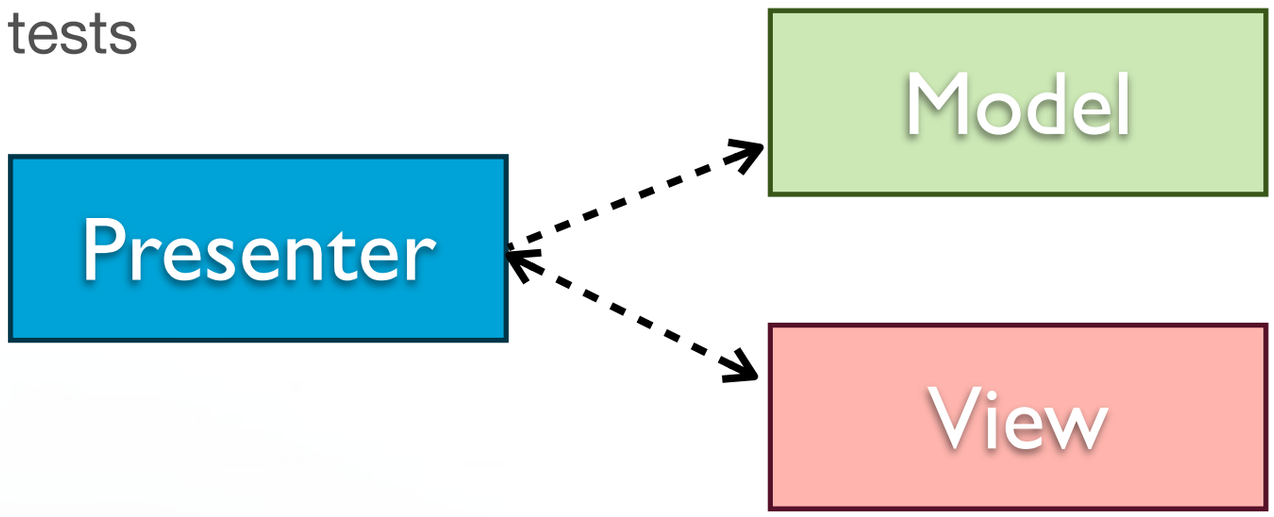
GUI & DB testing

PRAGMATISTS

What's hard about testing GUIs?

Passive View

- View knows NOTHING about the Model
- All interactions are being sent directly to the Presenter
- Presenter takes care of the whole information flow and updates the View
- There’s a bidirectional relationship between the View and the Presenter
- Since the View is just a stupid servant for the presenter, we can skip it in tests



Source : http://martinfowler.com/eaaDev/PassiveScreen.html

Passive View
View

```
public class EmployeesView extends JPanel {
  private EmployeesPresenter employeesPresenter;
  private JTextField firstName = new JTextField();
  private JTextField lastName = new JTextField();

  public EmployeesView() {
    employeesPresenter = new EmployeesPresenter(this);
    JButton addEmployeeButton = new JButton(new AbstractAction("Add Employee") {
      @Override
      public void actionPerformed(ActionEvent event) {
        employeesPresenter.addEmployee();
      }
    });
  }

  public String getFirstName() {
    return firstName.getText();
  }

  public String getLastName() {
    return lastName.getText();
  }

  public void showEmployeeAddedMessage() {
    JOptionPane.showMessageDialog(this, "Employee added");
  }
}
```

Passive View Presenter

```
public class EmployeesPresenter {

  private final EmployeesView view;

  public EmployeesPresenter(EmployeesView view) {
    this.view = view;
  }

  public void addEmployee() {
    persist(view.getFirstName(), view.getLastName());
    view.showEmployeeAddedMessage();
  }

  private void persist(String firstName, String lastName) {
    employeeService.addNew(new Employee(firstName, lastName));
  }
}
```

Passive View

If a Presenter is contacting the View via an interface, we can get indepenent from the technology.

```
public interface EmployeesView {
   String getFirstName();
   String getLastName();
   void showEmployeeAddedMessage();
}

public class SwingEmployeesView extends JPanel implements EmployeesView {

public class EmployeesPresenter {

  private final EmployeesView view;

  public EmployeesPresenter(EmployeesView view) {
    this.view = view;
  }
(...)
```

A bit less passive
Passive View

- The View sends to the Presenter all the values which it can easily get
- It makes both the presenter and the view simpler
- Less communication between the View and the Presenter
- Code coverage by tests is the same
- Downside: View gets dependent on the Presenter’s implementation

A bit less passive Passive View

```
public class EmployeesView extends JPanel {
  private EmployeesPresenter employeesPresenter;
  private JTextField firstName = new JTextField();
  private JTextField lastName = new JTextField();

  public EmployeesView() {
    employeesPresenter = new EmployeesPresenter(this);
    JButton addEmployeeButton = new JButton(new AbstractAction("Add Employee") {
      @Override
      public void actionPerformed(ActionEvent event) {
        employeesPresenter.addEmployee(firstName.getText(), lastName.getText());
      }
    });
  }

  public void showEmployeeAddedMessage() {
    JOptionPane.showMessageDialog(this, "Employee added");
  }
}


public class EmployeesPresenter {

  public void addEmployee(String firstName, String lastName) {
    persist(firstName, lastName);
    view.showEmployeeAddedMessage();
  }
```

Testing the DB

Why is it hard to test the DB?

Assumptions

- Tests should be isolated and independent
- We often test many operations at the same time (save + load)
- We can use DB tests to drive DB layer development
- We can verify the correctness of a data model to DB

Spring TestContext framework

What does it give us?

- Wiring spring beans
- declarative transactions
- spring context caching among test runs
- additional classes which support integration tests

Spring TestContext framework

Test class should extend

`AbstractTransactionalJUnit4SpringContextTests`

```
@ContextConfiguration(locations = { "/spring/annotations.xml", "/spring/test-data-sources.xml" })
@TransactionConfiguration(defaultRollback = true)
@Transactional
public class UserRepositoryTest extends AbstractTransactionalJUnit4SpringContextTests {

  @Autowired
  UserRepository repository;

  @Test
  @Rollback(false)
  public void shouldFetchAllUsers() {
    repository.persist(new UserModel("user 1"));
    repository.persist(new UserModel("user 2"));

    assertEquals(2, repository.fetchAll().size());
  }
}
```

Spring TestContext framework

Configure DB for the test

```
@ContextConfiguration(locations = { "/spring/annotations.xml", "/spring/test-data-sources.xml" })
@TransactionConfiguration(defaultRollback = true)
@Transactional
public class UserRepositoryTest extends AbstractTransactionalJUnit4SpringContextTests {

  @Autowired
  UserRepository repository;

  @Test
  @Rollback(false)
  public void shouldFetchAllUsers() {
    repository.persist(new UserModel("user 1"));
    repository.persist(new UserModel("user 2"));

    assertEquals(2, repository.fetchAll().size());
  }
}
```

Spring TestContext framework

Default rollback for all methods in this class

```
@ContextConfiguration(locations = { "/spring/annotations.xml", "/spring/test-data-sources.xml" })
@TransactionConfiguration(defaultRollback = true)
@Transactional
public class UserRepositoryTest extends AbstractTransactionalJUnit4SpringContextTests {

  @Autowired
  UserRepository repository;

  @Test
  @Rollback(false)
  public void shouldFetchAllUsers() {
    repository.persist(new UserModel("user 1"));
    repository.persist(new UserModel("user 2"));

    assertEquals(2, repository.fetchAll().size());
  }
}
```

Spring TestContext framework

Test is transactional

```
@ContextConfiguration(locations = { "/spring/annotations.xml", "/spring/test-data-sources.xml" })
@TransactionConfiguration(defaultRollback = true)
@Transactional
public class UserRepositoryTest extends AbstractTransactionalJUnit4SpringContextTests {

  @Autowired
  UserRepository repository;

  @Test
  @Rollback(false)
  public void shouldFetchAllUsers() {
    repository.persist(new UserModel("user 1"));
    repository.persist(new UserModel("user 2"));

    assertEquals(2, repository.fetchAll().size());
  }
}
```

Spring TestContext framework

We can also have tests which don't rollback

```
@ContextConfiguration(locations = { "/spring/annotations.xml", "/spring/test-data-sources.xml" })
@TransactionConfiguration(defaultRollback = true)
@Transactional
public class UserRepositoryTest extends AbstractTransactionalJUnit4SpringContextTests {

  @Autowired
  UserRepository repository;

  @Test
  @Rollback(false)
  public void shouldFetchAllUsers() {
    repository.persist(new UserModel("user 1"));
    repository.persist(new UserModel("user 2"));

    assertEquals(2, repository.fetchAll().size());
  }
}
```

Spring TestContext framework

You need to remember to flush and clear the session before verifying
(otherwise we just hit cache...)

```
@ContextConfiguration(locations = { "/spring/annotations.xml", "/spring/test-data-sources.xml" })
@TransactionConfiguration(defaultRollback = true)
@Transactional
public class UserRepositoryTest extends AbstractTransactionalJUnit4SpringContextTests {

  @Autowired
  UserRepository repository;

  @Test
  @Rollback(false)
  public void shouldFetchAllUsers() {
    repository.persist(new UserModel("user 1"));
    repository.persist(new UserModel("user 2"));
    repository.getHibernateTemplate().flush();
    repository.getHibernateTemplate().clear();

    assertEquals(2, repository.fetchAll().size());
  }
}
```

Spring TestContext framework Runner

Instead of inheritance, you can also use a runner

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = { "/spring/annotations.xml", "/spring/test-data-sources.xml" })
@TransactionConfiguration(defaultRollback = true)
@Transactional
public class UserRepositoryTest {

(...)

}
```

Spring TestContext framework additional annotations

```
@BeforeTransaction
public void verifyInitialDatabaseState() {
    (...)
}

@AfterTransaction
public void verifyFinalDatabaseState() {
    (...)
}
```

```
@Test
@NotTransactional
public void shouldDoSomethingNonTransactional() {
    (...)
}


@Test
@Repeat(3)
public void shouldDoSomethingThreeTimes() {
    (...)
}
```