Ink tutorial – 3

Written by: Lior Schachter

Empowering users to extend the system

In the previous tutorial you learned how you can empower your users to extend the application model with new classes.

In this tutorial we will further explore this subject by implementing another example with Ink.

The source code of this tutorial is located in the same project as previous tutorials and can be SVN checked out from this URL: http://svn.codespot.com/a/eclipselabs.org/ink/trunk/ink.tutorials

The example system

We will revisit a classic AOM example “Video-Store”, which was introduced in Johnson and Woolf “Type Object” pattern (http://www.cs.ox.ac.uk/jeremy.gibbons/dpa/typeobject.pdf).

The example describes a video store, which rents out videos to customers. Each movie has a number of properties, such as a title and a MPAA rating, and there can be many physical tapes of each movie. A videotape can be rented to a customer.

Requirements

We want to allow our end-user, the video-store owner, to manage its videos inventory. He may introduce new films; add more copies of an existing film etc.

Solution

Figure 1 is a class diagram of the video store example. A `Movie` is a metaclass, which defines the `title` and `rating` properties.

`Videotape` is an abstract class, instance of the `Movie` metaclass. It defines properties to be supplied by its instances, such as `isRented` and `renter`. Specific movies (e.g. "Spiderman") are represented as classes that extend the `Videotape` class and assign values to the `title` and `rating` properties. Physical tapes are represented as instances of those classes.

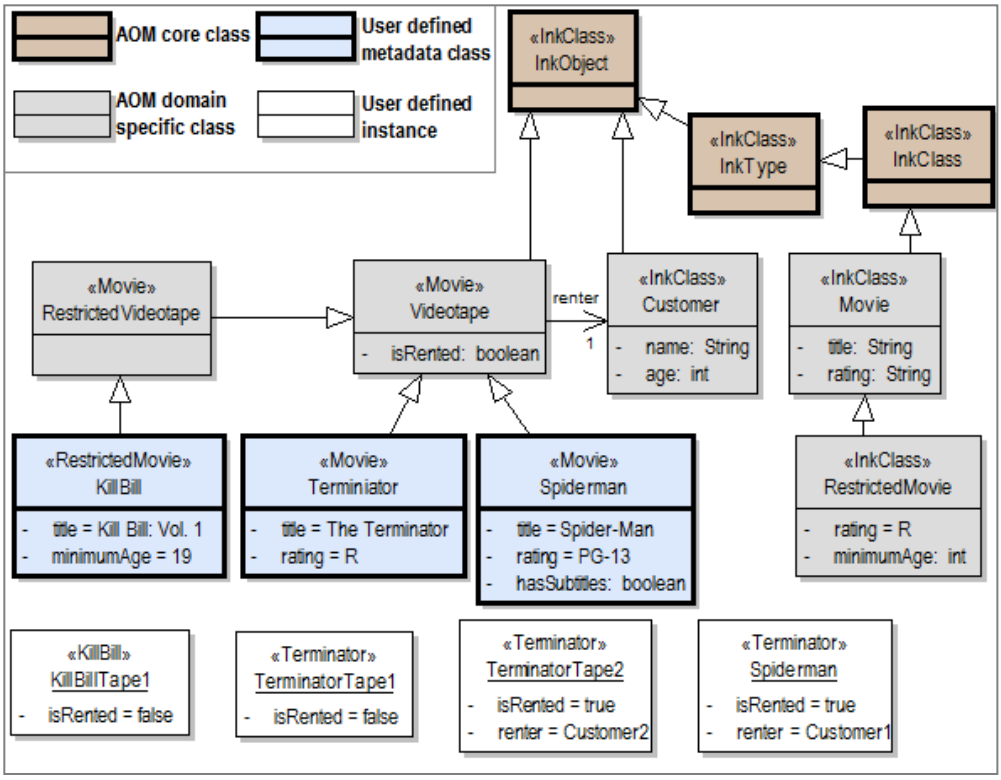

Figure 1. Example model, classes, meta-classes and instances

Listing 1a shows the Ink representation of `Movie` and `Videotape`. The `Movie` metaclass is a sub-class of (and an instance of) `InkClass`, and the `Videotape` class is an instance of `Movie` and a descendent of `InkObject`. The `java_mapping` property defines which Java counterparts the given class has. `Movie` and `Videotape` have the full Java structure – state class (`MovieState`, `VideotapeState`), behavior interface (`Movie`, `Videotape`) and behavior implementation class (`MovieImpl`, `VideotapeImpl`). Listing 1b shows the corresponding Java code. Generated code (State classes) is omitted.

The specific movies, represented by classes that extend `Videotape`, don't need specific behavior. The generic behavior in `VideotapeImpl` can be used. Thus their Java mapping is defined as `No_Java` (Listing 1c), at runtime they will be instantiated with their superclass Java counterparts (`VideotapeState` and `VideotapeImpl`). As such, they are perfectly suited for AOM implementation. Defining a new movie title doesn't require writing, editing or generating Java code or byte-code, and is well suited for non-programmers. Note that the `Spiderman` class modifies the inherited structure by adding the boolean property `hasSubtitles`.

```
Class id="Movie" class="ink.core:InkClass"
super="ink.core:InkClass"{
  java_path ""
  java_mapping "State_Behavior_Interface"
  properties {
    property class="ink.core:StringAttribute"{
      name "title"
      mandatory true
    }
    property class="ink.core:StringAttribute"{
      name "rating"
      mandatory true
    }
  }
}

Class id="Videotape" class="Movie"
super="ink.core:InkObject" abstract=true {
  java_path ""
  java_mapping "State_Behavior_Interface"
  properties {
    property class="ink.core:BooleanAttribute" {
      name "isRented"
      default_value false
    }
    property class="ink.core:Reference" {
      type ref="Customer"
      name "renter"
      mandatory false
    }
  }
}
```

Listing 1a. Movie and Videotape structure in Ink.

```
public class MovieImpl<S extends MovieState>
extends
InkClassImpl<S> implements Movie {
  public String getTitle() {
    return getState().getTitle();
  }
  …
}
```

```
public interface Videotape extends InkObject {
  public boolean canRent(Customer customer);
  …
}

public class VideotapeImpl<S extends
VideotapeState> extends
InkObjectImpl<S> implements Videotape {
  public boolean canRent(Customer customer) {
    return ! getState().getIsRented();
  }
  …
}
```

Listing 1b. Movie and Videotape behavior in Java.

```
Class id="Terminator" class="Movie"
super="Videotape" {
  java_mapping "No_Java"
  rating "R"
  title "The Terminator (1984)"
}

Class id="Spiderman" class="Movie"
super="Videotape" {
  java_mapping "No_Java"
  rating "PG-13"
  title "Spider-Man (2002)"
  properties {
    property class="ink.core:BooleanAttribute" {
      name "hasSubtitles"
      mandatory true
    }
  }
}
```

Listing 1c. Specific Videotape classes in Ink.

As time goes by, a new requirement arises: restricted movies should be classified as such, and for those movies the renter's age should be verified. The updated model and behavior appear in listings 2a and 2b. The `RestrictedMovie` metaclass sets the inherited `rating` property to a final value "R", and introduces a new property: `minimumAge`. `RestrictedVideotapeImpl` overrides the `canRent()` method and compares the renter's age with the minimum rental age as defined by its metaclass. In Listing 4 few Videotape instances are defined. These instances are being used in Listing 5 that demonstrates how a client code can interact with the video-store model.

```
Metaclass id="RestrictedMovie"
class="ink.core:InkClass" super="Movie" {
  java_path ""
  java_mapping "State_Behavior_Interface"
  component_type "Root"
  properties {
    property class="ink.core:IntegerAttribute" {
      name "minimumAge"
      mandatory true
      min_value 10
      max_value 100
    }
    property class="ink.core:StringAttribute" {
      name "rating"
      final_value "R"
    }
  }
}

Class id="RestrictedVideotape" class="RestrictedMovie"
super="Videotape"  abstract=true {
  java_path ""
  java_mapping "State_Behavior"
}

Class id="KillBill" class="RestrictedMovie"
super="RestrictedVideotape" {
  java_mapping "No_Java"
  title "Kill Bill: Vol. 1 (2003)"
  minimumAge 19
}
```

Listing 2a. Restricted movie implementation in Ink.

```
public interface RestrictedMovie extends Movie {
  public boolean canRent(CustomerState customer);
}

public class RestrictedMovieImpl<S extends
RestrictedMovieState>
extends MovieImpl<S> implements RestrictedMovie {
  public boolean canRent(CustomerState customer) {
    return getState().getMinimumAge() <=
customer.getAge();
  }
}

public class RestrictedVideotapeImpl<S extends
RestrictedVideotapeState>
extends VideotapeImpl<S> implements Videotape {
  public boolean canRent(CustomerState customer) {
    RestrictedMovie meta = getMeta();
    return super.canRent(customer)&&
meta.canRent(customer);
  }
}
```

Listing 2b. Restricted movie implementation in Java.

At runtime, classes can be created at runtime from AOM definitions, stored as metadata and edited via a GUI. A system can be developed and deployed with the build-time model (`Movie` and `Videotape`), at runtime, movie classes may be defined either in Ink SDL or in by means of a meta-object protocol (Mirror API). Listing 3 shows how a new class is created in the `ObjectEditorImpl` class. This method is invoked by the `createMovie(…)` method in Listing 5. After the values are set, the new class is registered in the Ink VM, and from that moment it becomes fully equivalent to classes that were written in the IDE.

```
public class ObjectEditorImpl<S extends ObjectEditorState>
        extends InkObjectImpl<S> implements ObjectEditor {
  …
  public ObjectEditor createDescendent(String
        descendentId){
    InkObjectState descendentState =
        targetState.cloneState();
    ObjectEditor descendentEditor =
        descendentState.reflect().edit();
    descendentEditor.setSuper(superState);
    descendentEditor.setId(descendentId);
    return descendentEditor;
  }
}
```

Listing 3. MOP implementation for creating new class using Mirror API.

```
Object id="Customer1" class="Customer" {
  name "Peter Parker"
  age 18
}
Object id="Customer2" class="Customer" {
  name "John Connor"
  age 33
}
Object id="TerminatorTape1" class="Terminator" {
  isRented false
}

Object id="TerminatorTape2" class="Terminator" {
  isRented true
  renter ref="Customer2"
}
Object id="SpidermanTape1" class="Spiderman" {
  isRented true
  renter ref="Customer1"
  hasSubtitles true
}
Object id="KillBillTape1" class="KillBill" {
  isRented false
}
```

Listing 4. Videotape instances.

```
@Test
public void testVideoStore() {
  Context context = InkVM.instance().getContext();
  Videotape terminatorTape1 = context.getObject("ink.tutorial:TerminatorTape1");
  Videotape terminatorTape2 = context.getObject("ink.tutorial:TerminatorTape2");
  Videotape spidermanTape1 = context.getObject("ink.tutorial:SpidermanTape1");
  Videotape killBillTape1 = context.getObject("ink.tutorial:KillBillTape1");

  CustomerState customer1 = context.getState("ink.tutorial:Customer1");
  CustomerState customer2 = context.getState("ink.tutorial:Customer2");
  assertTrue(terminatorTape1.canRent(customer1));
  assertTrue(!terminatorTape2.canRent(customer1));
  assertTrue(!killBillTape1.canRent(customer1));
  assertTrue(killBillTape1.canRent(customer2));
  assertTrue((Boolean)spidermanTape1.reflect().getPropertyValue("hasSubtitles");
}

@Test
public void testMOP() {
  Context context = InkVM.instance().getContext();
  CustomerState customer1 = context.getState("ink.tutorial:Customer1");

  createMovie("example.videostore:Shrek", "Shrek (2001)", "PG");
  Videotape shrekTape1 = createTape("ink.tutorial:Shrek");
  assertTrue(shrekTape1.canRent(customer1));

  createRestrictedMovie("ink.tutorial:FightClub", "Fight
       Club (1999)", 21);
  Videotape fightClubTape1 = createTape("ink.tutorial:FightClub");
  assertTrue(!fightClubTape1.canRent(customer1));
}

private Videotape createTape(String videotapeId) {
  VideotapeState newTapeState = InkVM.instance().getContext().newInstance(videotapeId);
  newTapeState.setIsRented(false);
  newTapeState.setRenter(null);
  return newTapeState.getBehavior();
}

public void createMovie(String id, String title, String rating) {
  Context context = InkVM.instance().getContext();
  InkObjectState videotape = context.getState("ink.tutorial:Videotape");
  ObjectEditor dynamicVideotapeEditor = videotape.reflect().edit().createDescendent(id);
  dynamicVideotapeEditor.setPropertyValue(InkClassState.p_java_path, "");
  dynamicVideotapeEditor.setPropertyValue(InkClassState.p_java_mapping, JavaMapping.No_Java);
  dynamicVideotapeEditor.setPropertyValue(MovieState.p_title, title);
  dynamicVideotapeEditor.setPropertyValue(MovieState.p_rating, rating);
  dynamicVideotapeEditor.save();
  context.register(dynamicVideotapeEditor.getEditedState());
}
```

Listing 5. Testing the videostore model.