Ink tutorial – 1

Written by: Atzmon Hen-tov and Olga Kogan

What is ink all about?

Ink is a framework for developing agile software using Java. It allows you to write code that can be customized and enhanced easily. Much of this possible customization can be achieved without writing any additional code.

Ink lets developers expose their code as a Domain Specific Language (DSL), a simple syntax to configure the behavior of your system.

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.

To try the tutorial code yourself, take a look at "Ink Tutorial– Installation Guide" at the end of this tutorial.

An example system

To demonstrate the advantages and usage of Ink, we will now describe the system which we will use for the examples in this tutorial.
This is a typical Java based system, to which we will gradually introduce the Ink framework as the tutorial progresses.

Even though it's a "fictitious" system, it shares many of the qualities of modern enterprise systems.

Imagine a magazine subscription management system. The system is used to subscribe readers to the magazine, renew their subscription, etc.

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

- A Magazine class, implementing the interface A_Product. Each instance represents a magazine, along with its price.

```
public interface A_Product {

   String getID();
   String getName();
   Double getPrice();
}
```

- A Customer class, implementing the interface A_Customer:
-

```
public interface A_Customer {

   public String getName();
   public String getEmail();
   public String getCreditCardNumber();
   public boolean isStudent();

}
```

- A Subscription class, implementing the A_Subscription interface. This class represents the subscription of a single customer to a particular magazine.

```
public interface A_Subscription {

   // The customer
   A_Customer getCustomer();

   // Subscription is to this magazine
   A_Product getMagazine();

   // List price
   double getPriceForSubscriptionPeriod();

   // 1 year, 2 years, etc.
   int getPeriods();

   // Start period of the subscription
   Date getPeriodStart();

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

   // Indicates if this subscription is in effect or just draft
   boolean isCommitted();


}
```

New requirement

One day, a new functionality is required from the system – add the possibility of promotions and discounts. Since "discount" and "promotion" nowadays can mean more or less anything, this new ability needs to be flexible and easily adaptable to the new ideas of marketing and sales managers. This is where Ink DSLs are handy, allowing this new functionality to be easily implemented and configurable upon need.

Solution

The new requirement implies that A_Subscription should include the special terms of the subscription.

We will implement a logic that will look for special offers that are relevant for the subscription, and select the best offer.

So, we will add the following methods to A_Subscription:

```
A_SpecialOffer getBestOffer();
```

Also, we will need to implement the promotions themselves: A_SpecialOffer interface (see below), an abstract base class (BaseOfferImpl), and two concrete implementations of promotions:

- percentage discount (x% of original price)
- fixed-price offer (pay x instead of y).

```
public interface A_SpecialOffer {

      double getPromotionalPrice();    // Price after discount
      int getFreeIssues();              // # of free issues
      String getPromotionalMessage();   // Promotional message
}
```

We'll implement these classes as an Ink DSL. This will allow us to create multiple variations of such promotions, without writing additional code, or even restarting the system.

Creating an Ink DSL

Ink DSLs actually define instances of Java objects, but are not written in Java.

In our case, a definition of a special offer would look something like this:

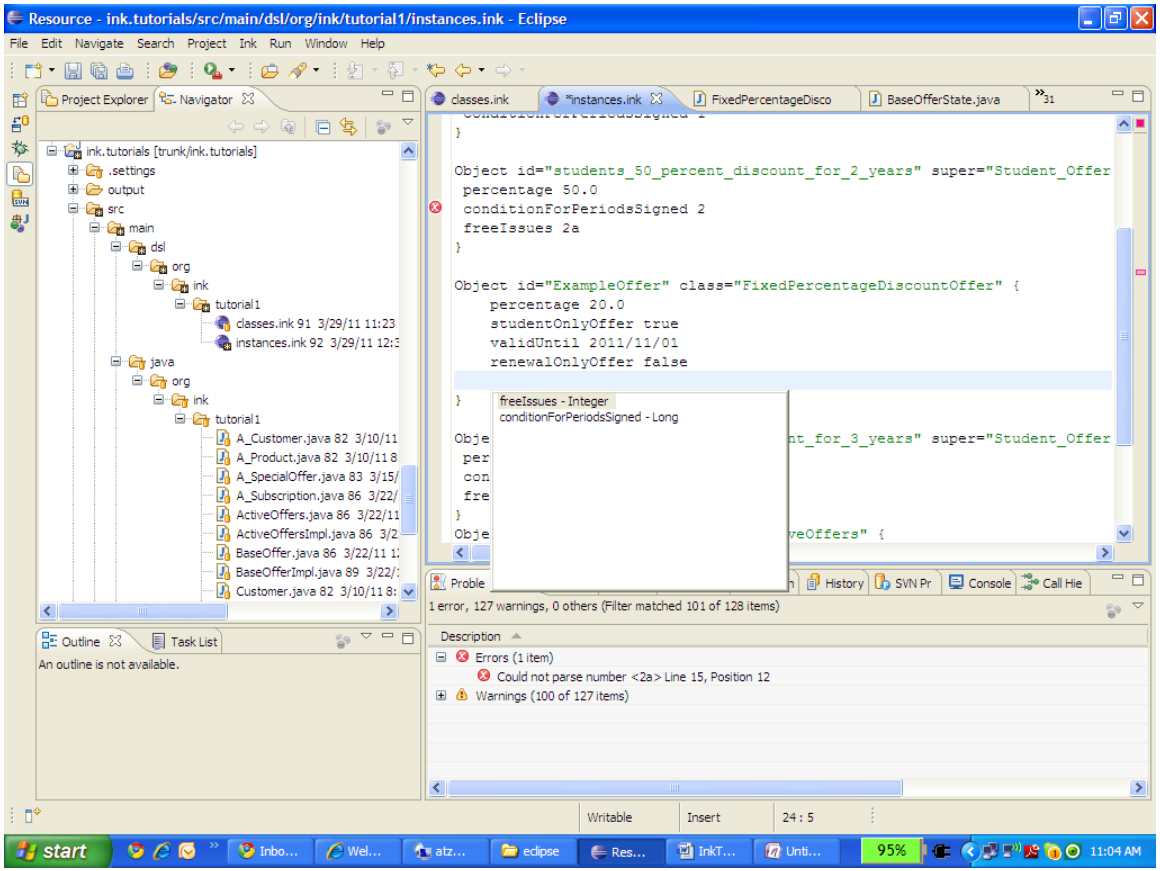
```
Object id="ExampleOffer" class="PercentageDiscountOffer" {
      percentage 20.0
      studentOnlyOffer true
      validUntil 2011/11/01
      renewalOnlyOffer false
      freeIssues 0
}
```

As you might have guessed, this is an offer that gives 20% discount to students that subscribe before November 1, 2011.

At runtime, the Ink VM reads the DSL script , creates the expected instance of PercentageDiscountOffer class and injects the values from the Ink object (ExampleOffer) to the Java instance.

Writing such scripts is especially easy since the Ink plugins for eclipse offer you powerful IDE features similar to what eclipse JDT provides to Java developers.

This includes auto-completion, incremental compilation, navigation commands, etc., (see table of useful shortcuts at the end of this tutorial.)


Resource - ink.tutorials/src/main/dsl/org/ink/tutorial1/instances.ink - Eclipse
File Edit Navigate Search Project Ink Run Window Help
Project Explorer
Navigator
ink.tutorials [trunk/ink.tutorials]
.settings
output
src
main
dsl
org
ink
tutorial1
classes.ink 91 3/29/11 11:23
instances.ink 92 3/29/11 12:3
java
org
ink
tutorial1
A_Customer.java 82 3/10/11
A_Product.java 82 3/10/11 8
A_SpecialOffer.java 83 3/15/
A_Subscription.java 86 3/22/
ActiveOffers.java 86 3/22/11
ActiveOffersImpl.java 86 3/2
BaseOffer.java 86 3/22/11 1
BaseOfferImpl.java 89 3/22/
Customer.java 82 3/10/11 8
Outline
Task List
An outline is not available.
classes.ink
*instances.ink
FixedPercentageDisco
BaseOfferState.java
}
Object id="students_50_percent_discount_for_2_years" super="Student_Offer
percentage 50.0
conditionForPeriodsSigned 2
freeIssues 2a
}
Object id="ExampleOffer" class="FixedPercentageDiscountOffer" {
percentage 20.0
studentOnlyOffer true
validUntil 2011/11/01
renewalOnlyOffer false
}
freeIssues - Integer
conditionForPeriodsSigned - Long
Obje nt_for_3_years" super="Student_Offer
per
con
fre
}
Obje veOffers" {
Proble
History
SVN Pr
Console
Call Hie
1 error, 127 warnings, 0 others (Filter matched 101 of 128 items)
Description
Errors (1 item)
Could not parse number <2a> Line 15, Position 12
Warnings (100 of 127 items)
Writable
Insert
24 : 5
start
Inbo...
Wel...
atz...
eclipse
Res...
InkT...
Unti...
95%
11:04 AM

How does it work?

Ink is a type-safe language. Scripts are checked for consistency by the incremental compiler every time you save an Ink file.

In order to use the Ink script from the previous section, some ground work has to be done.
Basically, what we have to do is to define a template of a BaseOffer instantiation.

These are the templates we've used:

```
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="PercentageDiscountOffer" class="ink.core:InkClass" super="BaseOffer" abstract=false{
	java_path ""
	java_mapping "State_Behavior"
	properties{
		property class="ink.core:DoubleAttribute"{
			name "percentage"
			mandatory true
		}
	}
}
```

As you can see, we've defined two **Ink classes**: BaseOffer and PercentageDiscountOffer. These classes define our "Marketing DSL" and allow users to use the DSL by writing instances of those classes. Each Ink class has a corresponding Java class used for executing the DSL.

In the classes above, PercentageDiscountOffer has one property, "percentage" of type Double.
Since it inherits the BaseOffer class (super=), it has the properties defined in BaseOffer as well.

Lookin again at the "ExampleOffer", you may observe it is an instance of PercentageDiscounterOffer (class=).

```
Object id="ExampleOffer" class="PercentageDiscountOffer" {
      percentage 20.0
      studentOnlyOffer true
      validUntil 2011/11/01
      renewalOnlyOffer false
      freeIssues 0
}
```

The syntax used by Ink is called SDL. You can learn more about it here: http://en.wikipedia.org/wiki/Simple_Declarative_Language

Note that the Ink class definitions are written in Ink as well.

Later you'll see that these model classes have corresponding Java classes, but for the purpose of authoring Ink scripts it is inconsequential.

Making the marketing department happy

All this mumbo-jumbo made us almost forget about actually implementing the requirement – adding promotions and discounts to the system.

Here we will define some promotions, and the "pool" of possible promotions.

For promotions that share common logic and behavior, we can use Ink's instance inheritance, just like with Java classes.

For example, let's define an abstract offer for students, and two inheriting offers.

```
Object id="Student_Offers_Template_For_2010" class="ink.tutorial1:BaseOffer" abstract=true {
 studentOnlyOffer true
 renewalOnlyOffer false
 validUntil 2011/11/01
}


Object id="students_30_percent_discount_for_1_year" super="Student_Offers_Template_For_2010"
class="ink.tutorial1:FixedPercentageDiscountOffer"{
 percentage 30.0
 conditionForPeriodsSigned 1
}

Object id="students_50_percent_discount_for_2_years" super="Student_Offers_Template_For_2010"
class="ink.tutorial1:FixedPercentageDiscountOffer"{
 percentage 50.0
 conditionForPeriodsSigned 2
 freeIssues 2
}
```

In this example, `students_30_percent_discount_for_1_year` inherits the values of "`studentOnlyOffer`", "`renewalOnlyOffer`" and "`validUntil`" from `Student_Offers_Template_For_2010`, and assigns values to "`percentage`" and "`conditionForPeriodsSigned`".

Note that Ink uses "abstract" and "super" keywords, very similarly to Java.

The next step is to define the “pool” of all the active offers.

This is an instance that contains references to other instances. This is done by “ref” keyword, and then specifying the IDs of specific instances.

```
Object id="Active_offers" class="ActiveOffers" {
      offers{
            offer ref="students_30_percent_discount_for_1_year"
            offer ref="students_50_percent_discount_for_2_years"
            offer ref="students_60_percent_discount_for_3_years"
      }
}
```

Now that we already mastered the basics, we know that the structure of `ActiveOffers` class should also be defined:

```
Class id="ActiveOffers" class="ink.core:InkClass" super="ink.core:InkObject" {
      java_path ""
      java_mapping "State_Behavior_Interface"
      properties {
            property class="ink.core:ListProperty"{
                  type ref="ink.core:List"
                  name "offers"
                  mandatory true
                  list_item class="ink.core:Reference"{
                        type ref="ink.tutorial1:BaseOffer"
                        name "offer"
                  }
            }

      }
}
```

The ActiveOffers class has one property of type “list of Base Offer”.

Looking again at the “ActiveOffers” instance, observe that each list member is referenced as “offer”.

```
Object id="Active_offers" class="ActiveOffers" {
      offers{
            offer ref="students_30_percent_discount_for_1_year"
            offer ref="students_50_percent_discount_for_2_years"
            offer ref="students_60_percent_discount_for_3_years"
      }
}
```

Now you can see that the word “offer” is dictated by the Ink template:

```
property class="ink.core:ListProperty"{
      type ref="ink.core:List"
      name "offers"
      mandatory true
      list_item class="ink.core:Reference"{
            type ref="ink.tutorial1:BaseOffer"
            name "offer"
      }
}
```

The definition of “list of Base Offer” specifies the name of the property (“offers”), as well as the name of each item in the list (“offer”).

Glue from Java to Ink

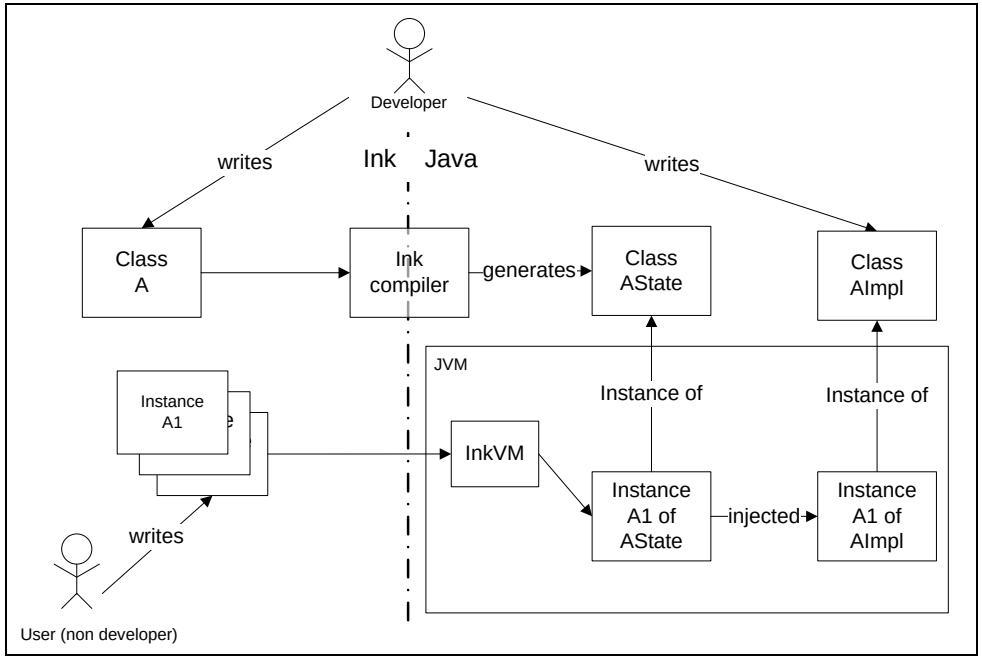
The last piece of the puzzle, to complete and run our code, is the part connecting Ink to Java.
An Ink class is mapped into two Java classes (see figure below)

Structure class (class AState in the diagram)

The structure class is named the same as the Ink class + "State" suffix. It contains the properties as defined in the Ink class (e.g. percentage).
This class is generated by the Ink compiler.
In our example, it would be `PercentageDiscountOfferState`.

Behavioral class (class AImpl in the diagram)

The behavioral class is named the same as the Ink class + "Impl" suffix. This class is written by the DSL developer (that's you ☺), and implements the required behavior.
In our example, it would be `PercentageDiscountOfferImpl`.



When Ink VM instantiates the behavioral class, it injects to the new instance the corresponding instance of the structure class.
The basic idea behind Ink DSLs is that behavior classes use the injected state instances as configuration.
The injected ink instance (Java structure class) is available to the behavior class by the `getState()` method (see below).

For example, let's see how the java uses "percentage" property from our first DSL:

```
Object id="ExampleOffer" class="PercentageDiscountOffer" {
      percentage 20.0
      CustomerType Student
      validUntil 2011/11/01
      renewalOnlyOffer false
      freeIssues 0
}
```

```
package org.ink.tutorial1;

public class FixedPercentageDiscountOfferImpl<S extends FixedPercentageDiscountOfferState>
            extends BaseOfferImpl<S> implements BaseOffer {


      @Override
      public double getPromotionalPrice(A_Subscription subscription) {
            double result = 0.0;


            if (isEligible(subscription)) {

              result = subscription.getPrice()
                        * (100.0 - getState().getPercentage()) / 100.0;

            } else {
              result = subscription.getPrice();
            }

            return result;
      }
…
}
```

Note: We don't need to cast the state to `PercentageDiscountOfferState`, because we use generics in the class definition.

Calling Ink from non-Ink Java

When you integrate Ink into your existing Java application, you need to instantiate the Ink VM. This is done by calling the method

```
static instance()
```

of `InkVM` class. Any time you need to access the instances you defined using Ink, it's simply done as in the example below,

See implementation of getBestOffer in Subscription class.

```
public A_SpecialOffer getBestOffer() {
  A_SpecialOffer bestOffer;

  ActiveOffers
      offers = InkVM.instance().getContext().getState("ink.tutorial1:Active_offers").getBehavior();

  bestOffer = offers.getBestOffer(this);
  return bestOffer;
}
```

Exercise

If you want to try by yourself, Implement FixedPriceOffer:

Write an Ink instance, a java "Impl" class, and enhance the unit-tests (TestTutorial1Test) to cover it.

Summary

In this tutorial you learned how to create a new Ink DSL, by the following steps:

- Language developer defines the DSL – by writing Ink classes (templates)

- Language developer implements the DSL – by writing Java behavior classes (“Impl”)
- Language user uses the DSL – by writing Ink instances

Note that using the DSL does not require changes in Java-code, and thus provides a faster way to deliver functionality to production.

Also, person who modifies and writes Ink instances does not even need to be a programmer.

In the next tutorial you will learn how to empower users (non developers) to define new, functioning Ink classes.

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

Ink Tutorial– Installation Guide

1. Install Eclipse IDE:

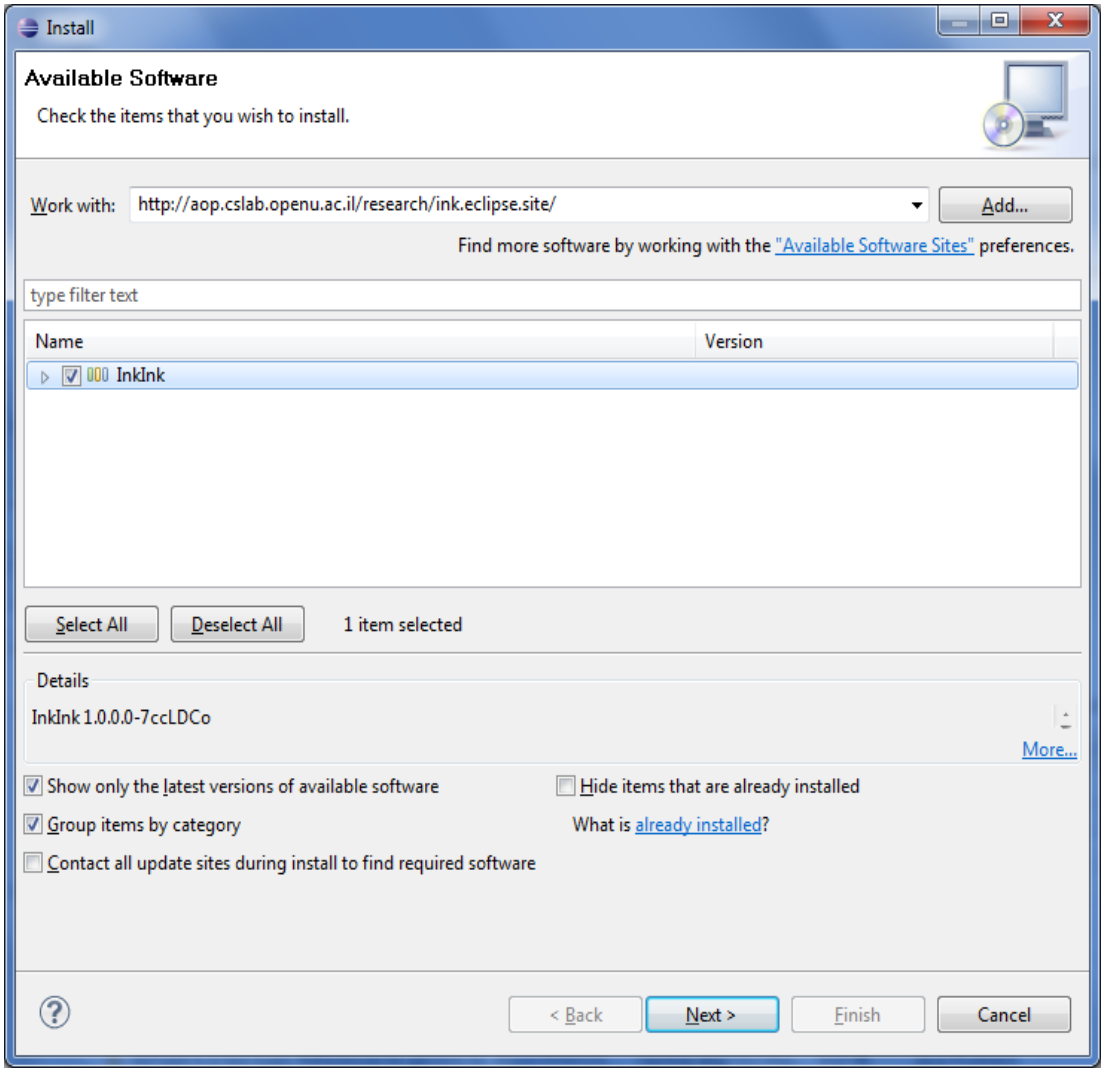
Download the Eclipse IDE latest version for Java developers from

http://www.eclipse.org/downloads/packages/eclipse-ide-java-developers/heliossr2

Use the eclipse version for your operating system.

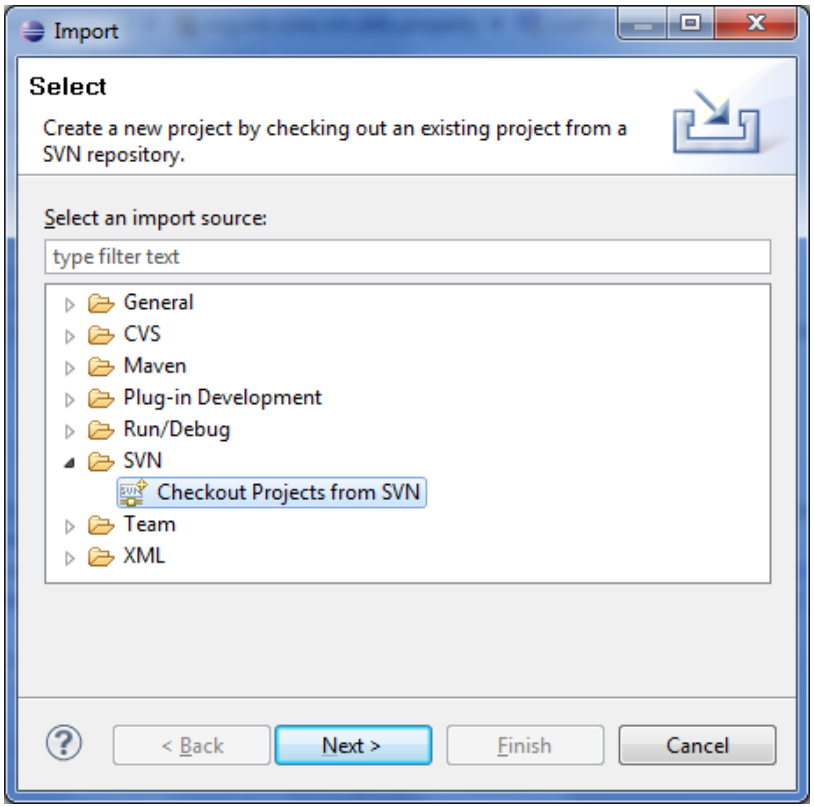
2. Install the Ink-Framework Eclipse plugin:
    a. In Eclipse, go to 'Help'→'Install New Software…'
    b. Type in the Ink-Framework update site URL : http://aop.cslab.openu.ac.il/research/ink.eclipse.site/ :

Mark the ‘InkInk’ flag and press ‘Next’.

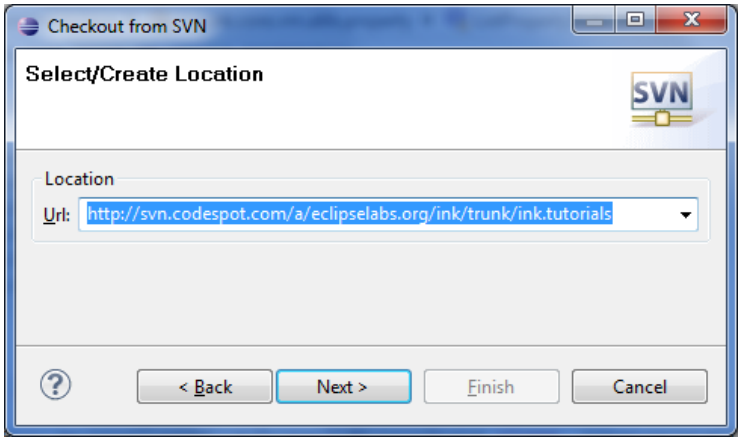


c. After the installation process is completed, restart Eclipse.

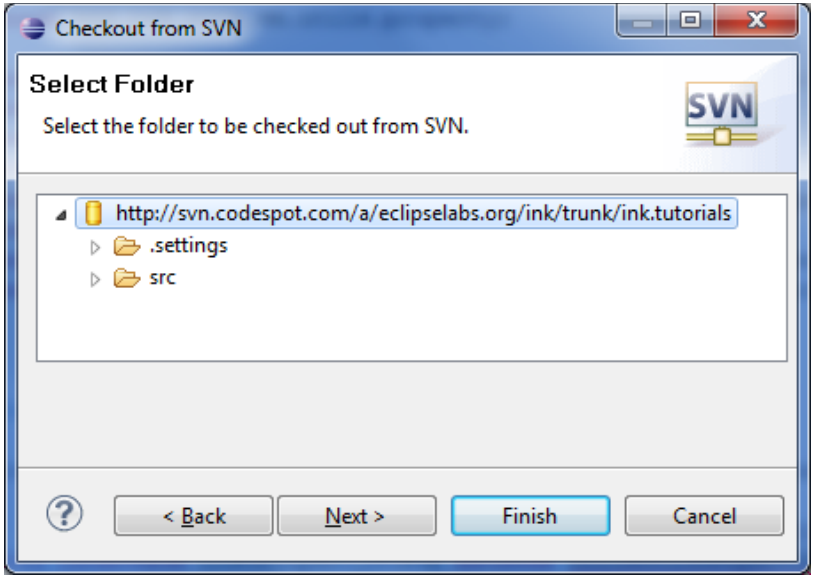
3. Install Eclipse SVN plugin :
   a. Follow the instructions here : http://subclipse.tigris.org/servlets/ProjectProcess?pageID=p4wYuA
      Or
   b. Simply go to 'Help'→'Install New Software' → type in the update site URL http://subclipse.tigris.org/update_1.6.x → 'Next'…
4. Download the 'ink-tutorial' Eclipse project:



a. Go to 'File'→'Import' and choose 'Checkout Projects from SVN'.
b. Choose 'Create a new repository location' → 'Next'.
c. Paste the URL: http://svn.codespot.com/a/eclipselabs.org/ink/trunk/ink.tutorials and press 'Next'.



d. Choose the root element and press ‘Finish’.



e. Restart Eclipse IDE.

f. Launch Clean-Build : ‘Project’→’Clean’ (make sure ‘Clean all projects’ is marked)→Press ‘OK’.