Introduction to the Java language and Development Environment

By Marc-André Laverdière

Introduction to the Java language and Development Environment

Complete reference and tutorials available online at www.java.sun.com

Java is intended to be a multiplatform-independent programming language. However, different virtual machines on different platforms can behave differently.

Introduction to the Java language and Development Environment

Topics Covered Today:

- Java Language Syntax
- Classes & Inheritance in Java
- Exceptions & Exception Handling
- Java Language basic API
- System.* I/O
- Using Java from the command line
- JBuilder & SunOne Development Environments

Java Language Syntax

- Classes Only
- All Pointers, but Simple Memory Management
- Integrated Declaration-Definition
- Structuring a program
- Syntax of control structures

Classes Only

- Java has been built close to initial object concepts
- All in Java is based around classes
- Classes are organized within packages
- The general syntax is almost identical to C/C++

All Pointers, but Simple Memory Management

- (Almost) all variables in Java are pointers.
- This means that you must initialize all variables that are not basic types.
- The keyword null represents a null pointer.
- The keyword new is used to create new instances of a class
- Example:

```
String a; //This is just a pointer to a string, its value is null
String b = “” ; // this is an empty string
String c = new String(); //empty string, but built with the new operator
```

All Pointers, but Simple Memory Management

- Rely on the garbage collector for memory de-allocation.
- The garbage collector runs in a thread and detects objects that are no longer in use and destroys them. It is done implicitly.

Integrated Declaration-Definition

- Java classes and operations are defined directly.
- In C++:

```
Class a {
  private:
    int x;
  public:
    int function();
};
int a::function(){return x;}
```

- In Java:

```
Class a {
  private int x;
  public int function() {return x;}
  }
```

Structuring a Program

- Java forces only one class per .java file
- Classes are structured within packages
- Packages are the equivalent of the header files in C
- One of the classes will have a main function

Structuring a Program

Packages

- Define a class as part of a package:

package <packageName>;

- Using a package or a specific class

import java.lang.*; //import the whole java.lang package

import java.lang.String; //import the String class in java.lang

- Specifying that a class is public or private

public class a {…} //This class is accessible outside the package, and thus is part of the package's public interface

private class b {…}//This class is not accessible outside the package. Only classes within the package can use it.

Structuring a Program

Main

- The main class will have this method:

```
public static void main(String[] args) {
//Things that Main does…
}
```

Syntax of Control Structures

- **The syntax is meant to be identical to C/C++ syntax**
- **If Statement:**

if (<condition>) {…}
else {…}

- **Switch Statement:**

switch (<identifier>) {
case <case>: …
break;
default: …
}

- **While loop:**

while (<condition>) {…}

- **For Loop:**

for (<initialize>; <check>; <incrementation>){…}

- **Do while loop:**

Do {…} while(<condition>)

Classes & Inheritance in Java

- Specifying Encapsulation
- Constructors & Destructors
- Direct Inheritance
- Abstract Classes
- Implementation Inheritance
- Polymorphism

Specifying Encapsulation

- All definitions within a function must be precedented by its accessibility identifier.
- Private: Accessible internally only
- Protected: Accessible internally, inheritable
- Public: Accessible externally

```
public class A{ //example for a Singleton
  protected A instance;
  protected A() {…}
  public A getInstance(){…}
}
```

Constructors & Destructors

- Constructors are specified by using the same name as the class
- Destructors are specified by the finalize() function, which is called by the garbage collector. Use it for closing files and so on.

```
public class aBuffer{
  protected byte[] buffer;
  public aBuffer () { buffer = new byte[1024];}
…
protected void finalize(){
//not needed, really }
```

Direct Inheritance

- The direct inheritance is implemented by using the extends keyword:

```
public class myString extends String{
  public myString (){ //constructor overload
    super(); // call the ancestor's constructor
  }
}
```

- If you want your class not to be derived (and I don't see why), use the final keyword

public final class <name> …

Abstract Classes

- Abstract classes are classes with abstract methods.
- Abstract methods are methods with no definition.
- Abstract methods need to be implemented in derived classes that are not abstract.

```
Public abstract class A{
  …
  public abstract void someOperation();
}
```

Implementation Inheritance

- The implementation inheritance allows to implement interface classes
- Not used very often

```
public interface myInterface …

public class threadProgram implements Runnable{
  …
  public void run(){…} //This method is defined in the interface and must be defined here
}
```

Polymorphism

- Members of the same class hierarchy can be used polymorphically.
- All classes are implicitly derived from Object.
- For example, a general-purpose class acting on any class can use Object as parameters.

```
public void push_stack (Object to_push){…}
```

Exceptions & Exception handling

- Exception is a newer concept for error handling.
- In C, the convention was to have all functions return a negative value defined in constants.
- This complexifies error handling and increases coupling.

Exceptions & Exception handling

- Java defines a Exception class, as well as many derived classes, such as IOException.
- You can define your own exceptions and use them as a powerful tool for exception handling
- You can also catch exceptions in order to encapsulate error handling inside a package and so on.

Exceptions & Exception handling

- It is possible to define a method as throwing a certain type of exception

public void read() throws IOException{…}

- Java will then force us to use exception handling in our program when calls to such methods are indicated.

try { … } //This code is run, perhaps partially. If an exception arises, the rest of the try block is not executed, and the catch code is run

catch (Exception e){ …} //The catch code is run whenever an exception is throwed

finally {…} // This code is run after the try or the catch block. It might need its own try/catch sequence. The Finally block is optional.

Java Language Basic API

- Basic Data Types
- Constants
- Operators
- Arrays & Cloning
- Strings
- Vector
- Object
- Math

Java Language Basic API

- All the details on the API and all classes is detailed on: http://java.sun.com/j2se/1.3/docs/api/
- Today's presentation merely summarizes a some of the more important classes

Basic Data Types

- long:Big-endian 64-bit signed integer
- int: Big-endian 32-bit signed integer
- short: Big-endian 16-bit signed integer
- byte: Used in arrays, is a 8-bit signed integer
- float: 32-bit IEEE 754 floating-point
- double: 64-bit IEEE 754 floating-point
- char: 2 unsigned bytes
- boolean: true (1), false (0)

Basic Data Types

- All the basic data types have a wrapper class, which is spelled with the first letter as a capital.
- Ex.: int -> class Int
- They are mostly useful when parsing a specific value from a String.

Operators

The operators are the same as in C, which is why we won't cover them in detail.
However, take note that Java won't allow operator overloading

Operator	Meaning
+	Addition on numbers
+	Concatenation on String
==	Equality. You should use the .equal() method when not dealing with numbers
!=	Difference
>	Greater than
<	Smaller than

Constants

Java Allows you to define constants using the final keyword.

```
public final int MAX_CONNECTIONS = 127;
```

There is no enumeration available as in C.

Arrays & Cloning

- Arrays are declared as following:

Base_type[] <identifier_name>;

byte[] byteStream;

- Arrays are initialized similarly as in C++

byte[] byteStream = new byte[1024];

- Arrays support cloning. When passing arrays around to classes who store array information, you should use the .clone() method to get a copy of the array and ensure that the data won’t get corrupted.

Arrays & Cloning

- They are similar to C arrays, only that you will receive a ArrayIndexOutOfBoundsException if you try to access beyond the allocated memory
- Use the .length data member to determine the defined length of the array.

Strings

- Strings are atomic UTF-8 strings.
- This differs from C, where strings were simple arrays of 8-bit characters
- Package: java.lang.String;
- They support practically all string-related operations
- Useful methods:

endsWith, equalsIgnoreCase , lastIndexOf, replace, startsWith, toLowerCase, toUpperCase, trim, valueOf

Vector

- Typical class implementing list, queues, stacks…
- Vector is a lot more useful than arrays in order to manage data and accesses.
- Vector is self-adjusting, so you don’t have to worry too much about memory management.
- Package: java.util.Vector;

Method Summary

void	add(int index, Object element) Inserts the specified element at the specified position in this Vector.
void	addElement(Object obj) Adds the specified component to the end of this vector, increasing its size by one.
void	clear() Removes all of the elements from this Vector.
Object	clone() Returns a clone of this vector.
boolean	contains(Object elem) Tests if the specified object is a component in this vector.
Object	elementAt(int index) Returns the component at the specified index.
int	indexOf(Object elem) Searches for the first occurence of the given argument, testing for equality using the equals method.

void	insertElementAt(Object obj, int index) Inserts the specified object as a component in this vector at the specified index.
boolean	isEmpty() Tests if this vector has no components.
Object	lastElement() Returns the last component of the vector.
Object	remove(int index) Removes the element at the specified position in this Vector.
boolean	removeElement(Object obj) Removes the first (lowest-indexed) occurrence of the argument from this vector.
void	setElementAt(Object obj, int index) Sets the component at the specified index of this vector to be the specified object.
int	size() Returns the number of components in this vector.
Object[]	toArray() Returns an array containing all of the elements in this Vector in the correct order.

Object

- Object is the base class of all classes in Java. As such, it can polymorphically represent all objects.
- Some classes will return Object types, and so you need to cast them to be useful
- For example:

String aString = “12345”; Vector aVector = new Vector();

aVector.addElement(aString); String result = (String) aVector.elementAt(0);

Math

- Math is a catchall object that puts together the major mathematical functions
- Package: java.lang.Math;

Method Summary

static double	abs(double a) Returns the absolute value of a double value.
static double	acos(double a) Returns the arc cosine of an angle, in the range of 0.0 through *pi*.
static double	asin(double a) Returns the arc sine of an angle, in the range of -*pi*/2 through *pi*/2.
static double	atan(double a) Returns the arc tangent of an angle, in the range of -*pi*/2 through *pi*/2.
static double	atan2(double a, double b) Converts rectangular coordinates (b, a) to polar (r, *theta*).
static double	ceil(double a) Returns the smallest (closest to negative infinity) double value that is not less than the argument and is equal to a mathematical integer.

static double	cos(double a) Returns the trigonometric cosine of an angle.
static double	exp(double a) Returns the exponential number *e* (i.e., 2.718...) raised to the power of a double value.
static double	floor(double a) Returns the largest (closest to positive infinity) double value that is not greater than the argument and is equal to a mathematical integer.
static double	IEEEremainder(double f1, double f2) Computes the remainder operation on two arguments as prescribed by the IEEE 754 standard.
static double	log(double a) Returns the natural logarithm (base *e*) of a double value.
static double	max(double a, double b) Returns the greater of two double values.
static double	min(double a, double b) Returns the smaller of two double values.

static double	pow(double a, double b) Returns of value of the first argument raised to the power of the second argument.
static double	random() Returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0.
static long	round(double a) Returns the closest long to the argument.
static double	sin(double a) Returns the trigonometric sine of an angle.
static double	sqrt(double a) Returns the correctly rounded positive square root of a double value.
static double	tan(double a) Returns the trigonometric tangent of an angle.
static double	toDegrees(double angrad) Converts an angle measured in radians to the equivalent angle measured in degrees.
static double	toRadians(double angdeg) Converts an angle measured in degrees to the equivalent angle measured in radians.

System.* I/O

- System is the variable that represents the computer system
- It has 3 major classes:
- System.out
- System.err
- System.in
- We will not cover the details of I/O in this introduction, but only enough so you can do simple input/output programs.

System.out & System.err

- System.out prints to STDOUT
- System.err prints to STDERR
- They are of type PrintStream, which captures all exceptions
- Significant methods: print, prinln
- It acts as an equivalent to cout and cerr in C++

System.in

- System.in reads from STDIN
- It is of type InputStream, which is very limited for reading
- As such, it is normally combined

```
BufferedReader stdin = new BufferedReader(new
    InputStreamReader(System.in));
```

- Calls such as .readline() are very convenient.
- However, the exceptions are not handled automatically.

Using Java from the command line

The process is in 2 steps:

- Compilation: javac *.java
- Execution: java [name of main class]
- javac will tell you syntax errors and will pre-compile the .java files into .class files
- java will run the main class that you specify

JBuilder & SunOne Development Environments

- JBuilder is only available in H905
- SunOne is available in all labs
- You can download a copy of the personal version on the internet for free.
- JBuilder: http://www.borland.com/products/downloads/download_jbuilder.html#
- SunOne: http://wwws.sun.com/software/sundev/jde/buy/index.html

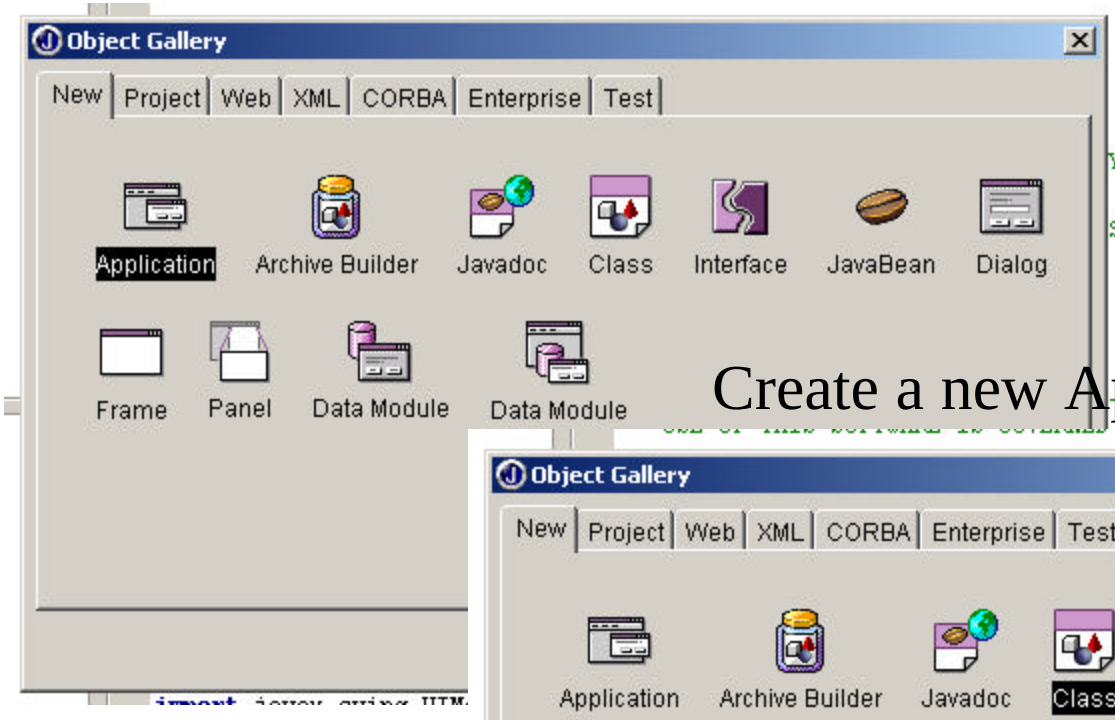
JBuilder IDE

- JBuilder is my personal favorite
- As-you-type syntax error notification
- Pop-up help indicating available methods
- Method parameter popping-up
- Automatic adding of missing bracket
- Very good help system
- Integrates Javadoc, CVS, UI development
- The version in the lab is older and not as good as the one you can download

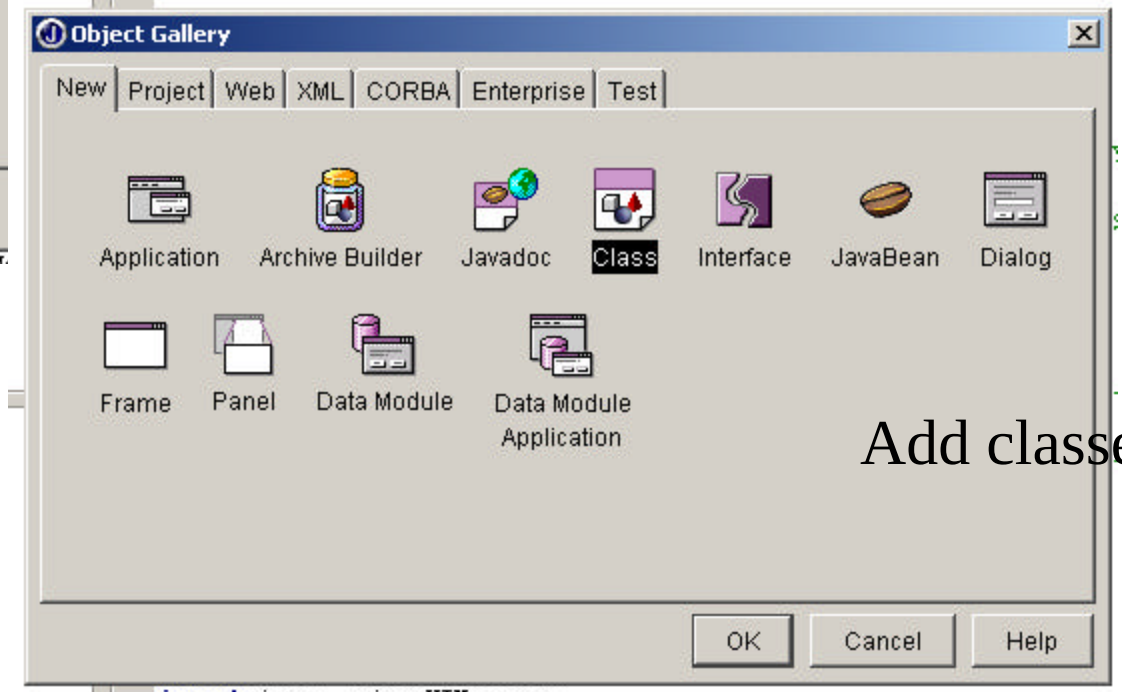
JBuilder IDE

- JBuilder provides many wizards to help you create projects, applications, classes…
- Tabbed editing view allows to simply handle multiple classes open at the same time

JBuilder IDE



Create a new Application

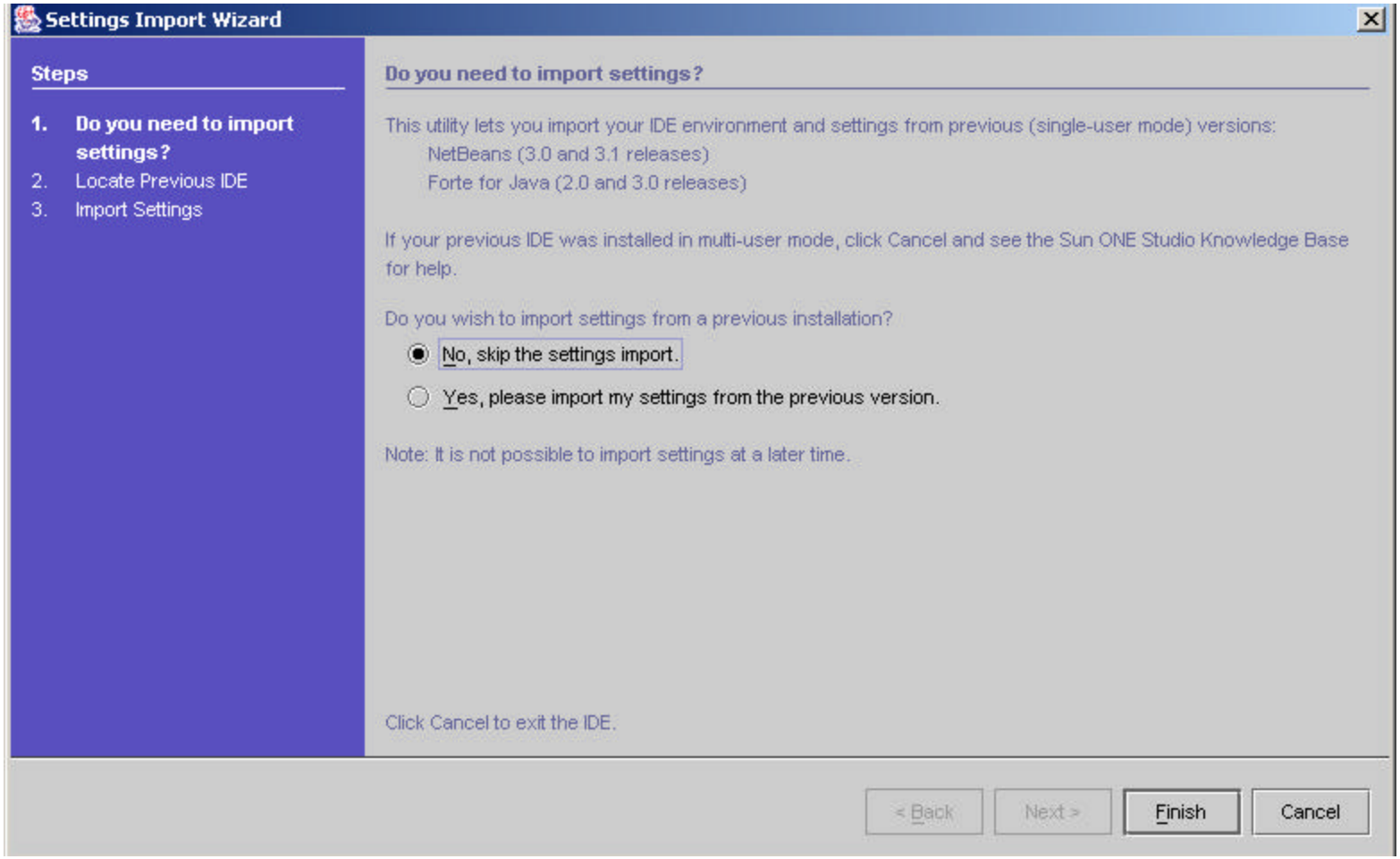


Add classes

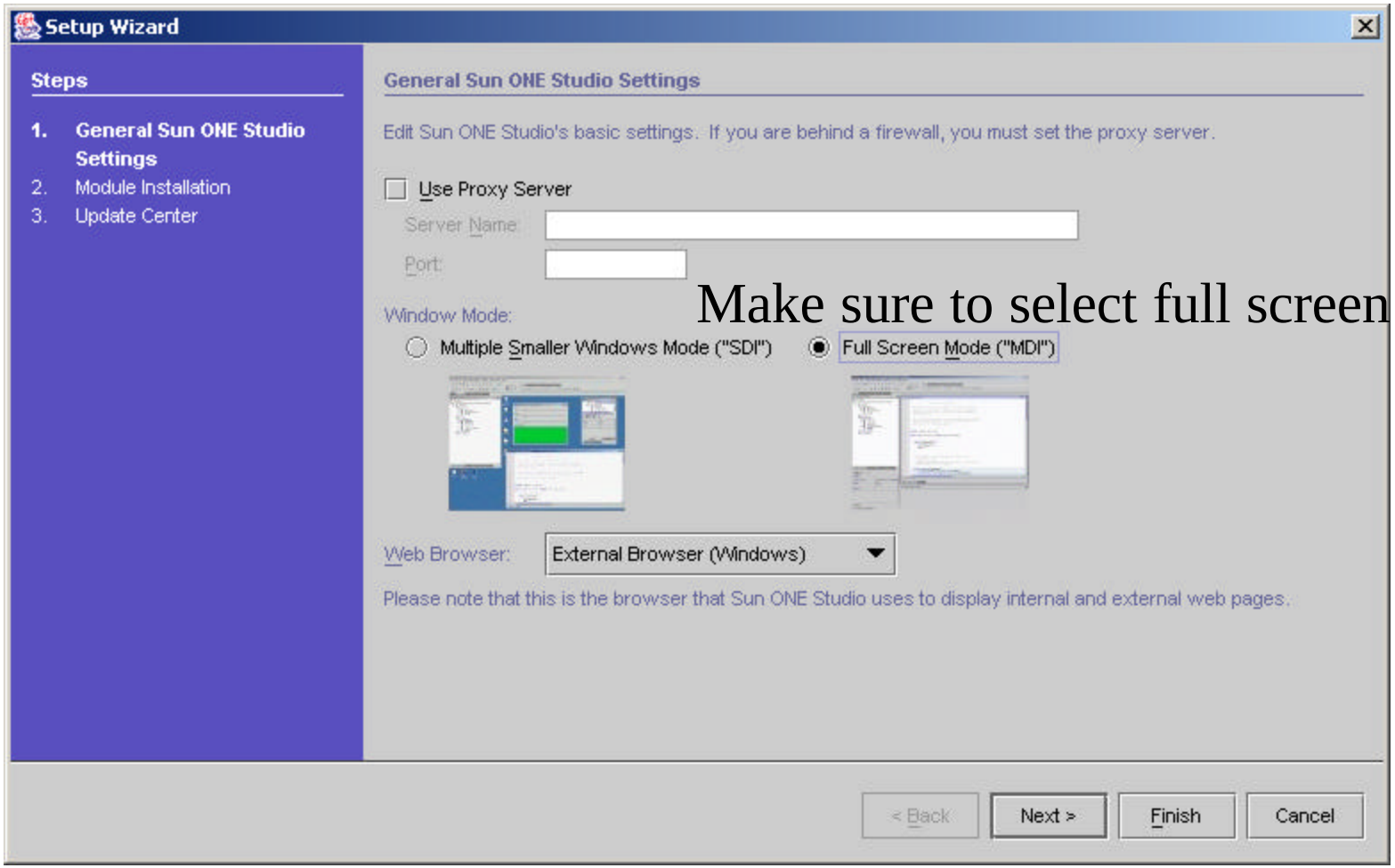
SunOne IDE

- Slower than JBuilder
- Clumsy help system
- If misconfigured, too many windows!
- Integrates CVS, Javadoc, UI development
- Large set of templates to choose from
- I’ll tell you more about it, since it’s the default tool in the labs

SunOne IDE

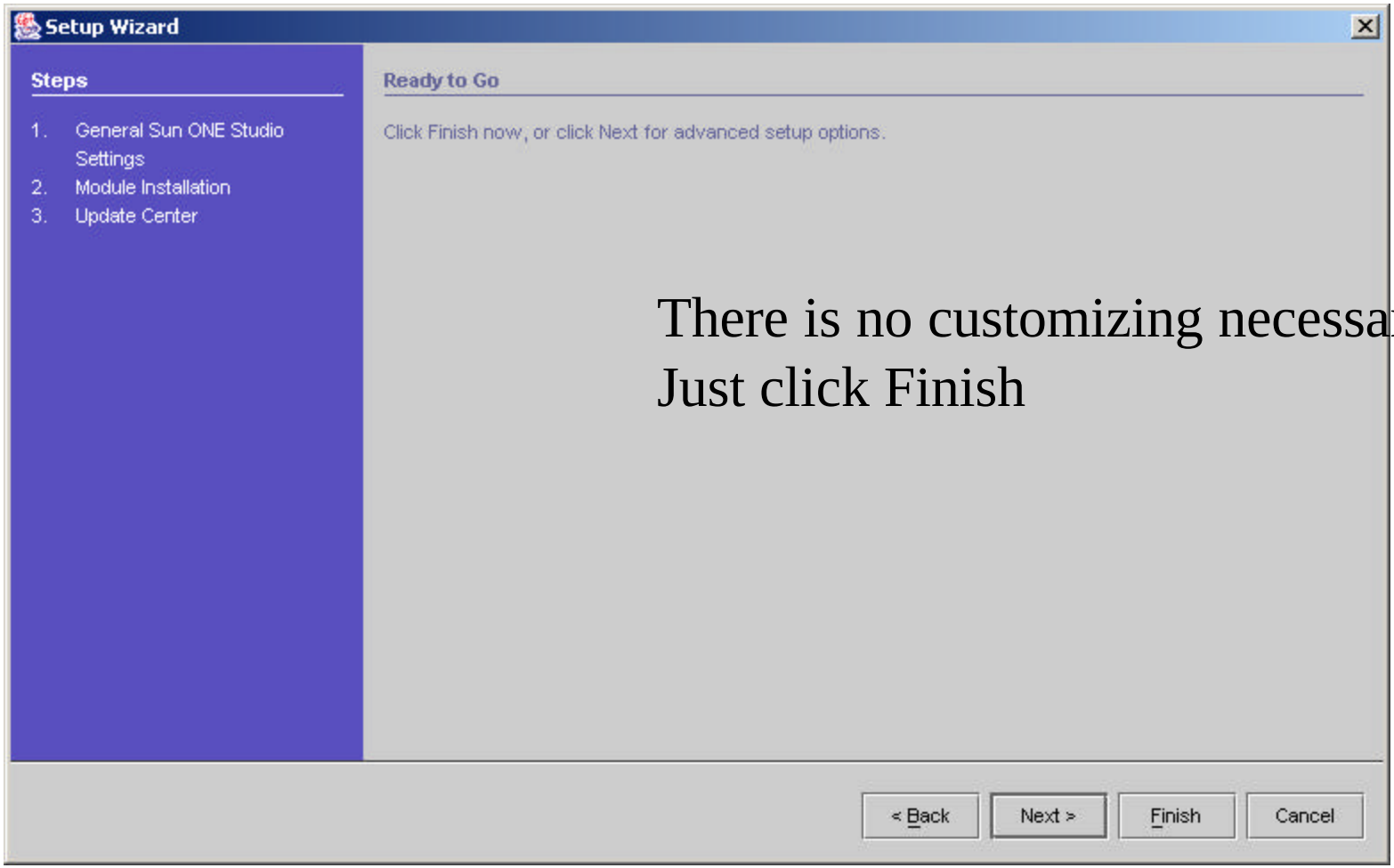

Settings Import Wizard
Steps
1. Do you need to import settings?
2. Locate Previous IDE
3. Import Settings
Do you need to import settings?
This utility lets you import your IDE environment and settings from previous (single-user mode) versions:
NetBeans (3.0 and 3.1 releases)
Forte for Java (2.0 and 3.0 releases)
If your previous IDE was installed in multi-user mode, click Cancel and see the Sun ONE Studio Knowledge Base for help.
Do you wish to import settings from a previous installation?
No, skip the settings import.
Yes, please import my settings from the previous version.
Note: It is not possible to import settings at a later time.
Click Cancel to exit the IDE.
< Back
Next >
Finish
Cancel

SunOne IDE



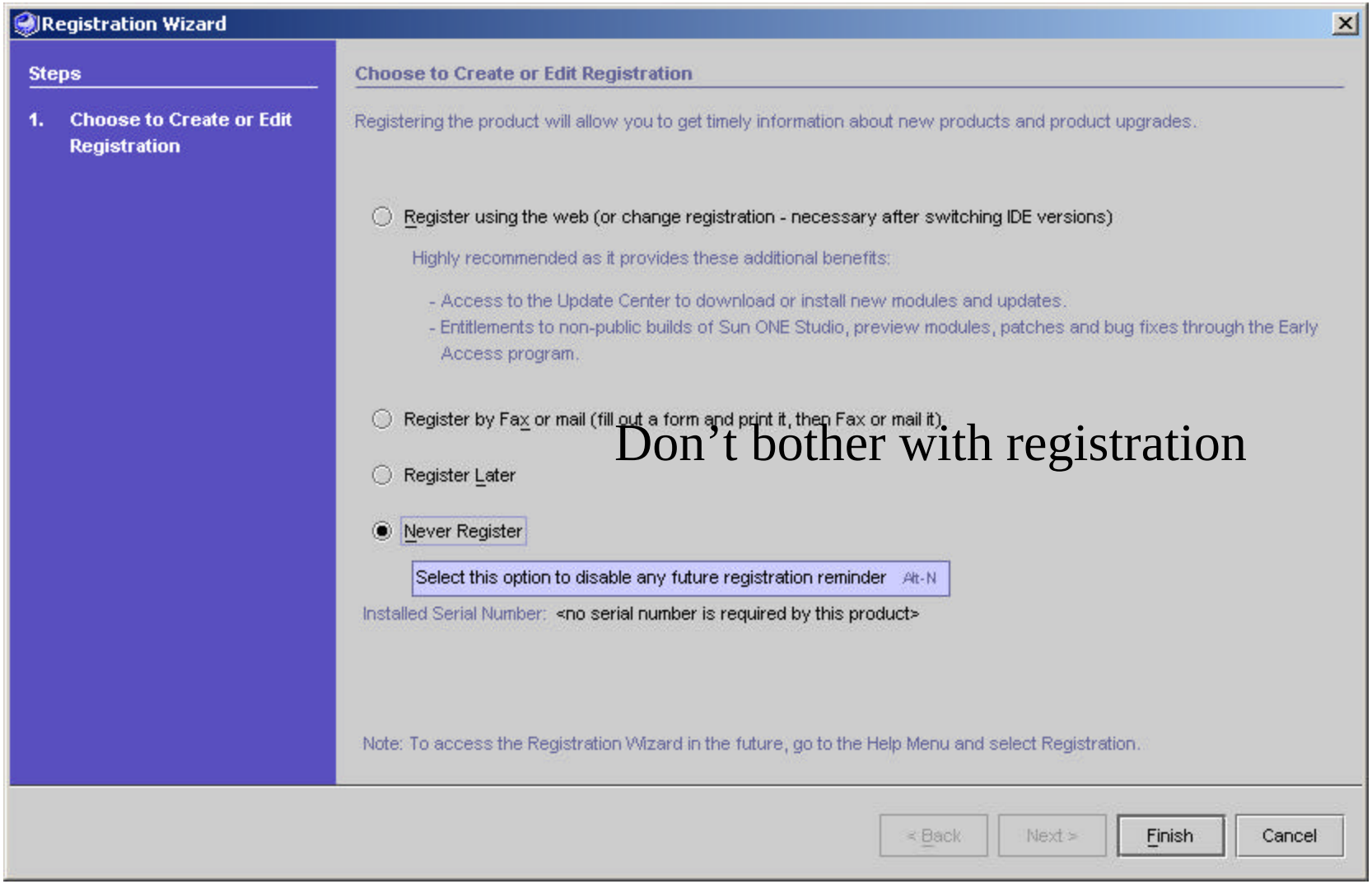
Make sure to select full screen mode

SunOne IDE



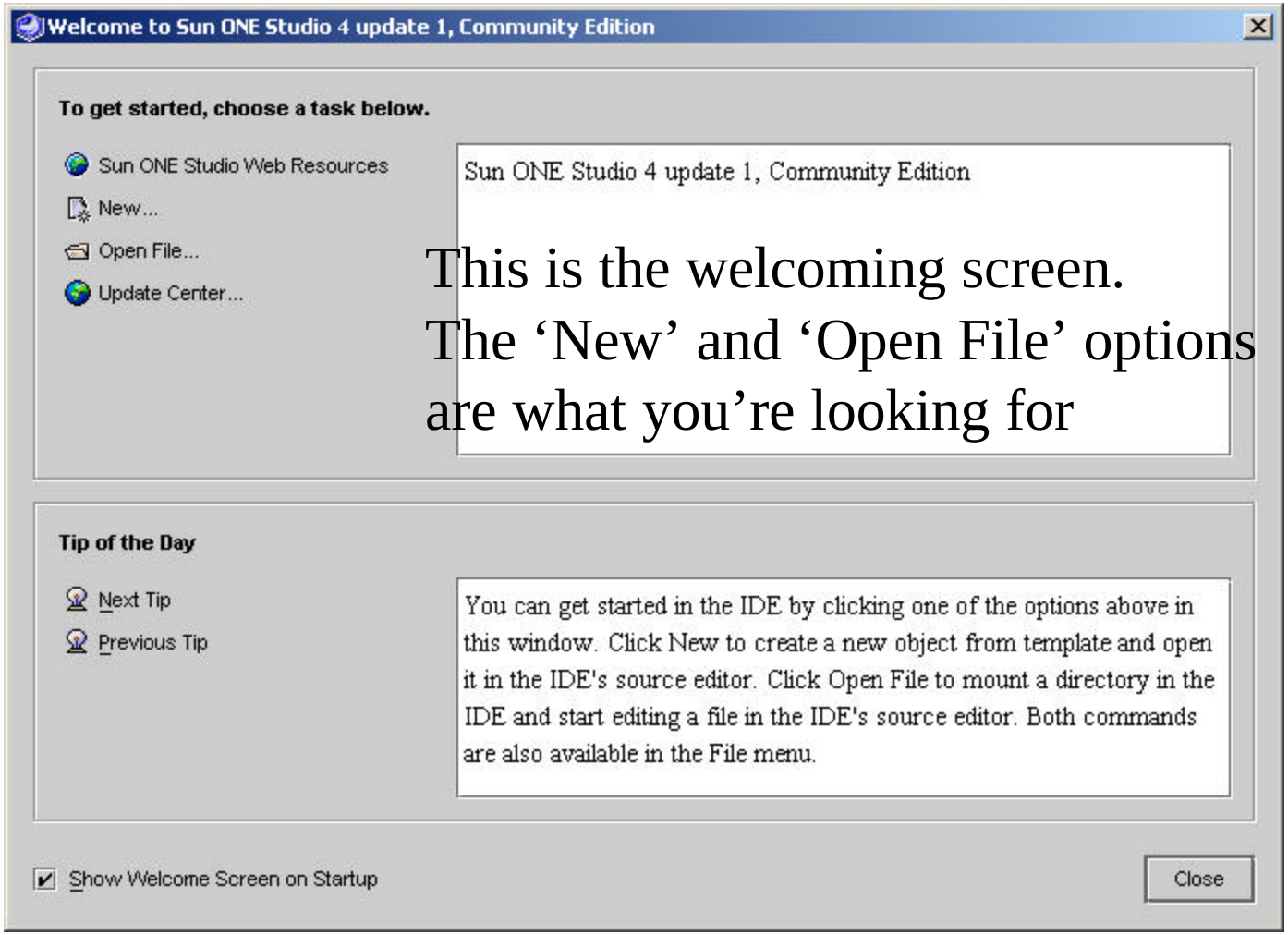
There is no customizing necessary.
Just click Finish

SunOne IDE



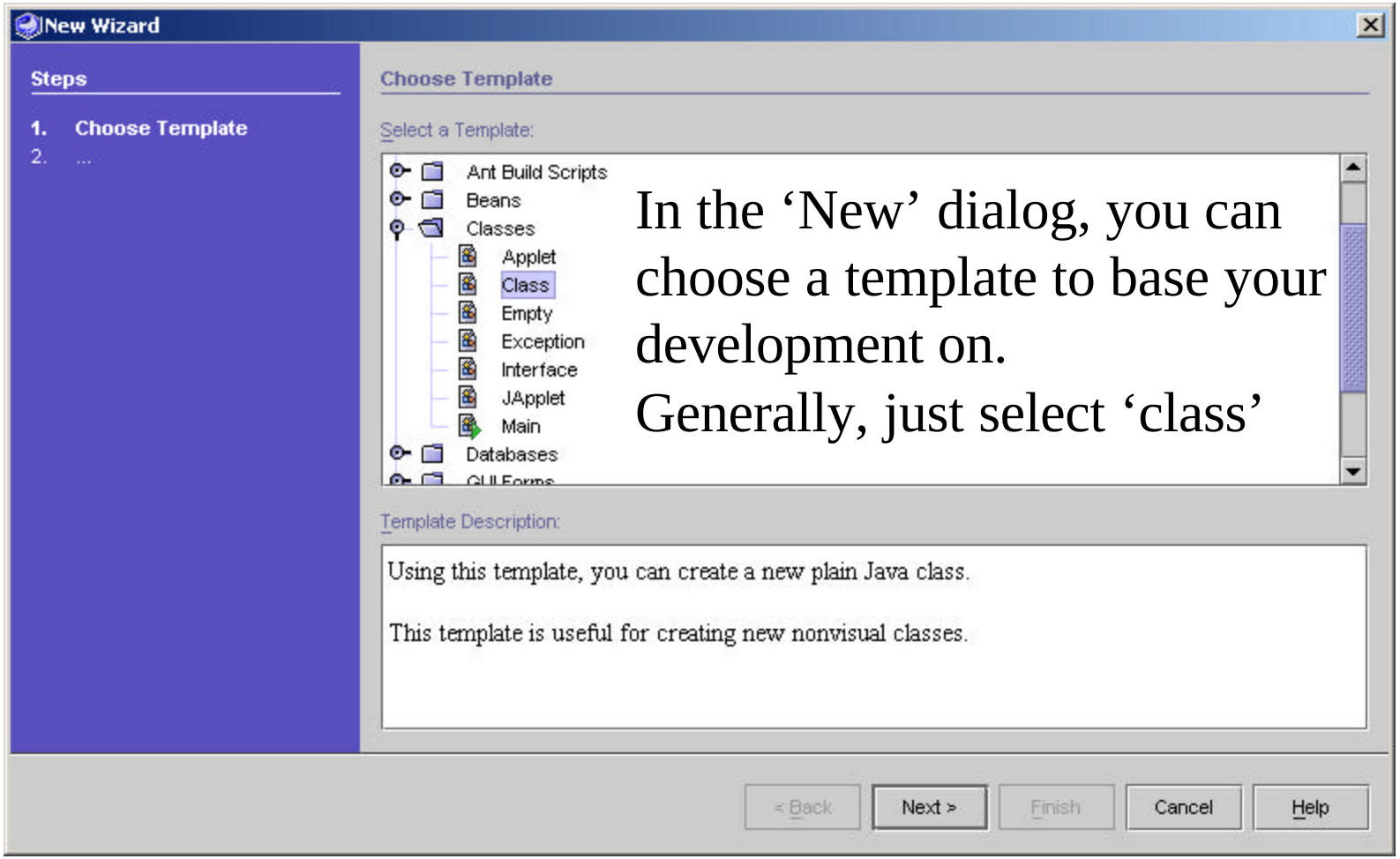
Don’t bother with registration

SunOne IDE

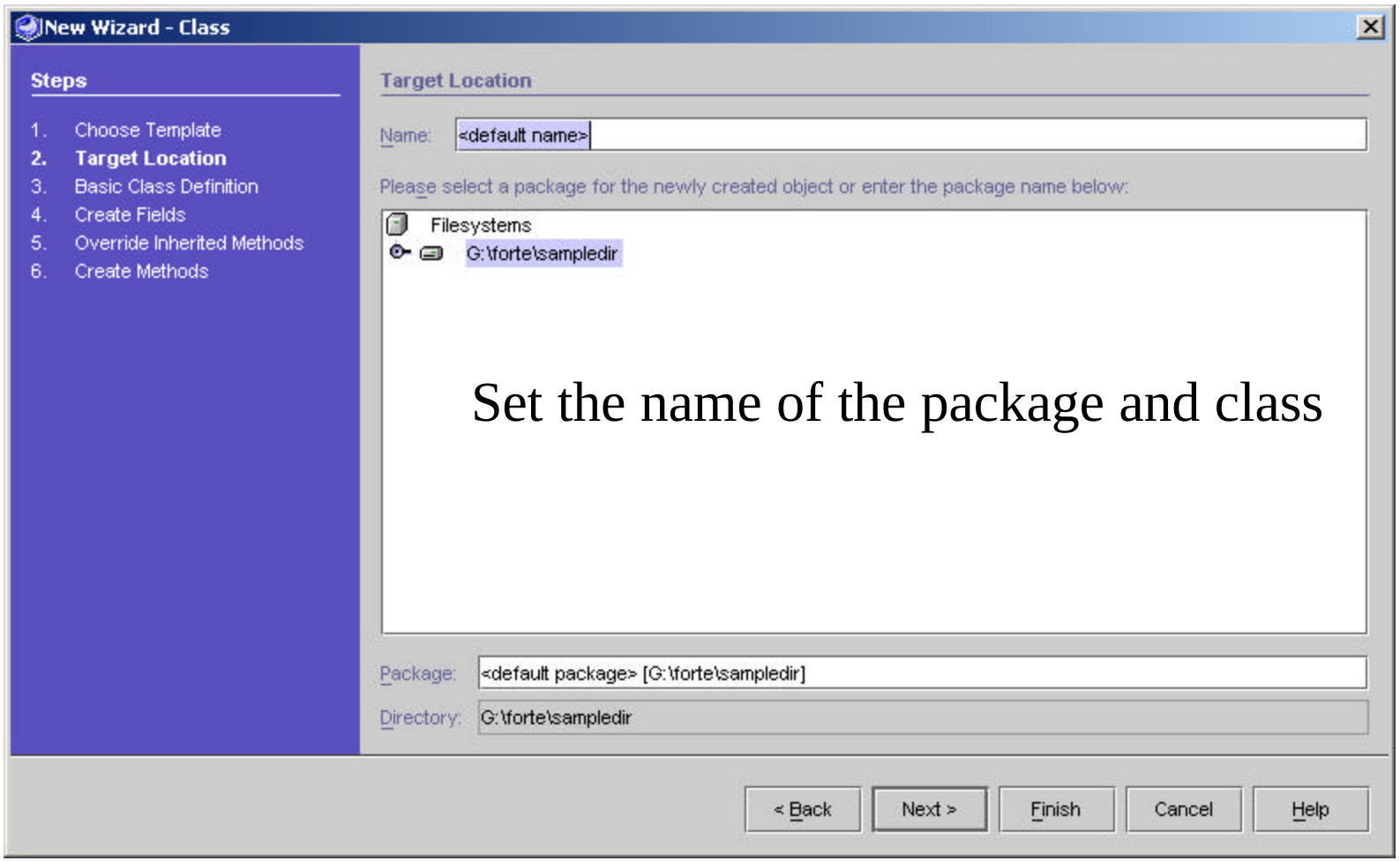


This is the welcoming screen. The ‘New’ and ‘Open File’ options are what you’re looking for

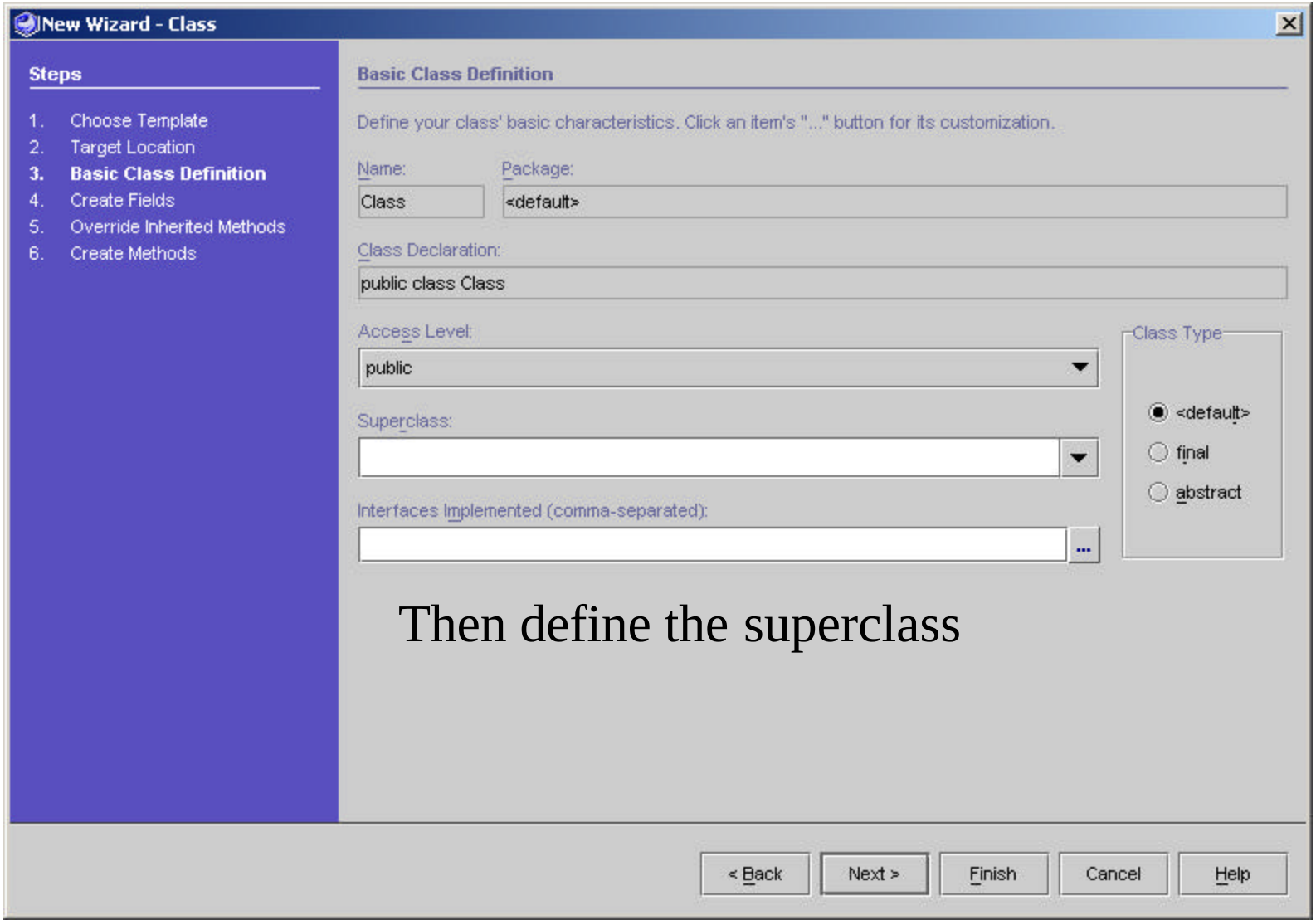
SunOne IDE


New Wizard
Steps
1. Choose Template
2. ...
Choose Template
Select a Template:
Ant Build Scripts
Beans
Classes
Applet
Class
Empty
Exception
Interface
JApplet
Main
Databases
In the ‘New’ dialog, you can choose a template to base your development on.
Generally, just select ‘class’
Template Description:
Using this template, you can create a new plain Java class.
This template is useful for creating new nonvisual classes.
< Back
Next >
Finish
Cancel
Help

SunOne IDE

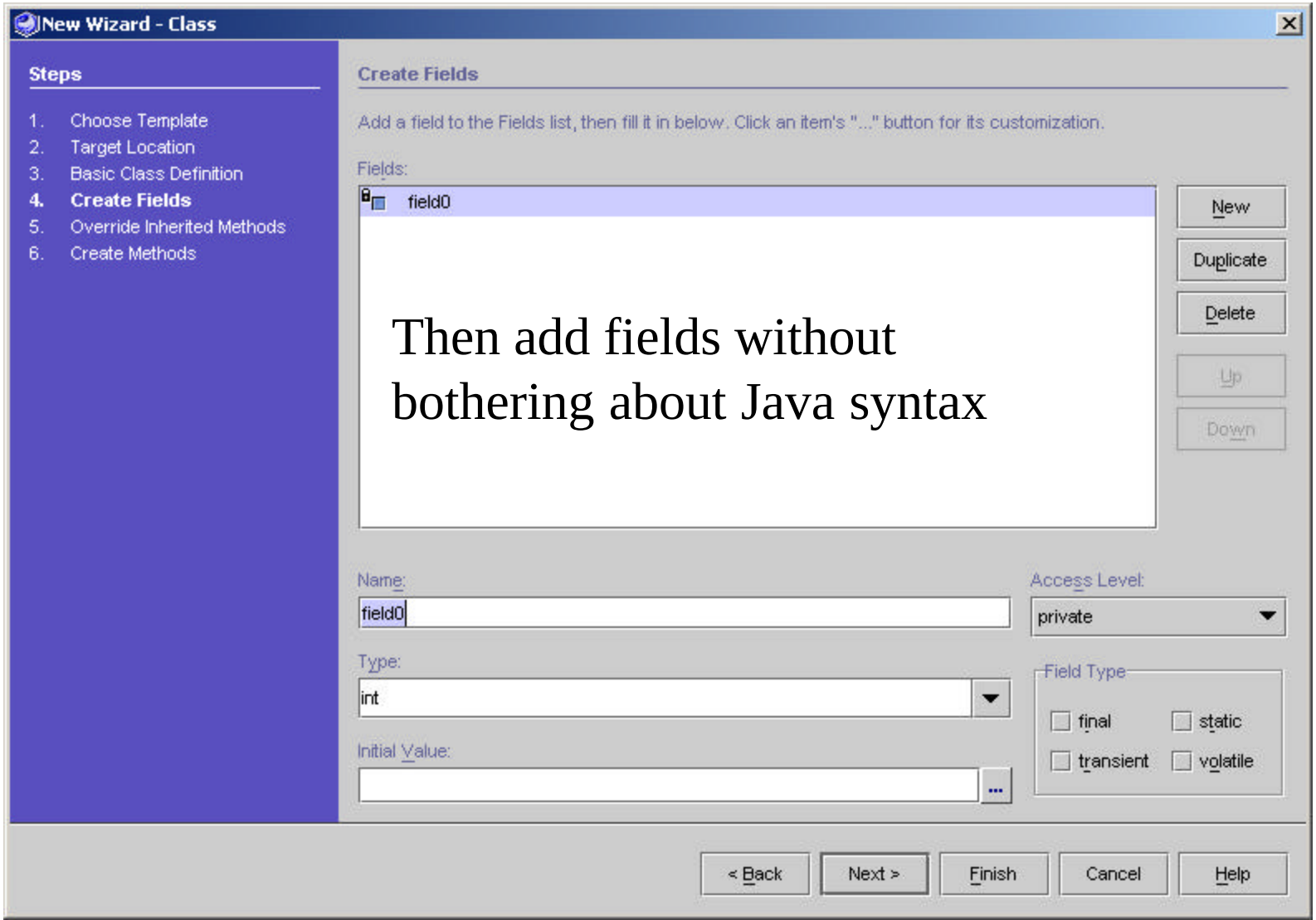

New Wizard - Class
Steps
1. Choose Template
2. Target Location
3. Basic Class Definition
4. Create Fields
5. Override Inherited Methods
6. Create Methods
Target Location
Name: <default name>
Please select a package for the newly created object or enter the package name below:
Filesystems
G:\forte\sampledir
Set the name of the package and class
Package: <default package> [G:\forte\sampledir]
Directory: G:\forte\sampledir
< Back
Next >
Finish
Cancel
Help

SunOne IDE

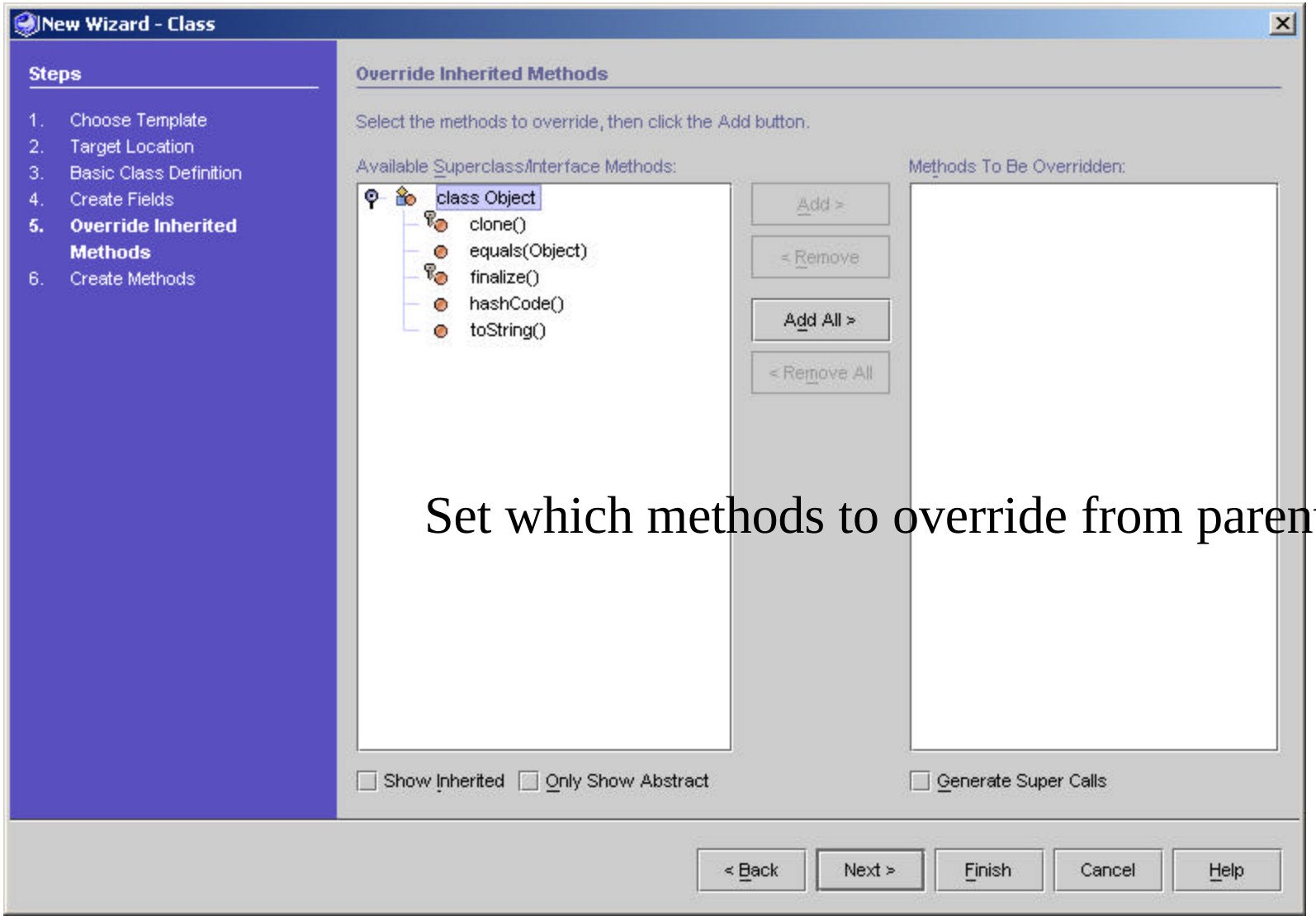


Then define the superclass

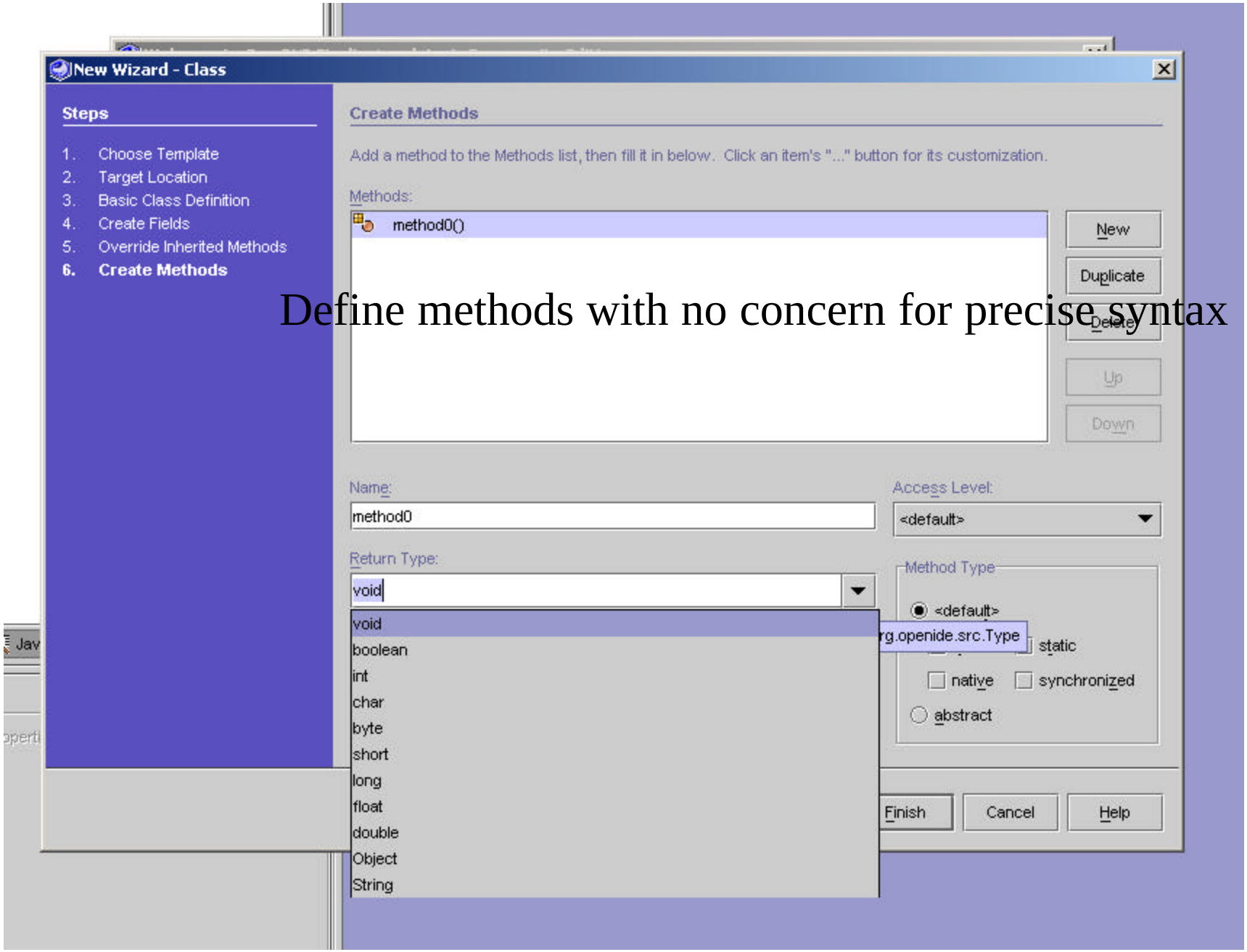
SunOne IDE


New Wizard - Class
Steps
1. Choose Template
2. Target Location
3. Basic Class Definition
4. Create Fields
5. Override Inherited Methods
6. Create Methods
Create Fields
Add a field to the Fields list, then fill it in below. Click an item's "..." button for its customization.
Fields:
field0
New
Duplicate
Delete
Up
Down
Then add fields without bothering about Java syntax
Name:
field0
Access Level:
private
Type:
int
Field Type
final
static
transient
volatile
Initial Value:
...
< Back
Next >
Finish
Cancel
Help

SunOne IDE

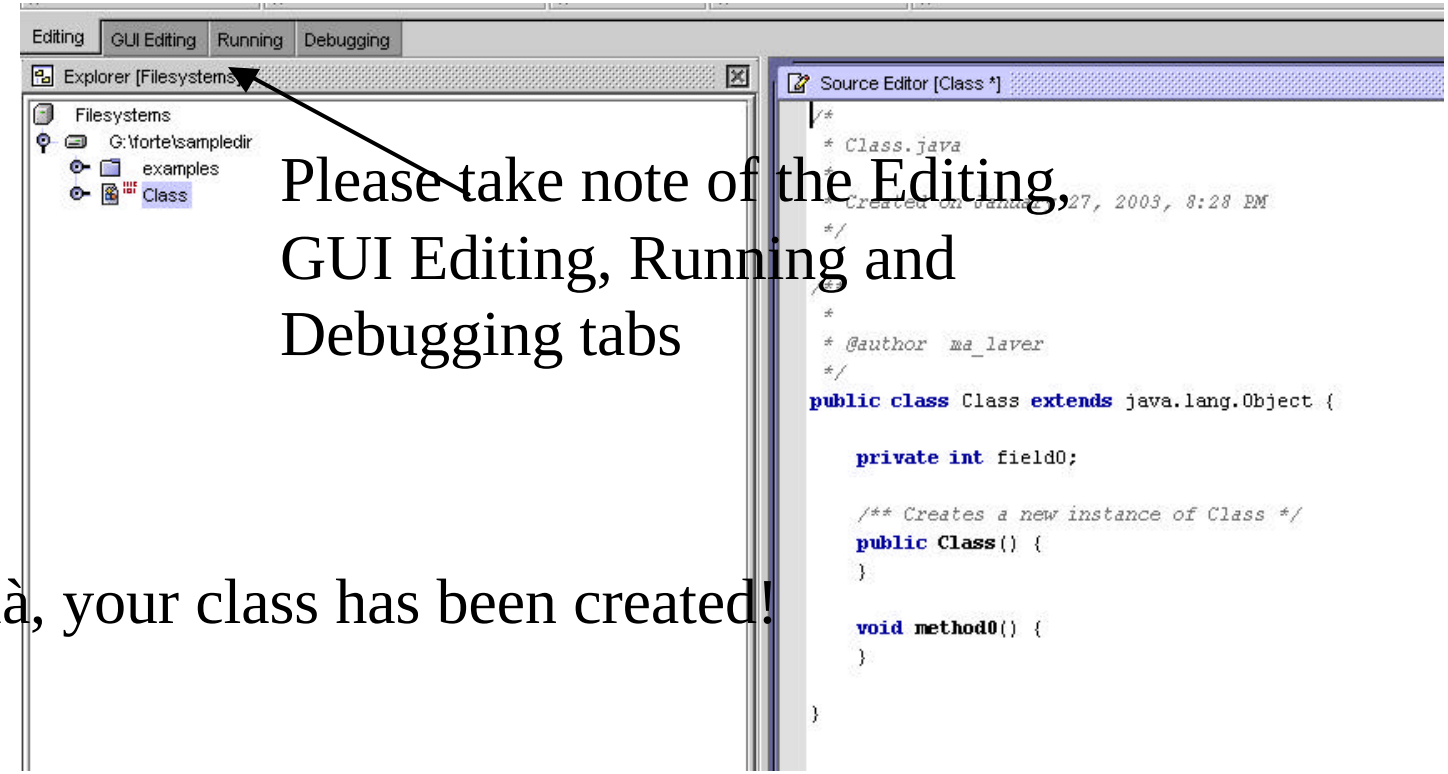


Set which methods to override from parent



Define methods with no concern for precise syntax

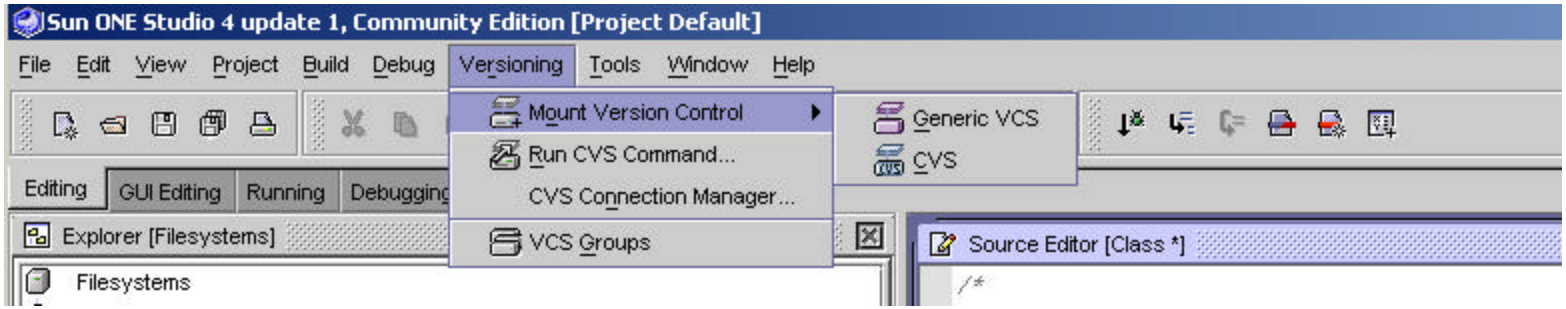
SunOne IDE



Please take note of the Editing, GUI Editing, Running and Debugging tabs

Voilà, your class has been created!

SunOne IDE



You can set CVS directly, so you don't need to bother learning CVS commands

In the labs, CVS repositories are located on the u:\ drive

SunOne IDE

Generating Javadoc is automated.

However, displaying th generated javadoc is nc straightforward.

The javadoc is stored o g:\forte\javadoc for the default configuration.