# Ink tutorial – 1

# Ink tutorial for beginners

Ink is a framework for developing agile software. Ink lets developers expose their code as Domain Specific Language (DSL) so that others can use it in a simple way. So simple, that you don’t have to be a developer to use a DSL. Ink’s architecture supports agility in the technical sense as well. DSLs are interpreted by the Ink framework at runtime in a just-in-time manner. This helps to keep the edit-execute cycle in the scale of seconds rather than minutes.

In this tutorial you will learn the concept of declarative DSLs and how to implement a simple DSL in Ink.

## The example

For the purpose of the tutorials we’ll use a fictitious Subscription Management System for a magazine. The system manages subscriptions to magazines and allows new users to subscribe to the magazine, renew their subscription, etc.

## New requirement

Suppose you need to add functionality for special promotional prices allowing the Program Manager to define benefits that encourage people to subscribe.

E.g. A benefit can be a percentage off the list price. For example, Subscribe to “New Yorker” until 1/12 and get discount - 1 year, 20% discount, 2 years, 25% discount, 3 years 50% discount.  
A benefit can also be a specific number of bonus issues added to the subscription. For example, buy a 2 year subscription to Car Magazine and receive two free issues.  
A benefit can be a combination of % off the list price + extra issues. For example, buy a 3 year subscription to Car Magazine and receive a 25% discount off the list price + 4 free issues.

In this tutorial we show how DSLs can help in this use-case.

## Existing system

The system already has the following classes implemented in regular Java (no Ink):

* Magazine class, implementing interface A\_Product. This keeps tracks of the various magazines available and their list price:

**public** **interface** A\_Product {

String getID();

String getName();

Double getPrice();

}

* Customer class, implementing interface A\_Customer:

**public** **interface** A\_Customer {

**public** String getName();

**public** String getEmail();

**public** String getCreditCardNumber();

**public** **boolean** isStudent();

}

* Subscription class – represents a subscription, implements A\_Subscription interface:

**public** **interface** A\_Subscription {

// List price

**double** getPrice();

// Subscription is to this magazine

A\_Product getMagazine();

// 1 year, 2 years, etc.

**int** getPeriod();

// Start period of the subscription

Date getPeriodStart();

// Indicates if this subscription is in effect or just draft

**boolean** isCommitted();

// When the customer commits to the subscription, call commit()

**boolean** commit();

// The customer

A\_Customer getCustomer();

}

## Solution design

We’ll add the following methods to A\_Subscription in order to show the user the special offer terms and benefits:

**double** getPromotionalPrice(); // Price after discount

**int** getFreePeriods(); // # of free issues

String getPromotionalMessage(); // Promotional message

The implementation of these methods will look for special offers that are relevant for the subscription and select the best offer.

Enter Ink.

We’ll leave the implementation of the above methods to later in the tutorial as they comprise the glue code to Ink.

The implementation of a special-offer includes A\_SpecialOffer interface (see below), an abstract base class (BaseOfferImpl) and two concrete implementations for fixed-percentage discount and fixed-price offer (see below).

**public** **interface** A\_SpecialOffer {

// Is this subscription eligible to the special offer?

**boolean** isEligible(A\_Subscription subscription);

**double** getPromotionalPrice(A\_Subscription subscription);

**int** getFreeIssues(A\_Subscription subscription);

String getPromotionalMessage(A\_Subscription subscription);

}

We’ll implement these classes as an Ink DSL. This will allow us to use the DSL in order to define new offers (as instances of those offers).

## Creating an Ink DSL

First thing to know about Ink DSLs in Ink is that they are merely definition of Java instances written outside Java. In our case a definition of a special offer can look like this:

Object id="ExampleOffer" class="FixedPercentageDiscountOffer" {

percentage 20.0

studentOnlyOffer true

validUntil 2011/11/01

renewalOnlyOffer false

freeIssues 0

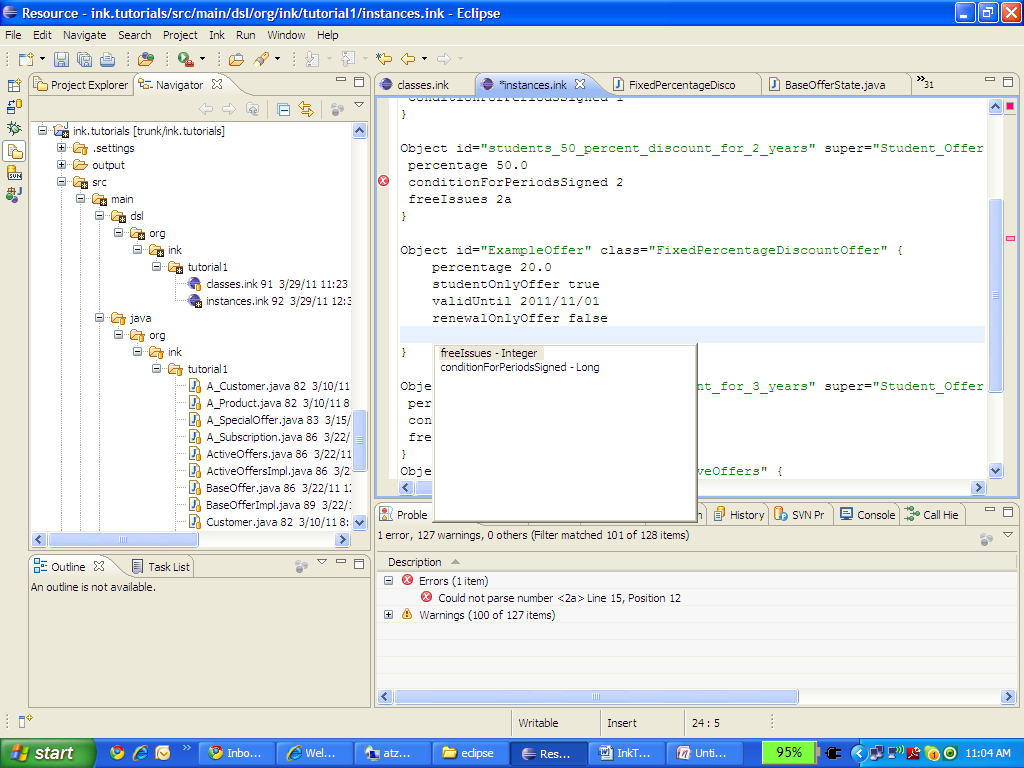
}

In terms of the Specia-Offers DSL, this is a definition of a special offer that gives you 20% discount if you are a student and you subscribe before 1/11/2011.

In fact, this is a definition of a Java instance for the FixedPercentageDiscountOffer class. At runtime, the Ink VM reads this DSL script and creates the Java instance.

When you edit the DSL script you get tool support from Ink plugins, similar to what you get from the Eclipse-JDT when you edit Java code. This includes auto-completion, incremental compilation, navigation commands, etc.,

See table of useful tools at the end of this tutorial.



Ink is a type-safe language. Scripts are checked for consistency by the incremental compiler every time you save an Ink file. The consistency checks and the other tool support are based on definition of the DSL provided by the DSL author. For the example above, here is the corresponding definition:

Class id="BaseOffer" class="ink.core:InkClass" super="ink.core:InkObject" abstract=true {

java\_path ""

java\_mapping "State\_Behavior\_Interface"

properties{

property class="ink.core:BooleanAttribute"{

name "studentOnlyOffer"

mandatory true

}

property class="ink.core:BooleanAttribute"{

name "renewalOnlyOffer"

mandatory true

}

property class="ink.core:LongAttribute"{

name "conditionForPeriodsSigned"

mandatory false

}

property class="ink.core:IntegerAttribute" {

name "freeIssues"

mandatory false

}

property class="ink.core:DateAttribute"{

name "validUntil"

mandatory true

}

}

}

Class id="FixedPercentageDiscountOffer" class="ink.core:InkClass" super="BaseOffer" abstract=false{

java\_path ""

java\_mapping "State\_Behavior"

properties{

property class="ink.core:DoubleAttribute"{

name "percentage"

mandatory true

}

}

}

This definition comprises of two model classes: BaseOffer and FixedPercentageDiscountOffer. Later you’ll see that these model classes has corresponding Java classes, but for the purpose of authoring Ink scripts only the model classes are used. The model classes define the DSL and dictate which Java classes need to be written.

A model class definition states the properties of the class for which instances need to supply value.

In this example, FixedPercentageDiscounterOffer has one property, “percentage” of type Double. Since it inherits the BaseOffer class, it has the properties defined in BaseOffer as well.

Each instance specifies it’s class in the class attribute:

Object id="ExampleOffer" class="**FixedPercentageDiscountOffer**" {

percentage 20.0

studentOnlyOffer true

validUntil 2011/11/01

renewalOnlyOffer false

freeIssues 0

}

Note that the model class definitions are written in Ink as well. This is because they are also instances of their meta-classes. But lets leave this to a later tutorial.

Ink uses SDL concrete syntax, you can learn more about it here: <http://en.wikipedia.org/wiki/Simple_Declarative_Language>

Actual instances with inheritance

Active offers

## Glue from Java to Ink

## Ink tools

|  |  |  |
| --- | --- | --- |
| Tool | Activated by | Does |
| Compiler | Saving an ink file | Validates consistency of the Ink scripts |
| Open Ink Element (similar to alt-ctrl-R in Java) | alt-ctrl-q | Search for and open an Ink element by it’s name (id) |
| Open declaration (similar to F3 in Java) | F3 | Navigate to selected Ink element |
| Open Java implementation | F4 | Navigate from Ink script to it’s Java implementation class |

# Open issues

* Explain mapping
* Explain glue technique – Calling VM to instantiate, BaseOffer inherits foreign interface A\_SpecialOffer
* Explain debug settings to skip proxy - org.ink.core.vm.proxy
* Explain abstract instance – the template students offer
* Prepare for AOM in tutorial 2 – additional benefit (every property of type A\_Benefit).
* Tutorial 3 – Validation

1. How to write a DSL in Ink
2. Validation
3. AOM
4. Aspects