SampleTest

```
/*
 * Copyright 2005 The Apache Software Foundation.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */


package org.apache.jdo.tck.api.persistencemanager;

import java.util.Collection;
import java.util.Iterator;

import javax.jdo.Query;
import javax.jdo.Transaction;

import org.apache.jdo.tck.pc.mylib.PCPoint;
import org.apache.jdo.tck.pc.mylib.PCRect;
import org.apache.jdo.tck.util.BatchTestRunner;

/**
 *<B>Title:</B> Only one instance of persistent object in cache per
 PersistenceManager
 *<BR>
 *<B>Keywords:</B> cache
 *<BR>
 *<B>Assertion ID:</B> A5.4-10.
 *<BR>
 *<B>Assertion Description: </B>
JDO implementations must manage the cache of JDO instances such that there is
only one JDO instance, associated with each <code>PersistenceManager</code>
representing the persistent state of each corresponding data store object.
 */

public class OneInstanceOfObjectPerPersistenceManager extends
        PersistenceManagerTest {

    /** */
    private static final String ASSERTION_FAILED =
        "Assertion A5.4-10 (OneInstanceOfObjectPerPersistenceManager) " +
                "failed: ";

    /**
     * The <code>main</code> is called when the class
     * is directly executed from the command line.
     * @param args The arguments passed to the program.
     */
    public static void main(String[] args) {
        BatchTestRunner.run(OneInstanceOfObjectPerPersistenceManager.class);
    }

    /**
         * This test creates objects in one transaction and commits.
         * The important object is p1.
         * Then, in a second transaction, it gets an object p1a by id,
         * gets another object p1b by navigation, and a third object p1c by
         * query. All of these represent the same datastore object and
         * therefore must be identical in the same PersistenceManager.
     */
    public void test() {
        /** The getPM method is declared in a superclass.
         * This is the standard way to get a PersistenceManager.
         * The method automatically gets a PersistenceManagerFactory,
         * gets a PersistenceManager, and puts the PersistenceManager into
         * the field pm.
         */
        getPM();
        /** This is the standard way to get a Transaction.
         */
        Transaction tx = pm.currentTransaction();

        /** Any values for these flags should be set before
         * beginning a transaction.
         */
        tx.setRetainValues(false);
        tx.setRestoreValues(false);

        /** This is the standard way to begin a transaction.
         */
        tx.begin();
        /** Create new objects to be persisted.
         */
        PCPoint p1 = new PCPoint(10, 20);
        PCPoint p2 = new PCPoint(20, 40);
        PCRect rect = new PCRect(0, p1, p2);
        /** This test relies on persistence by reachability.
         */
        pm.makePersistent(rect);
        /** This is the standard way to commit a transaction.
         */
        tx.commit();

        /** Begin a new transaction so that the navigation
         * uses the object id to load the target object into the cache.
         * The RetainValues flag false guarantees that the object fields
         * are no longer loaded.
         */
        tx.begin();
        Object p1Id = pm.getObjectId(p1);
        /** Retrieves the field values from the datastore.
         */
        PCPoint p1a = (PCPoint)pm.getObjectById(p1Id, true);
        /** Navigate to the point.
         */
        PCPoint p1b = rect.getUpperLeft();
        /** Query for the point by its values in the datastore.
         */
        PCPoint p1c = findPoint(10, 20);
        tx.commit();
        tx = null;

        /** Use a StringBuffer to collect results.
         */
        StringBuffer results = new StringBuffer();

        /** Compare the original object with the object obtained
         * by getObjectById.
         */
        if (p1 != p1a) {
            results.append("getObjectById results differ. ");
        }

        /** Compare the original object with the object obtained
         * by navigating from another object.
         */
        if (p1 != p1b) {
            results.append("navigation results differ. ");
        }
        /** Compare the original object with the object obtained
```

```
         * by query.
         */
        if (p1 != p1c) {
            results.append("query results differ. ");
        }
        if (results.length() != 0) {
            fail(ASSERTION_FAILED + results.toString());
        }

    /** The standard way to end each test method is to simply return.
     * Exceptions are caught by JUnit.
     * The tearDown method ends the transaction and closes
     * the PersistenceManager.
     */
    }

    /** */
    private PCPoint findPoint (int x, int y) {
        Query q = getPM().newQuery (PCPoint.class);
        q.declareParameters ("int px, int py");
        q.setFilter ("x == px & y == py");
        Collection results = (Collection)q.execute (new Integer(x),
                        new Integer(y));
        Iterator it = results.iterator();
        PCPoint ret = (PCPoint)it.next();
        return ret;
    }

}
```