Lesson plan for Gries and Gries's *ProgramLive*

This lesson plan divides a course into six units, each of which requires about two weeks of work, more or less. There are five optional units, on multidimensional arrays, exception handling, interfaces, recursion, and applets, each of which takes about a week to cover.

Below, we give an overview of each unit followed by a checklist for the activities in it. Check off each item as you complete it.

Unit 1. Introduction

This first part of this unit introduces the mechanics of a livetext. Do spend half an hour on lesson 0 of *ProgramLive*, learning about the features of the livetext. In addition to the activities, it offers a plethora of learning tools such as the glossary, index, exercises and labs.

The second part of this unit provides a look at the programming language Java. This may be your first contact with a programming language, and you will see lots of new terminology and concepts. Don't expect to remember everything from this first look at Java. You will also learn how to run Java programs on the computer, using either an IDE (Interactive Development Environment) or a UNIX or PC command-line environment.

In learning to run Java programs, use material that is appropriate to the system you are using —perhaps lesson 18 (for IDE Visual Cafe), lesson 19 (for IDE CodeWarrior), Appendix B (for a command-line system), or whatever else is provided with your IDE.

☐ Introduction to livetexts	*PL* Lesson 0-1
☐ Activities	*PL* Lesson 0-2
☐ The lesson book page	*PL* Lesson 0-3
☐ Global features	*PL* Lesson 0-4
☐ Page controls	*PL* Lesson 0-5
☐ Dealing with Java programs	*PL* Lesson 0-6
☐ Learning effectively	*PL* Lesson 0-7
☐ Hardware and software	*PL* Lesson 1-1, page 41
☐ Some simple Java programs	*PL* Lesson 1-2, page 43

Pay attention to the filing cabinet metaphor, for it is a good analogy for understanding just what a class is. We use it throughout *ProgramLive*.

☐ Components of a Java program	*PL* Lesson 1-3, page 47

The sooner you learn the terminology introduced on this page, the better off you will be. The notions of *variable* and *expression* will probably not be new to you; you just have to learn how they are used in Java.

☐ Three statements *PL* Lesson 1-4, page 51
This is your first hard look at what it means to execute a statement, using three of Java's most common statements. Memorize what it means to execute these statements, and also become familiar with the statement-comment.

☐ Conventions for indentation *PL* Lesson 13-3, page 189
You need not listen to the activities now. Just read the lesson page and the two footnotes concerning the if-statement and the if-else-statement.

☐ Input/Output *PL* Lesson 1-5, page 56
Definitely listen to the first activity on output. Skip the second activity unless your instructor requires it; there is no need to know about it now. But the activity on `JLiveWindow` is important, because `JLiveWindow` will be used often, You will also want to listen to the activities on drawing lines, circles, etc., because that stuff is fun.

☐ Do Lab PGL-1 (assignment) of this lesson. *PL* Lesson 1

☐ Do Lab PGL-2 (if-statement) of this lesson. *PL* Lesson 1

☐ Do Lab PGL-3 (if-else-statement) of this lesson. *PL* Lesson 1

☐ Types *PL* Lessons 6-1, page 117, and 6-2, page 118
You already know that a type defines a set of values and operations on them. You can declare variables of type `int` and write simple expressions, and you can also write simple `String` expressions. At some point, you will want to study all of lesson 6 carefully, to get a more precise understanding of Java's primitive types. For now, just read these two lesson pages, and you can skip casting for now.

☐ Introduction to your IDE. Use whatever materials are available for your method of running Java programs.

Unit 2. Methods

This unit introduces you to a basic building block of programs, the method. A method is like a recipe: a sequence of instructions to be executed to get something done. You will be writing methods as well as reading them, so we want to give you some insight into how to write them and test them. In this unit, then, we introduce you to five things:

1. The "assertion" as a way of understanding methods,
2. The definition of methods,
3. Statements that call a method in order to get its task performed,
4. Top-down programming: a strategy for developing methods, and
5. How to test a method.

We also ask you to look at a few activities in *ProgramLive* on programming style, which, for organizational purposes, are in lesson 13.

☐ Good programming practices — *PL* Lesson 13-1, page 185
This is just a short essay for your reading enjoyment.

☐ Assertions in programs — *PL* Lesson 1-6, page 58

☐ Methods — *PL* Lesson 2-1, page 61
A method is just a recipe for doing something. Here, you learn the "user's" view of a method, i.e. the view of someone who wants to have a method executed.

☐ Method bodies and method calls — *PL* Lesson 2-2, page 64
We now look more closely at procedure bodies, and we also define precisely how a procedure call is executed. Memorize the steps!

☐ Two components of method bodies — *PL* Lesson 2-3, page 67
The local variable and the return statement can ease the task of writing a procedure.

☐ Functions — *PL* Lesson 2-4, page 70
The *function* is another kind of method. A procedure call is a statement; a function call is an expression.

☐ Testing and debugging — *PL* Lesson 14-1, page 193
This lesson page introduces terminology concerning testing and debugging and gives you some guidelines that, if followed, will reduce the chance of your programs having errors. The single activity on this page uses an example that contains a loop, which you don't need to know about yet. Skip the example.

☐ Testing strategies — *PL* Lesson 14-2, page 194
Listen only to activities 1 and 2 and concentrate mainly on the first one, unless your instructor tells you to concentrate on the second one.

☐ Debugging — *PL* Lesson 14-4, page 196
The two activities will give you some idea about how to track down bugs.

☐ Do Lab PGL-4 (writing simple functions) of this lesson. — *PL* Lesson 2

☐ Naming conventions — *PL* Lesson 13-2, page 185
Read the lesson page and listen to only the first activity on naming parameters, the second activity on naming local variables, and the second-last activity on naming methods.

☐ Conventions for indentation — *PL* Lesson 13-3, page 189
Listen to the two activities and read the footnote for indenting a method body.

☐ Guidelines for writing methods — *PL* Lesson 13-4, page 190
The first two activities try to convince you of the importance of method specifications. Get in the habit of writing a spec first! The last three activities provide insight on how to structure method bodies when they get long.

☐ Do Lab PGL-1 (Statement-comments) of this lesson. — *PL* Lesson 2

☐ Top-down programming — *PL* Lesson 2-5, page 72

Unit 3. Classes

This unit introduces you to some fundamental concepts of object-oriented programming and their realization in Java. As mentioned earlier, the *class* is a basic building block in Java. In our examples, a class corresponds to a drawer of a filing cabinet. The class is used to describe two things:

1. Class variables and methods, which go into the drawer, and
2. Objects, which also go into the drawer when they are created.

You will see how to create and use objects.

☐ Classes — *PL* Lesson 3-1, page 75
This lesson page introduces the first use of a class. There is a description of class methods and variables, which go in its file-cabinet drawer.

☐ Class `Math` — *PL* Lesson 3-2, page 77
Class `Math` contains useful static methods (i.e. class methods) and static fields (i.e. class fields). There's nothing to "memorize" here; just become familiar with the items in class `Math`, so that you can find and use them when you need them.

☐ Classes and objects — *PL* Lesson 3-3, page 79
This lesson page begins the discussion of the second use of a class: as a description of or template for objects of the class.

☐ Creating and initializing objects. — *PL* Lesson 3-4, page 82
Static methods and field are in the class drawer when execution of a program begins. Objects are created (and stored in the drawer) during execution of the program.

☐ Class `String` — *PL* Lesson 5-3, page 106
Lesson page 3-4 has an activity on creating objects of class `String`. This is a good time to become thoroughly familiar with this class, by studying Lesson page 5-3.

☐ Scope boxes and constructors — *PL* Lesson 3-5, page 85
The introduction of classes forces us to introduce scope boxes into the model of execution. The constructor, a special kind of method, is used to initialize the fields of objects.

☐ Do Lab PGL-1 (Writing constructors) of this lesson. — *PL* Lesson 3

☐ Nonstatic methods — *PL* Lesson 3-6, page 87
Static methods go in the class drawer; nonstatic methods, or *instance methods*, belong in each instance of the class.

☐ Do Lab PGL-2 (Drawing objects) of this lesson. — *PL* Lesson 3

☐ Consequences of using objects — *PL* Lesson 3-7, page 90
We (1) extend the model of execution to include calls on instance methods, (2) talk about equality of object names, as opposed to equality of objects, and (3) introduce, as a convention, method `toString`.

☐ Do Lab PGL-3 (Drawing frames) of this lesson. — *PL* Lesson 3

☐ Style considerations concerning classes *PL* Lesson 13-2, page 185
Listen only to the third activity (p. 187) on naming instance variables and class variables; the last activity (p. 188) on naming classes; and the footnote on lesson page 13-3 (p. 189) on indenting components of a class.

☐ Describing variables *PL* Lesson 13-5, page 191

☐ Testing strategies *PL* Lesson 14-2, page 194
Listen only to the activity on using assertions (p. 195), and read the footnote at the bottom of the page.

☐ Numerical wrapper classes *PL* Lesson 5-1, page 105
Now that you know about classes, you can learn about the "wrapper classes" for the primitive types. This lesson page describes wrapper class `Integer`, which wraps an **int**. The wrapper classes for the other numerical types are similar; refer to the footnotes for them when you need them. Lesson page 5-2 describes wrapper classes `Boolean` and `Character`. You don't have to look at them now, but remember their existence and look at them when you need them.

☐ Object-oriented design *PL* Lesson 3-8, page 93

Unit 4. Subclasses

This unit introduces subclasses and superclasses, inherited methods, overriding methods, and casting an object. This will require some preparatory study of primitive types and casting. After studying this unit, you will know all the basics of object-oriented programming in Java (except for interfaces) and will be able to understand just about all parts of Java programs —finally!

This material requires the idea of casting, so some material in lesson 6 on types is included here. If you have already covered this material, skip it.

☐ The integral types *PL* Lesson 6-2, page 118
The important part here is the conversion (casting) from one type to another. Study mainly the activities on promoting values to a wider type (p. 119) and casting integer values (p. 119).

☐ Do Lab PGL-2 (Casting among integral types) of this lesson. *PL* Lesson 6

☐ Subclasses *PL* Lesson 4-1, page 97
The subclass is an important tool in object-oriented programming. Without it, our tools for organizing programs would be very limited.

☐ Do Lab PGL-1 (Drawing objects II) of this lesson. *PL* Lesson 4

☐ Constructors and inherited methods *PL* Lesson 4-2, page 99
This lesson page introduces a variety of concepts that stem from having subclasses. The final activity tells you about the important class `Object`, the "superest" class of them all.

- ☐ Do Lab PGL-2 (Writing constructors II) of this lesson. *PL* Lesson 4
- ☐ Casting and method calls *PL* Lesson 4-3, page 101
  The idea of "casting" an object of a subclass to a superclass but still having the overriding methods of the subclass be used is extremely important for object-oriented programming. Be sure you understand it. At this point, we can give the final model of execution.
- ☐ Do Lab PGL-3 (Drawing frames II) of this lesson. *PL* Lesson 4
- ☐ Selecting test cases and checking them *PL* Lesson 14-3, page 195
  We finish the lesson on testing and debugging with a study of different kinds of testing.
- ☐ (Optional) Do Lab PGL-3 (Formatting in locales) of this lesson. *PL* Lesson 6
- ☐ Object-oriented design with subclasses *PL* Lesson 4-4, page 103
- ☐ (Optional) Abstract classes *PL* Lesson 4-5, page 104
  Although this lesson page is optional, it is short, and we recommend that you study it.
- ☐ (Optional) Do Lab PGL-4 (Practice with shapes) of this lesson. *PL* Lesson 4

Unit 5. Loops

To iterate means to repeat over and over again. Well, then, to reiterate should mean to do it again, that means, to again repeat over and over again. So it goes, with English.

In Java (and other programming languages), execution of a "loop" causes a statement, called its "repetend" or "body", to be executed over and over again. The loop is an extremely useful statement, but it is much harder to understand then the statements that you have learned already. That's why a complete unit is devoted to it.

- ☐ Iteration (only the first activity) *PL* Lesson 7-1, page 127
  This activity takes you through one execution of a **while** loop. It introduces some terminology, and it shows you a flaw chart that describes how a loop is executed. Memorize this material before proceeding!
- ☐ Do Lab PGL-1 (Executing a while loop) of this lesson. *PL* Lesson 7
- ☐ Iteration (all but the first activity) *PL* Lesson 7-1, page 127
  Knowing how a loop is executed is not enough; you have to know how to understand what happens within a loop, and you have to be able to explain a loop to others. This requires the notion of a loop invariant. Study this material carefully.
- ☐ Several examples of loops *PL* Lesson 7-2, page 132
  The first activity discusses the development loop invariants. The rest of them develop three algorithms. Study them with an eye to understanding the development process. You may want to obtain the spiral program and play with it.

☐ Do Lab PGL-2 (Developing loops from invariants) of this lesson. *PL* Lesson 7

☐ (Optional) Do Lab PGL-3 (Developing loops . . . II) of this lesson. *PL* Lesson 7

☐ Conventions for indentation *PL* Lesson 13-3, page 189
Read the footnote on indenting loops, near the bottom of the lesson page.

☐ Loop schemata *PL* Lesson 7-3, page 134
Rather than write each loop from scratch, learn to use loop schemata.

☐ Do Lab PGL-4 (Using loop schemata) of this lesson. *PL* Lesson 7

☐ The **for** loop *PL* Lesson 7-4, page 136
The **for** loop is an abbreviation of a **while** loop that uses a "loop counter". It is extremely useful when the number of iterations to perform is known before execution of the loop. You'll see lots of **for** loops in Java programs.

☐ (Optional) Do Lab PGL-4 (Translating **while**s into **for**s). *PL* Lesson 7

☐ Making progress and stopping *PL* Lesson 7-5, page 138
Many people will get by without studying this lesson page. However, you will have a much better understanding of loops if you study it carefully.

☐ Miscellaneous points about loops *PL* Lesson 7-6, page 140
Do read the warning note at the top of the page. It will take only a few seconds.

The first two activities illustrate an important use of the statement-comment: abstraction helps us say that we *never* think of nested loops (well, . . .).

You need *not* study the information about the do-while loop, the **continue** statement, and the **break** statement. In general, these constructs are not needed at this point of your programming career. If you see them in a program and want to find out about them, look in the *ProgramLive* glossary or index.

Unit 6. Arrays

An array is a collection of elements of the same (primitive or class) type —`int`, `String`, `JLiveWindow`, etc. Arrays are used in many situations; for example, to hold a list of courses offered by a college.

☐ Introduction to arrays *PL* Lesson 8-1, page 143
This lesson page introduces all the technical details concerning arrays.

☐ Talking about array segments *PL* Lesson 8-2, page 146
The notation described here, including pictures, makes it easier to discuss algorithms that manipulate arrays.

☐ Some programs that use arrays *PL* Lesson 8-3, page 148
Activities 1 and 2 develop two useful schemata for processing arrays. You will use them often. The next five activities develop algorithms that manipulate arrays. Study them all! Particularly important are the last two; they show you how to test for array equality and show you that a function can return an array.

☐ Do Lab PGL-1 (Using arrays) of this lesson. *PL* Lesson 8

☐ Arrays and classes —a student report *PL* Lesson 8-4, page 153
The first two activities develop a first application of arrays, showing how arrays are actually used.

☐ (Optional) Arrays and classes —dynamic arrays *PL* Lesson 8-4, page 153
You may also want to look at class `Vector` on lesson page 5-5 (p. 111).

☐ Some basic array algorithms *PL* Lesson 8-5, page 156
Eight basic algorithms on arrays are developed on this page. You need not study them all at this point. At a minimum, though, study the first activity on finding the first value, the activities on finding the minimum value, partitioning an array segment, and the important activity on binary search.

☐ Selection sort and Insertion sort *PL* Lesson 8-6, page 161
Do study the two activities on Selection sort. Insertion sort is optional.

☐ (Optional) Do Lab PGL-2 (Timing execution) of this lesson. *PL* Lesson 8

Unit 7. Multidimensional Arrays (optional)

This unit, which is optional, shows you how you can have two- and three-dimensional (and more) arrays in Java. You can live without this material in your first exposure to programming and Java.

☐ Multidimensional arrays *PL* Lesson 9-1, page 163
This lesson page presents the technical details you need to work with two- and three-dimensional arrays. It also presents a non-Java notation that makes talking about multidimensional arrays easier.

☐ Programs that use two-dimensional arrays *PL* Lesson 9-2, page 165
Don't skip the second activity on the page, which develops a surprisingly simple algorithm. Finding a loop invariant first and following the methodology for developing loops that was explained in lesson 7 leads to an algorithm that otherwise is extremely difficult to discover.

☐ Do Lab PGL-1 (Rectangular arrays) of this lesson. *PL* Lesson 9

☐ The Java concept of a multidimensional array *PL* Lesson 9-3, page 168
You will discover that we didn't give the whole story on lesson page 9-1. For example, you will see that a two-dimenssional array is really an array whose elements are arrays —which can be of different lengths!

☐ Programs that use ragged arrays *PL* Lesson 9-4, page 170

Unit 8. Exception handling (optional)

This material is optional.

☐ Output of thrown Exceptions and Errors — *PL* Lesson 10-1, page 173
Explains some basic error messages.

☐ The throwable object. — *PL* Lesson 10-2, page 173
Shows how to pass an error message out of a nested method call without crashing the program.

☐ Catching a thrown exception — *PL* Lesson 10-3, page 174
Deals with exceptions within your own code.

☐ The throw-statement — *PL* Lesson 10-4, page 175
Tell what to do when your program encounters some problems and how to tell the user which problem occurred.

☐ Checked exceptions and the throws clause — *PL* Lesson 10-5, page 176

☐ Hints on using exceptions — *PL* Lesson 10-5, page 176

Unit 9. Interfaces (optional)

This material is optional.

☐ Interfaces — *PL* Lesson 12-1, page 179
Introduces the interface and its implementation.

☐ The interface as a type — *PL* Lesson 12-2, page 180
Classes can implement more than one interface, and interfaces can be extended to cover more methods.

☐ Interface `Comparable` — *PL* Lesson 12-3, page 182

☐ Interfaces `Enumeration` and `Iterator` — *PL* Lesson 12-4, page 183
An advanced topic.

Unit 10. Recursion (optional)

Anything you can do with iteration you can do with recursion (methods that can call themselves), and in many cases recursion will be the simpler tool to use. You already have all the technical tools you need to understand recursion; there is really nothing new about it.

☐ Recursion *PL* Lesson 15-1, page 197
This lesson page discusses what it means for a method to call itself, develops three recursive methods, and talks about the "recursive pattern". Understand this pattern, and you understand recursion.

☐ Execution of calls on recursive methods (optional) *PL* Lesson 15-2, page 200
Do only the first activity. You will see that you already know how to execute recursive calls by hand. The model of execution already discussed needs no changing.

☐ Execution of calls on recursive methods *PL* Lesson 15-2, page 200
Java doesn't execute recursive methods as efficiently as they might be —Java wastes space. The activities after the first one are optional. They tell you about tail recursion and show you how to change a method to eliminate tail-recursive calls, thus saving the space that they require for frames. It's neat stuff, but not necessary at this point.

☐ Interesting recursive methods *PL* Lesson 15-3, page 202
Of the three algorithms developed here, "Tiling Elaine's kitchen" is the most impressive. Recursion can make a seemingly impossible problem appear relatively simple.

☐ Quicksort *PL* Lesson 15-4, page 202
Quicksort is the most famous and most widely used sorting algorithm. Every computer scientist and professional programmer should know it. The first version is relatively simple. In order to save space and make it more efficient, the first method has to be manipulated a bit.

Unit 11. Applets (optional)

An *application* is a Java program whose execution starts when the system calls method `main` of some class. An *applet* is a Java program whose execution starts when a browser (e.g. Netscape or Internet Explorer) loads an html page that contains a command to start the applet.

Some instructors prefer to use applications in their course; others, applets. For those who prefer applets, this Unit may be placed earlier in the lesson plan, perhaps after Unit 2 (Methods). This Unit on applets is short and should take only one lecture.

☐ Applets *PL* Lesson 16-1, page 205
Here, we show only what an applet looks like, not how it is called.

☐ HTML and applet commands *PL* Lesson 16-2, page 206
Html is the language in which files that appear in a browser are written. This is a brief introduction to html.

☐ Examples of applets *PL* Lesson 16-3, page 208