```
import java.io.File;

import org.junit.Before;
import org.junit.Test;

import junit.framework.TestCase;

public class DatabaseTest extends TestCase {
      private static IDatabase database = Database.getDatabase();
      String empty, expectedOutput, stringExpectedOutput = null;
      ArrayFormat value, stringArray = null;
      ObjectFormat obj, intObject = null;
      ICursor intCursor, stringCursor, doubleCursor, arrayCursor,
objectCursor = null;
      ITransaction firstTransaction = null;
      ITransaction secondTransaction = null;

      @Before
      public void setUp() {
            value = new ArrayFormat();
            stringArray = new ArrayFormat();
            obj = new ObjectFormat();
            intObject = new ObjectFormat();

      value.put(12).put("cs696").put(1678.90).put(intObject);

      stringArray.put("grapes").put("strawberry").put("pine");
            intObject.put("first", 1234).put("second",
9876).put("third", -986.12).put("fourth", "Integer");
            obj.put("integer", 678).put("string",
"summer").put("double", 189.789).put("array", stringArray);
            database.put("updated", "no").put("cs645", 100);
            database.put("Cursor Int", 777);
            database.put("Cursor String", "Assignment3");
            database.put("Cursor Array", value);
            database.put("Cursor Object", obj);
            database.put("Cursor Double", -34.4934);
            intCursor = database.getCursor("Cursor Int");
            stringCursor = database.getCursor("Cursor String");
            arrayCursor = database.getCursor("Cursor Array");
            objectCursor = database.getCursor("Cursor Object");
            doubleCursor = database.getCursor("Cursor Double");
            database.put("object value", obj).put("deleted",
"no").put("cs545", 900).put("delete array", value);
            expectedOutput = "[Key Cursor Double : Value 89.989]";
            stringExpectedOutput = "[Key Cursor String : Value
Done]";
            firstTransaction = database.transaction();
            secondTransaction = database.transaction();
      }

      @Test
```

```
     public void testDatabase() throws DataTypeMisMatchException
{
          /** database Put() and Get() test **/
          database.put("array", value).put("object",
obj).put("int", 19800).put("double", -34.567);
          database.put("string", "Oops
Assignment").put("getter", 190);
          try {
               database.put("empty value", empty);
               fail("Expected an IllegalArgumentException to be
thrown");
          } catch (IllegalArgumentException
anIllegalArgumentException) {

     assertEquals(anIllegalArgumentException.getMessage(), "Value
cannot be null");
          }
          assertEquals(value.toString(),
database.getArray("array").toString());
          assertEquals(obj.toString(),
database.getObject("object").toString());
          assertEquals(19800, database.getInt("int"));
          assertEquals("Oops Assignment",
database.getString("string"));
          assertEquals(-34.567, database.getDouble("double"));
          try {
               database.getString("int");
               fail("Expected an DataTypeMisMatchexception to be
thrown");
          } catch (DataTypeMisMatchException
anDataTypeMisMatchException) {

     assertEquals(anDataTypeMisMatchException.getMessage(),
"value is not of string type");
          }
          assertEquals(190, database.get("getter"));
          try {
               database.get("illegal");
               fail("Expected an IllegalArgumentException to be
thrown");
          } catch (IllegalArgumentException
anIllegalArgumentException) {

     assertEquals(anIllegalArgumentException.getMessage(), "Key
not found");
          }

          /** database modify test **/
          assertTrue(database.modify("updated", "no"));
          assertTrue(database.modify("cs645", 89));
          assertTrue(database.modify("object value", value));
          try {
```

```
                database.modify("updated", empty);
                fail("Expected an IllegalArgumentException to be
thrown");
          } catch (IllegalArgumentException
anIllegalArgumentException) {

     assertEquals(anIllegalArgumentException.getMessage(), "Value
cannot be null");
          }
          try {
                database.modify("illegal", 789);
                fail("Expected an IllegalArgumentException to be
thrown");
          } catch (IllegalArgumentException
anIllegalArgumentException) {

     assertEquals(anIllegalArgumentException.getMessage(), "Key
not found");
          }

          /** database delete test **/
          assertEquals("no", database.delete("deleted"));
          assertEquals(900, database.delete("cs545"));
          assertEquals(value.toString(), database.delete("delete
array").toString());
          assertNull(database.delete("empty"));

          /** database Undo test **/
          database.put("add undo", 199);
          database.undo();
          try {
                database.modify("add undo", 789);
                fail("Expected an IllegalArgumentException to be
thrown");
          } catch (IllegalArgumentException
anIllegalArgumentException) {

     assertEquals(anIllegalArgumentException.getMessage(), "Key
not found");
          }
     }

     @Test
     public void testCursor() throws DataTypeMisMatchException {
          assertEquals(777, intCursor.value());
          assertEquals("Assignment3", stringCursor.value());
          database.put("Cursor Int", 999);
          database.put("Cursor String", "Next Assignment4");
          assertEquals(999, intCursor.value());
          assertEquals("Next Assignment4",
stringCursor.value());
          assertEquals(999, intCursor.getInt());
```

```
          assertEquals(obj.toString(),
objectCursor.getObject().toString());
          try {
               intCursor.getString();
               fail("Expected an DataTypeMisMatchexception to be
thrown");
          } catch (DataTypeMisMatchException
anDataTypeMisMatchException) {

     assertEquals(anDataTypeMisMatchException.getMessage(),
"value is not of string type");
          }
          assertEquals("Next Assignment4",
stringCursor.getString());
          try {
               stringCursor.getInt();
               fail("Expected an DataTypeMisMatchexception to be
thrown");
          } catch (DataTypeMisMatchException
anDataTypeMisMatchException) {

     assertEquals(anDataTypeMisMatchException.getMessage(),
"value is not of int type");
          }
          try {
               objectCursor.getArray();
               fail("Expected an DataTypeMisMatchexception to be
thrown");
          } catch (DataTypeMisMatchException
anDataTypeMisMatchException) {

     assertEquals(anDataTypeMisMatchException.getMessage(),
"value is not of array type");
          }
          assertEquals(value.toString(),
arrayCursor.getArray().toString());
          FooObserver foo = new FooObserver();
          doubleCursor.addObserver(foo);
          database.put("Cursor Double", 89.9890);
          assertEquals(foo.toString(), expectedOutput);
          doubleCursor.removeObserver(foo);
          database.put("Cursor Double", 99.9999);
          assertEquals(foo.toString(), expectedOutput);
          assertEquals(99.9999, doubleCursor.getDouble());
          stringCursor.addObserver(foo);
          database.put("Cursor String", "Done");
          assertEquals(foo.toString(), stringExpectedOutput);
          stringCursor.removeObserver(foo);
          database.put("Cursor String", "Litchi");
          assertEquals(foo.toString(), stringExpectedOutput);
     }
```

```
      @Test
      public void testTransaction() throws
DataTypeMisMatchException, TransactionNotValidException {
            secondTransaction.put("history",
"gandhi").put("algebra", -980.87).put("kannada", stringArray)
                        .put("geography", intObject).put("math",
123);
            firstTransaction.put("fruit", "litchi").put("cost",
1234).put("weight", 90.8765).put("fruit names", stringArray)
                        .put("cost object", intObject);
            assertTrue(firstTransaction.isActive());
            assertTrue(firstTransaction.commit());
            try {
                  firstTransaction.isActive();
                  fail("Expected an TransactionNotValidexception to
be thrown");
            } catch (TransactionNotValidException
anTransactionNotValidException) {

      assertEquals(anTransactionNotValidException.getMessage(),
"transaction is not valid");
            }
            assertTrue("second transaction is still active",
secondTransaction.isActive());
            assertEquals(intObject.toString(),
secondTransaction.getObject("geography").toString());
            try {
                  secondTransaction.getString("algebra");
                  fail("Expected an DataTypeMisMatchexception to be
thrown");
            } catch (DataTypeMisMatchException
anDataTypeMisMatchException) {

      assertEquals(anDataTypeMisMatchException.getMessage(),
"value is not of string type");
            }
            assertEquals("gandhi",
secondTransaction.getString("history"));
            try {
                  secondTransaction.getInt("kannada");
                  fail("Expected an DataTypeMisMatchexception to be
thrown");
            } catch (DataTypeMisMatchException
anDataTypeMisMatchException) {

      assertEquals(anDataTypeMisMatchException.getMessage(),
"value is not of int type");
            }
            try {
                  secondTransaction.getArray("history");
                  fail("Expected an DataTypeMisMatchexception to be
thrown");
```

```
          } catch (DataTypeMisMatchException
anDataTypeMisMatchException) {

     assertEquals(anDataTypeMisMatchException.getMessage(),
"value is not of array type");
          }
          assertEquals(stringArray.toString(),
secondTransaction.getArray("kannada").toString());
          try {
               secondTransaction.get("illegal");
               fail("Expected an IllegalArgumentException to be
thrown");
          } catch (IllegalArgumentException
anIllegalArgumentException) {

     assertEquals(anIllegalArgumentException.getMessage(), "Key
not found");
          }

          assertTrue(secondTransaction.modify("history",
"lincoln"));
          assertTrue(secondTransaction.modify("kannada", 890));

     assertTrue(secondTransaction.modify("algebra", -812.35));
          try {
               secondTransaction.modify("history", empty);
               fail("Expected an IllegalArgumentException to be
thrown");
          } catch (IllegalArgumentException
anIllegalArgumentException) {

     assertEquals(anIllegalArgumentException.getMessage(), "Value
cannot be null");
          }
          try {
               secondTransaction.modify("illegal", 789);
               fail("Expected an IllegalArgumentException to be
thrown");
          } catch (IllegalArgumentException
anIllegalArgumentException) {

     assertEquals(anIllegalArgumentException.getMessage(), "Key
not found");
          }
          assertEquals(123, secondTransaction.delete("math"));
          assertEquals(intObject.toString(),
secondTransaction.delete("geography").toString());
          assertNull(secondTransaction.delete("empty"));
          assertTrue(secondTransaction.abort());

     }
```

```
      @Test
      public void testRecoverAndSnapShot() throws Exception {
            database.snapShot(new File("testCommands.txt"), new
File("testSnapshot.txt"));
            /**
             * tested having testcommands.txt 5 add and one delete
commands
             *
**************************************************************
****
             * add@updated@yes@String ||| add@cs645@111
@Integeradd@Cursor
             *
Array@[12,"cs696",1678.9,{"map":{"third":-986.12,"second":9876,
             * "fourth":"Integer","first":1234}}]@Array |||
add@Cursor
             *
Object@{"string":"summer","integer":678,"double":189.789,"array":
{
             * "arrayData":["grapes","strawberry","pine"]}}@Object
||||| add@Cursor
             * Double@-34.4934@Double |||| delete@getter@190
@Integer
             **/

            database.recover(new File("testCommands.txt"), new
File("testSnapshot.txt"));
            assertEquals("yes", database.getString("updated"));
            assertEquals(111, database.getInt("cs645"));
            assertEquals(-34.4934, database.getDouble("Cursor
Double"));
            assertEquals(
                        "[12,\"cs696\",1678.9,{\"map\":{\"third
\":-986.12,\"second\":9876,\"fourth\":\"Integer\",\"first
\":1234}}]",
                        database.getArray("Cursor
Array").toString());
            assertEquals(
                        "{\"integer\":678,\"string\":\"summer\",
\"double\":189.789,\"array\":{\"arrayData\":[\"grapes\",
\"strawberry\",\"pine\"]}}",
                        database.getObject("Cursor
Object").toString());
            try {
                  database.get("getter");
                  fail("Expected an IllegalArgumentException to be
thrown");
            } catch (IllegalArgumentException
anIllegalArgumentException) {

      assertEquals(anIllegalArgumentException.getMessage(), "Key
not found");
```

```
                }
            }

        }
```