

# **Persistence Testing with Files**

Roberto A. Bittencourt

# Persistence Testing

# Persistence Testing

---

- ▶ Adding persistence to a system complicates testing because files/databases end up saving state that can disturb the test cases that come after another test case
- ▶ A simple approach is the following:
  1. Start your testing with no files/database (or an empty file/database)
  2. Set up the test fixture to create the class that contains the DAO
  3. Exercise the test case that may save into file/database
  4. Tear down the test fixture by deleting/emptying the file/database
  5. The unit test framework will repeat steps 2, 3 and 4 for each test method, which will now be consistent

# Recalling how the DAO pattern affects testing

---

- ▶ Our testing code will handle two different issues:
  - ▶ Testing whether the collection is working correctly
  - ▶ Testing whether the persistence is also working correctly
- ▶ We deal with them separately by using a system property called **autosave**
  - ▶ By default, autosave is off: your tests check collections only
  - ▶ If you turn **autosave** on: tests check both collections and persistence

# Recalling the order of your testing process

---

- ▶ When you run a SchoolTest test case:
  - ▶ First, in setUp(), it constructs a School object
  - ▶ Then, the School's constructor creates the StudentDAO object
  - ▶ The StudentDAO's constructor recovers information about the autosave property and file directories and names
    - ▶ If autosave is true, it reads the files to initialize the StudentDAO with the saved collection of students
  - ▶ Then, SchoolTest exercises school operations and assertions
  - ▶ Finally, in tearDown(), SchoolTest cleans up the used files so that the next test case starts from a clean environment

# Recall we created a Configuration class to hold information about persistence...

---

```
class Configuration:
```

```
    # Note the lack of a constructor
```

```
    # Also note below the class variables ("static")
```

```
    # autosave is initially false, but integration tests
```

```
    # may change that in the setUp()
```

```
    autosave = False
```

```
    filename = 'students.dat'
```

# PersistenceSchoolTest (1 of 6)

## (setUp and tearDown)

---

```
import os
from unittest import TestCase
from unittest import main
from student import Student
from school import School

class SchoolTest(TestCase):

    def setUp(self):
        # set autosave to True to test persistence
        conf = Configuration()
        self.conf.__class__.autosave = True
        self.filename = conf.__class__.filename
        self.school = School('Central High')
        self.school.student_dao.filename = 'students.dat'

    def tearDown(self):
        if os.path.exists(self.school.student_dao.filename):
            os.remove(self.filename)
```

# PersistenceSchoolTest (2 of 6)

---

```
# continuing SchoolTest (2)
def test_create_search(self):
    student1 = Student('Peter', 123, 2010)
    student1a = Student('Peter', 123, 2010)
    student2 = Student('Ali', 234, 2011)
    student3 = Student('Kala', 345, 2009)

    # add new student
    self.assertTrue(self.school.create('Peter', 123, 2010), "creating student1
should return success")
    self.assertIsNotNone(self.school.search(123), "student1 is registered at
school")
    self.assertEqual(student1a, self.school.search(123),
        "registered student1 data should be the same as student1a")

    # add more students
    self.assertTrue(self.school.create('Ali', 234, 2011), "creating student2
should return success")
    self.assertTrue(self.school.create('Kala', 345, 2009), "creating student3
should return success")
    self.assertEqual(student1, self.school.search(123), "student1 remains
registered")
    self.assertEqual(student2, self.school.search(234), "student2 is
registered")
    self.assertEqual(student3, self.school.search(345), "student3 is
registered")
```

---

# PersistenceSchoolTest (3 of 6)

---

```
# continuing SchoolTest (3)
def test_retrieve(self):
    student1 = Student('Peter Jackson', 123, 2010)
    student2 = Student('Peter Parker', 234, 2011)
    student3 = Student('Kala', 345, 2009)

    self.assertEqual(0, len(self.school.retrieve('Peter')), "empty student
collection")

    # add some students
    self.school.create('Peter Jackson', 123, 2010)
    self.school.create('Peter Parker', 234, 2011)
    self.school.create('Kala', 345, 2009)

    # Retrieve a singleton list
    retrieved = self.school.retrieve('Kala')
    self.assertEqual(1, len(retrieved),
        "there should be 1 student named Kala")
    self.assertEqual(student3, retrieved[0], "Retrieving Kala")

    # Retrieve a list with more than one element
    retrieved = self.school.retrieve('Peter')
    self.assertEqual(2, len(retrieved),
        "there should be 2 students named Peter")
    self.assertEqual(student1, retrieved[0], "Retrieving Peter Jackson")
    self.assertEqual(student2, retrieved[1], "Retrieving Peter Parker")
```

# PersistenceSchoolTest (4 of 6)

---

```
# continuing SchoolTest (4)
def test_update(self):
    student1 = Student('Peter', 123, 2010)
    student1a = Student('Peter Jackson', 123, 1995)
    student2 = Student('Ali', 234, 2011)
    student2a = Student('Ali Mesbah', 102, 2011)
    student3 = Student('Kala', 345, 2009)

    self.assertFalse(self.school.update(123, 'Peter Jackson', 123, 1995),
"empty student collection")

    # add some students
    self.school.create('Peter', 123, 2010)
    self.school.create('Ali', 234, 2011)
    self.school.create('Kala', 345, 2009)

    # Do not update an unregistered student
    self.assertFalse(self.school.update(300, 'Latifa', 300, 2001),
"Unregistered student")

    # Update a registered student, maintaining key
    self.assertTrue(self.school.update(123, 'Peter Jackson', 123, 1995),
"Updating Peter, same key")

    # Update a registered student, changing key
    self.assertTrue(self.school.update(234, 'Ali Mesbah', 102, 1998),
"Updating Ali, different key")
```

---

# PersistenceSchoolTest (5 of 6)

---

```
# continuing SchoolTest (5)
def test_delete(self):
    student1 = Student('Peter', 123, 2010)
    student2 = Student('Ali', 234, 2011)
    student3 = Student('Kala', 345, 2009)

    self.assertFalse(self.school.delete(123), "empty student collection")

    # add some students
    self.school.create('Peter', 123, 2010)
    self.school.create('Ali', 234, 2011)
    self.school.create('Kala', 345, 2009)

    # Do not delete an unregistered student
    self.assertFalse(self.school.delete(300), "Unregistered student")

    # Delete a registered student
    self.assertTrue(self.school.delete(123), "Deleting Peter")

    # Do not delete a student that has already been deleted
    self.assertFalse(self.school.delete(123), "Deleting Peter again should not
be possible")

    # Delete another registered student
    self.assertTrue(self.school.delete(234), "Deleting Ali")
```

# PersistenceSchoolTest (6 of 6)

```
# continuing SchoolTest (6)
def test_list_all(self):
    student1 = Student('Peter', 123, 2010)
    student2 = Student('Ali', 234, 2011)
    ----- student3 = Student('Kala', 345, 2009) -----
    student4 = Student('Jin', 567, 2007)
    student5 = Student('Marcos', 789, 2008)

    self.assertEqual(0, len(self.school.list_all()), "empty student collection")

    # add one student
    self.school.create('Peter', 123, 2010)

    # List a singleton list
    listed = self.school.list_all()
    self.assertEqual(1, len(listed), "there should be 1 student in the list")
    self.assertEqual(student1, listed[0], "Listing Peter")

    # add some students
    self.school.create('Ali', 234, 2011)
    self.school.create('Kala', 345, 2009)

    # List with three students
    listed = self.school.list_all()
    self.assertEqual(3, len(listed), "there should be 3 students in the list")
    self.assertEqual(student1, listed[0], "Listing Peter")
    self.assertEqual(student2, listed[1], "Listing Ali")
    self.assertEqual(student3, listed[2], "Listing Kala")

    # delete some students
    self.school.delete(345)
    self.school.delete(123)

    # List is a singleton list again
    listed = self.school.list_all()
    self.assertEqual(1, len(listed), "there should be 1 student in the list")
    self.assertEqual(student2, listed[0], "Listing Ali")

    # add some more students
    self.school.create('Jin', 567, 2007)
    self.school.create('Marcos', 789, 2008)

    # List is back with three students, different ones
    listed = self.school.list_all()
    self.assertEqual(3, len(listed), "there should be 3 students in the list")
    self.assertEqual(student2, listed[0], "Listing Ali")
    self.assertEqual(student4, listed[1], "Listing Jin")
    self.assertEqual(student5, listed[2], "Listing Marcos")

    # delete all students
    self.school.delete(567)
    self.school.delete(234)
    self.school.delete(789)
    -----

    # List is empty again
    self.assertEqual(0, len(self.school.list_all()), "list should be empty")
```

# Stricter Persistence Testing

---

- ▶ To be stricter in your persistence tests, you will also want to test that each method that affects persistence:
  - ▶ Get access to persistence first and load the data
  - ▶ Execute each CRUD operation that persists information
  - ▶ **After each CRUD operation that persists information, reset the persistence mechanism, loading the data again**
- ▶ By following that stricter approach, you may catch persistence bugs inside each test method

# Stricter PersistenceSchoolTest (1 of 6): (adding persistence reset)

---

```
import os
from unittest import TestCase
from unittest import main
from student import Student
from school import School

class SchoolTest(TestCase):

    def setUp(self):
        conf = Configuration()
        self.conf.__class__.autosave = True
        self.filename = conf.__class__.filename
        self.school = School('Central High', True)
        self.school.student_dao.filename = 'students.dat'

    def tearDown(self):
        if os.path.exists(self.school.student_dao.filename):
            os.remove(self.school.student_dao.filename)

    def reset_persistence(self):
        self.school = School('Central High')
```

---

# Stricter PersistenceSchoolTest (2 of 6)

```
def test_create_search(self):
    student1 = Student('Peter', 123, 2010)
    student1a = Student('Peter', 123, 2010)
    student2 = Student('Ali', 234, 2011)
    student3 = Student('Kala', 345, 2009)

    # add new student
    self.assertTrue(self.school.create('Peter', 123, 2010), "creating student1
should return success")
    self.reset_persistence()
    self.assertIsNotNone(self.school.search(123), "student1 is registered at school")
    self.reset_persistence() # Do we really need to reset persistence here?
    self.assertEqual(student1a, self.school.search(123),
        "registered student1 data should be the same as student1a")
    # add more students
    self.assertTrue(self.school.create('Ali', 234, 2011), "creating student2 should
return success")
    self.reset_persistence()
    self.assertTrue(self.school.create('Kala', 345, 2009), "creating student3
should return success")
    self.reset_persistence()
    self.assertEqual(student1, self.school.search(123), "student1 remains
registered")
    self.assertEqual(student2, self.school.search(234), "student2 is registered")
    self.assertEqual(student3, self.school.search(345), "student3 is registered")

    # code continues with the other stricter test methods...
```



# Stricter PersistenceSchoolTest (3 of 6)

```
# continuing SchoolTest (3)
def test_retrieve(self):
    student1 = Student('Peter Jackson', 123, 2010)
    student2 = Student('Peter Parker', 234, 2011)
    student3 = Student('Kala', 345, 2009)

    self.assertEqual(0, len(self.school.retrieve('Peter')), "empty student
collection")

    # add some students
    self.school.create('Peter Jackson', 123, 2010)
    self.school.create('Peter Parker', 234, 2011)
    self.school.create('Kala', 345, 2009)
    self.reset_persistence()

    # Retrieve a singleton list
    retrieved = self.school.retrieve('Kala')
    self.assertEqual(1, len(retrieved),
        "there should be 1 student named Kala")
    self.assertEqual(student3, retrieved[0], "Retrieving Kala")

    # Retrieve a list with more than one element
    retrieved = self.school.retrieve('Peter')
    self.assertEqual(2, len(retrieved),
        "there should be 2 students named Peter")
    self.assertEqual(student1, retrieved[0], "Retrieving Peter Jackson")
    self.assertEqual(student2, retrieved[1], "Retrieving Peter Parker")
```



# Stricter PersistenceSchoolTest (4 of 6)

```
# continuing SchoolTest (4)
```

```
def test_update(self):
```

```
-----student1 = Student('Peter', 123, 2010)-----
    student1a = Student('Peter Jackson', 123, 1995)
    student2 = Student('Ali', 234, 2011)
    student2a = Student('Ali Mesbah', 102, 2011)
    student3 = Student('Kala', 345, 2009)

    self.assertFalse(self.school.update(123, 'Peter Jackson', 123, 1995),
"empty student collection")

    # add some students
    self.school.create('Peter', 123, 2010)
    self.school.create('Ali', 234, 2011)
    self.school.create('Kala', 345, 2009)
    self.reset_persistence()

    # Do not update an unregistered student
    self.assertFalse(self.school.update(300, 'Latifa', 300, 2001),
"Unregistered student")

    # Update a registered student, maintaining key
    self.assertTrue(self.school.update(123, 'Peter Jackson', 123, 1995),
"Updating Peter, same key")
    self.reset_persistence()

    # Update a registered student, changing key
    self.assertTrue(self.school.update(234, 'Ali Mesbah', 102, 1998),
"Updating Ali, different key")
    self.reset_persistence()
-----
```



# Stricter PersistenceSchoolTest (5 of 6)

```
# continuing SchoolTest (5)
def test_delete(self):
    student1 = Student('Peter', 123, 2010)
    student2 = Student('Ali', 234, 2011)
    student3 = Student('Kala', 345, 2009)

    self.assertFalse(self.school.delete(123), "empty student collection")

    # add some students
    self.school.create('Peter', 123, 2010)
    self.school.create('Ali', 234, 2011)
    self.school.create('Kala', 345, 2009)
    self.reset_persistence()

    # Do not delete an unregistered student
    self.assertFalse(self.school.delete(300), "Unregistered student")

    # Delete a registered student
    self.assertTrue(self.school.delete(123), "Deleting Peter")
    self.reset_persistence()

    # Do not delete a student that has already been deleted
    self.assertFalse(self.school.delete(123), "Deleting Peter again should not
be possible")

    # Delete another registered student
    self.assertTrue(self.school.delete(234), "Deleting Ali")
    self.reset_persistence()
```

# Stricter PersistenceSchoolTest (6 of 6)

# continuing SchoolTest (6)

```
def test_list_all(self):
    student1 = Student('Peter', 123, 2010)
    student2 = Student('Ali', 234, 2011)
    student3 = Student('Kala', 345, 2009)
    student4 = Student('Jin', 567, 2007)
    student5 = Student('Marcos', 789, 2008)
    self.assertEqual(0, len(self.school.list_all()), "empty student collection")
    # add one student
    self.school.create('Peter', 123, 2010)
    self.reset_persistence()
    # List a singleton list
    listed = self.school.list_all()
    self.assertEqual(1, len(listed), "there should be 1 student in the list")
    self.assertEqual(student1, listed[0], "Listing Peter")
    # add some students
    self.school.create('Ali', 234, 2011)
    self.school.create('Kala', 345, 2009)
    self.reset_persistence()
    # List with three students
    listed = self.school.list_all()
    self.assertEqual(3, len(listed), "there should be 3 students in the list")
    self.assertEqual(student1, listed[0], "Listing Peter")
    self.assertEqual(student2, listed[1], "Listing Ali")
    self.assertEqual(student3, listed[2], "Listing Kala")
    # delete some students
    self.school.delete(345)
    self.school.delete(123)
    self.reset_persistence()
    # List is a singleton list again
    listed = self.school.list_all()
    self.assertEqual(1, len(listed), "there should be 1 student in the list")
    self.assertEqual(student2, listed[0], "Listing Ali")
    # add some more students
    self.school.create('Jin', 567, 2007)
    self.school.create('Marcos', 789, 2008)
    self.reset_persistence()
    # List is back with three students, different ones
    listed = self.school.list_all()
    self.assertEqual(3, len(listed), "there should be 3 students in the list")
    self.assertEqual(student2, listed[0], "Listing Ali")
    self.assertEqual(student4, listed[1], "Listing Jin")
    self.assertEqual(student5, listed[2], "Listing Marcos")
    # delete all students
    self.school.delete(567)
    self.school.delete(234)
    self.school.delete(789)
    self.reset_persistence()
    # List is empty again
    self.assertEqual(0, len(self.school.list_all()), "list should be empty")
```

# More advanced integration tests

---

- ▶ The solution we found for testing persistence is somehow naïve and a bit expensive
- ▶ There are better ways to test persistence
  - ▶ Work with a simpler DAO object (**mock**) when not testing persistence
  - ▶ When testing persistence, use the real DAO object that persists information (these are the real **integration tests**)
- ▶ A course on Software Testing teaches **mocking** techniques
- ▶ Here we will use the simple naïve approach to persistence testing