,

```
public String toString()
{
    StringBuffer s = new StringBuffer();
    int i =0;
    for(Node node=head; node !=null; node=node.next, i++)
    {
        s.append("[" + i + "] =" +node.element + "\n");
    }
    return s.toString();
}
```

Serialization

- *Serialization* is the process of transforming an object to a stream of bytes.
- *Deserialization* is the reverse process.
- Objects of classes that implement java.io.Serializable can be serialized and deserialized.
- We will talk more about how to code the readObject() and writeObject() methods of this interface later in the semester.

Unit Testing

- The testing of software systems is split into phases:
  - Unit Testing: Test each component independently before the units are integrated into the whole system.
  - Integration and System Testing: Test the system as a whole.
  - Acceptance testing: Validate that the system functions and performs as the customer or end-user expects.

Simple Unit Testing

- One can write a simple Test class with a main() method that creates instances of the unit to be tested.
- The test performs some operations on these instances and compares these values against the expected output.

JUnit

- When dealing with a large and extensive set of cases the approach described in the previous slide is too clumsy.
- It is useful to have a unit testing tool.
- One such tool is JUnit which is available from: http://www.junit.org.

A Typical JUnit test program

```
public class MyTest extends TestCase
{
    public MyTest(String name)
    {
        super(name);
    }
    public void testCase_1(){/*test and compare results*/}
    …
    public void testCase_1(){/*test and compare results*/}
    public static Test suite(){ return new
       TestSuite(MyTest.class);
}
```

Compiling and running a JUnit test program

(1) add the path to JUnit.jar file to your CLASSPATH

(2) javac -source 1.5 MyTest.java

Then do one of:

(1) java -ea junit.textui.TestRunner MyTest (text version)

(2) java -ea junit. swingui.TestRunner MyTest (GUI version)